HMMER User's Guide

Biological sequence analysis using profile hidden Markov models

http://hmmer.wustl.edu/
Version 2.3.1; June 2003

Sean Eddy
Howard Hughes Medical Institute and Dept. of Genetics
Washington University School of Medicine
660 South Euclid Avenue, Box 8232
Saint Louis, Missouri 63110, USA
http://www.genetics.wustl.edu/eddy/
With contributions by Ewan Birney (*birney@sanger.ac.uk*)

Contents

1 Introduction

HMMER is an implementation of profile hidden Markov models (profile HMMs) for biological sequence analysis (Krogh et al., 1994; Eddy, 1998; Durbin et al., 1998).

How to avoid reading this manual

I hate reading documentation. If you're like me, you're sitting here thinking, 95 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 95 exhausting pages of someone's documentation. For you cynics that have seen one too many software packages that don't work:

- Follow the quick installation instructions on page 9. An automated test suite is included, so you will know immediately if something went wrong.
- Go to the tutorial, section 3 on page 19, which walks you through some examples of using HMMER on real data.

Everything else, you can come back and read later.

What profile HMMs are

Profile HMMs are statistical models of multiple sequence alignments. 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 (Krogh et al., 1994), 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 (Churchill, 1989), 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" were introduced by Gribskov and colleagues (Gribskov et al., 1987; Gribskov et al., 1990), and several other groups introduced similar approaches at about the same time, such as "flexible patterns" (Barton, 1990), and "templates"(Bashford et al., 1987; Taylor, 1986). The term "profile" has stuck.[1] All of the profile methods 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 (Altschul et al., 1990), FASTA (Pearson and Lipman, 1988), or the Smith/Waterman algorithm (Smith and Waterman, 1981)) 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. For example, a profile HMM can be trained from unaligned sequences, if a trusted alignment isn't yet known. Another consequence is that HMMs have a consistent theory behind gap and insertion scores. In most details, profile

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

HMMs are a slight improvement over a carefully constructed profile – but less skill and manual intervention are necessary to use profile HMMs. This allows 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 (Sonnhammer et al., 1997; Bateman et al., 2002), which is a significant part of the Interpro protein domain annotation system (Mulder et al., 2003). The construction and use of Pfam is tightly tied to the HMMER software package.

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] HMMs make poor models of RNAs, for instance, because an HMM cannot describe base pairs. Also, compare protein "threading" methods, which usually include scoring terms for nearby amino acids in a three-dimensional protein structure.

Applications of profile HMMs

One 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 (Bateman et al., 2002) and SMART (Letunic et al., 2002) in the Interpro database consortium (Mulder et al., 2003). (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?

And 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 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 (Bateman et al., 2002) 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.

[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).

HMMER has reportedly been used to model music, speech, and automobile engine telemetry. If you use it for something particularly strange, I'd be curious to hear about it (but I never want to see my error messages showing up on the console of my Honda).

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. Some are listed in the table below.

Software	URL
HMMER	http://hmmer.wustl.edu/
SAM	http://www.cse.ucsc.edu/research/compbio/sam.html
PFTOOLS	http://www.isrec.isb-sib.ch/ftp-server/pftools/
HMMpro	http://www.netid.com/html/hmmpro.html
GENEWISE	http://www.ebi.ac.uk/Wise2/
PROBE	ftp://ftp.ncbi.nih.gov/pub/neuwald/probe1.0/
META-MEME	http://metameme.sdsc.edu/
BLOCKS	http://www.blocks.fhcrc.org/
PSI-BLAST	http://www.ncbi.nlm.nih.gov/BLAST/newblast.html

HMMER, SAM, PFTOOLS, and HMMpro are the most closely related to the profile HMM methods introduced by Krogh et al. HMMpro is commercial, not free software.

How to learn more about profile HMMs

Reviews of the profile HMM literature have been written by myself (Eddy, 1996; Eddy, 1998) and by Anders Krogh (Krogh, 1998).

For details on how profile HMMs and probabilistic models are used in computational biology, see the pioneering 1994 paper from Krogh et al. (Krogh et al., 1994) or our book *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids* (Durbin et al., 1998).

To learn more about HMMs from the perspective of the speech recognition community, an excellent tutorial introduction has been written by Rabiner (Rabiner, 1989).

> ▷ ***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.wustl.edu/. If you must use a paper reference, the best one to use is my 1998 profile HMM review (Eddy, 1998).*

2 Installation

Quick installation instructions

configuring, compiling, and installing a source code distribution

Download the source tarball (**`hmmer.tar.gz`**) from ftp://ftp.genetics.wustl.edu/pub/eddy/hmmer/ or http://hmmer.wustl.edu/.

Unpack the software:

```
> tar xvf hmmer.tar.gz
```

Go into the newly created top-level directory (named **`hmmer-xx`**, where **`xx`** is a release number):

```
> cd hmmer-2.3.1
```

Configure for your system, and build the programs:

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

Run the automated testsuite. This is optional. All these tests should pass:

```
> make check
```

The programs are now in the **`src/`** subdirectory. The man pages are in the **`documentation/man`** subdirectory. You can manually move or copy all of these to appropriate locations if you want. You will want the programs to be in your $PATH.

Optionally, you can install the man pages and programs in system-wide directories. If you are happy with the default (programs in **`/usr/local/bin/`** and man pages in **`/usr/local/man/man1`**), do:

```
> make install
```

(You might need to be root when you install, depending on the permissions on your /usr/local directories.)

That's all. Each of these steps is documented in more detail below, including how to change the default installation directories for **`make install`**.

configuring and installing a precompiled binary distribution

Alternatively, you can obtain a precompiled binary distribution of HMMER from http://hmmer.wustl.edu/. Thanks to generous hardware support from many manufacturers, binary distributions are available for most common UNIX and UNIX-like OS's. For example, the distribution for Intel x86/GNU Linux machines is **`hmmer-2.3.1.bin.intel-linux.tar.gz`**.

After you download a binary distribution, unpack it:

```
> tar xvf hmmer.bin.intel-linux.tar.gz
```

HMMER is now in the newly created top-level directory (named **`hmmer-xx`**, where **`xx`** is a release number). Go into it:

```
> cd hmmer-2.3.1
```

You don't really need to do anything else. The programs are in the **`binaries/`** subdirectory. The man pages are in the **`documentation/man`** subdirectory. The PDF copy of the Userguide is in the top level HMMER directory (Userguide.pdf). You can manually move or copy all of these to appropriate locations if you want. You will want the programs to be in your $PATH.

However, you'll often want to install in a more permanent place. To configure with the default locations (programs in **`/usr/local/bin/`** and man pages in **`/usr/local/man/man1`**) and install everything, do:

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

If you want to install in different places than the defaults, keep reading; see the beginning of the section on running the **configure** script.

System requirements and portability

HMMER is designed to run on UNIX platforms. The code is POSIX-compliant ANSI C. You need a UNIX operating system to run it. You need an ANSI C compiler if you want to build it from source.

Linux and Apple Macintosh OS/X both count as UNIX. Microsoft operating systems do not. However, HMMER is known to be easy to port to Microsoft Windows and other non-UNIX operating systems, provided that the platform supports ANSI C and some reasonable level of POSIX compliance.

Running the testsuite (**make check**) requires that you have Perl (specifically, /usr/bin/perl). However, Perl isn't necessary to make HMMER work.

HMMER has support for two kinds of parallelization: POSIX multithreading and PVM (Parallel Virtual Machine) clustering. Both are optional, not compiled by default; they are enabled by passing the **--enable-threads** or **--enable-pvm** options to the **./configure** script before compilation. The precompiled binary distributions generally support multithreading but not PVM.

Running the configure script

The GNU **configure** script that comes with HMMER has a number of options. You can see them all by doing:

```
> ./configure --help
```

Most customizations of HMMER are done at the **./configure** command line.

setting installation targets

The most important **configure** options set the installation directories for **make install** to be appropriate to your system. What you need to know is that HMMER installs only two types of files: the programs, and man pages. It installs the programs in **--bindir** (which defaults to **/usr/local/bin**), and the man pages in the **man1** subdirectory of **--mandir** (default **/usr/local/man**). So, say you want **make install** to instead install programs in **/usr/bioprogs/bin/** and man pages in **/usr/share/man/man1**; you could configure with:

```
> ./configure --mandir=/usr/share/man --bindir=/usr/bioprogs/bin
```

That's really all you need to know, since HMMER installs so few different types of files. But just so you know; a GNU **configure** script is very flexible, and it has shortcuts that accomodates several standard conventions for where programs get installed.

One common strategy is to install all files under one directory, like the default **/usr/local**. To change this prefix to something else, say **/usr/mylocal/** (so that programs go in **/usr/mylocal/bin** and man pages in **/usr/mylocal/man/man1**, you can use the **--prefix** option:

```
> ./configure --prefix=/usr/mylocal
```

Another common strategy (especially in multiplatform environments) is to put programs in an architecture-specific directory like **/usr/share/Linux/bin** while keeping man pages in a shared, architecture-independent directory like **/usr/share/man/man1**. GNU configure uses **--exec-prefix** to set the path to architecture dependent files. This normally defaults to being the same as **--prefix**, but you could change it, for example, by:

```
> ./configure --prefix=/usr/share --exec-prefix=/usr/share/Linux/
```

In summary, a complete list of the **./configure** installation options that affect HMMER:

Option	Meaning	Default
--prefix=PREFIX	architecture independent files	**/usr/local/**
--exec-prefix=EPREFIX	architecture dependent files	PREFIX
--bindir=DIR	programs	EPREFIX/bin/
--mandir=DIR	man pages	PREFIX/man/

These are the only **configure** options that work for both the full source distribution of HMMER, and for the binary distributions. The remaining options only affect the configuration of the source distribution; they have no effect on a precompiled binary-only distribution.

setting compiler and compiler flags

By default, **configure** searches first for the GNU C compiler **gcc**, and if that is not found, for a compiler called **cc**. This can be overridden by specifying your compiler with the **CC** environment variable.

By default, the compiler's optimization flags are set to **-g -O2** for **gcc**, or **-g** for other compilers. This can be overridden by specifying optimization flags with the **CFLAGS** environment variable.

For example, to use an Intel C compiler in **/usr/intel/ia32/bin/icc** with optimization flags **-O3 -ip**, you could do:

```
> env CC=/usr/intel/ia32/bin/icc CFLAGS="-O3 -ip" ./configure
```

which is a one-line shorthand that does the same thing as the C-shell commands:

```
> setenv CC /usr/intel/ia32/bin/icc
> setenv CFLAGS "-O3 -ip"
> ./configure
```

If you are using a non-GNU compiler, you almost certainly want to set **CFLAGS** to some sensible optimization flags for your platform and compiler. The **-g** default generated unoptimized code. *At a minimum, turn on your compiler's default optimizations with **CFLAGS=-O***. Otherwise, HMMER will run much slower than it should. If speed is crucial to you, it is often worth a little playing around with different compiler optimization options.

turning on POSIX thread support for multiprocessors

HMMER can run in parallel on multiple processors. To enable this, use the **--enable-threads** option:

```
> ./configure --enable-threads
```

You probably should turn this on. If you're going to run HMMER on a multiprocessor machine, you want this option. Even if you aren't, it doesn't hurt. (Some platforms don't support POSIX threads, so mulltithreading is not configured by default.)

If you turn on threads support, by default, the multithreaded HMMER programs **hmmcalibrate**, **hmmsearch**, and **hmmpfam** will use *all* available CPUs. You can alter this behavior in three ways: by defining **HMMER_NCPU** in the **src/config.h** file that **configure** generates (see "The config.h header file", below); by defining **HMMER_NCPU** as a shell environment variable where you run HMMER (see "Shell environment variables understood by HMMER", below), or using the **--cpu <x>** option with the multithreaded programs (see the "Manual pages" section later in the guide).

turning on PVM support for clusters

HMMER can also run in parallel on distributed clusters, using the Parallel Virtual Machine (PVM) message passing library. To enable the optional PVM support, use the **`--enable-pvm`** flag:

```
> ./configure --enable-pvm
```

You need to have PVM installed first, and the environment variables **`PVM_ROOT`** and **`PVM_ARCH`** must be set appropriately, so HMMER can find the PVM headers and library. You can obtain the free PVM library from Oak Ridge National Laboratory, at http://www.csm.ornl.gov/pvm/.

turning on LFS support for files >2 GB

HMMER can access large files (> 2 GB) even on 32-bit operating systems, using Large File Summit (LFS) extensions. Most modern UNIX systems have LFS extensions. To enable this optional code, use the **`--enable-lfs`** flag:

```
> ./configure --enable-lfs
```

turning on Altivec optimization for Macintosh PowerPC

HMMER contains code specifically optimized for the Macintosh PowerPC, contributed by Erik Lindahl at Stanford University. This code is about four times faster than the normal code. To enable this optional code, use the **`--enable-altivec`** flag:

```
> ./configure --enable-altivec
```

The **`configure`** script will automatically check whether you're on a machine that supports Altivec instructions or not, so it doesn't hurt to try to turn this on if you're not sure.

other options, used in development code

The remaining two features selectable with the **`configure`** script will only be useful to you if you're a developer.

The **`--enable-ccmalloc`** option configures HMMER to be instrumented by the **`ccmalloc`** memory checking library, a free software library from Armin Biere at ETH Zurich. You have to have **`ccmalloc`** installed. **`ccmalloc`** can be obtained from http://www.inf.ethz.ch/personal/biere/projects/ccmalloc/.

The **`--enable-debugging`** option is the same as passing **`CFLAGS=-g`** in the environment. **`--enable-debugging`** can also take a numeric argument from 1-3, which sets the verbosity of additional debugging output from all the programs. **`--enable-debugging=1`** is the least, and **`--enable-debugging=3`** is the most verbose. There is no telling what will spew out of the programs if you set these, because I tend to change the debugging output on a whim.

example configuration

The Intel GNU/Linux version installed in St. Louis is configured as follows:

```
> env CC=icc CFLAGS="-O -static" ./configure --enable-threads --enable-pvm
--enable-lfs --prefix=/usr/seshare/ --exec-prefix=/usr/seshare/‘uname‘
```

The config.h header file

The **configure** script generates a **src/config.h** file that contains most of the configuration information that the source code uses. You don't have to edit this file. There are two things that are worth knowing about, though.

controlling memory footprint with RAMLIMIT

HMMER has two different implementations of its alignment algorithms. One implementation is fast, but consumes memory proportional to the product of the query and target sequence lengths. The other implementation is about two-fold slower, but consumes memory proportional to the *sum* of the query and target lengths, which can be orders of magnitude less memory than the normal algorithm. Having both algorithms allows HMMER to efficiently deal with almost any size of query model and target sequence.

HMMER switches to the small memory variant algorithm when the normal algorithm would take more than a certain number of megabytes of memory. This threshold is set by the definition of RAMLIMIT in **config.h**. By default, RAMLIMIT is 32.

On a multiprocessor, this means 32 MB per thread, e.g. per CPU. On a 64-processor machine, for example, HMMER expects by default to have 2 GB of memory (64 × 32 MB).

This number only affects the switchover point from the normal algorithm to the small memory variants, so it is not a perfect estimate of the actual total memory consumed by HMMER. But since the alignment algorithms dominate HMMER's memory usage, the correspondence is pretty good.

Therefore, if you know you will be running HMMER on a machine with less than 32 MB of memory per CPU, you might want to reduce RAMLIMIT before compiling. (Even 0 will work – it will force the small memory variant algorithms to be used for all target sequences.) Or, if you know you will always have a lot of memory available, you can increase RAMLIMIT and the programs will run faster, depending on how many more target sequences get processed by the faster algorithms.

limiting the default number of processors with HMMER_NCPU

By default, the multithreaded version of HMMER (e.g. what you get with passing **--enable-threads** to **configure**) will use *all* available CPUs. This might not be socially acceptable behavior, especially on a shared resource.

You can compile in a different default limit by defining the **HMMER_NCPU** variable in **config.h**.

If you're compiling HMMER for a large multiprocessor system that's shared by a number of people, and you want most HMMER jobs to access only a small number of processors by default, setting **HMMER_NCPU** is a good idea.

This "limit" is only a default. It can still be overridden in the shell environment by setting an environment variable **HMMER_NCPU**, or by the user on the command line using a **--cpu <x>** option.

the other stuff in config.h

The other stuff in **config.h** is dangerous to change, unless you know what you're doing, or are willing to learn. One example of when you might change it is if you're warping HMMER code to work on non-biological alphabets. HMMER can be configured for arbitrary symbol alphabets by changing key constants in **config.h**, plus code in **globals.h** where the alphabet is declared, and **alphabet.c** where the alphabet gets set up. I haven't done this myself but it is apparently fairly straightforward. HMMER has reportedly been adapted to speech recognition, recorded music reconstruction, and automobile engine telemetry applications, and possibly others that I haven't heard about.

Running make

When you run **`configure`**, it also creates the Makefiles that will actually compile and install the code. There are a variety of make targets in the toplevel Makefile. Typing

```
> make
```

is equivalent to **`make all`** - it compiles all the source code, and leaves the executables in the **`src/`** subdirectory. The other make targets that you might use, in the order you're likely to use them, are:

check Runs the testsuite; verifies that HMMER has compiled correctly.

install Installs the programs in BINDIR and the man pages in MANDIR. Creates the target directories if they don't already exist.

clean Cleans up the directory, removing files generated in the compilation process, while leaving the original distribution files, plus the Makefiles and the programs.

uninstall Remove a previously installed HMMER, reversing a previous **`make install`**. This assumes you left the toplevel Makefile untouched since you did the **`make install`**.

distclean Like 'make clean', but removes everything that isn't part of a pristine source distribution. Use this only if you're going to start configuring and compiling HMMER all over again.

Shell environment variables understood by HMMER

HMMER is built to coexist peacefully with the BLAST suite of database search programs (Altschul, 1991; Altschul et al., 1997). HMMER reads the following environment variables (the examples given use UNIX csh syntax):

`BLASTDB` Location of sequence databases that **`hmmsearch`** will look in, in addition to the current working directory. Multiple directories are allowed, separated by colons. A trailing slash on each path is important to BLAST, but not to HMMER.
Examples:

```
> setenv BLASTDB /nfs/databases/
> setenv BLASTDB /nfs/databases/:/nfs/moredatabases/
```

`BLASTMAT` Location of substitution matrices that **`hmmbuild --pam`** (the PAM prior option) can read. Although HMMER can parse a colon-separated list, BLAST must have a single directory path here. Example:

```
> setenv BLASTMAT /nfs/databases/matrix/
```

`HMMERDB` Location of HMMs, PFAM, or other HMMER specific data files. Any program that reads an HMM file looks in both HMMERDB and the current working directory. Multiple directories are allowed, colon-separated. Examples:

```
> setenv HMMERDB /usr/local/lib/hmmer/
> setenv HMMERDB /usr/local/lib/hmmer/:/nfs/databases/pfam/
```

HMMER_NCPU On multiprocessors that support POSIX threads (this includes almost all modern UNIX multiprocessors; for example, SGI Origin servers), the programs **hmmcalibrate**, **hmmpfam**, and **hmmsearch** run as parallelized, multithreaded applications. Normally they will take over all available CPUs in the machine. HMMER_NCPU sets a maximum number of CPUs to utilize, so HMMER searches are "good citizens", leaving some CPU power for other jobs. An example of configuring HMMER to use only 16 processors on a 32-processor Origin:

```
> setenv HMMER_NCPU 16
```

If you have installed BLAST, you probably already have the two BLAST environment variables set in system-wide or user-specific .cshrc files.

All four variables are optional. HMMER looks for any sequence file first in **BLASTDB**, then in the current working directory. It looks for any HMM file first in **HMMERDB**, then in the current working directory. Thus if these are set up, you can simplify command lines to:

```
> hmmpfam Pfam my.query
> hmmsearch my.hmm swiss35
```

instead of:

```
> hmmpfam /some/long/path/to/databases/Pfam my.query
> hmmsearch my.hmm /some/long/path/to/databases/swiss35
```

HMMER only uses BLAST scoring matrices with the **hmmbuild --pamprior** option. If you use this, the matrix is looked for in the **BLASTMAT** directory first, then in the current directory.

The **HMMER_NCPU** environment variable takes precedence over any **HMMER_NCPU** processor number limit defined in **config.h** – see the discussion of this in the previous section.

Configuring a PVM cluster for HMMER

The following only applies to people installing HMMER on a distributed cluster using PVM. I assume you're already reasonably familiar with PVM applications (at least, as familiar as I am – I'm only familiar enough with it to make it work), and that you have PVM installed.

Designate one machine as the "master", and the other machines as "slaves". You will start your HMMER process on the master, and the master will spawn jobs on the slaves using PVM.[3]

Install PVM on the master and all the slaves. On the master, make sure the environment variables **PVM_ROOT** and **PVM_ARCH** are set properly (ideally, in a system-wide .cshrc file). You may also want to have **PVM_RSH** set to **ssh** (we do).

If you're using rsh to connect to the slaces, Add the master's name to your .rhosts or /etc/hosts.equiv file on the slaves, so the slaves accept passwordless rsh connections from the master. Or, if you're using ssh, do the equivalent with ssh authentication (and make sure you have **PVM_RSH** set to **ssh** in the environment). Test this by rsh'ing or ssh'ing into the slaves from the master's command line, independent of PVM and HMMER.

Put copies of HMMER executables in a directory on the master and all the slaves. For each PVM-capable program (**hmmcalibrate**, **hmmpfam**, and **hmmsearch**, there is a corresponding slave PVM program (**hmmcalibrate-pvm**, **hmmpfam-pvm**, and **hmmsearch-pvm**). The master machine needs copies of

[3] HMMER will not parallelize on PVM capable systems that cannot run a strict master-slave model, that is, where the master is running a different program than its slaves. For example, Cray T3E systems are in principle PVM-capable, but only when all processors instantiate the same program, and that program figures out for itself whether it's the master calling the shots, or a slave communicating back to the master. This style of message passing parallelization is more MPI like, and becoming more standard – but is not supported by HMMER.

all the HMMER programs, including the slave PVM programs. The slaves only need copies of the three slave PVM programs. (You never need to start the slave programs yourself; PVM does that. You just need to make sure they're installed where PVM can see them.)

The PVM implementation of **`hmmpfam`** needs a copy of any HMM databases you may search to be installed on the master and every slave. All HMM databases must be indexed with **`hmmindex`**. The reason is that **`hmmpfam`** is I/O bound; the PVM implementation can't distribute an HMM database fast enough over a typical cluster's Ethernet. Instead, each PVM node accesses its own copy of the HMM database, distributing the I/O load across the nodes. **`hmmcalibrate`** and **`hmmsearch`**, in contrast, are freestanding. Only the master node needs to be able to access any HMM and/or sequence files.

Write a PVM hostfile for the cluster. Specify the location of the HMMER executables using the ep= directive. Specify the location of **`pvmd`** on the slaves using the dx= directive (alternatively, you can make sure **`PVM_ROOT`** and **`PVM_ARCH`** get set properly on the slaves). For the slaves, use the wd= directive to specify the location of the HMM databases for **`hmmpfam`** (alternatively, you can make sure HMMERDB gets set properly on the slaves). Use the sp= directive to tell HMMER how many processors each node has (and hence, how many independent PVM processes it should start); sp=1000 means 1 CPU, sp=2000 means 2 CPUs, sp=4000 means 4 CPUS, etc.

Start the PVM by typing

```
> pvm hostfile
```

(where "hostfile" is the name of your hostfile) on the master. Make sure all the nodes started properly by typing

```
> conf
```

at the PVM console prompt. Type

```
> quit
```

to exit from the PVM console, which leaves the PVM running in the background. You should only need to start PVM once. (We have a PVM running continuously on our network right now, waiting for HMMER jobs.)

Once PVM is running, at any time you can run HMMER programs on the master and exploit your PVM, just by adding the option **`--pvm`**; for instance,

```
> hmmpfam --pvm Pfam my.query
```

parallelizes a search of a query sequence in the file my.query against the Pfam database.

Once PVM is properly configured and your slave nodes have the required slave programs (and databases, in the case of **`hmmpfam`**), the only difference you will notice between the serial and the PVM version is a (potentially massive) increase in search speed. Aside from the addition of the **`--pvm`** option on the command line, all other options and input/output formats remain identical.

example of a PVM cluster

The St. Louis Pfam server runs its searches using HMMER on a PVM cluster called Wulfpack. I'll use it as a specific example of configuring a PVM cluster. It's a little more intricate than you'd usually need for personal use, just because of the details of running PVM jobs in a standalone way from CGI scripts on a Web server. [4]

The master node is the Web server itself, **`fisher`**. The slave nodes are eight dual processor Intel/Linux boxes called **`wulf01`** through **`wulf08`**.

PVM 3.3.11 is installed in **`/usr/local/pvm3`** on the master and all the slaves.

[4] These instructions were made for an older version of Wulfpack; machine names and numbers have changed since then.

On fisher, all HMMER executables are installed in **/usr/local/bin**. On the wulf slave nodes, the three PVM slave executables are installed in **/usr/local/wulfpack**.

Pfam and PfamFrag, two Pfam databases, are installed on the wulf slave nodes in **/usr/local/wulfpack.** They are converted to binary format using **hmmconvert -b**, then indexed using **hmmindex**. (Using binary format databases is a performance win for **hmmpfam** searches, because **hmmpfam** is I/O bound and binary HMM databases are smaller.)

An **ls** of **/usr/local/wulfpack** on any wulf node looks like:

```
[eddy@wulf01 /home]$ ls /usr/local/wulfpack/
Pfam             PfamFrag         hmmcalibrate-pvm   hmmsearch-pvm
Pfam.ssi         PfamFrag.ssi     hmmpfam-pvm
```

The PVM hostfile for the cluster looks like:

```
# Config file for Pfam Web server PVM
#
* ep=/usr/local/bin sp=1000
fisher.wustl.edu
* lo=pfam dx=/usr/local/pvm3/lib/pvmd ep=/usr/local/wulfpack sp=2000
wulf01
wulf02
wulf03
wulf04
wulf05
wulf06
wulf07
wulf08
```

A wrinkle specific to configuring Web servers: the web server is running HMMER as user “nobody” because it’s calling HMMER from a CGI script. We can’t configure a shell for “nobody” on the slaves, so we create a dummy user called “pfam” on each wulf node. The lo= directive in the PVM hostfile is telling the master to connect to the slaves as user “pfam”. On each slave, there is a user “pfam” with a .rhosts that looks like:

```
   fisher nobody
   fisher.wustl.edu nobody
```

which tells the wulf node to allow fisher’s user “nobody” to connect to the wulf node as user “pfam”.

Also note how we use the sp= directive to tell HMMER (via PVM) that the wulf nodes are dual processors. fisher is actually a dual processor too, but by setting sp=1000, HMMER will only start one PVM process on it (leaving the other CPU free to do all the things that keep Web servers happy).

The trickiest thing is making sure **PVM_ROOT** and **PVM_ARCH** get set properly. For my own private PVM use, my **.cshrc** contains the lines:

```
        setenv PVM_ROOT    /usr/local/pvm3
        setenv PVM_ARCH    ‘$PVM_ROOT/lib/pvmgetarch‘
```

But for the web server PVM, it’s a little trickier. At boot time, we start the Web server’s pvmd as user “nobody” on fisher using a local init script, **/etc/rc.d/init.d/pvm_init**. With its error checking deleted for clarity, this script basically looks like:

```
#!/bin/sh
wulfpack_conf=/home/www/pfam/pfam-3.1/wulfpack.conf
. /usr/local/pvm3/.pvmprofile
$PVM_ROOT/lib/pvmd $wulfpack_conf >/dev/null &
```

We call this at boot time by adding the line **`su nobody -c "sh /etc/rc.d/init.d/pvm_init"`** to our **`rc.local`** file. **`.pvmprofile`** is a little PVM-supplied script that properly sets **`PVM_ROOT`** and **`PVM_ARCH`**, and **`wulfpack.conf`** is our PVM hostfile.

The relevant lines of the CGI Perl script that runs HMMER jobs from the Web server (again, heavily edited for clarity) are:

```
# Configure environment for PVM
$ENV{'HMMERDB'}    = "/usr/local/wulfpack:/home/www/pfam/data/"
$ENV{'PVM_EXPORT'} = "HMMERDB";
$output = `/usr/local/bin/hmmpfam --pvm Pfam /tmp/query`;
```

The trick here is that we export the HMMERDB environment variable via PVM, so the PVM processes on wulf nodes will know where to find their copy of Pfam.

▷ ***Why is HMMER crashing when I try to use it under PVM?*** *It appears that the PVM port may not be as good as it should be. It happens to work well in St. Louis. It is used 24/7 on the servers backing the St. Louis Pfam search server at http://pfam.wustl.edu. However, I receive occasional reports of problems at other sites. John Blanchard at Incyte has contributed a number of improvements to the PVM code, but despite his generous hard work, scattered problems persist. Unfortunately, PVM is painful for me to debug, and I have been unable to reproduce any of these latest problems in St. Louis. Since it works fine for me, I can't reproduce the issues; and nobody has sent me a small, detailed, and reproducible bug report, so I'm not able to track these phantoms down. But it seems only fair to warn you that this is an area of the code where bugs may lurk.*

▷ ***Does HMMER support MPI?*** *Not yet.*

3 Tutorial

Here's a tutorial walk-through of some small projects with HMMER. This section should be sufficient to get you started on work of your own, and you can (at least temporarily) skip the rest of the Guide.

The programs in HMMER

There are currently nine programs supported in the HMMER 2 package:

`hmmalign` Align sequences to an existing model.

`hmmbuild` Build a model from a multiple sequence alignment.

`hmmcalibrate` Takes an HMM and empirically determines parameters that are used to make searches more sensitive, by calculating more accurate expectation value scores (E-values).

`hmmconvert` Convert a model file into different formats, including a compact HMMER 2 binary format, and "best effort" emulation of GCG profiles.

`hmmemit` Emit sequences probabilistically from a profile HMM.

`hmmfetch` Get a single model from an HMM database.

`hmmindex` Index an HMM database.

`hmmpfam` Search an HMM database for matches to a query sequence.

`hmmsearch` Search a sequence database for matches to an HMM.

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:

`globins50.msf` An alignment file of 50 aligned globin sequences, in GCG MSF format.

`globins630.fa` A FASTA format file of 630 unaligned globin sequences.

`fn3.sto` An alignment file of fibronectin type III domains, in Stockholm format. (From Pfam 8.0.)

`rrm.sto` An alignment file of RNA recognition motif domains, in Stockholm format. (From Pfam 8.0).

`rrm.hmm` An example HMM, built from rrm.sto

`pkinase.sto` An alignment file of protein kinase catalytic domains, in Stockholm format. (From Pfam 8.0).

`Artemia.fa` A FASTA file of brine shrimp globin, which contains nine tandemly repeated globin domains.

7LES_DROME A SWISSPROT file of the *Drosophila* Sevenless sequence, a receptor tyrosine kinase with multiple domains.

RU1A_HUMAN A SWISSPROT file of the human U1A protein sequence, which contains two RRM domains.

Create a new directory that you can work in, and copy all the files in **tutorial** there. I'll assume for the following examples that you've installed the HMMER programs in your path; if not, you'll need to give a complete path name to the HMMER programs (e.g. something like **/usr/people/eddy/hmmer-2.2/binaries/hmmbu** instead of just **hmmbuild**).

Format of input alignment files

HMMER starts with a multiple sequence alignment file that you provide. HMMER can read alignments in several common formats, including the output of the CLUSTAL family of programs, Wisconsin/GCG MSF format, the input format for the PHYLIP phylogenetic analysis programs, and "alighed FASTA" format (where the sequences in a FASTA file contain gap symbols, so that they are all the same length).

HMMER's native alignment format is called Stockholm format, the format of the Pfam protein database that allows extensive markup and annotation. All these formats are documented in a later section.

The software autodetects the alignment file format, so you don't have to worry about it.

Most of the example alignments in the tutorial are Stockholm files. **rrm.sto** is a simple example (generated by stripping all the extra annotation off of a Pfam RNA recognition motif seed alignment). **pkinase.sto** and **fn3.sto** are original Pfam seed alignments, with all their annotation.

Searching a sequence database with a single profile HMM

One common use of HMMER is to search a sequence database for homologues of a protein family of interest. You need a multiple sequence alignment of the sequence family you're interested in.

> ▷ ***Can I build a model from unaligned sequences?*** *In principle, profile HMMs can be trained from unaligned sequences; however, this functionality is temporarily withdrawn from HMMER. I recommend CLUSTALW as an excellent, freely available multiple sequence alignment program. The original* ***hmmt*** *profile HMM training program from HMMER 1 is also still available, from ftp://ftp.genetics.wustl.edu/pub/eddy/hmmer/hmmer-1.8.4.tar.Z.*

build a profile HMM with hmmbuild

Let's assume you have a multiple sequence alignment of a protein domain or protein sequence family. To use HMMER to search for additional remote homologues of the family, you want to first build a profile HMM from the alignment. The following command builds a profile HMM from the alignment of 50 globin sequences in **globins50.msf**:

```
> hmmbuild globin.hmm globins50.msf
```

This gives the following output:

```
hmmbuild - build a hidden Markov model from an alignment
HMMER 2.3 (April 2003)
Copyright (C) 1992-2003 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Alignment file:                    globins50.msf
```

```
File format:                      MSF
Search algorithm configuration:   Multiple domain (hmmls)
Model construction strategy:      MAP (gapmax hint: 0.50)
Null model used:                  (default)
Prior used:                       (default)
Sequence weighting method:        G/S/C tree weights
New HMM file:                     globin.hmm
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Alignment:           #1
Number of sequences: 50
Number of columns:   308

Determining effective sequence number    ... done. [2]
Weighting sequences heuristically        ... done.
Constructing model architecture          ... done.
Converting counts to probabilities       ... done.
Setting model name, etc.                 ... done. [globins50]

Constructed a profile HMM (length 143)
Average score:      189.04 bits
Minimum score:      -17.62 bits
Maximum score:      234.09 bits
Std. deviation:      53.18 bits

Finalizing model configuration           ... done.
Saving model to file                     ... done.
//
```

The process takes a second or two. **`hmmbuild`** create a new HMM file called **`globin.hmm`**. This is a human and computer readable ASCII text file, but for now you don't care. You also don't care for now what all the stuff in the output means; I'll describe it in detail later. The profile HMM can be treated as a compiled model of your alignment.

calibrate the profile HMM with hmmcalibrate

This step is optional, but doing it will increase the sensitivity of your database search.

When you search a sequence database, it is useful to get "E-values" (expectation values) in addition to raw scores. When you see a database hit that scores x, an E-value tells you the number of hits you would've expected to score x or more just by chance in a sequence database of this size.

HMMER will always estimate an E-value for your hits. However, unless you "calibrate" your model before a database search, HMMER uses an analytic upper bound calculation that is extremely conservative. An empirical HMM calibration costs time (about 10% the time of a SWISSPROT search) but it only has to be done once per model, and can greatly increase the sensitivity of a database search. To empirically calibrate the E-value calculations for the globin model, type:

```
> hmmcalibrate globin.hmm
```

which results in:

```
hmmcalibrate -- calibrate HMM search statistics
HMMER 2.3 (April 2003)
Copyright (C) 1992-2003 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HMM file:                 globin.hmm
Length distribution mean: 325
Length distribution s.d.: 200
Number of samples:        5000
random seed:              1051632537
```

```
histogram(s) saved to:    [not saved]
POSIX threads:            4
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

HMM    : globins50
mu     :   -39.897396
lambda :     0.226086
max    :    -9.567000
//
```

This might take several minutes, depending on your machine. Go have a cup of coffee. When it is complete, the relevant parameters are added to the HMM file. (Note from the "POSIX threads: 4" line that I'm running on 4 CPUs on a quad-processor box. I'm impatient.)

Calibrated HMMER E-values tend to be relatively accurate. E-values of 0.1 or less are, in general, significant hits. Uncalibrated HMMER E-values are also reliable, erring on the cautious side; uncalibrated models may miss remote homologues.

> ▷ ***Why doesn't hmmcalibrate always give the same output, if I run it on the same HMM?*** *It's fitting a distribution to the scores obtained from a random (Monte Carlo) simulation of a small sequence database, and this random sequence database is different each time. You can make* **`hmmcalibrate`** *give reproducible results by making it initialize its random number generator with the same seed, using the* **`--seed <x>`** *option, where* **`x`** *is any positive integer. By default, it chooses a "random" seed, which it reports in the output header. You can reproduce an* **`hmmcalibrate`** *run by passing this number as the seed. (Trivia: the default seed is the number of seconds that have passed since the UNIX "epoch" - usually January 1, 1970.* **`hmmcalibrate`** *runs started in the same second will give identical results. Beware, if you're trying to measure the variance of HMMER's estimated $\hat{\lambda}$ and $\hat{\mu}$ parameters...)*

search the sequence database with hmmsearch

As an example of searching for new homologues using a profile HMM, we'll use the globin model to search for globin domains in the example *Artemia* globin sequence in **`Artemia.fa`**:

```
> hmmsearch globin.hmm Artemia.fa
```

The output comes in several sections, and unlike building and calibrating the HMM, where we treated the HMM as a black box, now you *do* care about what it's saying.

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

```
hmmsearch - search a sequence database with a profile HMM
HMMER 2.3 (April 2003)
Copyright (C) 1992-2003 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HMM file:                   globin.hmm [globins50]
Sequence database:          Artemia.fa
per-sequence score cutoff:  [none]
per-domain score cutoff:    [none]
per-sequence Eval cutoff:   <= 10
per-domain Eval cutoff:     [none]
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query HMM:   globins50
Accession:   [none]
Description: [none]
  [HMM has been calibrated; E-values are empirical estimates]
```

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:

```
Scores for complete sequences (score includes all domains):
Sequence Description                                    Score    E-value  N
-------- -----------                                    -----    ------- ---
S13421   S13421 GLOBIN - BRINE SHRIMP                   474.3   1.7e-143   9
```

The first field is the name of the target sequence, then followed by the description line for the sequence. The last three fields are the raw score (in units of “bits”), the estimated E-value, and the total number of domains detected in the sequence. By default, every sequence with an E-value less than 10.0 is listed in this output.

The second section is the *domain top hits* list. By default, for every sequence with an E-value less than 10, every domain with a raw score greater than 0 is listed. (Read that carefully. In a later chapter we’ll discuss some caveats about how **`hmmsearch`** identifies domains, and how to control its output in different ways.) Each domain detected in the search is output in a list ranked by E-value:

```
Parsed for domains:
Sequence Domain  seq-f seq-t    hmm-f hmm-t      score  E-value
-------- ------- ----- -----    ----- -----      -----  -------
S13421     7/9     932  1075 ..     1   143 []    76.9  7.3e-24
S13421     2/9     153   293 ..     1   143 []    63.7  6.8e-20
S13421     3/9     307   450 ..     1   143 []    59.8  9.8e-19
S13421     8/9    1089  1234 ..     1   143 []    57.6  4.5e-18
S13421     9/9    1248  1390 ..     1   143 []    52.3  1.8e-16
S13421     1/9       1   143 [.     1   143 []    51.2    4e-16
S13421     4/9     464   607 ..     1   143 []    46.7  8.6e-15
S13421     6/9     775   918 ..     1   143 []    42.2    2e-13
S13421     5/9     623   762 ..     1   143 []    23.9  6.6e-08
```

The first field is the name of the target sequence. The second field is the number of this domain: e.g. “6/9” means the sixth domain of nine total domains detected.

The fields marked “seq-f” and “seq-t” mean “sequence from” and “sequence to”: the start and end points of the alignment on the target sequence. After these two fields is a shorthand annotation for whether the alignment is “global” with respect to the sequence or not. A dot (.) means the alignment does not go all the way to the end; a bracket ([or]) means it does. Thus, .. means that the alignment is local within the sequence; [. means that the alignment starts at the beginning of the sequence, but doesn’t go all the way to its end; .] means the alignment starts somewhere internally and goes all the way to the end; and [] means the alignment includes the entire sequence.

Analogously, the fields marked “hmm-f” and “hmm-t” indicate the start and end points with respect to the consensus coordinates of the model, and the following field is a shorthand for whether the alignment is global with respect to the *model*. Here, for instance, all the globin domains in the *Artemia* sequence are complete matches to the entire globin model – *because, by default,* ***`hmmbuild`*** *built the HMM to only look for those kinds of alignments*. We’ll discuss later how to modify the profile HMM for other search styles.

The final two fields are the raw score in bits and the estimated E-value, *for the isolated domain*. The scores for the domains sum up to the raw score of the complete sequence.

The next section is the *alignment output*. By default, every domain that appeared in the domain top hits list now appears as a BLAST-like alignment. For example:

```
Alignments of top-scoring domains:
S13421: domain 7 of 9, from 932 to 1075: score 76.9, E = 7.3e-24
                   *->eekalvksvwgkveknveevGaeaLerllvvyPetkryFpkFkdLss
                      +e a vk+ w+ v+ ++  vG  +++ l++ +P+ +++FpkF d+
      S13421   932    REVAVVKQTWNLVKPDLMGVGMRIFKSLFEAFPAYQAVFPKFSDVPL 978

                   adavkgsakvkahgkkVltalgdavkkldd...lkgalakLselHaqklr
```

```
              d++++++ v +h   V t+l++ ++ ld++ +l+   ++L+e H+  lr
S13421   979 -DKLEDTPAVGKHSISVTTKLDELIQTLDEpanLALLARQLGEDHIV-LR 1026

             vdpenfkllsevllvvlaeklgkeftpevqaalekllaavataLaakYk<
             v+   fk +++vl+  l++ lg+ f+  ++ +++k+++++++ +++  +
S13421  1027 VNKPMFKSFGKVLVRLLENDLGQRFSSFASRSWHKAYDVIVEYIEEGLQ  1075

             -*

S13421     -    -
```

The top line is the HMM consensus. The amino acid shown for the consensus is the highest probability amino acid at that position according to the HMM (not necessarily the highest *scoring* amino acid, though). Capital letters mean "highly conserved" residues: those with a probability of > 0.5 for protein models, or > 0.9 for DNA models.

The center line shows letters for "exact" matches to the highest probability residue in the HMM, or a "+" when the match has a positive score and is therefore considered to be "conservative" according to the HMM's view of *this particular position in the model* – not the usual definition of conservative changes in general.

The third line shows the sequence itself, of course.

> ▷ ***Why does alignment output from hmmsearch or hmmpfam include some strange almost-blank lines with -'s and *'s?*** *The consensus line includes leading and trailing symbols* **`*->`** *and* **`<-*`**, *representing the nonemitting profile HMM state paths $S \to N \to B$ and $E \to C \to T$. This little flourish makes some sense if you know something about state paths and profile HMMs, but contributes nothing terribly useful to the output from a user's perspective – except that it may confuse your output parsing scripts when the extra symbols cause the alignment to unexpectedly wrap around to a blank final block, as happened in this example.*

The next section of the output is the *score histogram*. It shows a histogram with raw score increasing along the Y axis, and the number of sequence hits represented as a bar along the X axis. In our example here, since there's only a single sequence, the histogram is very boring:

```
Histogram of all scores:
score    obs    exp  (one = represents 1 sequences)
-----    ---    ---
  474      1      0|=
```

Notice though that it's a histogram of the whole sequence hits, not the domain hits.

You can ignore the rest of the **`hmmsearch`** output:

```
% Statistical details of theoretical EVD fit:
              mu =   -39.8974
          lambda =     0.2261
chi-sq statistic =     0.0000
  P(chi-square)  =          0

Total sequences searched: 1

Whole sequence top hits:
tophits_s report:
     Total hits:           1
     Satisfying E cutoff:  1
     Total memory:         16K

Domain top hits:
tophits_s report:
```

```
Total hits:           9
Satisfying E cutoff:  9
Total memory:         21K
```

This is just some trailing internal info about the search that used to be useful to me.

searching major databases like NCBI NR or SWISSPROT

HMMER reads all major database formats and does not need any special database indexing. You can search any large sequence database you have installed locally just by giving the full path to the database file, e.g. something like:

```
> hmmsearch globin.hmm /nfs/databases/swiss35/sprot35.dat
```

If you have BLAST installed locally, it's likely that you have a directory (or directories) in which the BLAST databases are kept. The location of these directories is specified to BLAST by a shell environment variable called **BLASTDB**, which contains a colon-delimited list of one or more directories. HMMER will read the same environment variable. For example, if you have BLAST databases in directories called **/nfs/databases/blast-db/** and **/nfs/databases/golden-path/blast/**, and you want to search **/nfs/databases/blast-db/swissprot**, the following commands will work (in a C shell):

```
> setenv BLASTDB /nfs/databases/blast-db/:/nfs/databases/golden-path/blast/
> hmmsearch globin.hmm swissprot
```

You'd tend to have the **setenv** command as part of the local configuration of your machine, rather than typing it at the command line.

local alignment versus global alignment

This is important. HMMER does not do local (Smith/Waterman) and global (Needleman/Wunsch) style alignments in the same way that most computational biology analysis programs do it. To HMMER, whether local or global alignments are allowed is part of the *model*, rather than being accomplished by running a different *algorithm*. (This will be discussed in greater detail later; it is part of the "Plan7" architecture of the new HMMER2 models.)

Therefore, you need to choose what kind of alignments you want to allow *when you build the model* with **hmmbuild**. By default, **hmmbuild** builds models which allow alignments that are global with respect to the HMM, local with respect to the sequence, and allows multiple domains to hit per sequence. Such models will only find complete domains.

hmmbuild provides some standard options for common alignment styles. The following table shows the four alignment styles supported by **hmmbuild**, and also shows the equivalent old HMMER 1.x search program style (to orient old-school HMMER users).

Command	w.r.t. sequence	w.r.t. HMM	multidomain	HMMER 1 equivalent
hmmbuild	local	global	yes	hmmls
hmmbuild -f	local	local	yes	hmmfs
hmmbuild -g	local	global	no	hmms
hmmbuild -s	local	local	no	hmmsw

In brief, if you know you only want to find complete domains, use the **hmmbuild** default. If you need to find fragments (local alignments) too, and are willing to give up some sensitivity on complete domains to see them, use **hmmbuild -f**. If you want maximal search sensitivity, build two models and search with both of them.

Searching a query sequence against a profile HMM database

A second use of HMMER is to look for known domains in a query sequence, by searching a single sequence against a library of HMMs. (Contrast the previous section, in which we searched a single HMM against a sequence database.) To do this, you need a library of profile HMMs. One such library is the PFAM database (Sonnhammer et al., 1997; Bateman et al., 2002). You can also create your own.

creating your own profile HMM database

HMM databases are just concatenated single HMM files. You can build them either by invoking the **-A** "append" option of **hmmbuild**, or by concatenating HMM files you've already built. For example, here's two ways to build an HMM database called **myhmms** that contains models of the rrm RNA recognition motif domain, the fn3 fibronectin type III domain, and the pkinase protein kinase catalytic domain:

```
> hmmbuild -F rrm.hmm rrm.sto
> hmmbuild fn3.hmm fn3.sto
> hmmbuild pkinase.hmm pkinase.sto
> cat rrm.hmm fn3.hmm pkinase.hmm > myhmms
> hmmcalibrate myhmms
```

or:

```
> hmmbuild -A myhmms rrm.sto
> hmmbuild -A myhmms fn3.sto
> hmmbuild -A myhmms pkinase.sto
> hmmcalibrate myhmms
```

Notice that **hmmcalibrate** can be run on HMM databases as well as single HMMs.

Also note the **-F** option on that first **hmmbuild** command line; that's the "force" option which allows overwriting an existing HMM file. If you really followed instructions, you already have a file **rrm.hmm** that came with the tutorial files. HMMER will refuse to create a new HMM if it would mean overwriting an existing one. Usually this is just annoying, but sometimes can be a useful safety check.

parsing the domain structure of a sequence with hmmpfam

Now that you have a small HMM database called **myhmms**, let's use it to analyze the *Drosophila* Sevenless sequence, **7LES_DROME**:

```
> hmmpfam myhmms 7LES_DROME
```

Like **hmmsearch**, the **hmmpfam** output comes in several sections. The first section is the *header*:

```
hmmpfam - search one or more sequences against HMM database
HMMER 2.3 (April 2003)
Copyright (C) 1992-2003 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HMM file:                 myhmms
Sequence file:            7LES_DROME
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query sequence: 7LES_DROME
Accession:      P13368
Description:    SEVENLESS PROTEIN (EC 2.7.1.112).
```

The next section is the *sequence family classification* top hits list, ranked by E-value. The scores and E-values here reflect the confidence that this query sequence contains one *or more* domains belonging to a domain family. The fields have the same meaning as in **hmmsearch** output, except that the name and description are for the HMM that's been hit.

```
Scores for sequence family classification (score includes all domains):
Model    Description                                    Score    E-value  N
-------- -----------                                    -----    ------- ---
pkinase  Protein kinase domain                          314.6    1.4e-94   1
fn3      Fibronectin type III domain                    176.6    4.7e-53   6
rrm      RNA recognition motif. (a.k.a. RRM, RBD, or    -40.4        1.6   1
```

The E-values for pkinase and fn3 are excellent (much less than 1), so this sequence has domains that belong to the protein kinase and the fibronectin type III domain families. An rrm hit is also reported (because it is less than E=10) but at 1.6, it is not significant; we expect to see 1.6 hits of this quality just by chance in a search of this size. By default, like BLAST, **hmmsearch** and **hmmpfam** report well down into the noise. If you want the output to be cleaner, set an E-value threshold; for example **hmmpfam -E 0.1**.

The next section is the *domain parse* list, ordered by position on the sequence (not by score). Again the fields have the same meaning as in **hmmsearch** output:

```
Parsed for domains:
Model    Domain  seq-f seq-t    hmm-f hmm-t      score  E-value
-------- ------- ----- -----    ----- -----      -----  -------
fn3        1/6     437   522 ..     1    84 []    48.3  2.1e-14
fn3        2/6     825   914 ..     1    84 []    13.4  1.6e-05
fn3        3/6    1292  1389 ..     1    84 []    15.9  9.4e-06
fn3        4/6    1799  1891 ..     1    84 []    63.5  5.3e-19
fn3        5/6    1899  1978 ..     1    84 []    15.2  1.1e-05
fn3        6/6    1993  2107 ..     1    84 []    20.3  3.7e-06
pkinase    1/1    2209  2483 ..     1   294 []   314.6  1.4e-94
rrm        1/1    2223  2284 ..     1    77 []   -40.4      1.6
```

Note how it's still showing us that "rrm" hit - **7LES_DROME** doesn't have any RRM domains.

The final output section is the *alignment output*, just like **hmmsearch**:

```
Alignments of top-scoring domains:
fn3: domain 1 of 6, from 437 to 522: score 48.3, E = 2.1e-14
                CS    C CCCCEEEEEECCTTCCEEEEECCC CCCCCCCEEEEE.ECCCCCC
                   *->P.saPtnltvtdvtstsltlsWsppt.gngpitgYevtyRqpkngge
                      P saP   + +++ ++ l ++W p +  ngpi+gY++++ +++ g+
  7LES_DROME   437    PiSAPVIEHLMGLDDSHLAVHWHPGRfTNGPIEGYRLRL-SSSEGNA 482

                CS CCCCEEECCCCCECECCEEEEECCCCEEEEEECCC CCCC
                   wneltvpgtttsytltgLkPgteYevrVqAvnggG.GpeS<-*
                   + e+ vp    sy+++ L++gt+Y++ +  +n +G+Gp
  7LES_DROME   483 TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGeGPVA    522
...
```

The alignment report has an extra line in this example, a "CS" (consensus structure) line, indicating residues expected to be in α-helix (H), coil (C), or β-sheet (E). (The fibronectin type III domain is a 7-stranded β-sandwich.) This line was picked up from the consensus secondary structure annotation (#=GC SS_cons) in the **fn3.sto** alignment file, which many curated Pfam alignments now have. **hmmbuild** can pick up several kinds of optional information from an annotated alignment file.

obtaining the PFAM database

The PFAM database is available from either **`http://pfam.wustl.edu/`** or **`http://www.sanger.ac.uk/Pfam/`**. Download instructions are on the Web page. The PFAM HMM library is a single large file, containing several hundred models of known protein domains. Install it in a convenient directory and name it something simple like **`Pfam`**.

HMMER will look for PFAM and other files in a directory (or directories) specified by the **`HMMERDB`** environment variable. For instance, if you install the PFAM HMM library as **`/nfs/databases/hmmer/pfam`**, the following commands will search for domains in **`7LES_DROME`**:

```
> setenv HMMERDB /nfs/databases/hmmer/
> hmmpfam pfam 7LES_DROME
```

Creating and maintaining multiple alignments with hmmalign

Another use of profile HMMs is to create multiple sequence alignments of large numbers of sequences. A profile HMM can be build of a "seed" alignment of a small number of representative sequences, and this profile HMM can be used to efficiently align any number of additional sequences.

This is how the PFAM database is updated automatically as the primary sequence databases increase exponentially in size. The PFAM seed alignments are curated, representative alignments that are (relatively) stable from release to release. PFAM full alignments are created automatically by searching a nonredundant database with the seed model and aligning all the significant hits into a multiple alignment using **`hmmalign`**.

For example, to align the 630 globin sequences in **`globins630.fa`** to our globin model **`globin.hmm`**, and create a new alignment file called **`globins630.ali`**, we'd do:

```
> hmmalign -o globins630.ali globin.hmm globins630.fa
```

which would result in:

```
hmmalign - align sequences to an HMM profile
HMMER 2.3 (April 2003)
Copyright (C) 1992-2003 HHMI/Washington University School of Medicine
Freely distributed under the GNU General Public License (GPL)
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
HMM file:             globin.hmm
Sequence file:        globins630.fa
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Alignment saved in file globins630.ali
```

Using the **`-o`** option to specify a save file for the final alignment is a good idea. Else, the alignment will be displayed on the screen as output – and an alignment of several hundred sequences will give a fairly voluminous output.

General notes on using the programs in HMMER

getting quick help on the command line

If you forget the command-line syntax or available options of any of the programs, you can type the name of the program with no other arguments and get a short help message, including summaries of the most common options, e.g.

```
> hmmbuild
```

If you call any program with an option **-h**, you get an augmented help message, including version info (the software version number is helpful if you report bugs or other problems to me) and a complete summary of all the available options, including the expert/experimental ones, e.g.

```
> hmmbuild -h
```

Commonly used options are generally small letters, like **-a**. More infrequently used options are generally large letters, like **-A**. Expert or experimental options are generally in the GNU long form, like **--null2**.

sequence file formats

Unaligned sequence files are usually in FASTA format, but HMMER can read many common file formats including Genbank, EMBL, and SWISS-PROT format.

Aligned sequence files are expected to be in Stockholm format (HMMER's native format, used by the Pfam and Rfam databases), but HMMER can read many common alignment formats including Clustal, GCG MSF, aligned FASTA, and Phylip format. It can also read a simple format (SELEX format) of one line per sequence, containing the name first, followed by the aligned sequence. Alignment files can also be used where unaligned format files are required; the sequences will be read in one at a time and their gaps removed.

HMMER autodetects what format its input sequence files are in. You don't have to worry about reformatting sequences or setting options.

The autodetector is robust - so long as your file really is in one of the formats HMMER expects. If you pass HMMER a file that is not a sequence file, or is in an unexpected format, the autodetector may screw up. (SELEX format, for instance, is so permissive that many non-sequence files look like SELEX to the autodetector.) You can turn off the autodetector and manually specify an input sequence file format using the **--informat** option to many of the programs. This is particularly useful when driving HMMER with unattended scripts - you can increase the robustness of your annotation pipeline by specifying **--informat fasta**, for instance, so HMMER will be able to check that your files really are in FASTA format.

using compressed files

HMMER will automatically read sequence files compressed with gzip, but you have to tell it what format the file is in: format autodetection doesn't work on compressed files. For instance, if I have a compressed FASTA file of the nonredundant NCBI database as **nr.gz**, I can search it with:

```
> hmmsearch --informat fasta my.hmm nr.gz
```

HMMER simply reacts to the **.gz** suffix to detect a gzip'ed file. You must have **gzip** installed and in your **$PATH** for this to work; HMMER simply invokes **gzip -dc** on the file, and starts reading the data from the resulting pipe.

reading from pipes

HMMER can also read sequence data from standard input, through a pipe. Again (as with gzip'ed files) format autodetection doesn't work, so you must specify the file format manually. To read from **stdin**, pass a **-** as the file name; for example, if I have a script that retrieves a bunch of FASTA records, and I want to search each retrieved sequence against a model, I could do:

```
> cool_sequence_fetching_script.pl | hmmsearch --informat fasta my.hmm -
```

> ▷ ***Why can't HMMER autodetect sequence file formats from gzip'ed files or pipes?*** *Format autodetection looks at the file once; then it rewinds to the beginning of the file; then it reads the data, in a second*

pass across the file. When the input is from a pipe rather than a file, a UNIX program can't assume that it can rewind a pipe. Therefore HMMER skips the autodetection pass when reading from a pipe. HMMER could be recoded to store the file as it reads it in for autodetection, so it only makes a single pass over the data – but that's not the way it's coded now.

protein analysis versus nucleic acid analysis

HMMER2 is only recommended for protein sequence analysis.

Nonetheless, HMMER can work with either nucleic acid models and sequences using a nucleotide alphabet, or with protein models and sequences using an amino acid alphabet. HMMER will autodetect whether your files contain RNA, DNA, or protein sequence. (An exception is hmmpfam, which for technical reasons does not do sequence type autodetection; to run hmmpfam on a nucleic acid query sequence, you must specify the **`-n`** option.)

> ▷ ***Why is HMMER recommended only for protein sequence analysis, if it can build and search with DNA models too?*** *The problem is just that I haven't systematically tested HMMER2 on DNA sequences yet, so I don't want to recommend it. I'm a little wary because unlike HMMER1, HMMER2 has been tuned in various ways for Pfam-style protein alignments.* ***`hmmbuild`****, for instance, by default uses a model construction algorithm that is tuned to the expected information content of protein alignments. The search programs also do not expect to have to deal with chromosome-scale target sequences, though they use memory-efficient algorithms that can deal with large proteins. Feel free to explore it (warily) yourself, or you can download HMMER1 (which was used extensively for DNA analysis) from the web site.*

Since HMMER2 is aimed at protein analysis, there isn't even any option for searching both strands of a DNA database. If you do use HMMER2 for DNA analysis, be forewarned that the search programs will only search the top strand!

HMMER cannot search protein model queries against nucleic acid sequence databases; you must translate your target database in all six frames externally and search the resulting peptide database, if this is what you want to do. (Alternatively, use Ewan Birney's GENEWISE program, which can – slowly – search a HMMER protein model against a nucleic acid database.)

environment variables

You can control some aspects of HMMER's behavior, particularly the directories in which it will automatically look for sequence and HMM databases, using shell environment variables. These are documented on page 14, in the section "Shell environment variables understood by HMMER". The person who installed HMMER may have set these environment variables system-wide; as a user, you can override them in your own shell environment, for instance by adding lines to your **`.cshrc`**.

exit status from the programs

HMMER programs always return a normal (zero) POSIX exit status to the shell when everything goes right. Upon any type of failure, HMMER programs will return a nonzero exit status. (The exact nonzero number returned upon a failure is usually uninformative – for errors caught cleanly by HMMER, it will simply exit with status 1. Serious crashes and other faults caught by the operating system may return some other nonzero code.)

If you are wrapping any kind of script (Perl, shell, whatever) around HMMER, you can check this exit status to be sure that your calls are succeeding.

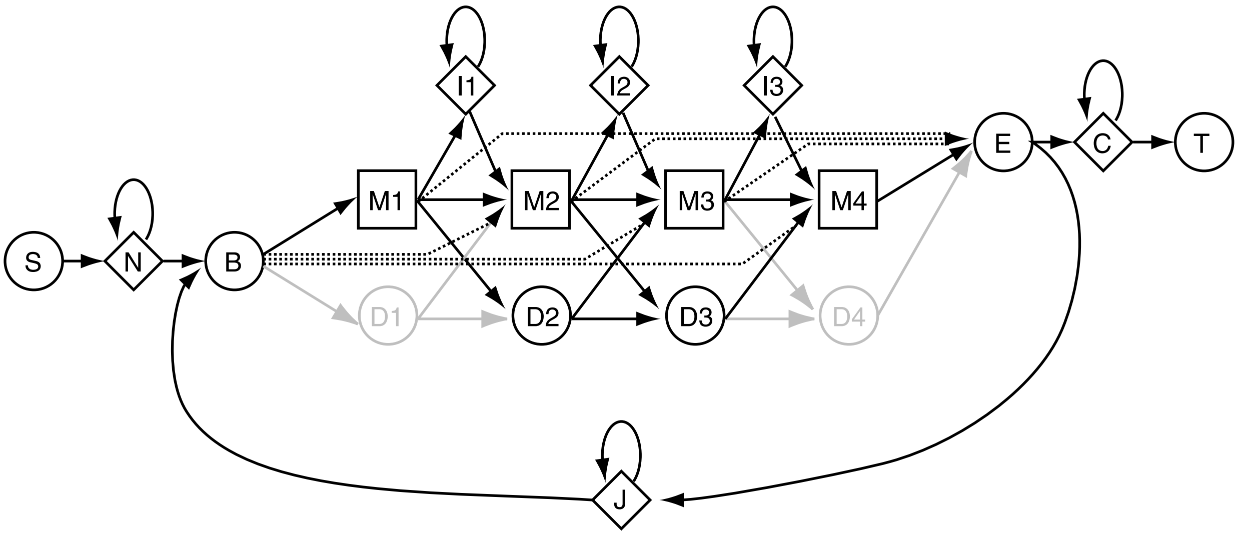


Figure 1: *The Plan7 architecture. Squares indicate match states (modeling consensus positions in the alignment). Diamonds indicate insert states (modeling insertions relative to consensus) and special random sequence emitting states. Circles indicate delete states (modeling deletions relative to consensus) and special begin/end states. Arrows indicate state transitions.*

4 How HMMER builds profile HMMs from alignments

This section walks you through the steps of building a model from a multiple sequence alignment, in gory detail. You have a fair amount of control over many of these steps, using options to the **`hmmbuild`** program.

We'll start with a description of the "Plan 7" profile HMM architecture used by HMMER, and the philosophy behind it. It differs in small but important ways from the original Krogh/Haussler architecture. Specifically, Plan 7 is a fully probabilistic model of both local and global profile alignment.

The Plan 7 profile HMM architecture

HMMER uses a profile HMM architecture called Plan 7. This is somewhat more complex than the original profile HMM architecture introduced by Krogh, Haussler, and coworkers (Krogh et al., 1994).

The heart of a profile HMM is a linear set of match (M) states, one per consensus column in the multiple alignment. Each M state emits (aligns to) a single residue, with a probability score that is determined by the frequency that residues have been observed in the corresponding column of the multiple alignment. Each match state therefore carries a vector of 20 probabilities, for scoring the 20 amino acids.

Some analysis approaches, including BLOCKS (Henikoff and Henikoff, 1994b), are essentially equivalent to a profile HMM composed only of M states. These methods are called *weight matrices* or *position-specific score matrices* (PSSMs); they look for *ungapped* alignments to a consensus.

A profile HMM is capable of modeling gapped alignments, e.g. including insertions and deletions, which lets the software describe a complete conserved domain (rather than just a small ungapped motif). Insertions and deletions are modeled using (surprise!) insertion (I) states and deletion (D) states. Each match state has an I and a D state associated with it. HMMER calls a group of three states (M/D/I) at the same consensus position in the alignment a "node".

These states are interconnected with arrows called *state transition probabilities*. The transitions are arranged so that at each node, either the M state is used (and a residue is aligned and scored) or the D state is used (and no residue is aligned, resulting in a deletion-gap character, '-'). Insertions occur between nodes, and I states have a self-transition, allowing one or more inserted residues to occur between consensus columns. The transition to an I state for the first inserted residue, followed by zero or more I →I self-transitions for each subsequent inserted residue, is the probabilistic equivalent of the familiar gap-open and gap-extend *affine gap* penalty system, where one pays a (usually) high cost for opening a gap and a (usually) lower cost for extending it.

The model begins and ends with dummy non-emitting states, B and E.

This core section of the Plan 7 model, composed of M, D, and I nodes, flanked by B and E states, is essentially a Krogh/Haussler profile HMM. This is the "main model". The main model controls the *data dependent* features of the model. All the probability parameters in the main model are generally estimated from observed frequencies of residues and transitions in a multiple sequence alignment.

Unlike the original Krogh/Haussler and HMMER model architecture, Plan 7 has no D → I or I → D transitions. This reduction from 9 to 7 transitions per node in the main model is one of the origins of the name Plan 7. (The original Krogh/Haussler architecture is called Plan 9 in parts of the code.)[5]

The other states (S,N,C,T,J) are "special states". They (combined with special entry probabilities from B and exit probabilities to E) control *algorithm dependent* features of the model: e.g. how likely the model is to generate various sorts of local or multihit alignments. The algorithm dependent parameters are typically not learned from data, but rather set externally by choosing a desired alignment style.

The abbreviations for the states are as follows:

M$_x$ Match state x. Has K emission probabilities.

D$_x$ Delete state x. Non-emitter.

I$_x$ Insert state x. Has K emission probabilities.

S Start state. Non-emitter.

N N-terminal unaligned sequence state. Emits *on transition* with K emission probabilities.

B Begin state (for entering main model). Non-emitter.

E End state (for exiting main model). Non-emitter.

C C-terminal unaligned sequence state. Emits *on transition* with K emission probabilities.

J Joining segment unaligned sequence state. Emits *on transition* with K emission probabilities.

the philosophy of Plan 7

HMMs are *generative probabilistic models*. Generative models address a serious theoretical problem. To do correct statistical inference, we need to be able to calculate a probability distribution $P(S|M)$ for the

[5] The true origin of the codename is left undocumented, as it reveals an inordinate fondness for bad science fiction movies. Frighteningly, David Haussler understood immediately.

probability of sequences S given a model M, and have this quantity sum to one over the "space" of all sequences. But the number of possible different sequences S is effectively infinite, because they can be of any length. We can't just enumerate this distribution, or we'd have to estimate an infinite number of parameters. How do we make a *finite* model of an *infinite* data space? Generative models work by *recursive* enumeration of possible sequences from a finite set of rules – rules that in an HMM are represented by states, state transitions, and symbol emission probabilities.

(In a profile HMM, the recursion is trivial: the self-transition probability of the insert states, which allows insertions of any length between two positions in the consensus. This simple model implies a geometric distribution over insertion lengths – as does any linear or affine gap scoring system, in fact. Biologically, insertions empirically follow a distribution with a longer tail, but more realistic insertion models are computationally more expensive. The marginal expected gain in performance is not thought to be worth the extra computational effort.)

As described in (Krogh et al., 1994), profile HMMs were models of *global alignment*: the S that the model generates usually starts at the first match state and ends at the last match state.

In real alignments of real biological sequences, global alignment is rarely useful. A profile HMM is usually a model of a conserved domain, not necessarily a model of a complete target sequence. A conserved domain may be buried in a longer target sequence; there might be more than one conserved domain in the target sequence; and maybe the target sequence only contains a smaller fragment of the conserved domain. In real applications, we prefer *local alignment* algorithms, which look for a high-scoring alignment between a subsequence of the target sequence and a part of the query model. The common tools of sequence analysis – BLAST, FASTA, and Smith/Waterman alignment – are all local alignment algorithms.

It is straightforward to wrap local alignment style algorithms around the core Krogh/Haussler model of global alignment to a domain. This is what early versions of profile HMM software did, including HMMER1. But some of the power of probabilistic modeling gets thrown away; the resulting local alignment scores are no longer interpretable as probabilities, because we have to introduce arbitrary nonprobabilistic parameters for the local alignment (such as the usual score of 0 for starting or ending a Smith/Waterman local alignment).

At least from a probabilistically purist perspective, it would be sensible to revise the original Krogh/Haussler model to generate *complete* target sequences – that is, modeling one or more (possibly incomplete) matches to the domain model, and also modeling the *unaligned* sequence in the target. This is what Plan 7 does.

the fully probabilistic "local" alignment models of Plan 7

The Plan 7 architecture explicitly models the *entire* target sequence, regardless of how much of that sequence matches the main model. *All alignments to a Plan 7 model are "global" alignments (!),* in that the complete target sequence is aligned to a complete path through the model from a start state (S) to the terminal state (T). However, some (possibly most) of the sequence may be assigned to Plan 7 states (N,C,J) that generate nonhomologous, unaligned, "random" sequence that is not aligned to the main model. Thus, the algorithm dependent parts of the model control the *apparent* locality of the alignments.

Local alignments with respect to the sequence (i.e., allowing a match to the main model anywhere internal to a longer sequence) are controlled by the N and C states. If the N → N transition is set to 0, alignments are constrained to start in the main model at the first node. Similarly, if the C → C transition is set to 0, alignments are constrained to end from the very last node.

Local alignments with respect to the model (i.e., allowing fragments of the model to match the sequence) are controlled by B → M "entry" transitions and M → E "exit" transitions, shown as dotted lines in the Plan 7 figure. Setting all entries but the B → M_1 transition to 0 forces a partially "global" alignment in which all alignments to the main model must start at the first match or delete state. Setting all exits to 0 but the final M → E transition (which is always 1.0) forces a partially global alignment in which all alignments to the main model must end at the final match or delete state.

Multiple hit alignments are controlled by the E → J transition and the J state. If the E → J transition is set to 0, a sequence may only contain one domain (one alignment to the main model). If it is nonzero, more than one domain per sequence can be aligned to the main model. The J → J transition controls the expected length of the intervening sequence between domains; the lower this probability, the more clustered the domains are expected to be.

available alignment modes

Because the alignment mode in Plan 7 is determined by the HMM configuration, not by the search algorithm, you choose your alignment style when you build the model – not when you do the search.

The **`hmmbuild`** program currently allows you to choose one of four alignment modes, as follows. They are called **`hmmls`**, **`hmmfs`**, **`hmmsw`**, and **`hmms`** mode for historical reasons – these were the names of the corresponding search programs in HMMER1.

`hmmls` This is like "profile" alignment or "glocal" alignment (as Waterman and others call it): global with respect to the profile, and local with respect to sequence. Multiple nonoverlapping domains are allowed in the target sequence. t_{NN} and t_{CC} set to t_{GG} from the null model (see below). t_{EJ} set to 0.5. Internal entries and exits set to zero. This is the default mode for HMMER.

`hmmfs` Multihit Smith/Waterman alignment: fully local with respect to both the main model and the sequence; in addition, multiple nonoverlapping alignments to more than one domain in the same target sequence are allowed. t_{NN} and t_{CC} set to t_{GG} from the null model. t_{EJ} is set to 0.5. Internal entries are set to $0.5/(M-1)$ for a model with M match states. Exit probabilities are set such that the overall chance of exiting from an internal match state is 0.5, and the posterior probability distribution over which match state is exited is equal (I'm wording this carefully; the M → E probabilities are not equal, because there's a small compensation for the fact that if you can leave from M_5, then the chance that you'll even get to M_6 is reduced.) Activated by the **`hmmbuild -f`** option.

`hmmsw` Smith/Waterman "classic" local alignment, single best alignment per target. Same as **`hmmfs`** above, but t_{EJ} is set to zero. Activated by the **`hmmbuild -s`** option.

`hmms` Needleman/Wunsch "classic" global alignment, single best alignment per target. t_{NN} and t_{CC} set to t_{GG} from the null model. t_{EJ} set to zero. Internal entries and exits set to zero. Activated by the **`hmmbuild -g`** option.

One advantage of Plan 7 is great flexibility in choosing an alignment style. Complicated alignment styles are easily encoded in the model parameters without changing the alignment algorithm. For example, say you wanted to model human L1 retrotransposon elements. Because of the way L1 elements are inserted by reverse transcriptase (RVT), L1 elements tend to have a defined 3' end (RVT starts replication at the same place in each new L1) but a ragged 5' end (RVT prematurely falls off a new L1 in an unpredictable fashion). A specialized L1 model could define non-zero internal entry probabilities and zero internal exit probabilities to model this biological situation.

> ▷ ***Why does Pfam distribute two sets of models, called `Pfam_ls` and `Pfam_fs`?*** *A disadvantage of Plan 7 is that if you decide you want to do both local and global alignments, you need two different models. This wouldn't be too terrible, except for the fact that the algorithm-dependent parameters strongly affect the values of the μ and λ parameters that E-value statistics depend on. If the algorithm dependent parameters are changed, these parameters are invalidated and the model needs to be recalibrated with* **`hmmcalibrate`** *– but* **`hmmcalibrate`** *is slow, so this can't be done "on the fly" inside one of the search programs. It turns out that ls mode is the most sensitive alignment mode, so long as a complete domain alignment can be found; whereas fs mode is needed to find domain fragments. Therefore Pfam is distributed with both sets of models; for maximum detection ability of all homologies, both sets of models have to be used.*

wing retraction in Plan 7 dynamic programming

A rather technical point...

In the figure of the Plan 7 architecture, you may have noticed that the first and last delete states are greyed out. Internally in HMMER, these delete states exist in the "probability form" of the model (when the model is being worked with in every way except alignments) but they are carefully removed in the "search form" of the model (when the model is converted to log-odds scores and used for alignments). This process is called "wing retraction" in the code; an analogy to a swept-wing fighter changing from a wings-out takeoff and landing configuration to a wings-back configuration for high speed flight.

The technical problem that this is addressing is that the Plan 7 model allows cycles through the J state. If a continuous nonemitting "mute cycle" were allowed (J, B, D states, E, and back to J), dynamic programming recursions would fail. This is why special mute states like delete states must be handled carefully in HMM dynamic programming algorithms (Durbin et al., 1998). The easiest way to prevent a mute cycle is to disallow them: make sure that the model must pass through at least one match state per path through the main model. Usually this can be done by breaking the chain somewhere and absorbing the offending probability into a non-mute path.

Wing retraction involves folding the probabilities of the terminal delete paths into the Plan 7 entry and exit probabilities. For example, in wing retraction the "algorithm dependent" B → M_3 entry probability is incremented by the probability of the "data dependent" path B → D_1 → D_2 → M_3.

Having the wing retraction step, rather than *always* folding these probabilities together, is a design decision, preserving a distinction between the "algorithm dependent" and "data dependent" parts of the model.

Parsing the residue information in input multiple sequence alignments

alphabet type: DNA or protein

By default, the alphabet type (DNA or protein) is determined automatically from the sequences in the alignment. This determination primarily relies on the fact that some amino acid residues ("EFIPQZ") do not occur in DNA sequences, even in the IUPAC degenerate code. Rarely, on small alignments, the autodetection may resort to guesswork (and print a warning), or may fail altogether. You can explicitly force the alphabet type using the **`--amino`** or **`--nucleic`** option.

> ▷ ***Can HMMER make models of sequences other than DNA or protein?*** *By default, HMMER requires DNA (RNA) or protein sequences. But if you're willing to hack the code, the alphabet is configurable; see the notes on the* **`config.h`** *file, page 13 in the installation section.*

case insensitivity

HMMER reads sequence residues in its input files in a *case-insensitive* manner. (Internally, residues are handled digitally, rather than alphabetically.) Other aspects of the file, such as sequence names, preserve their case.

> ▷ ***Why doesn't HMMER recognize Santa Cruz's SAM A2M format properly?*** *HMMER can read an aligned FASTA format that is close, but crucially not quite the Santa Cruz a2m format. The real Santa Cruz a2m format, used by the SAM profile HMM package, encodes match versus insert column information in an upper vs. lower case convention in its input files, and a - versus . convention for its gap characters. Since HMMER digitizes residues case-insensitively, and treats all gap characters identically in input files, this information is lost. HMMER cannot currently read SAM a2m files directly - even though the aligned fasta format is sometimes called a2m! This is a historical misunderstanding of mine about what the Santa Cruz format was; I thought it was simple...*

handling of degenerate residue codes

For amino acid sequences, HMMER accepts the standard IUPAC degeneracy codes (BZX). It reads U (selenocysteine) as a valid amino acid, and treats it as a serine (S). (Selenocysteine is derived from serine in vivo, so this seems as good a choice as any.) It does not accept the IUPAC `*` character for stop codons.

For nucleic acid sequences, it accepts the standard IUPAC degeneracy codes (NRYMKSWHBVD). It treats U and T as the same residue, and thus does not distinguish in any way between RNA and DNA. It treats X as if it were N, to humor the many bioinformatics software packages that aren't IUPAC-compliant.

It handles degenerate codes probabilistically, by assuming a uniform distribution over the possible residues. That is, an amino acid of X counts as a probability $\frac{1}{20}$ for each of the 20 residues.

Model architecture construction

Not all of the columns of the multiple alignment will necessarily be treated as consensus (match) columns. Some may be treated as insertions relative to the consensus. Only the consensus columns will get a model node (a trio of M/D/I states) assigned to them. The assignment of alignment columns as either "consensus" or "insert" is called *model architecture construction*. HMMER allows a choice of three architecture construction algorithms.

maximum a posteriori model architecture construction, the default

By default, HMMER uses a maximum a posteriori (MAP) architecture algorithm, e.g. one that is guaranteed to find the model architecture with the highest posterior probability for the alignment data (Durbin et al., 1998). This is a dynamic programming algorithm, and it is almost always fast enough that you won't notice that it's running – the rate limiting step in model construction is more often the sequence weighting algorithm (see below).

The MAP algorithm incorporates a prior distribution over expected model size. The prior holds the model sizes down and has the effect of requiring a certain amount of information content per column before it becomes worthwhile to make it a consensus column. This prior is a geometric distribution, controlled by a single parameter which can be set by the **`--archpri <x>`** option to any number $0 \leq x \leq 1$. By default, it is set to 0.85. Increasing it favors longer models; decreasing it favors shorter models. The default setting was determined empirically, and should be considered to be an *ad hoc* parameter, since it implies a mean expected model length of only 7 consensus residues and is unrelated to the observed distribution of protein domain consensus lengths.

> ▷ ***Why did hmmbuild build a model of length 0 for my alignment?*** *It is rare but possible for the MAP architecture algorithm to decide that the optimal architecture is* nothing *– e.g. that there is so little information in the alignment, it isn't even worth modeling it as a consensus, because the null model is a better model of it. This can happen if your alignment contains unrelated sequences, or if it has a lot of sequence fragments (and hence a lot of gaps). You should probably take this as a warning that your alignment may not be very good. If you want to work around the effect, either increase the architecture prior parameter using the* ***--archpri*** *option, or use one of the other two architecture construction algorithms,* ***--fast*** *or* ***--hand****.*

fast model architecture construction

The **--fast** option selects an alternative, heuristic model architecture algorithm: all columns that contain more than a certain fraction x of gap characters will be assigned as an insert column. This is the original Krogh/Haussler method (Krogh et al., 1994).

By default, x is 0.5. It can be set to something else using the **--gapmax <x>** option. x can be anything between 0 and 1. The larger x is, the more columns will be included in the consensus; at $x = 1$, all columns will be included, and at $x = 0$, only columns that contain all aligned residues (e.g. no gaps at all) will be called consensus columns.

hand model architecture construction

If the input alignment is in annotated Stockholm (or SELEX) format, it can have an RF (reference) coordinate line. The **--hand** option tells **hmmbuild** to parse the RF line, and use it to determine the architecture. Any column marked with a non-gap character (x, for instance) in the RF line is assigned as a consensus column; any column marked with a gap character in the RF line is assigned as an insertion column.

Hand construction can come in, well, handy. For example, one can pick a "canonical" sequence in the alignment, copy that aligned sequence as the RF annotation, and a hand-built model will then have its model nodes numbered exactly in the coordinate system of the canonical sequence. (I do this for some families where I know one sequence so well, I remember some of its residues by number, and I want the HMM to number aligned residues the way I expect.)

HMMER propagates an appropriate RF annotation line into every new multiple alignment it builds with **hmmalign**. If you build a new model from that new alignment, **hmmbuild --hand** causes the new model to have the same consensus architecture as the first model (e.g. the model that made the alignment) – for instance, the column that was aligned to match state 43 in the first model gets assigned to match state 43 in the new model. This provides a way to maintain a fixed model architecture through iterations of **hmmbuild**/**hmmsearch**/**hmmalign**, even as the number of sequences changes.

If you select **--hand** but an RF annotation line is not present on the alignment, **hmmbuild** will fail with an error message, of course.

Collecting observed emission/transition counts

Having assigned all the columns as consensus columns or insertion columns, **hmmbuild** now collects counts of symbol emissions (residues) and state transitions (insertions/deletions) from the alignment. The raw counts are adjusted by a few heuristics as follows.

sequence weighting

A profile HMM has no notion of phylogeny. It assumes, in fact, that all the observed sequences are independent, uncorrelated samples from a single model. This is clearly not true in any biological sequence data set, because sequences are related to each other by speciation and gene duplication events on a phylogenetic tree – not to mention trivial redundancy in the sequence databases. Therefore, HMMER applies a sequence weighting algorithm to downweight counts from closely related sequences, and upweight distantly related sequences.

The default weighting algorithm is the Gerstein/Sonnhammer/Chothia tree weighting algorithm (Gerstein et al., 1994).

Several weighting schemes are available as options, as follows:

Option	Weighting method	Reference
--wblosum	Henikoff simple filter weights	(Henikoff and Henikoff, 1992)
--wgsc	GSC tree weights (default)	(Gerstein et al., 1994)
--wme	maximum entropy (ME)	(Krogh and Mitchison, 1995)
--wpb	Henikoff position-based weights	(Henikoff and Henikoff, 1994a)
--wvoronoi	Sibbald/Argos Voronoi weights	(Sibbald and Argos, 1990)
--wnone	don't do any weighting	

Most of the available weighting algorithms will become rather slow for large numbers of sequences (scaling with the square of the number of sequences). If you are trying to weight by the GSC, BLOSUM, or Voronoi methods, for alignments over a threshold of n sequences, the weighting algorithm is instead switched automatically to the **--wpb** style, Henikoff position based weights (Henikoff and Henikoff, 1994a). The default switching threshold is 1000 sequences; this can be adjusted with the **--pbswitch <n>** option. Set **--pbswitch** to something very large if you don't want this algorithm switching behavior to happen.

The **--wblosum** scheme works by single linkage clustering of all sequences related by more than some threshold x of fractional sequence identity, then distributing a weight of 1 across each cluster. By default x is 0.62 (62%), the rule used for weighting the BLOSUM62 scoring matrix (Henikoff and Henikoff, 1992). The **--idlevel <x>** option can be used to set the threshold to something else. Beware, **--idlevel** also affects the effective sequence number calculation, as described in the next subsection.

There is no satisfactory theory behind setting the weights; this is an *ad hoc* step. The maximum entropy method comes with the most theoretical motivation, but even it is a bit of a stretch as far as biological realism. A correct model would incorporate an explicit model of phylogeny (e.g. "tree HMMs" (Mitchison and Durbin, 1995)). There is a growing body of literature from Jotun Hein, Ian Holmes, Bill Bruno, Richard Goldstein and others, moving towards combining profile HMMs and explicit phylogenetic modeling.

determining the effective sequence number

For N sequences, sequence weighting algorithms typically redistribute a total of N counts, such that some sequences get weights less than 1, some have weights larger than 1, but overall each sequence has an average weight of 1. In principle, though, the total sequence weight could be an arbitrary number, not necessarily N; here, call it X, the *effective sequence number*.

For example, if our alignment contained 20 sequences, but 10 of them were identical copies of the same sequence due to trivial database redundancy, we'd not only want to downweight those sequences so that they counted 1/10 as much as the other ones; we'd want to treat the alignment as a total of 11 sequences, not 20.

Normally, sequence weighting algorithms ignore this issue because it cancels out: if we were going to estimate probability parameters as simple frequencies (e.g. maximum likelihood parameter estimation), the choice of X is irrelevant.

Profile HMMs do not estimate probability parameters solely from counts, though. Observed count data are combined with *priors*, which incorporate *a priori* knowledge about what homologous, aligned sequences usually look like; profile HMM parameters are *mean posterior estimates*, not maximum likelihood estimates (Durbin et al., 1998; Sjölander et al., 1996). The less count data there are, the more the priors affect the parameter estimates, so the choice of effective sequence number X becomes relevant.

HMMER determines effective sequence number by the BLOSUM clustering rule: it counts the number of single linkage clusters above some threshold fractional identity x, where x defaults to 0.62. The threshold can be changed with the **`--idlevel <x>`** option (beware, that changes the behavior of the rarely used **`--wblosum`** weighting option too). Note that $X \leq N$. At **`--idlevel 0`**, all sequences are clustered into one group, and $X = 1$; at **`--idlevel 1`**, no sequences are clustered, and $X = N$.

The effective sequence number calculation can be shut off using the **`--noeff`** option.

There is no satisfactory theory behind the *ad hoc*-ery of the effective sequence number. Explicit phylogenetic methods will someday make it obsolete.

adjustments to the alignment

Don't be shocked, but if you used a **`--hand`** or **`--fast`** architecture construction strategy, the alignment that HMMER builds its model from is not necessarily exactly the alignment that you provided.

The reason is that the Plan 7 design disallows D → I and I → D transitions, but your alignment may imply them. For instance, consider a hand-built model from this alignment:

```
#=GC RF   xx..xx
seq1      C-a.YH
seq2      CM.v-H
```

seq1 implies a $D_2 \rightarrow I_2$ transition, and seq2 implies an $I_2 \rightarrow D_3$ transition. These illegal moves have to be dealt with somehow, before a valid Plan7 model can be built. HMMER does something truly ugly to get around this - an evil little routine, **`modelmaker.c:trace_doctor()`**, quietly *revises* the alignment to make it compatible with Plan 7. D → I moves are fixed by shoving the insertion to the left, filling the deletion and turning it into a match. I → D moves are fixed by shoving the insertion to the right. There are a few other types of illegal moves at the beginning and end of the model that **`trace_doctor()`** also deals with.

Fortunately this is rare. If you want to see what HMMER did to your beautiful alignment, you can use the **`-o <alifile>`** option to output a copy of your alignment *after* hmmbuild has worked its evil magic on it - the match columns will be annotated with an RF line, and some residues may have been realigned. For instance, a resaved copy of the above example, after **`hmmbuild --hand -o`**, is:

```
# STOCKHOLM 1.0
#=GF AU    HMMER 2.2g

seq1        CAYH
seq2        CMVH
#=GC RF     xxxx
//
```

and the gaps have (not so mysteriously, now?) disappeared.

Estimating probability parameters from counts

Counts c_x are converted to mean posterior probability estimates $\hat{p}_x$ using mixture Dirichlet priors. The equation is

$$\hat{p}_x = \sum_k P(k|\vec{c}) \frac{c_x + \alpha_x^k}{\sum_y (c_y + \alpha_y^k)}$$

for a mixture of Dirichlet parameters α, with each individual Dirichlet mixture component numbered by k. The probability of mixture k given the count vector $\vec{c}$, $P(k|\vec{c})$, is a mess of gamma functions; its equation is given by Sjölander (Sjölander et al., 1996; Durbin et al., 1998).

In the case of a single-component Dirichlet, this equation reduces to:

$$\hat{p}_x = \frac{c_x + \alpha_x}{\sum_y (c_y + \alpha_y)}$$

where the α terms are often thought of as *pseudocounts*, because they are added to the count data as if they were observations. The bigger the alpha terms, the more weight the prior has, and the more count data it takes to override the prior.

Each emission and transition probability distribution in the data-dependent parts of a Plan 7 model have a prior assigned to them. HMMER has hardcoded defaults; copies of these defaults are in **`tutorial/amino.pri`** and **`tutorial/nucleic.pri`**, in HMMER's prior file format (see page 58).

For proteins, the match emission prior is the nine-component Blocks9 mixture from Sjölander et al. (Sjölander et al., 1996). The insert emission prior is a slightly hydrophilic-biased distribution estimated from inserted residues in an early version of the Pfam database. The insert prior has then been artificially scaled to very high values of α, which has the effect of fixating the insert emission distributions in HMMER: all insertion states have virtually identical emission distributions, rather than learning from the observed data. The transition priors for M, D, and I states are single-component Dirichlets, estimated by maximum likelihood on an early version of Pfam by Graeme Mitchison.

No attempt has been made to estimate good nucleic acid priors. The transition priors are copied from the protein set. The match and insert emission priors are *plus-one* (Laplace) priors.

using customized priors

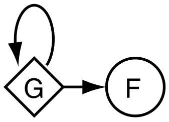
The default priors can be overridden on the command line, by providing a prior file with the **`--prior <f>`** option to **`hmmbuild`**.

the *ad hoc* PAM prior

An alternative to Dirichlet priors is available, called PAM priors, which uses an amino acid scoring matrix to contribute prior information. It is activated by providing a scoring matrix file with the **`--pam <f>`** option. There is a weighting parameter A that affects how much weight (in counts) the prior has; by default $A = 20$, but this can be set with the **`--pamwgt <x>`** option. This implements the "substitution matrix mixture" strategy discussed by Durbin et al. on p.117-119 (Durbin et al., 1998).

Calculating scores from counts

The alignment scores reported by HMMER are log-odds scores: the log of the probability of the sequence given the HMM, divided by the probability of the sequence given a "null hypothesis" that the sequence is just a random sequence, unrelated to the HMM (Barrett et al., 1997). The "null model" is also a probabilistic model, but a simpler one than the profile:



The G state has a symbol emission probability distribution for K symbols in the alphabet. By default, this distribution is set either to the average amino acid composition of SWISSPROT 34, or to 0.25 for each nucleotide. The G → G transition controls the expected length of observed random sequences; in practice, this transition probability is so close to 1 that it has very little effect. (It is set to 350/351 for protein models, 1000/1001 for DNA models.) The F state is just a dummy end state like the T state in the Plan 7 architecture.

The score of residues x in match or insert states are:

$$s_x = \frac{p_x}{g_x}$$

where p_x is the probability according to the HMM, and g_x is the probabilitity according to the null model. Transition scores are also calculated as log odds parameters. For details, see the source documentation in **`plan7.c:P7Logoddsify()`**.

As with the priors, copies of the default null model parameters are provided, in the files **`tutorial/amino.null`** and **`tutorial/nucleic.null`**.

Alternative null model files may be provided using the **`--null <f>`** option to **`hmmbuild`**.

Setting the alignment mode

The model is then configured in one of four different alignment modes, as described above. The default is hmmls mode: global with respect to the model, multihit-local in the target sequence(s). The other modes are selected with options:

`-f` hmmfs, "fragment" mode: local in model, multihit-local in sequence.

`-g` hmms, "global" mode: global in model, single hit global in sequence.

`-s` hmmsw, "Smith/Waterman" mode: local in model, single hit local in sequence.

Naming and saving the HMM

HMM files have names (enabling them to be used in multi-HMM databases for hmmpfam, like Pfam). The name must be one word, containing no whitespace. The name is obtained in one of three ways:

- if a name **`<s>`** is explicitly provided by the **`-n <s>`** option, that's the name;
- else if the alignment has a name (Stockholm `#=GF ID` or SELEX `#=ID` annotation), that's the name;

- else the name is constructed by stripping off any suffix from the alignment file name. For instance, the alignment file “rrm.phylip” would result in an HMM named “rrm”.

Finally, the HMM is saved. The default is to save it to the file named on the **`hmmbuild`** command line in a flat text, readable ASCII format (page 51. If the file already exists, HMMER refuses to overwite it unless the **`-F`** option was used to force the overwrite, or the **`-A`** option was used, which appends to an existing HMM file rather than making a new HMM file.

A more space-efficient but undocumented binary format is available, selected with the **`--binary`** option. (Do not append binary HMMs to ASCII files or vice versa; the program doesn’t check to be sure you’re not doing this.) Minor but significant speed gains can be obtained by working with large HMM databases in binary format. Usually, one would convert an entire database to binary at once, using the **`hmmconvert`** program.

5 How HMMER scores alignments and determines significance

The search programs **`hmmsearch`** and **`hmmpfam`** give you a ranked list of hits in a sequence database. Which ones are likely to be true homologous and which ones are likely to be nonhomologous to your query HMM?

HMMER gives you at least two scoring criteria to judge by: the HMMER raw score, and an E-value. Additionally, Pfam models carry a third set of criteria: six expert-calibrated raw score cutoffs that the Pfam database maintainers set. How should you interpret all this information?

Executive summary

- The best criterion of statistical significance is the E-value. The E-value is calculated from the bit score. It tells you how many false positives you would have expected to see at or above this bit score. Therefore a low E-value is best; an E-value of 0.1, for instance, means that there's only a 10% chance that you would've seen a hit this good in a search of nonhomologous sequences. *Typically, I trust the results of HMMER searches at about E=0.1 and below, and I examine the hits manually down to E=10 or so.*

- HMMER bit scores are a stricter criterion: they reflect whether the sequence is a better match to the profile model (positive score) or to the null model of nonhomologous sequences (negative score). A HMMER bit score above $\log_2$ of the number of sequences in the target database is likely to be a true homologue. For current NR databases, this rule-of-thumb number is on the order of 20 bits. Whereas the E-value measures how statistically significant the bit score is, the bit score itself is telling you how well the sequence matches your HMM. Because these things should be strongly correlated, usually, true homologues will have both a good bit score and a good E-value. However, sometimes (and these are the interesting cases), you will find remote homologues which do not match the model well (and so do not have good bit scores – possibly even negative), but which nonetheless have significant E-values, indicating that the bit score, though "bad", is still better than you would've expected by chance, so it is suggestive of homology.

- For Pfam HMMs, you can also examine six other numbers that represent bit score thresholds: two TC (trusted cutoff) scores, two GA (gathering) scores, and two NC (noise cutoff) scores. The meaning of these numbers is described below.

> ▷ ***What does it mean when I have a negative bit score, but a good E-value?*** *The negative bit score means that the sequence is not a good match to the model. The good E-value means that it's still a better score than you would've expected from a random sequence. The usual interpretation is that the sequence is homologous to the sequence family modeled by the HMM, but it's not "within" the family - it's a distant homologue of some sort. This happens most often with HMMs built from "tight" families of high sequence identity, aligned to remote homologues outside the family. For example, an actin HMM aligned to an actin-related protein will show this behavior - the bit score says the sequence isn't an actin (correct) but the E-value says it is significantly related to the actin family (also correct).*

In more detail: HMMER bit scores

The bit score is a log-odds score in log base two (thus, in units of *bits*. Specifically, it is:

$$S = \log_2 \frac{P(\text{seq}|\text{HMM})}{P(\text{seq}|\text{null})}.$$

$P(\text{seq}|\text{HMM})$ is the probability of the target sequence according to your HMM. $P(\text{seq}|\text{null})$ is the probability of the target sequence given a “null hypothesis” model of the statistics of random sequence. In HMMER, this null model is a simple one-state HMM that says that random sequences are i.i.d. sequences with a specific residue composition (this “null model distribution” is part of the HMM save file, and it can be altered when you do an **`hmmbuild`**).

Thus, a positive score means the HMM is a better model of the target sequence than the null model is (e.g. the HMM gives a higher probability).

`hmmsearch` reports two hit lists, with different types of scores. The first list is ranked by *per-sequence* scores, and the second is ranked by *per-domain* scores. The per-sequence score is the score of the entire target sequence according to the HMM. A per-domain score is a score of one matching subsequence (domain) from the target sequence: the per-domain score is calculated as if that subsequence was the entire target sequence.

Because HMMER’s Plan 7 model is capable of looping, and therefore of modeling more than one copy of a particular domain in a target sequence (for instance, a closely spaced array of immunoglobulin superfamily domains), HMMER may well report more than one domain per target sequence.

For single domain proteins, the per-sequence and per-domain scores are identical. For multi-domain proteins, the per-sequence score is the sum of the individual per-domain scores.

You can specify cutoffs for per-sequence and/or per-domain scores and/or E-values when you use **`hmmsearch`** or **`hmmpfam`**. For a specific sensible use of such cutoffs, read about how the Pfam TC/NC/GA cutoffs are set and used (below).

interaction of multihit alignment with negative bit scores

HMMER will always report a score for the one best domain it finds in the target sequence, even if that hit is a negative bit score. The Plan 7 model architecture forces a target sequence to pass through the model at least one time.

However, each *subsequent* domain must have a positive bit score to be identified. If an additional domain has a negative bit score, HMMER can find a better scoring alignment by using the “nonhomologous” N, C, or J states to account for the residues in that domain – *even if the domain would have a significant E-value*.

> ▷ ***Why does HMMER give this domain a good E-value when I just give it the domain sequence, but it doesn’t find the domain at all in the context of the whole sequence the domain came from?*** *See above. The behavior of being able to report the single best hit, regardless of score, produces one counterintuitive behavior: statistically significant domains will be invisible in a sequence if they have negative bit scores. You can find cases where you can find additional domains with good E-values but negative bit scores by searching against a sequence that you’ve cut into smaller pieces. This is not a bug; it’s a limitation of the approach.*

In more detail: HMMER E-values

The E-value is the expected number of false positives with scores at least as high as your hit.

Unlike the raw score, the E-value is dependent on the size of the database you search. If you detect a hit with an E-value of 0.1 in a search of a sequence database with 100,000 sequences, then you happen to re-score the sequence all by itself, you will get an E-value 100,000 times better. The E-value is quite literally the expected number of false positives at this raw score; the larger the database you search, the greater the number of expected false positives.

▷ ***Why do I get a different E-value when I search against a file containing my sequence, than I got when I searched the database?*** *See above. This behavior is shared with BLAST and FASTA P-value and E-values, so it should not be unfamiliar to most users. However, it can cause consternation: a related phenomenon is that a hit that is marginally significant this year may no longer be significant next year, when the database is twice as large. You can specify a database size with the* ***`-Z`*** *option.*

▷ ***Why do hmmsearch and hmmpfam give me different E-values?*** *From the above discussion, it should also be clear that if you use* ***`hmmpfam`*** *to search a sequence against Pfam (say, ~ 3000 models) to find a matching HMM, you will get a different E-value than using* ***`hmmsearch`*** *to search that single HMM against a sequence database to find the original sequence, because the size of the search spaces is entirely different. For hmmpfam, the search space size Z is the number of* models*; for hmmsearch, the search space size Z is the number of* sequences*. Some people want to argue that this is wrong, but it's something you have to deal with; that's statistics for you.*

HMMER has two ways of calculating E-values. One way is inaccurate but analytical (and fast); the other way is more accurate but empirical (and slow).

If your HMM has not been calibrated with **`hmmcalibrate`**, HMMER uses the analytic calculation. This is a conservative calculation, meaning that the "true" E-value will be lower; sometimes much lower. The calculation is essentially the same as that given in (Barrett et al., 1997).

It is highly recommended that you calibrate HMMs with **`hmmcalibrate`**. **`hmmcalibrate`** writes two parameters into your HMM file on a line labeled "EVD": these parameters are the μ (location) and λ (scale) parameters of an extreme value distribution (EVD) that best fits a histogram of scores calculated on randomly generated sequences of about the same length and residue composition as SWISS-PROT. You only need to do this calibration once for a given HMM. All the Pfam HMMs come pre-calibrated.

fitting extreme value distributions to HMMER score histograms

Now, there's good news and bad news about extreme value distributions.

Frst the good news. The extreme value distribution fitting is done with a rather robust and personally satisfying chunk of maximum likelihood code (see **`histogram.c`** in the codebase). The ML approach was suggested to me by Stephen Altschul, who directed me to a lovely textbook by Lawless (Lawless, 1982). A brief technical report on HMMER's EVD fitting is available at ftp://ftp.genetics.wustl.edu/pub/eddy/papers/evd.pdf. Any EVD fitting code that is relying on linear regression fits to log-log plots is bound to be less accurate, judging from my tests.

However, the down side is that most profile HMM scores don't fit the extreme value distribution well. Fully local alignments (models built with **`hmmbuild -f`** or **`hmmbuild -s`** fit the EVD fairly well, as expected for Smith/Waterman style alignments, for the same reasons that gapped BLAST or FASTA alignment scores have been shown to be approximately EVD-distributed. By default, though, **`hmmbuild`** makes models for "glocal" alignments which are global with respect to the model, and multi-hit-local with respect to the sequence. Also called "profile scores" by Waterman, this sort of alignment score is known not to be EVD-distributed (Goldstein and Waterman, 1994).

Nonetheless, empirically, the tail of the distribution around E=1 is falling off more or less like an EVD, and this is the region of interest. HMMER does a "censored EVD fit" from the peak of the distribution down to the right tail, and does not include the data to the left of the peak in its fit. This produces a reasonable fit in the important region of the scores. Empirically, HMMER E-values tend to be accurate in the critical region (E 1), despite the lack of mathematical foundation.

The end of **`hmmsearch`** output shows you the observed histogram, and (if the model is calibrated) an expected curve according to the EVD fit. You should observe that the relevant tail (around E=1) is more or less well fitted. The bulk of the distribution, particularly at lower scores, is usually poorly fitted, but this is not the area of interest.

In more detail: Pfam TC/NC/GA cutoffs

When a Pfam model is built, the Pfam curation team keeps track of per-sequence and per-domain scores of every sequence in a large nonredundant database. They record three types of score cutoffs on Pfam HMM files:

GA (gathering cutoffs) : the scores used as cutoffs in constructing Pfam. All domains that are in a sequence satisfying the GA1 per-sequence score, and which themselves satisfy the GA2 per-domain score, will be included in the Pfam "full alignment".

TC (trusted cutoffs) : the scores of the lowest-scoring hit(s) that were included as true member(s) of the Pfam family. The per-domain TC2 score is the score of the lowest scoring domain *in a sequence with a per-sequence score over the TC1 cutoff*; therefore, the TC1 and TC2 scores could conceivably come from different targets. Hits above these scores are "within" the Pfam family and almost certainly members.

NC (noise cutoffs) : the scores of the highest-scoring hit(s) that were *not* included as true members of the Pfam family, because they were considered to be the top of the noise. Calculated analogously to TC1 and TC2. Hits above the NC cutoff are above the top scoring noise in the Pfam NR database search, so are likely homologues, but not as trustworthy as hits over the GA or TC cutoffs.

In order of increasing conservativeness, the cutoffs rank: NC, GA, and TC.

The GA cutoffs, being the actual cutoffs used in constructing Pfam, are a very good choice to use to collate large-scale automated data, like counting protein family members in a sequenced genome.

The TC and NC cutoffs are less useful, and only really there as documentation of Pfam construction. In general, the TC cutoffs would be extremely conservative cutoffs to use in a database search, more conservative than GA. The NC cutoffs are less conservative than GA.

Why use GA (or the other cutoffs) instead of the E-value? Pfam artificially imposes a "flat", nonhierarchical structure on protein sequence space. Pfam asserts that no Pfam family is related to any other Pfam family. This is obvious nonsense: many Pfam families are in fact homologous. The different 7-TM (G-protein coupled receptor) families are one example; the different GTPase families are another. HMMER often detect significant relationships between families that Pfam chooses to suppress. In these cases, the Pfam GA cutoff will be elevated to artifically separate two homologous but distantly related subgroups of the same structural superfamily.

> ▷ ***Why isn't sequence X included in a Pfam full alignment? It has a significant score!*** *For the reasons above, the sequences in Pfam full alignments are harvested using curated GA thresholds, rather than using score or E-value thresholds. Interpro-based counts of domains in genome analyses also use curated GA thresholds. Please don't go writing a paper that claims HMMs don't detect some similarity until you've done the experiment with HMMs instead of just looking at curated Pfam classifications. Yes, such paper(s) have been written. Sigh.*

The mechanism that HMMER uses to incorporate up these cutoffs is general: Stockholm format multiple sequence alignments can carry appropriate TC, NC, and GA markup lines. This means that you can use a Pfam-like cutoff system if you like, just by adding the appropriate Stockholm markup to your collection of alignments. When these numbers are available in the HMM, HMMER search programs provide options (**`--cut_ga`**, etc.) for setting search cutoffs to GA, TC, or NC automatically.

Biased composition filtering: the null2 model

I've lied. HMMER bit scores are actually calculated as log odds scores relative to *two* null hypotheses. The first is the null model built into the profile HMM when it was built with **`hmmbuild`**. The second, called *null2,* is an *ad hoc* model calculated on the fly for each alignment, from the characteristics of that alignment. The purpose of null2 is to compensate for false positive hits caused by simple biased composition regions in the target sequence.

Common biased composition filters like XNU, DUST, and SEG are qualitative filters – if a region is detected as biased composition, it is masked (the residues are converted to X's). A drawback of this approach is that some real domains are biased composition – collagens, for example, are almost completely masked by standard filters, and you won't see true collagen homologies. The HMMER composition filter is a quantitative filter, that tests whether the sequence is a better match to the profile HMM, the null (random composition) model, or a null2 model of biased amino acid composition.

This is a Good Thing, but on the other hand, the null2 model is not very sophisticated. It is a single-state HMM just like the main null model, which means it only captures residue composition, like DUST; no attempt is made in HMMER to filter short-period repetitive sequences like the XNU algorithm does. (The XNU masking algorithm is available as an option, **`--xnu`**.)

The null2 model was motivated by an artifact that appeared in the Plan 7 implementation of HMMER2. In HMMER1, insertion emissions always scored zero: the insertion emission distribution was assumed to be identical to overall amino acid composition. But that loses a small amount of information. On average, insertions tend to occur in surface loops of proteins. Inserted residues have a small but significant hydrophilic bias. HMMER2's parameterization takes that into account. As a result, insertions of hydrophilic residues get slightly positive scores, while insertions of hydrophobic residues get slightly negative scores.

> ▷ ***Why is my alignment annotating inserted residues with +'s? Shouldn't +'s only appear for aligned consensus residues?*** *This is not a bug. See above; certain inserted residues can contribute positive score to the alignment, because they are slighly more likely to be seen in insertions than in overall amino acid composition.*

If you give hydrophilic insertions positive score, and you combine that with the ability to do Smith/Waterman local alignment, you get an undesirable artifact: it's possible to get alignments that enter the model at some match state, align a residue to that match state, then proceed to use the insertion state for a long hydrophilic stretch before exiting the model from the next match state: thus, a "high scoring" alignment that aligns to only two consensus positions in the model, with a long insertion. It's this artifact that motivated the development of the null2 filter, though the null2 filter also happens to work well on a variety of other biased-composition scenarios.

A different null2 model is calculated for every alignment. The 20 emission probabilities of the null2 model are calculated as the occurrence-weighted average emission probability of all the states in the alignment's state path π. For example, if the state path is 52 emissions long and contains M_{32}, 50 inserted residues aligned to I_{32}, and M_{33}, the null2 model will be calculated by averaging 52 emission distributions (50 copies of I_{32}, 1 each of M_{32} and M_{33}).

But now we've got *two* null hypotheses. We said we report a bit score that's a log-odds ratio of our model likelihood and *one* null hypothesis likelihood. How do we calculate a score if we have more than one null hypothesis? HMMER does a bit of algebraic sleight of hand here, to arrive at an additive correction to the original score that it calls the "null2 score correction".

Because the mysterious null2 correction generated a fair amount of faq-ish email traffic in the past, let's now finally slog through and document the null2 score correction calculation, and the unpublished (but relatively trivial) theory behind it.

derivation of the null2 score correction

We arrived at the parameters of the null2 model in a very *ad hoc* way. However, after that, the way HMMER arrives at the final bit score once the null2 parameters have been determined is clean (e.g. derivable) Bayesian probability theory, and sort of a novel idea for quantitative sequence masking, I think.

If we take the Bayesian view, we're interested in the probability of a hypothesis H given some observed data D:

$$P(H|D) = \frac{P(D|H)P(H)}{\sum_{H_i} P(D|H_i)P(H_i)},$$

an equation which forces us to state explicit probabilistic models not just for the hypothesis we want to test, but also for the alternative hypotheses we want to test against. Up until now, we've considered two hypotheses for an observed sequence D: either it came from our profile HMM (call that model M), or it came from our null hypothesis for random, unrelated sequences (call that model N). If these are the only two models we consider, the Bayesian posterior for the model M is:

$$P(M|D) = \frac{P(D|M)P(M)}{P(D|M)P(M) + P(D|N)P(N)}$$

Recall that the log odds score reported by HMMER's alignment algorithms is

$$s = \log \frac{P(D|M)}{P(D|N)}.$$

Let's assume for simplicity that *a priori*, the profile and the null model are equiprobable, so the priors $P(M)$ and $P(N)$ cancel. Then the log odds score s is related to the Bayesian posterior by a sigmoid function,

$$P(M|D) = \frac{e^s}{e^s + 1}.$$

(We don't have to assume that the two hypotheses are equiprobable; keeping these around would just add an extra $\pi = \log P(M)/P(N)$ factor to s. We'll reintroduce these prior log odds scores π shortly.)

The simple sigmoid relationship between the posterior and the log odds score suggests a plausible basis for calculating a score that includes contributions of more than one null hypothesis: **we desire a generalized score S such that:**

$$\frac{e^S}{e^S + 1} = P(M|D),$$

for *any* number of alternative hypotheses under consideration.

So, let N_i represent any number of alternative null models N_i. Then, by algebraic rearrangement of Bayes' theorem,

$$S = \log \frac{P(S|M)P(M)}{\sum_i P(S|N_i)P(N_i)}.$$

We saw above that HMMER internally calculates a log odds score s, of the model relative to the first null hypothesis. Let's now call that s_M, the alignment score of the model. HMMER extends that same scoring system to all additional competing hypotheses, calculating a log odds score relative to the first null hypothesis for any additional null hypotheses $i > 1$:

$$s_i = \log \frac{P(D|N_i)}{P(D|N_1)}$$

We can also state prior scores π_i for how relatively likely each null hypothesis is, relative to the main one:

$$\pi_i = \log \frac{P(N_i)}{P(N_1)}$$

(Remember that we assumed $\pi_M = 0$; but we're going to put it back in anyway now.)

Now we can express S in terms of the internal scores s and prior scores π:

$$S = \log \frac{e^{s_M + \pi_M}}{1 + \sum_{i>1} e^{s_i + \pi_i}},$$

which therefore simply amounts to an additive correction of the original score, $(s_M + \pi_M)$:

$$S = (s_M + \pi_M) - \log \left(1 + \sum_{i>1} e^{s_i + \pi_i} \right)$$

So, to calculate its reported score, HMMER uses four quantities:

s_M The (simple, uncorrected) log odds score for the model, calculated by optimal alignment of the model to the sequence.

π_M The log odds of the priors, $\log P(M)/P(N_1)$. HMMER implicitly assumes this factor to be 0.

s_2 The (simple, uncorrected) log odds score for the null2 hypothesis, calculated by rescoring the residues of the alignment under the null2 model.

π_2 The log odds of the priors, $\log P(N_2)/P(N_1)$. HMMER arbitrarily assumes that the null2 model is $\frac{1}{256}$ as likely as the main null model, so this factor is -8 bits.

Thus, if you read the code in **`masks.c:TraceScoreCorrection()`**, you'll see:

- we start with an alignment.
- the null2 model is calculated from the state path, by averaging over the emission distributions of all M and I states that appear in the path.
- The null2 emission probabilities are converted to log odds scores, relative to the main null model.
- The residues in the alignment are rescored under null2, which gives s_2.
- We subtract the arbitrary 8 bit prior penalty from s_2, so now we have $s_2 + \pi_2$.
- We calculate the value $\log(1 + e^{s_2 + \pi_2})$. This quantity is the "null2 score correction"; we return it.

The null2 correction is usually close to zero, for random sequences, but becomes a significant quantitative penalty on biased composition sequences. It gets added to the original alignment score to form HMMER's final bit score.

(Finally, in order to guarantee that the scores of individual domains add up to the score of a complete sequence, HMMER calculates and applies separate null2 corrections for each aligned domain.)

6 File formats

HMMER save files

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

```
HMMER2.0  [2.3]
NAME  rrm
ACC   PF00076
DESC  RNA recognition motif. (a.k.a. RRM, RBD, or RNP domain)
LENG  77
ALPH  Amino
RF    no
CS    yes
MAP   yes
COM   ../src/hmmbuild -F rrm.hmm rrm.sto
NSEQ  90
DATE  Tue Apr 29 11:01:43 2003
CKSUM 8325
GA    15.2 0.0
TC    15.2 0.3
XT      -8455     -4  -1000  -1000  -8455     -4  -8455     -4
NULT      -4  -8455
NULE     595  -1558     85    338   -294    453  -1158    (...)
   -21   -313     45    531    201    384  -1998   -644
HMM        A      C      D      E      F      G      H    (...)
         m->m   m->i   m->d   i->m   i->i   d->m   d->d   b->m   m->e
          -16      *  -6492
     1  -1084    390  -8597  -8255  -5793  -8424  -8268   (...)  1
     -   -149   -500    233     43   -381    399    106   (...)
     C     -1 -11642 -12684   -894  -1115   -701  -1378    -16      *
     2  -2140  -3785  -6293  -2251   3226  -2495   -727   (...)  2
     -   -149   -500    233     43   -381    399    106   (...)
     C     -1 -11642 -12684   -894  -1115   -701  -1378      *      *
(...)
    76  -2255  -5128   -302    363   -784  -2353   1398   (...)  103
     -   -149   -500    233     43   -381    399    106   (...)
     E     -1 -11642 -12684   -894  -1115   -701  -1378      *      *
    77   -633    879  -2198  -5620  -1457  -5498  -4367   (...)  104
     -      *      *      *      *      *      *      *   (...)
     C      *      *      *      *      *      *      *      *      0
//
```

The HMMER2 format provides all the necessary parameters to compare a protein sequence to a HMM, including the search mode of the HMM (local vs. global, and more), the null (background) model, and the statistics to evaluate the match on the basis of a previously fitted extreme value distribution.

The format consists of one or more HMMs. Each HMM starts with the identifier “HMMER2.0” and ends with // on a line by itself. The identifier allows backward compatibility as the HMMER software evolves. The closing // allows multiple HMMs to be concatenated into a single file to provide a database of HMMs.

The format for an HMM is divided into two regions. The first region contains text information and miscalleneous parameters in a (roughly) tag-value scheme, akin to EMBL formats. This section is ended by a line beginning with the keyword **`HMM`**. The second region is of a more fixed whitespace-limited format that contains the main model parameters. It is ended by the // that ends the entire definition for a single profile HMM.

Both regions contain numbers representing the probabilities that parameterize the HMM. These are stored as integers which are related to the probability via a log-odds calculation. The log-odds score calculation is defined in **`mathsupport.c`** and is:

$$\text{score} = (\texttt{int})\texttt{floor}(0.5 + (\texttt{INTSCALE} * \log_2(\text{prob}/\text{null-prob})))$$

so conversely, to get a probability from the scores in an HMM save file:

$$\text{prob} = \text{null-prob} * 2^{\text{score}/\texttt{INTSCALE}}$$

`INTSCALE` is defined in **`config.h`** as 1000.

Notice that you must know a null model probability to convert scores back to HMM probabilities.

The special case of prob = 0 is translated to “*”, so a score of * is read as a probability of 0. Null model probabilities are not allowed to be 0.

This log-odds format has been chosen because it has a better dynamic range than storing probabilities as ASCII text, and because the numbers are more meaningful to a human reader to a certain extent: positive values means a better than expected probability, and negative values a worse than expected probability. However, because of the conversion from probabilities, it should be noted that *you should not edit the numbers in a HMMER save file directly*. The HMM is a probabilistic model and expects state transition and symbol emission probability distributions to sum to one. If you want to edit the HMM, you must understand the underlying Plan7 probabilistic model, and ensure the correct summations yourself.

A more detailed description of the format follows.

header section

In the header section, each line after the initial identifier has a unique tag of five characters or less. For shorter tags, the remainder of the five characters is padded with spaces. Therefore the first six characters of these lines are reserved for the tag and a space. The remainder of the line starts at the seventh character. The parser does require this.

`HMMER2.0` File format version; a unique identifier for this save file format. Used for backwards compatibility. *Not* necessarily the version number of the HMMER software that generated it; rather, the version number of the last HMMER that changed the format so much that a whole new function had to be introduced to do the parsing. (i.e., HMMER 2.8 might still be writing save files that are headed **`HMMER2.0`**). The remainder of the line after the HMMER2.0 tag is free text: HMMER’s **`hmmbuild`** currently writes the real version number there in brackets, e.g. **`[2.3]`** in the example ab **Mandatory.**

`NAME <s>` Model name; **`<s>`** is a single word name for the HMM. No spaces or tabs may occur in the name. **`hmmbuild`** will use the `#=GF ID` line from a Stockholm alignment file to set the name. If this is not present, or the alignment is not in Stockholm format, **`hmmbuild`** sets the HMM name using the name of the alignment file, after removing any file type suffix. For example, an HMM built from the alignment file **`rrm.slx`** would be named **`rrm`** by default. **Mandatory.**

`ACC <s>` Accession number; **`<s>`** is a one-word accession number for an HMM. This is picked up from the `#=GF AC` line in a Stockholm format alignment. Accessions are stable identifiers for Pfam models, whereas names may change from release to release. Added in v2.1.1. **Optional.**

`DESC <s>` Description line; **`<s>`** is a one-line description of the HMM. **`hmmbuild`** will use the `#=GF DE` line from a Stockholm alignment file to set the description line. If this is not present, or the alignment is not in Stockholm format, the description line is left blank; one can be added manually (or by Perl script) if you wish. **Optional.**

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

`ALPH <s>` Symbol alphabet; **`<s>`** must be either **`Amino`** or **`Nucleic`**. This determines the symbol alphabet and the size of the symbol emission probability distributions. If **`Amino`**, the alphabet size is set to 20 and the symbol alphabet to "ACDEFGHIKLMNPQRSTVWY" (alphabetic order). If **`Nucleic`**, the alphabet size is set to 4 and the symbol alphabet to "ACGT". Case insensitive. **Mandatory.**

`RF <s>` Reference annotation flag; **`<s>`** must be either **`no`** or **`yes`** (case insensitive). If set to **`yes`**, a character of reference annotation is read for each match state/consensus column in the main section of the file (see below); else this data field will be ignored. Reference annotation lines are currently somewhat inconsistently used. The only major use in HMMER is to specify which columns of an alignment get turned into match states when using the **`hmmbuild --hand`** manual model construction option. Reference annotation can only be picked up from Stockholm or SELEX format alignments. See description of Stockholm and SELEX formats for more details on reference annotation lines. **Optional**; assumed to be no if not present.

`CS <s>` Consensus structure annotation flag; **`<s>`** must be either **`no`** or **`yes`** (case insensitive). If set to **`yes`**, a character of consensus structure annotation is read for each match state/consensus column in the main section of the file (see below); else this data field will be ignored. Consensus structure annotation lines are currently somewhat inconsistently used. Consensus structure annotation can only be picked up from Stockholm or SELEX format alignments. See description of Stockholm and SELEX formats for more details on consensus structure annotation lines. **Optional**; assumed to be no if not present.

`MAP <s>` Map annotation flag; **`<s>`** must be either **`no`** or **`yes`** (case insensitive). If set to **`yes`**, each line of data for the match state/consensus column in the main section of the file is followed by an extra number. This number gives the index of the alignment column that the match state was made from. This information provides a "map" of the match states ($1..M$) onto the columns of the alignment ($1..alen$). It is used for quickly aligning the model back to the original alignment, e.g. when using **`hmmalign --mapali`**. Added in v2.0.1. **Optional**; assumed to be no if not present.

`COM <s>` Command line log; **`<s>`** is a one-line command. There may be more than one **`COM`** line per save file. These lines record the command line for every HMMER command that modifies the save file. This helps us automatically log Pfam construction strategies, for example. **Optional.**

`CKSUM <d>` Training alignment checksum; **`<d>`** is a nonzero positive integer. This number is calculated from the training alignment and stored when **`hmmbuild`** is used. It is used in conjunction with the alignment map information to to verify that some alignment is indeed the alignment that the map is for. Added in v2.0.1. **Optional.**

`GA <f> <f>` Pfam gathering thresholds GA1 and GA2. This is a feature in progress. See Pfam documentation of GA lines. Added in v2.1.1. **Optional.**

`TC <f> <f>` Pfam trusted cutoffs TC1 and TC2. This is a feature in progress. See Pfam documentation of TC lines. Added in v2.1.1. **Optional.**

`NC <f> <f>` Pfam noise cutoffs NC1 and NC2. This is a feature in progress. See Pfam documentation of NC lines. Added in v2.1.1. **Optional.**

`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.**

`DATE <s>` Creation date; **`<s>`** is a date string. This field is only used for logging purposes. **Optional.**

`XT <d>*8` Eight “special” transitions for controlling parts of the algorithm-specific parts of the Plan7 model. The null probability used to convert these back to model probabilities is 1.0. The order of the eight fields is N → B, N → N, E → C, E → J, C → T, C → C, J → B, J → J. (Another way to view the order is as four transition probability distributions for N,E,C,J; each distribution has two probabilities, the first one for “moving” and the second one for “looping”.) For an explanation of these special transitions (and definition of the state names), read the Plan7 architecture documentation. **Mandatory.**

`NULT <d> <d>` The transition probability distribution for the null model (single G state). The null probability used to convert these back to model probabilities is 1.0. The order is G → G, G → F. **Mandatory.**

`NULE <d>*K` The symbol emission probability distribution for the null model (G state); consists of K (e.g. 4 or 20) integers. The null probability used to convert these back to model probabilities is $1/K$. (Yes, it’s a little weird to have a “null probability” for the null model symbol emission probabilities; this is strictly an aesthetic decision, so one can look at the null model and easily tell which amino acids are more common than chance expectation in the background distribution.) **Mandatory.**

`EVD <f> <f>` The extreme value distribution parameters μ and λ, respectively; both floating point values. λ is positive and nonzero. These values are set when the model is calibrated with **`hmmcalibrate`**. They are used to determine E-values of bit scores. If this line is not present, E-values are calculated using a conservative analytic upper bound. **Optional.**

`HMM` HMM flag line; flags the end of the header section. Otherwised not parsed. Strictly for human readability, the symbol alphabet is also shown on this line, aligned to the **`NULE`** fields and the fields of the match and insert symbol emission distributions in the main model. The immediately next line is also an unparsed human annotation line: column headers for the state transition probability fields in the main model section that follows. Both lines are **mandatory.**

main model section

All the remaining fields are **mandatory**, except for the alignment map.

The first line in the main model section is atypical; it contains three fields, for transitions from the B state into the first node of the model. *The only purpose of this line is to set the B → D transition probability*. The first field is the score for $1 - t(B \rightarrow D)$. The second field is always "*" (there is no B → I transition). The third field is the score for $t(B \rightarrow D)$. The null probability used for converting these scores back to probabilities is 1.0. In principle, only the third number is needed to obtain $t(B \rightarrow D)$. In practice, HMMER reads both the first and the third number, converts them to probabilities, and renormalizes the distribution to obtain $t(B \rightarrow D)$. [6]

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:

Match emission line The first field is the node number (1..M). The HMMER parser verifies this number as a consistency check (it expects the nodes to come in order). The next K numbers for match emission scores, one per symbol. The null probability used to convert them to probabilities is the relevant null model emission probability calculated from the **`NULE`** line.

If **`MAP`** was **`yes`**, then there is one more number on this line, representing the alignment column index for this match state. See **`MAP`** above for more information about the alignment map, and also see the man pages for **`hmmalign --mapali`**. Added in v2.0.1. This field is optional, for backwards compatibility with 2.0.

Insert emission line The first field is a character of reference annotation (RF), or "-" if there is no reference annotation. The remaining fields are K numbers for insert emission scores, one per symbol, in alphabetic order. The null probability used to convert them to probabilities is the relevant null model emission probability calculated from the **`NULE`** line.

State transition line The first field is a character of consensus structure annotation (CS), or "-" if there is no consensus structure annotation. The remaining 9 fields are state transition scores. The null probability used to convert them back from log odds scores to probabilities is 1.0. The order of these scores is given by the annotation line at the top of the main section: it is M → M, M → I, M → D; I → M, I → I; D → M, D → D; B → M; M → E.

The insert emission and state transition lines for the final node M are special. Node M has no insert state, so all the insert emissions are given as "*". (In fact, this line is skipped by the parser, except for its RF

[6] OK, it's more complicated than it has to be. I discovered this temporary insanity when I tried to document it.

annotation.) There is also no next node, so only the B → M and M → E transitions are valid; the first seven transitions are always "*". (Incidentally, the M → E transition score for the last node is always 0, because this probability has to be 1.0.)

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

renormalization

After the parser reads the file and converts the scores back to probabilities, it renormalizes the probability distributions to sum to 1.0 to eliminate minor rounding/conversion/numerical imprecision errors. If you're trying to emulate HMMER save files, it might be useful to know what HMMER considers to be a probability distribution. See **`Plan7Renormalize()`** in **`plan7.c`** for the relevant function.

null emissions The K symbol emissions given on the **`NULE`** line.

null transitions The two null model transitions given on the **`NULT`** line.

N,E,C,J specials Each of the four special states N,E,C,J have two state transition probabilities (move and loop). All four distributions are specified on the **`XT`** line.

B transitions M B → M entry probabilities are given by the 9th field in the state transition line of each of the M nodes. The B → D transition (from the atypical first line of the main model section) is also part of this state transition distribution.

match transitions One distribution of 4 numbers per node; $M \rightarrow M$, $M \rightarrow I$, $M \rightarrow D$, and $M \rightarrow E$ (fields 2, 3, 4, and 10 in the state transition line of each node). Note the asymmetry between B → M and M → E; entries are a probability distribution of their own, while exits are not.

insert transitions One distribution of 2 numbers per node; $I \rightarrow M$, $I \rightarrow I$ (fields 5 and 6 of the state transition line of each node).

delete transitions One distribution of 2 numbers per node; $D \rightarrow M$, $D \rightarrow D$ (fields 7 and 8 of the state transition line of each node).

match emissions One distribution of K numbers per node; the K match symbol emissions given on the first line of each node in the main section.

insert emissions One distribution of K numbers per node; the K insert symbol emissions given on the second line of each node in the main section.

note to developers

Though I make an effort to keep this documentation up to date, it may lag behind the code. For definitive answers, please check the parsing code in **`hmmio.c`**. The relevant function to see what's being written is **`WriteAscHMM()`**. The relevant function to see how it's being parsed is **`read_asc20hmm()`**.

HMMER null model files

A “null model” is used to calculate HMMER log odds scores. The null model states the expected background occurrence frequencies of the 20 amino acids or the 4 nucleotide bases. The null model also contains a parameter called p1, which is the $G \to G$ transition probability in the Plan7 null model (see the figure in the Introduction).

For protein models, by default, the 20 residue frequencies are set to the amino acid composition of SWISS-PROT 34, and p1 is set to 350/351 (which, because the Plan7 null model implies a geometric length distribution, states that the mean length of a protein is about 350 residues). For DNA/RNA models, by default, the 4 residue frequencies are set to 0.25 each, and p1 is set to 1000/1001. [In the code, see **`prior.c:P7DefaultNullModel()`**, and the amino acid frequencies are set in **`iupac.c:aafq`**.]

Each HMM carries its own null model (see above, HMM file format). The null model is determined when the model is built using **`hmmbuild`**. The default null model can be overridden using the **`--null <f>`** option to **`hmmbuild`**, where **`<f>`** is the name of a null model file.

Two example null model files, **`amino.null`** and **`nucleic.null`**, are provided in the **`tutorial`** subdirectory of the HMMER distribution. (They are copies of the internal default HMMER null model settings.) **`nucleic.null`** looks like this:

```
# nucleic.null
#
# Example of a null model file for DNA/RNA sequences.
# The values in this file are the HMMER 2 default
# settings.

Nucleic
0.25     # A
0.25     # C
0.25     # G
0.25     # T
0.999001 # p1
```

Anything on a line following a **`#`** is a comment, and is ignored by the software. Blank lines are also ignored. Valid fields are separated by blanks or new lines. Only the order that the fields occur in the file is important, not how they’re put on lines; for example, 20 required fields can all occur on one line separated by blanks, or on 20 separate lines.

There must be 6 or 22 non-comment fields in a null model file, occurring in the following order:

Alphabet type The first (non-comment) word in the file must be Nucleic or Amino, specifying what kind of sequence the null model is for.

Emission probabilities 4 or 20 background frequencies for the amino acids or nucleotides. These *must* come in alphabetical order (the A, C, G, T comments in the example above are only for easier human viewing, and aren’t parsed by the software).

p1 probability The $G \to G$ transition probability in the null model. Basically, if the expected mean length of target sequences is x, p1 should be $\frac{x}{x+1}$.

Null model files are parsed in **`prior.c:P7ReadNullModel()`**.

HMMER prior files

Observed counts of emissions (residues) and transitions (insertions and deletions) in a multiple alignment are combined with *Dirichlet priors* to convert them to probabilities in an HMM.

For protein models, by default, HMMER uses a nine-component mixture Dirichlet prior for match emissions, and single component Dirichlet priors for insert emissions and transitions. The nine-component match emission mixture Dirichlet comes from the work of Kimmen Sjölander (Sjölander et al., 1996).

For DNA/RNA models, by default, HMMER uses single component Dirichlets.

Two example null model files, **`amino.pri`** and **`nucleic.pri`**, are provided in the **`tutorial`** subdirectory of the HMMER distribution. (They are copies of the internal default HMMER prior settings.)

The way the format of these files is parsed is identical to null models: everything after a **`#`** on a line is a comment, the order of occurrence of the fields is important, and fields must be separated by either blanks or newlines.

A prior file consists of the following fields:

Strategy Must be the keyword **`Dirichlet`**. Currently this is the only available prior strategy in the public HMMER release.

Alphabet type Must be either **`Amino`** or **`Nucleic`**.

Transition priors 1 + 8a fields, where a is the number of transition mixture components. The first field is the number of transition prior components, a (often just 1). Then, for each component, eight fields follow: the prior probability of that mixture component (1.0 if there is only one component), then the Dirichlet alpha parameters for the seven transitions, in order of $M \rightarrow M$, $M \rightarrow I$, $M \rightarrow D$, $I \rightarrow M$, $I \rightarrow I$, $D \rightarrow M$, $D \rightarrow I$.

Match emission priors 1 + (5 or 21)b fields, where b is the number of match emission mixture components. The first field is the number of match emission mixture components, b. Then, for each component, 5 or 21 fields follows: the prior probability of that mixture component (1.0 if there is only one component), then the Dirichlet alpha parameters for the 4 or 20 residue types, in alphabetical order.

Insert emission priors 1 + (5 or 21)c fields, where c is the number of insert emission mixture components. The first field is the number of insert emission mixture components, c. Then, for each component, 5 or 21 fields follows: the prior probability of that mixture component (1.0 if there is only one component), then the Dirichlet alpha parameters for the 4 or 20 residue types, in alphabetical order.

In the code, prior files are parsed by **`prior.c:P7ReadPrior()`**.

Sequence files

supported file formats

HMMER can automatically recognize and parse a number of common file formats. The best supported of these formats are listed below. If you know your sequence database is in one of these formats, you can use the file. If you are formatting sequences yourself, see the section of FASTA format below for unaligned sequences, and the section on Stockholm format for alignments; these are the recommended native formats for HMMER.

Unaligned sequence formats

FASTA	Pearson FASTA format; BLAST databases
SWISS-PROT	SWISS-PROT protein sequence database
PIR	PIR protein sequence database
EMBL	EMBL DNA sequence database
GenBank	GenBank DNA database flat files
GCGdata	Wisconsin GCG sequence database format
GCG	Wisconsin GCG single sequence format

Multiple sequence alignment formats

Stockholm	The preferred format for HMMER/Pfam (see below)
SELEX	The old format for HMMER/Pfam
GCG MSF	GCG alignment format
CLUSTAL	CLUSTALW and CLUSTALV format
PHYLIP	Phylip phylogenetic software format

For programs that do sequential, database-style access (i.e. where you'd usually use an unaligned flat file), the alignment formats are read as if they were multiple unaligned sequences.

There is no provision for enforcing that single unaligned sequence formats (i.e. GCG unaligned sequence format) are single sequence only. HMMER will happily try to read more than one sequence if your file contains more than one. However, this may not give the results you expected.

Staden "experiment file" format is parsed using the EMBL file parser, but this functionality is relatively unsupported. There is one wrinkle in this. Staden experiment files use '-' characters to indicate 'N' – i.e., that a base is present in a sequence read, but its identity is unknown. Therefore, the software replaces any '-' in an EMBL sequence with an 'N'. Sometimes people use the unaligned formats to distribute aligned sequences by including gap characters. If EMBL files are used i this way for aligned strings, they must use a different character than '-' to indicate gaps.

FASTA unaligned sequence format

An example of a simple FASTA file:

```
>seq1 This is the description of my first sequence.
AGTACGTAGTAGCTGCTGCTACGTGCGCTAGCTAGTACGTCA
CGACGTAGATGCTAGCTGACTCGATGC
>seq2 This is a description of my second sequence.
CGATCGATCGTACGTCGACTGATCGTAGCTACGTCGTACGTAG
CATCGTCAGTTACTGCATGCTCG
```

FASTA is probably the simplest of formats for unaligned sequences. FASTA files are easily created in a text editor. Each sequence is preceded by a line starting with >. The first word on this line is the name of the sequence. The rest of the line is a description of the sequence (free format). The remaining lines contain the sequence itself. You can put as many letters on a sequence line as you want.

Blank lines in a FASTA file are ignored, and so are spaces or other gap symbols (dashes, underscores, periods) in a sequence. Any other non-amino or non-nucleic acid symbols in the sequence should produce an appropriately strident string of warnings on yur terminal screen when you try to use the file.

Stockholm, the recommended multiple sequence alignment format

While we recommend a community standard format (FASTA) for unaligned sequence files, the recommended multiple alignment file format is not a community standard. The Pfam Consortium developed a format (based on extended SELEX) called “Stockholm format”. The reasons for this are two-fold. First, there really is no standard accepted format for multiple sequence alignment files, so we don’t feel guilty about inventing a new one. Second, the formats of popular multiple alignment software (e.g. CLUSTAL, GCG MSF, PHYLIP) do not support rich documentation and markup of the alignment. Stockholm format was developed to support extensible markup of multiple sequence alignments, and we use this capability extensively in both RNA work (with structural markup) and the Pfam database (with extensive use of both annotation and markup).

a minimal Stockholm file

```
# STOCKHOLM 1.0

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

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

The simplest Stockholm file is pretty intuitive, easily generated in a text editor. It is usually easy to convert alignment formats into a “least common denominator” Stockholm format. For instance, GCG’s MSF format and the output of the CLUSTAL multiple alignment programs are similar interleaved formats.

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 or period 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 to the file 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.

Count vector files

hmmbuild saves a "count vector file" when the **--cfile** option is invoked. The count vector file contains observed weighted counts for match emissions, insert emissions, and state transitions. The intended use of the count vector file is for training mixture Dirichlet priors (something we do locally), but since it may be useful for other purposes, the format is documented here.

Each line of the file is an annotated count vector. The format of this line is:

Vector type The first field is a single letter M, I, or T, specifying whether the line is for a match emission, insert emission, or transition vector.

Counts The next several fields are the counts themselves.

For match and insert emission vectors, there are either 4 or 20 real numbers, depending on whether the alignment was DNA/RNA or protein, respectively; and the counts are in alphabetical order by residue ("ACGT" for DNA/RNA, "AC..WY" for protein).

For state transition vectors, there are 7 real numbers, in order MM, MI, MD, IM, II, DM, DD.

The counts are real numbers, not integers, because they're weighted. Both relative weights (default: tree weights) and absolute weights (effective sequence number) affect the observed counts. (They are exactly the numbers used by HMMER before addition of Dirichlet prior pseudocounts and renormalization.) Options affecting either relative or absolute weighting will therefore affect the count vectors. To see unweighted raw count vectors, add **--wnone --idlevel 1.0** to the **hmmbuild** command line.

Alignment name If the alignment was in Stockholm format (for example, a Pfam distribution), the name of the alignment is recorded in this field. (A count vector file may contain vectors from a large number of alignments.) If this information is unavailable, "-" appears instead.

Alignment column An integer indicating which alignment column this count vector corresponds to. In combination with the alignment name, this allows you to retrieve other annotation from the vicinity of this count vector in the original alignment file if you need to.

HMM node An integer indicating which HMM node this vector corresponds to. This number is always $\leq$ the alignment column index, because some alignment columns are skipped in constructing an HMM.

Note that count vectors are only recorded for the alignment columns that became HMM match states. If you want to collect a count vector from *every* alignment column, you need to make **hmmbuild** assign every column to a match state. You can do this by invoking the **--fast --gapmax 1.0** options to **hmmbuild**.

Structure annotation The next 2 fields are single characters representing structural annotation for this column in the alignment, if available. If not, “-” characters appear.

The first field is Stockholm format’s `#=GC SS_cons` secondary structure consensus annotation for the column. This is a single letter DSSP code.

The second field is Stockholm format’s `#=GC SA_cons` surface accessibility annotation for the column. This is a single character 0-9 (representing $<$10% to $<$100% accessibility), or X for unknown, or - or . for columns that are gaps in all the known structures in the alignment.

More structure annotation Transition vectors (only) have another two fields of structure annotation for the *next* (k+1’th) HMM node. Transition vectors are thought of as being between the k’th and the k+1’th HMM node, so most applications would probably want to see the annotation for both. Note that this does not necessarily correspond to the annotation on the next alignment column, because that column is not necessarily assigned to an HMM node.

7 Manual pages

`HMMER` - profile hidden Markov model software

Synopsis

hmmalign Align multiple sequences to a profile HMM.

hmmbuild Build a profile HMM from a given multiple sequence alignment.

hmmcalibrate Determine appropriate statistical significance parameters for a profile HMM prior to doing database searches.

hmmconvert Convert HMMER profile HMMs to other formats, such as GCG profiles.

hmmemit Generate sequences probabilistically from a profile HMM.

hmmfetch Retrieve an HMM from an HMM database

hmmindex Create a binary SSI index for an HMM database

hmmpfam Search a profile HMM database with a sequence (i.e., annotate various kinds of domains in the query sequence).

hmmsearch Search a sequence database with a profile HMM (i.e., find additional homologues of a modeled family).

Description

These programs use profile hidden Markov models (profile HMMs) to model the primary structure consensus of a family of protein or nucleic acid sequences.

Options

All **HMMER** programs give a brief summary of their command-line syntax and options if invoked without any arguments. When invoked with the single argument, **-h** (i.e., help), a program will report more verbose command-line usage information, including rarely used, experimental, and expert options. **-h** will report version numbers which are useful if you need to report a bug or problem to me.

Each **HMMER** program has its own man page briefly summarizing command line usage. There is also a user's guide that came with the software distribution, which includes a tutorial introduction and more detailed descriptions of the programs. See http://hmmer.wustl.edu/ for on-line documentation and the current HMMER release.

In general, no command line options should be needed by beginning users. The defaults are set up for optimum performance in most situations. Options that are single lowercase letters (e.g. **-a**) are ”common” options that are expected to be frequently used and will be important in many applications. Options that are single uppercase letters (e.g. **-B**) are usually less common options, but also may be important in some applications. Options that are full words (e.g. **--verbose**) are either rarely used, experimental, or expert options. Some experimental options are only there for my own ongoing experiments with HMMER, and may not be supported or documented adequately.

Sequence File Formats

In general, **HMMER** attempts to read most common biological sequence file formats. It autodetects the format of the file. It also autodetects whether the sequences are protein or nucleic acid. Standard IUPAC degeneracy codes are allowed in addition to the usual 4-letter or 20-letter codes.

Unaligned sequences Unaligned sequence files may be in FASTA, Swissprot, EMBL, GenBank, PIR, Intelligenetics, Strider, or GCG format. These formats are documented in the User's Guide.

Sequence alignments Multiple sequence alignments may be in CLUSTALW, SELEX, or GCG MSF format. These formats are documented in the User's Guide.

Environment Variables

For ease of using large stable sequence and HMM databases, **HMMER** looks for sequence files and HMM files in the current working directory as well as in system directories specified by environment variables.

BLASTDB Specifies the directory location of sequence databases. Example: **/seqlibs/blastdb/.** In installations that use BLAST software, this environment variable is likely to already be set.

HMMERDB Specifies the directory location of HMM databases. Example: **/seqlibs/pfam/.**

`hmmalign` - align sequences to an HMM profile

Synopsis

hmmalign *[options] hmmfile seqfile*

Description

hmmalign reads an HMM file from *hmmfile* and a set of sequences from *seqfile,* aligns the sequences to the profile HMM, and outputs a multiple sequence alignment.

seqfile may be in any unaligned or aligned file format accepted by HMMER. If it is in a multiple alignment format (e.g. Stockholm, MSF, SELEX, ClustalW), the existing alignment is ignored (i.e., the sequences are read as if they were unaligned - hmmalign will align them the way it wants).

Options

-h Print brief help; includes version number and summary of all options, including expert options.

-m Include in the alignment only those symbols aligned to match states. Do not show symbols assigned to insert states.

-o <*f*> Save alignment to file <*f*> instead of to standard output.

-q quiet; suppress all output except the alignment itself. Useful for piping or redirecting the output.

Expert Options

--informat <*s*> Assert that the input *seqfile* is in format <*s*>*;* do not run Babelfish format autodection. This increases the reliability of the program somewhat, because the Babelfish can make mistakes; particularly recommended for unattended, high-throughput runs of HMMER. Valid format strings include FASTA, GENBANK, EMBL, GCG, PIR, STOCKHOLM, SELEX, MSF, CLUSTAL, and PHYLIP. See the User's Guide for a complete list.

--mapali <*f*> Reads an alignment from file <*f*> and aligns it as a single object to the HMM; e.g. the alignment in <*f*> is held fixed. This allows you to align sequences to a model with **hmmalign** and view them in the context of an existing trusted multiple alignment. The alignment to the alignment is defined by a ”map” kept in the HMM, and so is fast and guaranteed to be consistent with the way the HMM was constructed from the alignment. The alignment in the file <*f*> must be exactly the alignment that the HMM was built from. Compare the **--withali** option.

--withali <*f*> Reads an alignment from file <*f*> and aligns it as a single object to the HMM; e.g. the alignment in <*f*> is held fixed. This allows you to align sequences to a model with **hmmalign** and view them in the context of an existing trusted multiple alignment. The alignment to the alignment is done with a heuristic (nonoptimal) dynamic programming procedure, which may be somewhat slow and is not guaranteed to be completely consistent with the way the HMM was constructed (though it should be quite close). However, any alignment can be used, not just the alignment that the HMM was built from. Compare the **--mapali** option.

`hmmbuild` - build a profile HMM from an alignment

Synopsis

hmmbuild *[options] hmmfile alignfile*

Description

hmmbuild reads a multiple sequence alignment file *alignfile* , builds a new profile HMM, and saves the HMM in *hmmfile.*

alignfile may be in ClustalW, GCG MSF, SELEX, Stockholm, or aligned FASTA alignment format. The format is automatically detected.

By default, the model is configured to find one or more nonoverlapping alignments to the complete model: multiple global alignments with respect to the model, and local with respect to the sequence. This is analogous to the behavior of the **hmmls** program of HMMER 1. To configure the model for multiple *local* alignments with respect to the model and local with respect to the sequence, a la the old program **hmmfs,** use the **-f** (fragment) option. More rarely, you may want to configure the model for a single global alignment (global with respect to both model and sequence), using the **-g** option; or to configure the model for a single local/local alignment (a la standard Smith/Waterman, or the old **hmmsw** program), use the **-s** option.

Options

-f Configure the model for finding multiple domains per sequence, where each domain can be a local (fragmentary) alignment. This is analogous to the old **hmmfs** program of HMMER 1.

-g Configure the model for finding a single global alignment to a target sequence, analogous to the old **hmms** program of HMMER 1.

-h Print brief help; includes version number and summary of all options, including expert options.

-n *<s>* Name this HMM *<s>*. *<s>* can be any string of non-whitespace characters (e.g. one ”word”). There is no length limit (at least not one imposed by HMMER; your shell will complain about command line lengths first).

-o *<f>* Re-save the starting alignment to *<f>*, in Stockholm format. The columns which were assigned to match states will be marked with x’s in an #=RF annotation line. If either the **--hand** or **--fast** construction options were chosen, the alignment may have been slightly altered to be compatible with Plan 7 transitions, so saving the final alignment and comparing to the starting alignment can let you view these alterations. See the User’s Guide for more information on this arcane side effect.

-s Configure the model for finding a single local alignment per target sequence. This is analogous to the standard Smith/Waterman algorithm or the **hmmsw** program of HMMER 1.

-A Append this model to an existing *hmmfile* rather than creating *hmmfile.* Useful for building HMM libraries (like Pfam).

-F Force overwriting of an existing *hmmfile.* Otherwise HMMER will refuse to clobber your existing HMM files, for safety's sake.

Expert Options

--amino Force the sequence alignment to be interpreted as amino acid sequences. Normally HMMER autodetects whether the alignment is protein or DNA, but sometimes alignments are so small that autodetection is ambiguous. See **--nucleic.**

--archpri <*x*> Set the ”architecture prior” used by MAP architecture construction to <*x*>, where <*x*> is a probability between 0 and 1. This parameter governs a geometric prior distribution over model lengths. As <*x*> increases, longer models are favored a priori. As <*x*> decreases, it takes more residue conservation in a column to make a column a ”consensus” match column in the model architecture. The 0.85 default has been chosen empirically as a reasonable setting.

--binary Write the HMM to *hmmfile* in HMMER binary format instead of readable ASCII text.

--cfile <*f*> Save the observed emission and transition counts to <*f*> after the architecture has been determined (e.g. after residues/gaps have been assigned to match, delete, and insert states). This option is used in HMMER development for generating data files useful for training new Dirichlet priors. The format of count files is documented in the User's Guide.

--fast Quickly and heuristically determine the architecture of the model by assigning all columns will more than a certain fraction of gap characters to insert states. By default this fraction is 0.5, and it can be changed using the **--gapmax** option. The default construction algorithm is a maximum a posteriori (MAP) algorithm, which is slower.

--gapmax <*x*> Controls the --*fast* model construction algorithm, but if --*fast* is not being used, has no effect. If a column has more than a fraction <*x*> of gap symbols in it, it gets assigned to an insert column. <*x*> is a frequency from 0 to 1, and by default is set to 0.5. Higher values of <*x*> mean more columns get assigned to consensus, and models get longer; smaller values of <*x*> mean fewer columns get assigned to consensus, and models get smaller. <*x*>

--hand Specify the architecture of the model by hand: the alignment file must be in SELEX or Stockholm format, and the reference annotation line (#=RF in SELEX, #=GC RF in Stockholm) is used to specify the architecture. Any column marked with a non-gap symbol (such as an 'x', for instance) is assigned as a consensus (match) column in the model.

--idlevel <*x*> Controls both the determination of effective sequence number and the behavior of the --*wblosum* weighting option. The sequence alignment is clustered by percent identity, and the number of clusters at a cutoff threshold of <*x*> is used to determine the effective sequence number. Higher values of <*x*> give more clusters and higher effective sequence numbers; lower values of <*x*> give fewer clusters and lower effective sequence numbers. <*x*> is a fraction from 0 to 1, and by default is set to 0.62 (corresponding to the clustering level used in constructing the BLOSUM62 substitution matrix).

--informat <*s*> Assert that the input *seqfile* is in format <*s*>*;* do not run Babelfish format autodection. This increases the reliability of the program somewhat, because the Babelfish can make mistakes; particularly recommended for unattended, high-throughput runs of HMMER. Valid format strings include FASTA, GENBANK, EMBL, GCG, PIR, STOCKHOLM, SELEX, MSF, CLUSTAL, and PHYLIP. See the User's Guide for a complete list.

--noeff Turn off the effective sequence number calculation, and use the true number of sequences instead. This will usually reduce the sensitivity of the final model (so don't do it without good reason!)

--nucleic Force the alignment to be interpreted as nucleic acid sequence, either RNA or DNA. Normally HMMER autodetects whether the alignment is protein or DNA, but sometimes alignments are so small that autodetection is ambiguous. See **--amino.**

--null <*f*> Read a null model from <*f*>. The default for protein is to use average amino acid frequencies from Swissprot 34 and p1 = 350/351; for nucleic acid, the default is to use 0.25 for each base and p1 = 1000/1001. For documentation of the format of the null model file and further explanation of how the null model is used, see the User's Guide.

--pam <*f*> Apply a heuristic PAM- (substitution matrix-) based prior on match emission probabilities instead of the default mixture Dirichlet. The substitution matrix is read from <*f*>. See **--pamwgt.** The default Dirichlet state transition prior and insert emission prior are unaffected. Therefore in principle you could combine **--prior** with **--pam** but this isn't recommended, as it hasn't been tested. (**--pam** itself hasn't been tested much!)

--pamwgt <*x*> Controls the weight on a PAM-based prior. Only has effect if **--pam** option is also in use. <*x*> is a positive real number, 20.0 by default. <*x*> is the number of "pseudocounts" contriubuted by the heuristic prior. Very high values of <*x*> can force a scoring system that is entirely driven by the substitution matrix, making HMMER somewhat approximate Gribskov profiles.

--pbswitch <*n*> For alignments with a very large number of sequences, the GSC, BLOSUM, and Voronoi weighting schemes are slow; they're O(Nˆ2) for N sequences. Henikoff position-based weights (PB weights) are more efficient. At or above a certain threshold sequence number <*n*> **hmmbuild** will switch from GSC, BLOSUM, or Voronoi weights to PB weights. To disable this switching behavior (at the cost of compute time, set <*n*> to be something larger than the number of sequences in your alignment. <*n*> is a positive integer; the default is 1000.

--prior <*f*> Read a Dirichlet prior from <*f*>, replacing the default mixture Dirichlet. The format of prior files is documented in the User's Guide, and an example is given in the Demos directory of the HMMER distribution.

--swentry <*x*> Controls the total probability that is distributed to local entries into the model, versus starting at the beginning of the model as in a global alignment. <*x*> is a probability from 0 to 1, and by default is set to 0.5. Higher values of <*x*> mean that hits that are fragments on their left (N or 5'-terminal) side will be penalized less, but complete global alignments will be penalized more. Lower values of <*x*> mean that fragments on the left will be penalized more, and global alignments on this side will be favored. This option only affects the configurations that allow local alignments, e.g. **-s** and **-f;** unless one of these options is also activated, this option has no effect. You have independent control over local/global alignment behavior for the N/C (5'/3') termini of your target sequences using **--swentry** and **--swexit.**

--swexit <*x*> Controls the total probability that is distributed to local exits from the model, versus ending an alignment at the end of the model as in a global alignment. <*x*> is a probability from 0 to 1, and by default is set to 0.5. Higher values of <*x*> mean that hits that are fragments on their right (C or 3'-terminal) side will be penalized less, but complete global alignments will be penalized more. Lower values of <*x*> mean that fragments on the right will be penalized more, and global alignments on this side will be favored. This option only affects the configurations that allow local alignments, e.g. **-s** and **-f;** unless one of these options is also activated, this option has no effect. You have independent control over local/global alignment behavior for the N/C (5'/3') termini of your target sequences using **--swentry** and **--swexit.**

--verbose Print more possibly useful stuff, such as the individual scores for each sequence in the alignment.

--wblosum Use the BLOSUM filtering algorithm to weight the sequences, instead of the default. Cluster the sequences at a given percentage identity (see **--idlevel);** assign each cluster a total weight of 1.0, distributed equally amongst the members of that cluster.

--wgsc Use the Gerstein/Sonnhammer/Chothia ad hoc sequence weighting algorithm. This is already the default, so this option has no effect (unless it follows another option in the --w family, in which case it overrides it).

--wme Use the Krogh/Mitchison maximum entropy algorithm to ”weight” the sequences. This supercedes the Eddy/Mitchison/Durbin maximum discrimination algorithm, which gives almost identical weights but is less robust. ME weighting seems to give a marginal increase in sensitivity over the default GSC weights, but takes a fair amount of time.

--wnone Turn off all sequence weighting.

--wpb Use the Henikoff position-based weighting scheme.

--wvoronoi Use the Sibbald/Argos Voronoi sequence weighting algorithm in place of the default GSC weighting.

`hmmcalibrate` - calibrate HMM search statistics

Synopsis

hmmcalibrate *[options] hmmfile*

Description

hmmcalibrate reads an HMM file from *hmmfile*, scores a large number of synthesized random sequences with it, fits an extreme value distribution (EVD) to the histogram of those scores, and re-saves *hmmfile* now including the EVD parameters.

hmmcalibrate may take several minutes (or longer) to run. While it is running, a temporary file called *hmmfile.xxx* is generated in your working directory. If you abort **hmmcalibrate** prematurely (ctrl-C, for instance), your original *hmmfile* will be untouched, and you should delete the *hmmfile.xxx* temporary file.

Options

-h Print brief help; includes version number and summary of all options, including expert options.

Expert Options

--cpu <*n*> Sets the maximum number of CPUs that the program will run on. The default is to use all CPUs in the machine. Overrides the HMMER_NCPU environment variable. Only affects threaded versions of HMMER (the default on most systems).

--fixed <*n*> Fix the length of the random sequences to <*n*>, where <*n*> is a positive (and reasonably sized) integer. The default is instead to generate sequences with a variety of different lengths, controlled by a Gaussian (normal) distribution.

--histfile <*f*> Save a histogram of the scores and the fitted theoretical curve to file <*f*>.

--mean <*x*> Set the mean length of the synthetic sequences to <*x*>, where <*x*> is a positive real number. The default is 350.

--num <*n*> Set the number of synthetic sequences to <*n*>, where <*n*> is a positive integer. If <*n*> is less than about 1000, the fit to the EVD may fail. Higher numbers of <*n*> will give better determined EVD parameters. The default is 5000; it was empirically chosen as a tradeoff between accuracy and computation time.

--pvm Run on a Parallel Virtual Machine (PVM). The PVM must already be running. The client program **hmmcalibrate-pvm** must be installed on all the PVM nodes. Optional PVM support must have been compiled into HMMER.

--sd <*x*> Set the standard deviation of the synthetic sequence length distribution to <*x*>, where <*x*> is a positive real number. The default is 350. Note that the Gaussian is left-truncated so that no sequences have lengths <= 0.

--seed <*n*> Set the random seed to <*n*>, where <*n*> is a positive integer. The default is to use **time()** to generate a different seed for each run, which means that two different runs of **hmmcalibrate** on the same HMM will give slightly different results. You can use this option to generate reproducible results for different **hmmcalibrate** runs on the same HMM.

`hmmconvert` - convert between profile HMM file formats

Synopsis

hmmconvert *[options] oldhmmfile newhmmfile*

Description

hmmconvert reads an HMM file from *oldhmmfile* in any HMMER format, and writes it to a new file *newhmmfile* in a new format. *oldhmmfile* and *newhmmfile* must be different files; you can't reliably overwrite the old file. By default, the new HMM file is written in HMMER 2 ASCII format. Available formats are HMMER 2 ASCII (default), HMMER 2 binary *(-b)* GCG profile *(-p)* , and Compugen XSW extended profile *(-P)*.

Options

-a Convert to HMMER 2 ASCII file. This is the default, so this option is unnecessary.

-b Convert to HMMER 2 binary file.

-h Print brief help; includes version number and summary of all options, including expert options.

-p Convert to GCG profile .prf format.

-A Append mode; append to *newhmmfile* rather than creating a new file.

-F Force; if *newhmmfile* already exists, and *-A* is not being used to append to the file, hmmconvert will refuse to clobber the existing file unless *-F* is used.

-P Convert the HMM to Compugen XSW extended profile format, which is similar to GCG profile format but has two extra columns for delete-open and delete-extend costs. (I do not believe that Compugen publicly supports this format; it may be undocumented.)

`hmmemit` - generate sequences from a profile HMM

Synopsis

hmmemit *[options] hmmfile*

Description

hmmemit reads an HMM file from *hmmfile* containing one or more HMMs, and generates a number of sequences from each HMM; or, if the **-c** option is selected, generate a single majority-rule consensus. This can be useful for various applications in which one needs a simulation of sequences consistent with a sequence family consensus. By default, **hmmemit** generates 10 sequences and outputs them in FASTA (unaligned) format.

Options

-a Write the generated sequences in an aligned format (SELEX) rather than FASTA.

-c Predict a single majority-rule consensus sequence instead of sampling sequences from the HMM's probability distribution. Highly conserved residues (p >= 0.9 for DNA, p >= 0.5 for protein) are shown in upper case; others are shown in lower case. Some insert states may become part of the majority rule consensus, because they are used in >= 50% of generated sequences; when this happens, insert-generated residues are simply shown as ”x”.

-h Print brief help; includes version number and summary of all options, including expert options.

-n <*n*> Generate <*n*> sequences. Default is 10.

-o <*f*> Save the synthetic sequences to file <*f*> rather than writing them to stdout.

-q Quiet; suppress all output except for the sequences themselves. Useful for piping or directing the output.

Expert Options

--seed <*n*> Set the random seed to <*n*>, where <*n*> is a positive integer. The default is to use **time()** to generate a different seed for each run, which means that two different runs of **hmmemit** on the same HMM will give slightly different results. You can use this option to generate reproducible results.

`hmmfetch` - retrieve an HMM from an HMM database

Synopsis

hmmfetch *[options] database name*

Description

hmmfetch is a small utility that retrieves an HMM called *name* from a HMMER model database called *database.* in a new format, and prints that model to standard output. For example, *hmmfetch* Pfam rrm retrieves the RRM (RNA recognition motif) model from Pfam, if the environment variable HMMERDB is set to the location of the Pfam database. The retrieved HMM file is written in HMMER 2 ASCII format.

The database must have an associated GSI index file. To index an HMM database, use the program **hmmindex.**

Options

-h Print brief help; includes version number and summary of all options, including expert options.

-n Interpret *name* as an HMM number instead of a name. Numbering starts at 0. For example, to fetch the first HMM from an HMM database called **foo,** you would do **hmmfetch -n 0 foo.**

`hmmindex` - create a binary SSI index for an HMM database

Synopsis

hmmindex *[options] database*

Description

hmmindex is a utility that creates a binary SSI (”squid sequence index” format) index for an HMM database file called *database.* The new index file is named *database.ssi.* An SSI index file is required for **hmmfetch** to work, and also for the PVM implementation of **hmmpfam.**

Options

-h Print brief help; includes version number and summary of all options, including expert options.

`hmmpfam` - search one or more sequences against an HMM database

Synopsis

hmmpfam *[options] hmmfile seqfile*

Description

hmmpfam reads a sequence file *seqfile* and compares each sequence in it, one at a time, against all the HMMs in *hmmfile* looking for significantly similar sequence matches.

hmmfile will be looked for first in the current working directory, then in a directory named by the environment variable *HMMERDB.* This lets administrators install HMM library(s) such as Pfam in a common location.

There is a separate output report for each sequence in *seqfile.* This report consists of three sections: a ranked list of the best scoring HMMs, a list of the best scoring domains in order of their occurrence in the sequence, and alignments for all the best scoring domains. A sequence score may be higher than a domain score for the same sequence if there is more than one domain in the sequence; the sequence score takes into account all the domains. All sequences scoring above the *-E* and *-T* cutoffs are shown in the first list, then *every* domain found in this list is shown in the second list of domain hits. If desired, E-value and bit score thresholds may also be applied to the domain list using the *--domE* and *--domT* options.

Options

-h Print brief help; includes version number and summary of all options, including expert options.

-n Specify that models and sequence are nucleic acid, not protein. Other HMMER programs autodetect this; but because of the order in which **hmmpfam** accesses data, it can't reliably determine the correct "alphabet" by itself.

-A <*n*> Limits the alignment output to the <*n*> best scoring domains. **-A0** shuts off the alignment output and can be used to reduce the size of output files.

-E <*x*> Set the E-value cutoff for the per-sequence ranked hit list to <*x*>, where <*x*> is a positive real number. The default is 10.0. Hits with E-values better than (less than) this threshold will be shown.

-T <*x*> Set the bit score cutoff for the per-sequence ranked hit list to <*x*>, where <*x*> is a real number. The default is negative infinity; by default, the threshold is controlled by E-value and not by bit score. Hits with bit scores better than (greater than) this threshold will be shown.

-Z <*n*> Calculate the E-value scores as if we had seen a sequence database of <*n*> sequences. The default is arbitrarily set to 59021, the size of Swissprot 34.

Expert Options

--acc Report HMM accessions instead of names in the output reports. Useful for high-throughput annotation, where the data are being parsed for storage in a relational database.

--compat Use the output format of HMMER 2.1.1, the 1998-2001 public release; provided so 2.1.1 parsers don't have to be rewritten.

--cpu <*n*> Sets the maximum number of CPUs that the program will run on. The default is to use all CPUs in the machine. Overrides the HMMER_NCPU environment variable. Only affects threaded versions of HMMER (the default on most systems).

--cut_ga Use Pfam GA (gathering threshold) score cutoffs. Equivalent to --globT <GA1> --domT <GA2>, but the GA1 and GA2 cutoffs are read from each HMM in *hmmfile* individually. hmmbuild puts these cutoffs there if the alignment file was annotated in a Pfam-friendly alignment format (extended SELEX or Stockholm format) and the optional GA annotation line was present. If these cutoffs are not set in the HMM file, **--cut_ga** doesn't work.

--cut_tc Use Pfam TC (trusted cutoff) score cutoffs. Equivalent to --globT <TC1> --domT <TC2>, but the TC1 and TC2 cutoffs are read from each HMM in *hmmfile* individually. hmmbuild puts these cutoffs there if the alignment file was annotated in a Pfam-friendly alignment format (extended SELEX or Stockholm format) and the optional TC annotation line was present. If these cutoffs are not set in the HMM file, **--cut_tc** doesn't work.

--cut_nc Use Pfam NC (noise cutoff) score cutoffs. Equivalent to --globT <NC1> --domT <NC2>, but the NC1 and NC2 cutoffs are read from each HMM in *hmmfile* individually. hmmbuild puts these cutoffs there if the alignment file was annotated in a Pfam-friendly alignment format (extended SELEX or Stockholm format) and the optional NC annotation line was present. If these cutoffs are not set in the HMM file, **--cut_nc** doesn't work.

--domE <*x*> Set the E-value cutoff for the per-domain ranked hit list to <*x*>, where <*x*> is a positive real number. The default is infinity; by default, all domains in the sequences that passed the first threshold will be reported in the second list, so that the number of domains reported in the per-sequence list is consistent with the number that appear in the per-domain list.

--domT <*x*> Set the bit score cutoff for the per-domain ranked hit list to <*x*>, where <*x*> is a real number. The default is negative infinity; by default, all domains in the sequences that passed the first threshold will be reported in the second list, so that the number of domains reported in the per-sequence list is consistent with the number that appear in the per-domain list. *Important* note: only one domain in a sequence is absolutely controlled by this parameter, or by **--domT.** The second and subsequent domains in a sequence have a de facto bit score threshold of 0 because of the details of how HMMER works. HMMER requires at least one pass through the main model per sequence; to do more than one pass (more than one domain) the multidomain alignment must have a better score than the single domain alignment, and hence the extra domains must contribute positive score. See the Users' Guide for more detail.

--forward Use the Forward algorithm instead of the Viterbi algorithm to determine the per-sequence scores. Per-domain scores are still determined by the Viterbi algorithm. Some have argued that Forward is a more sensitive algorithm for detecting remote sequence homologues; my experiments with HMMER have not confirmed this, however.

--informat <*s*> Assert that the input *seqfile* is in format <*s*>*;* do not run Babelfish format autodection. This increases the reliability of the program somewhat, because the Babelfish can make mistakes; particularly recommended for unattended, high-throughput runs of HMMER. Valid format strings include FASTA, GENBANK, EMBL, GCG, PIR, STOCKHOLM, SELEX, MSF, CLUSTAL, and PHYLIP. See the User's Guide for a complete list.

--null2 Turn off the post hoc second null model. By default, each alignment is rescored by a postprocessing step that takes into account possible biased composition in either the HMM or the target sequence. This is almost essential in database searches, especially with local alignment models. There is a very small chance that this postprocessing might remove real matches, and in these cases **--null2** may improve sensitivity at the expense of reducing specificity by letting biased composition hits through.

--pvm Run on a Parallel Virtual Machine (PVM). The PVM must already be running. The client program **hmmpfam-pvm** must be installed on all the PVM nodes. The HMM database *hmmfile* and an associated GSI index file *hmmfile*.gsi must also be installed on all the PVM nodes. (The GSI index is produced by the program **hmmindex**.) Because the PVM implementation is I/O bound, it is highly recommended that each node have a local copy of *hmmfile* rather than NFS mounting a shared copy. Optional PVM support must have been compiled into HMMER for **--pvm** to function.

--xnu Turn on XNU filtering of target protein sequences. Has no effect on nucleic acid sequences. In trial experiments, **--xnu** appears to perform less well than the default post hoc null2 model.

`hmmsearch` - search a sequence database with a profile HMM

Synopsis

hmmsearch *[options] hmmfile seqfile*

Description

hmmsearch reads an HMM from *hmmfile* and searches *seqfile* for significantly similar sequence matches.

seqfile will be looked for first in the current working directory, then in a directory named by the environment variable *BLASTDB.* This lets users use existing BLAST databases, if BLAST has been configured for the site.

hmmsearch may take minutes or even hours to run, depending on the size of the sequence database. It is a good idea to redirect the output to a file.

The output consists of four sections: a ranked list of the best scoring sequences, a ranked list of the best scoring domains, alignments for all the best scoring domains, and a histogram of the scores. A sequence score may be higher than a domain score for the same sequence if there is more than one domain in the sequence; the sequence score takes into account all the domains. All sequences scoring above the *-E* and *-T* cutoffs are shown in the first list, then *every* domain found in this list is shown in the second list of domain hits. If desired, E-value and bit score thresholds may also be applied to the domain list using the *--domE* and *--domT* options.

Options

-h Print brief help; includes version number and summary of all options, including expert options.

-A <*n*> Limits the alignment output to the <*n*> best scoring domains. **-A0** shuts off the alignment output and can be used to reduce the size of output files.

-E <*x*> Set the E-value cutoff for the per-sequence ranked hit list to <*x*>, where <*x*> is a positive real number. The default is 10.0. Hits with E-values better than (less than) this threshold will be shown.

-T <*x*> Set the bit score cutoff for the per-sequence ranked hit list to <*x*>, where <*x*> is a real number. The default is negative infinity; by default, the threshold is controlled by E-value and not by bit score. Hits with bit scores better than (greater than) this threshold will be shown.

-Z <*n*> Calculate the E-value scores as if we had seen a sequence database of <*n*> sequences. The default is the number of sequences seen in your database file <*seqfile*>.

Expert Options

--compat Use the output format of HMMER 2.1.1, the 1998-2001 public release; provided so 2.1.1 parsers don't have to be rewritten.

--cpu <*n*> Sets the maximum number of CPUs that the program will run on. The default is to use all CPUs in the machine. Overrides the HMMER_NCPU environment variable. Only affects threaded versions of HMMER (the default on most systems).

--cut_ga Use Pfam GA (gathering threshold) score cutoffs. Equivalent to --globT <GA1> --domT <GA2>, but the GA1 and GA2 cutoffs are read from the HMM file. hmmbuild puts these cutoffs there if the alignment file was annotated in a Pfam-friendly alignment format (extended SELEX or Stockholm format) and the optional GA annotation line was present. If these cutoffs are not set in the HMM file, **--cut_ga** doesn't work.

--cut_tc Use Pfam TC (trusted cutoff) score cutoffs. Equivalent to --globT <TC1> --domT <TC2>, but the TC1 and TC2 cutoffs are read from the HMM file. hmmbuild puts these cutoffs there if the alignment file was annotated in a Pfam-friendly alignment format (extended SELEX or Stockholm format) and the optional TC annotation line was present. If these cutoffs are not set in the HMM file, **--cut_tc** doesn't work.

--cut_nc Use Pfam NC (noise cutoff) score cutoffs. Equivalent to --globT <NC1> --domT <NC2>, but the NC1 and NC2 cutoffs are read from the HMM file. hmmbuild puts these cutoffs there if the alignment file was annotated in a Pfam-friendly alignment format (extended SELEX or Stockholm format) and the optional NC annotation line was present. If these cutoffs are not set in the HMM file, **--cut_nc** doesn't work.

--domE <*x*> Set the E-value cutoff for the per-domain ranked hit list to <*x*>, where <*x*> is a positive real number. The default is infinity; by default, all domains in the sequences that passed the first threshold will be reported in the second list, so that the number of domains reported in the per-sequence list is consistent with the number that appear in the per-domain list.

--domT <*x*> Set the bit score cutoff for the per-domain ranked hit list to <*x*>, where <*x*> is a real number. The default is negative infinity; by default, all domains in the sequences that passed the first threshold will be reported in the second list, so that the number of domains reported in the per-sequence list is consistent with the number that appear in the per-domain list. *Important* note: only one domain in a sequence is absolutely controlled by this parameter, or by **--domT.** The second and subsequent domains in a sequence have a de facto bit score threshold of 0 because of the details of how HMMER works. HMMER requires at least one pass through the main model per sequence; to do more than one pass (more than one domain) the multidomain alignment must have a better score than the single domain alignment, and hence the extra domains must contribute positive score. See the Users' Guide for more detail.

--forward Use the Forward algorithm instead of the Viterbi algorithm to determine the per-sequence scores. Per-domain scores are still determined by the Viterbi algorithm. Some have argued that Forward is a more sensitive algorithm for detecting remote sequence homologues; my experiments with HMMER have not confirmed this, however.

--informat *<s>* Assert that the input *seqfile* is in format *<s>;* do not run Babelfish format autodection. This increases the reliability of the program somewhat, because the Babelfish can make mistakes; particularly recommended for unattended, high-throughput runs of HMMER. Valid format strings include FASTA, GENBANK, EMBL, GCG, PIR, STOCKHOLM, SELEX, MSF, CLUSTAL, and PHYLIP. See the User's Guide for a complete list.

--null2 Turn off the post hoc second null model. By default, each alignment is rescored by a postprocessing step that takes into account possible biased composition in either the HMM or the target sequence. This is almost essential in database searches, especially with local alignment models. There is a very small chance that this postprocessing might remove real matches, and in these cases **--null2** may improve sensitivity at the expense of reducing specificity by letting biased composition hits through.

--pvm Run on a Parallel Virtual Machine (PVM). The PVM must already be running. The client program **hmmsearch-pvm** must be installed on all the PVM nodes. Optional PVM support must have been compiled into HMMER.

--xnu Turn on XNU filtering of target protein sequences. Has no effect on nucleic acid sequences. In trial experiments, **--xnu** appears to perform less well than the default post hoc null2 model.

8 Index of frequently asked questions

9 License terms

If you're just going to use HMMER to analyze sequence, it's free to you, and you can skip all the legalese that follows.

If you have a specially licensed (non-GPL) copy of HMMER from Washington University, you already have your own legalese, in the form of a separate written WashU licensing contract; this chapter does not apply to you either.

If you have a public, GPL'ed copy of HMMER and you're interested in modifying or borrowing the source code or distributing the software yourself, keep reading.

The gist of the license

HMMER is a copyrighted work. It is not in the public domain. It is distributed under an "open source" license, the GNU General Public License (GPL). In brief, what the GPL means is that:

- If you're just going to use HMMER to analyze sequence, it's free.
- If you want to redistribute *unmodified* copies of HMMER, you're free to do so. The license even permits you to sell copies of HMMER, or include it (in unmodified form) as part of an otherwise proprietary software package. The license and my copyright must remain unchanged. You don't need explicit permission from me, Washington University, nor HHMI.
- If you want to *modify* HMMER and use your modified copy *internally* (in your company or department), you're free to do that to.
- If you modify HMMER or borrow any part of it, and create a new, derivative work that you plan to distribute (either freely or for profit), then the license terms of the GPL kick in. In this case, you *must*:
  - freely distribute the derivative work under the GNU GPL, retaining my copyright on the parts I wrote; or
  - if you're a free software developer but you prefer a different open source license, contact me – I'll typically ship snippets of the code under any open source license; or
  - if you're a proprietary software developer, contact Washington University to arrange for non-GPL licensing terms. For commercial licenses, we will typically ask for a fee and/or royalties.

The intent of the license

Both you and I want the software to be downloadable from the Web. You don't want to hassle with signing any contract; I don't want to spend my time arranging licenses and sending out code on request. At the same time, my sense of fair play (not to mention my technology transfer office) doesn't want you to just take my software and sell it, unless you're sharing the revenue with us and helping to support our research.

These two points would seem to be in conflict. How can code be freely available, while still gently requiring that you play fair and share revenue from any commercialized versions? Certain open source licenses, including the GPL, have a widely unappreciated property: they enable an efficient dual licensing strategy. Most copies of HMMER are freely distributed under the GPL, but the GPL essentially inhibits commercialization. Other versions are provided under proprietary licenses and successfully commercialized.

If you get the free, GPL'ed version from the Web, you're choosing to play on a basic research playing field. You don't get any user support, and you're at the mercy of one academic's whims (mine), but you do get free, open access to complete HMMER source code. You can use it, modify it, redistribute it... on the conditions that my copyright stays on my code, and that you freely distribute anything that's derived from my work under exactly the same rules that I gave my work to you – e.g. under the GPL. This "viral" propagation of the GPL into derivative works means that you're generally going to have nonproprietary and noncommercial goals if you're willing to use the GPL'ed version. Freely redistributable source code with a virally propagating free license is generally anathema to commercial purposes – while being no significant hindrance on basic research.

Alternatively, you can choose to play on another, proprietary playing field. Contact the Washington University tech transfer people and negotiate a satisfactory, nonexclusive, proprietary license, free of the GPL. Unsurprisingly, we'll then happily ask you for fair license fees for this, essentially proportional to the ratio of our work versus your work in the final product. You're supporting research, not lining my pockets. License revenue goes into research funds for Washington University, the MRC Laboratory of Molecular Biology, and my lab. To the fullest extent I can, I decline to take any licensing revenue as personal income.[**] Several commercial versions of HMMER are in use, giving users several options to choose from for proper, responsive user support – and efficiently giving me only a couple of points of necessary contact to disseminate information, rather than me having to respond personally to an entire user community.

Importantly, it doesn't particularly matter whether you are sitting in academia, industry, or a nonprofit institute. What matters is how you're planning to distribute any derivative works. Plenty of companies do basic research, publish papers, and release their published work freely, and can work quite happily with the GPL. Many academics develop and sell proprietary, closed products through their tech transfer offices, which GPL'ed source code will (quite intentionally!) block, until a separate proprietary license is negotiated with Washington University.

Finally, the license!

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

[**] A minor fraction wanders the planet and in the end, cannot be routed to lab funds; the MRC-LMB's ineffable accounting office writes me a check every year that has to be treated as personal income, and I end up paying *both* UK and US taxes on the damn thing. From this faintly pathetic bureaucratic exercise, I net around a couple of thousand dollars a year.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The “Program”, below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   (b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   (c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

   (a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

   (c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

   It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

10 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 did inadvertently escape 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 is 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 has now settled into a comfortable middle age, like its author; still actively maintained, though dramatic changes are 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 latest release, 2.3, has been assembled in spring 2003.

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

Sequence format parsing (`sqio.c`) in HMMER is derived from an early release of the `READSEQ` package by Don Gilbert, Indiana University. Thanks to Don for an excellent piece of software; and apologies for the years of mangling I've put it through since I obtained it in 1992. The file `hsregex.c` is a derivative of Henry Spencer's regular expression library; thanks, Henry. Several miscellaneous functions in `sre_math.c` are taken from public domain sources and are credited in the code's comments. `masks.c` includes a modified copy of the XNU source code from David States and Jean-Michel Claverie.

John Blanchard (Incyte Pharmaceuticals) made several contributions to the PVM port; the remaining bugs are my own dumb fault. Dave Cortesi (Silicon Graphics) contributed much useful advice on the POSIX threads implementation. The blazing fast Altivec port to Macintosh PowerPC was contributed in toto by Erik Lindahl at Stanford.

In many other 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 machines, but is tested on a variety of hardware. Compaq, IBM, Intel, Sun Microsystems, Silicon Graphics, Hewlett-Packard, and Paracel have provided the 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; Cygnus' and others' egcs compiler; Conor Cahill's dbmalloc library; Bruce Perens' ElectricFence; Armin Biere's ccmalloc; the cast of thousands that develops CVS, the Concurrent Versioning System; Larry Wall's perl; LaTeX and TeX from Leslie Lamport and Don Knuth; Nikos Drakos' latex2html; Thomas Phelps' PolyglotMan; Linus Torvalds' Linux operating system; and the folks at Red Hat Linux and Mandrake Linux.

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

Altschul, S. F. (1991). Amino acid substitution matrices from an information theoretic perspective. *J. Mol. Biol.*, 219:555–565.

Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990). Basic local alignment search tool. *J. Mol. Biol.*, 215:403–410.

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

Barrett, C., Hughey, R., and Karplus, K. (1997). Scoring hidden Markov models. *Comput. Applic. Biosci.*, 13:191–199.

Barton, G. J. (1990). Protein multiple sequence alignment and flexible pattern matching. *Meth. Enzymol.*, 183:403–427.

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

Bateman, A., Birney, E., Cerruti, L., Durbin, R., Etwiller, L., Eddy, S. R., Griffiths-Jones, S., Howe, K. L., Marshall, M., and Sonnhammer, E. L. (2002). The Pfam protein families database. *Nucl. Acids Res.*, 30:276–280.

Churchill, G. A. (1989). Stochastic models for heterogeneous DNA sequences. *Bull. Math. Biol.*, 51:79–94.

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

Eddy, S. R. (1996). Hidden Markov models. *Curr. Opin. Struct. Biol.*, 6:361–365.

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

Gerstein, M., Sonnhammer, E. L. L., and Chothia, C. (1994). Volume changes in protein evolution. *J. Mol. Biol.*, 235:1067–1078.

Goldstein, L. and Waterman, M. S. (1994). Approximations to profile score distributions. *J. Cell. Biol.*, 1:93–104.

Gribskov, M., Luthy, R., and Eisenberg, D. (1990). Profile analysis. *Meth. Enzymol.*, 183:146–159.

Gribskov, M., McLachlan, A. D., and Eisenberg, D. (1987). Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, 84:4355–4358.

Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89:10915–10919.

Henikoff, S. and Henikoff, J. G. (1994a). Position-based sequence weights. *J. Mol. Biol.*, 243:574–578.

Henikoff, S. and Henikoff, J. G. (1994b). Protein family classification based on searching a database of blocks. *Genomics*, 19:97–107.

Krogh, A. (1998). An introduction to hidden Markov models for biological sequences. In Salzberg, S., Searls, D., and Kasif, S., editors, *Computational Methods in Molecular Biology*, pages 45–63. Elsevier.

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

Krogh, A. and Mitchison, G. (1995). Maximum entropy weighting of aligned sequences of proteins or DNA. *Proc. Int. Conf. on Intelligent Systems in Molecular Biology*, 3:215–221.

Lawless, J. F. (1982). *Statistical Models and Methods for Lifetime Data*, chapter 4, pages 141–202. John Wiley & Sons.

Letunic, I., Goodstadt, L., Dickens, N. J., Doerks, T., Schultz, J., Mott, R., Ciccarelli, F., Copley, R. R., Ponting, C. P., and Bork, P. (2002). Recent improvements to the SMART domain-based sequence annotation resource. *Nucl. Acids Res.*, 30:242–244.

Mitchison, G. J. and Durbin, R. M. (1995). Tree-based maximal likelihood substitution matrices and hidden Markov models. *J. Mol. Evol.*, 41:1139–1151.

Mulder, N. J., Apweiler, R., Attwood, T. K., Bairoch, A., Barrell, D., Bateman, A., Binns, D., Biswas, M., Bradley, P., Bork, P., Bucher, P., Copley, R. R., Courcelle, E., Das, U., Durbin, R., Falquet, L., Fleischmann, W., Griffiths-Jones, S., Haft, D., Harte, N., Hulo, N., Kahn, D., Kanapin, A., Krestyaninova, M., Lopez, R., Letunic, I., Lonsdale, D., Silventoinen, V., Orchard, S. E., Pagni, M., Peyruc, D., Ponting, C. P., Selengut, J. D., Servant, F., Sigrist, C. J., Vaughan, R., and Zdobnov, E. M. (2003). The InterPro database, 2003 brings increased coverage and new features. *Nucl. Acids Res.*, 31:315–318.

Pearson, W. R. and Lipman, D. J. (1988). Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448.

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, 77:257–286.

Sibbald, P. R. and Argos, P. (1990). Weighting aligned protein or nucleic acid sequences to correct for unequal representation. *J. Mol. Biol.*, 216:813–818.

Sjölander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I. S., and Haussler, D. (1996). Dirichlet mixtures: A method for improving detection of weak but significant protein sequence homology. *Comput. Applic. Biosci.*, 12:327–345.

Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197.

Sonnhammer, E. L. L., Eddy, S. R., and Durbin, R. (1997). Pfam: A comprehensive database of protein families based on seed alignments. *Proteins*, 28:405–420.

Taylor, W. R. (1986). Identification of protein sequence homology by consensus template alignment. *J. Mol. Biol.*, 188:233–258.