HMMER User's Guide

Biological sequence analysis using profile hidden Markov models

Sean R. Eddy
and the HMMER development team

http://hmmer.org
Version 3.2; June 2018

HMMER development is supported in part by the National Human Genome Research Institute of the US National Institutes of Health under grant number R01HG009116. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Contents

Introduction

Most protein sequences are composed from a relatively small number of ancestral protein domain families. Our sampling of common protein domain families has become comprehensive and deep, while raw sequence data continues to accumulate explosively. It has become advantageous to compare sequences against all known domain families, instead of all known sequences.

This makes protein sequence analysis more like speech recognition. When you talk to your smartphone, it doesn't compare your digitized speech to everything that's ever been said. It compares what you say to a prebuilt dataset of statistical models of common words and phonemes. Using machine learning techniques, each statistical model is trained on large datasets of examples spoken by different speakers in different accents. Similarly, for each protein domain family, there are typically thousands of known homologs that can be aligned into deep multiple sequence alignments. Sequence alignments reveal a specific pattern of evolutionary conservation particular to that domain's structure and function. These patterns can be captured by probabilistic models.

HMMER is a software package that provides tools for making probabilistic models of protein and DNA sequence domain families – called **profile hidden Markov models**, **profile HMMs**, or just **profiles** – and for using these profiles to annotate new sequences, to search sequence databases for additional homologs, and to make deep multiple sequence alignments. HMMER underlies several comprehensive collections of alignments and profiles of known protein and DNA sequence domain families, including the Pfam database.[1]

[1] pfam.org

How to avoid reading this manual

I hate reading documentation. If you're like me, you're thinking, 221 pages of documentation, you've got to be joking! First I want to know that the software compiles, runs, and gives useful results, before I'm going to plow through some 221 tedious pages of someone's documentation. For fellow cynics who have seen one too many software

packages that don't work:

- Follow the quick installation instructions on page 17. An automated test suite is included, so you will know immediately if something went wrong.[2]
- Go to the tutorial section on page 27, which walks you through examples of using HMMER.

  Everything else, you can come back and read later.

[2] Nothing should go wrong.

Background and brief history

A multiple sequence alignment of a homologous family of protein domains reveals patterns of site-specific evolutionary conservation. Key residues may be highly conserved at certain positions; some positions may tolerate certain substitutions while conserving physiochemical properties like hydrophobicity, charge, or size; some positions may be evolutionarily near-neutral and variable; insertions and deletions are tolerated at some positions better than others. A **profile** is a position-specific scoring model that describes which symbols are likely to be observed and how frequently insertions/deletions occur at each position (column) of a multiple sequence alignment.

Pairwise sequence alignment methods such as BLAST and FASTA use position-*independent* subsitution score matrices such as BLOSUM and PAM, but the desirability of position-*specific* models was recognized even before BLAST and FASTA were written.[3] Several groups introduced different position-specific alignment scoring approaches in the 1980's. The name "profiles", introduced by Gribskov and colleagues,[4] was a name that stuck.

[3] R. F. Doolittle. Similar amino acid sequences: Chance or common ancestry? *Science*, 214:149–159, 1981

[4] M. Gribskov, et al. Profile analysis: Detection of distantly related proteins. *Proc. Natl. Acad. Sci. USA*, 84:4355–4358, 1987

Profiles have a lot of parameters. The BLOSUM and PAM amino acid substitution matrices have only 210 free parameters (20×20, symmetric), and those parameters are averages over large collections of many different known sequence alignments. A profile typically has at least 22 parameters for each of the $\sim$200 consensus positions or so in a typical protein domain, and these thousands of parameters are estimated for one particular sequence family alignment, not averaged across all of them. Early profile methods were vexed by a lack of theoretical underpinnings for how to parameterize these models effectively, especially for insertion and deletions.

There's $\sim$22 parameters per position because there's 20 residue scores, plus gap-open and gap-extend penalties for starting or extending an insertion or deletion.

In the 1990's, Anders Krogh, David Haussler, and co-workers at UC Santa Cruz recognized a parallel between profiles and widely used speech recognition techniques, and they introduced **profile hidden Markov models (profile HMMs)**.[5] HMMs had been used in biology before, but the Krogh paper had dramatic impact because

[5] A. Krogh, et al. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.*, 235:1501–1531, 1994

HMM technology was so well suited to addressing the vexing parameterization problem. HMMs have a formal probabilistic basis, allowing the use of probability theory to set and to interpret the large number of free parameters in a profile, including the position-specific gap and insertion parameters. The methods are mathematically consistent and therefore automatable, which was crucial in allowing people to make libraries of hundreds of profile HMMs and apply them on a large scale to whole genome analysis. One such database of protein domain models is Pfam.[6] Historically, Pfam and HMMER have been developed in parallel.

The first implementations of profile HMMs were computationally intensive, including HMMER1 (1995) and HMMER2 (1998), but HMMER3 is now typically faster than BLASTP or FASTA searches even though it uses more complex models.

[6] E. L. L. Sonnhammer, et al. Pfam: A comprehensive database of protein families based on seed alignments. *Proteins*, 28:405–420, 1997

For DNA searches, BLASTN remains about two orders of magnitude faster than HMMER DNA searches, but is less sensitive.

Problems HMMER is designed for

Sensitive homology searches. You're working on a specific sequence family, and you've carefully constructed a representative multiple sequence alignment. The HMMER `hmmbuild` program lets you build a profile from your alignment, and the `hmmsearch` program lets you search your profile against a sequence database looking systematically for more homologs.

... even for single sequence queries. HMMER3 also works for single sequence comparisons, not just for multiple alignments. Pairwise sequence comparison is just a special case of profile HMMs. HMMER can use a BLOSUM substitution matrix to parameterize a profile built from just one sequence. HMMER3 includes two programs for searching protein databases with single query sequences: `phmmer` and `jackhmmer`. I believe `phmmer` is superior in many respects to BLASTP, and `jackhmmer` to PSI-BLAST.

Automated annotation of protein domains. Various large databases of curated alignments and HMMER models of known domains are available, including Pfam and others. Many top ten lists of protein domains, a *de rigueur* table in genome analysis papers, depend on HMMER-based annotation. HMMER3's `hmmscan` program lets you scan a sequence against a profile database to parse the sequence into its component domains.

Curated collections of deep multiple alignments. There are thousands of sequence families, some of which comprise hundreds of thousands of sequences, and the raw sequence databases continue to double

every year or so. Clustering the entire sequence database into family alignments is a hopeless task for manual curation, but some sort of manual curation remains necessary for high-quality, biologically relevant multiple alignments. Databases like Pfam are constructed by distinguishing between a stable curated "seed" alignment of a small number of representative sequences, and "full" alignments of all detectable homologs. HMMER is used to make a model of the seed and to search the database for homologs, and the `hmmalign` program can automatically produce the full alignment by aligning every sequence to the seed consensus. `hmmalign` scales to alignments of millions of sequences.

HMMER uses ensemble algorithms, not *optimal alignment*

Most sequence search tools look for optimal high-scoring alignments. However, sequence alignments are uncertain, and the more distantly related sequences are, the more uncertain their alignment is. Instead of using optimal alignment algorithms, HMMER uses ensemble algorithms that consider all possible alignments, weighted by their relative likelihood. This is one reason that HMMER gets more power than tools that depend on single optimal alignment.

In HMM jargon, HMMER uses the Forward algorithm (and variants of it), not the Viterbi algorithm.

The use of ensemble algorithms shows up in several HMMER features:

Explicit representation of alignment uncertainty. When HMMER shows an alignment, it also calculates how much probability mass that this alignment has in the ensemble – which means HMMER can annotate a probabilistic confidence in an alignment, or in each individual aligned residue. Some downstream analyses that depend on alignments (such as phylogenetic tree inference) benefit from being able to distinguish confidently aligned residues.

Sequence scores, not alignment scores. HMMER's log-odds scores for a sequence aren't optimal alignment scores; they are summed over the posterior alignment ensemble. Statistical inference theory says that scores based on a single optimal alignment are an approximation that breaks down when alignments are uncertain. HMMER's calculation is the full, unapproximated calculation.

Different speed heuristics. The ensemble (Forward) algorithm is more computationally intensive than optimal alignment algorithms. HMMER3's acceleration strategy is quite different from BLAST's.[7]. HMMER implements heuristic accelerations of the HMM Forward

[7] S. R. Eddy. Accelerated profile HMM searches. *PLOS Comp. Biol.*, 7:e1002195, 2011

algorithm using vectorization technology available on modern processors.

Individually, none of these points is new. As far as alignment ensembles go, one reason why hidden Markov models were so theoretically attractive in the first place for sequence analysis is that they are good probabilistic models for explicitly dealing with alignment uncertainty. The SAM profile HMM software from UC Santa Cruz has always used full probabilistic inference (the HMM Forward/Backward algorithms) as opposed to optimal alignment scores (the HMM Viterbi algorithm). HMMER2 had the full HMM inference algorithms available as command-line options, but it used Viterbi optimal alignment by default, in part for speed reasons.

One reason why it's been hard to deploy sequence scores for practical large-scale use is that it wasn't known how to accurately calculate the statistical significance of a log-odds score that's been summed over alignment uncertainty. Accurate statistical significance estimates are essential when one is trying to discriminate homologs from millions of unrelated sequences in a large sequence database search. The statistical significance of optimal local alignment scores is calculated by Karlin/Altschul statistics.[8] Karlin/Altschul statistics are one of the most important and fundamental advances introduced by BLAST. However, Karlin/Altschul theory *doesn't* apply to HMMER's ensemble log-odds sequence scores (HMM "Forward scores"). The statistical significance (E-values, or expectation values) of HMMER sequence scores is determined by using a theoretical conjecture about the statistical properties of ensemble log-odds scores which have been supported by numerical simulation experiments.[9]

And as far as speed goes, the pioneers of heuristic acceleration of sequence database searches are the folks behind BLAST and FASTA, who developed effective heuristics that closely approximate an unaccelerated Smith/Waterman dynamic programming search. The first implementations of profile HMM methods used dynamic programming without heuristics (the profile HMM Viterbi algorithm is essentially Smith/Waterman, just with position-specific probability scores), so they were more comparable in speed to Smith/Waterman than to BLAST. Using the Forward algorithm slowed them down still more. It was a while before I invested the time to develop heuristic acceleration of profile HMM methods. A principal design goal in HMMER3 was to achieve at least rough speed parity with BLAST and FASTA.

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

[9] S. R. Eddy. A probabilistic model of local sequence alignment that simplifies statistical significance estimation. *PLOS Comput. Biol.*, 4:e1000069, 2008

Assumptions and limitations of profile HMMs

Profile HMMs are primary sequence consensus models. They assume that the residue at a particular position is independent of the residues at all other positions, and they neglect any higher-order correlations. Profile HMMs are often not good models of structural RNAs, for instance, because an HMM is not an adequate model of correlated base pairs.

A profile HMM also lacks any explicit model of the phylogenetic relationships among a set of homologous sequences. Sequences are instead assumed to be independently generated from the profile, which is tantamount to assuming a star phylogeny with fixed branch lengths. Ad hoc sequence weighting techniques are used to compensate for the fact that typical multiple alignments include many redundant, closely related sequences.

Our Infernal software provides better tools for structural RNA sequence analysis, using **profile stochastic context-free grammars**, a more complex class of probability model than HMMs.

How to learn more

Our book *Biological Sequence Analysis*[10] describes the basic theory behind profile HMMs and HMMER.

HMMER's open source development code is available on GitHub.[11] The GitHub issue tracker is the best way to give me suggestions, feature requests, bug reports, and pull requests.

Cryptogenomicon[12] is a blog where I sometimes talk about issues as they arise in HMMER, and where you can comment or follow the discussion.

[10] R. Durbin, et al. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge UK, 1998

[11] github.com/EddyRivasLab/hmmer

[12] cryptogenomicon.org

How to cite HMMER

There has never been a paper on the HMMER software.[13] The best citation is to the web site, hmmer.org.

You should also cite what version of the software you used. I archive all old versions, so anyone should be able to obtain the version you used, when exact reproducibility of an analysis is an issue. The version number is in the header of most output files. To see it quickly, do something like `hmmscan -h` to get a help page, and the header will say:

```
# hmmscan :: search sequence(s) against a profile database
# HMMER 3.2 (June 2018); http://hmmer.org/
# Copyright (C) 2018 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

So (from the second line there) this is from HMMER 3.2.

If an unenlightened, url-unfriendly journal forces you to cite dead trees, you can cite the 2011 paper on HMMER3 acceleration.[14]

[13] And the way things are going, there may never be!

[14] S. R. Eddy. Accelerated profile HMM searches. *PLOS Comp. Biol.*, 7:e1002195, 2011

How to report a bug

Open an issue on our issue tracker at GitHub,[15] or email me.

[15] github.com/EddyRivasLab/hmmer/issues

Please give me enough information that I can reproduce the bug, including any files. Ideally, I'd like to have a small, reproducible test case. So if you're reporting a bug, please send me:

- A brief description of what went wrong.
- The command line(s) that reproduce the problem.
- Copies of any files I need to run those command lines.
- Information about what kind of hardware you're on, what operating system, and (if you compiled the software yourself rather than running precompiled binaries), what compiler and version you used, with what configuration arguments.

Depending on how glaring the bug is, I may not need all this information, but any work you can put into giving me a clean reproducible test case doesn't hurt and often helps.

The information about hardware, operating system, and compiler is often important. Bugs are frequently specific to particular configurations of hardware/OS/compiler. I have a wide variety of systems available for trying to reproduce bugs, and I'll try to match your system as closely as we can.

If you first see a problem on some huge compute (like running a zillion query sequences over a huge profile database), it will really, really help me if you spend a bit of time yourself trying to isolate whether the problem really only manifests itself on that huge compute, or if you can isolate a smaller test case for me. The ideal bug report (for me) gives me everything I need to reproduce your problem in one email with at most some small attachments.

If I'm in my usual good mood, I'll reply quickly. I'll probably tell you we fixed the bug in our development code, and that the fix will appear in the next HMMER release. This of course doesn't help you much, since nobody knows when the next HMMER release is going to be. So if possible, I'll usually try to describe a workaround for the bug.

If the code fix is small, I might also tell you how to patch and recompile the code yourself. You may or may not want to do this.

Remember, I'm not a company with dedicated support staff – I'm one person, I've got other stuff to do, the Xfam team is asking me when HMMER4's going to be ready, and I'm as busy as you. I'll need to drop what I'm doing to try to help you out. Work with me to save me some time, and I'm more likely to stay in my usual good mood.

When's HMMER4 coming?

HMMER4 has been in development since 2011.[16] Some of the stuff it will include:

[16] OK, *slow* development, but hey.

The return of glocal alignment. Slow old HMMER2 was capable of "glocal" alignment, in which it would align a complete profile to a subsequence of a target sequence; this was great for annotating domain structure of protein sequences, among other things. In developing our speed heuristic for HMMER3, for numerical reasons, I had to sacrifice glocal alignment; HMMER3 *only* does local alignment. In HMMER4, I've solved the problems that prevented H3 from using glocal alignment. H4 uses a new *dual-mode* profile architecture, combining local and glocal alignment modes in a single probability model.

Memory efficiency. The HMMER ensemble alignment algorithms (the HMM Forward and Backward algorithms) are expensive in memory. For most uses, you don't notice, but there are extreme cases where you may. H3 can require as much as $\sim 36L^2$ bytes of memory for a query sequence of length L, and for a 35Kaa titin sequence, that's 44GB of RAM. In HMMER4, I've solved this with a variety of old and new techniques.

Ensemble calculations everywhere. HMMER uses ensemble calculations (i.e., integrates over alignment uncertainty) for scoring homologous sequences and for calculating the confidence in individual aligned residues. However, when it decides how many domains are in a sequence, and where they are, it uses an *ad hoc* procedure that uses ensemble information, but is not well defined. In HMMER4, we've solved that problem with a new domain definition algorithm.

More processor support. One of the attractive features of the HMMER "MSV" acceleration algorithm is that it is a very tight and efficient piece of code. The bad news is, it's a very tight and efficient piece of *assembly* code. We have to write processor-specific SIMD vector instructions for each platform that HMMER runs on. HMMER currently only supports x86 (Intel/AMD) and PowerPC platforms (big-endian AIX PowerPC's, not the newer crop of little-endian Linux PowerPC's). HMMER4 will also include support for Linux/PowerPC and ARM NEON. It also can use the latest x86 vector instructions (AVX and AVX-512).

Better parallelization. HMMER is so fast that it's often input-bound, rather than CPU-bound: that is, it takes longer just to get the sequences from your disk than it takes to compare them to a profile. That's been taxing the simple parallelization methods we use. HMMER's multithreaded parallelization really doesn't scale well beyond 2-4 processors, on most machines; and possibly worse, if you're

on a slow filesystem (for example, if you're reading data from a network filesystem instead of from local disk). In H4, we're working on improving our parallelization and our data input.

What's still missing

Two of the more important holes for me are:

Translated comparisons. I'd love to have the HMM equivalents of BLASTX, TBLASTN, and TBLASTX. They'll come. In the meantime, I translate DNA sequence to six frames, and search hypothetical ORFs as protein sequences.

Profile/profile comparison. A number of pioneering papers and software packages have demonstrated the power of profile/profile comparison for even more sensitive remote homology detection. Check out HHBLITS from Johannes Söding's group.[17]

[17] toolkit.tuebingen.mpg.de/#/tools/hhblits

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

Other implementations of profile HMM methods and position-specific scoring matrix methods are available, including:

Software	URL
HH-SUITE	www.soeding.genzentrum.lmu.de/software-and-servers-2
PSI-BLAST	blast.ncbi.nlm.nih.gov
PFTOOLS	web.expasy.org/pftools
SAM	compbio.soe.ucsc.edu/sam.html

Installation

Choose one of the following three sections depending on whether you want to install a precompiled HMMER package for your system, compile from our source code distribution,[1] or compile source code from our github repository.[2] We recommend that you use one of the first two options. You can skip the gory details section unless you're already proficient and you want to use optional configuration or installation parameters.

[1] hmmer.org

[2] github.com/EddyRivasLab/hmmer

Quickest: install a precompiled binary package

The easiest way to install HMMER is to install a precompiled package for your operating system.[3] Some examples that I know of:

[3] Thanks to all the people who do the packaging!

```
% brew install hmmer              # OS/X, HomeBrew
% port install hmmer              # OS/X, MacPorts
% apt install hmmer               # Linux (Ubuntu, Debian...)
% dnf install hmmer               # Linux (Fedora)
% yum install hmmer               # Linux (older Fedora)
% conda install -c biocore hmmer  # Anaconda
```

Quick-ish: compile the source code

You can obtain the source code as a compressed `.tar.gz` tarball from hmmer.org in your browser, or you can also `wget` it on the command line from eddylab.org/software/hmmer/hmmer-3.2.tar.gz. Uncompress and untar it, and switch into the `hmmer-3.2` directory. For example:

```
% wget http://eddylab.org/software/hmmer/hmmer-3.2.tar.gz
% tar xf hmmer-3.2.tar.gz
% cd hmmer-3.2
```

To compile:

```
% ./configure
% make
```

Optionally, to compile and run a test suite:

```
% make check
```

The newly compiled binaries are now in the `src` directory. You can run them from there, or manually copy them wherever. You don't have to install HMMER programs to run them. Optionally, to install the programs and man pages in standard locations on your system, do:

```
% make install
```

By default, programs are installed in `/usr/local/bin` and man pages in `/usr/local/share/man/man1/`. You may need root privileges to do this, so you might need something like `sudo make install`.

You can change the `/usr/local` prefix to any directory you want when you do the `./configure` step, using the `./configure --prefix` option, as in `./configure --prefix /the/directory/you/want`. For example, you might do `./configure --prefix=${HOME}`, for installation in `bin/` and `share/man/man1` subdirectories in your own home directory.

Optionally, you can also install a set of additional small tools ("miniapps") from our Easel library. We don't do this by default, in case you already have a copy of Easel separately installed (from Infernal, for example). To install Easel miniapps and their man pages too:

```
% cd easel; make install
```

If you decide you did something wrong after the `./configure`, `make distclean` will clean up everything that got built and restore the distribution to a pristine state, and you can start again.

Geeky: compile source from our github repository

Alternatively, you can clone our git repository master branch:[4]

```
% git clone https://github.com/EddyRivasLab/hmmer hmmer-3.2
% cd hmmer-3.2
```

This is now essentially the same as if you unpacked a tarball, so from here, follow the `./configure; make` instructions above.

One difference is that our distribution tarballs include this user guide as a PDF, in addition to its LaTeX source code. The github repo only has the source LaTeX files. To compile the PDF, see "compiling the user guide" in the gory details below.

[4] As of 3.2, our git master branch is the stable current release, as the git deities prefer. This wasn't true for us in the past.

Gory details

System requirements

Operating system: HMMER is designed for POSIX-compatible platforms, including UNIX, Linux, and Mac OS/X. The POSIX standard essentially includes all operating systems except Microsoft Windows.[5] We test most extensively on Intel/Linux and on Apple OS/X.

[5] Windows 10 includes a Linux subsystem that allows you to install a Linux OS inside Windows, with a bash command shell, and this should work fine. For older Windows, there are add-on products available for making Windows more POSIX-compliant and more compatible with GNU-ish configures and builds. One such product is Cygwin, www.cygwin.com, which is freely available.

Processor: HMMER depends on vector parallelization methods that are processor-specific. H3 requires either an x86-compatible (Intel/AMD) processor that supports the SSE2 vector instruction set, or a big-endian (AIX, not Linux) PowerPC processor that supports the Altivec/VMX instruction set. SSE2 is supported on Intel processors from Pentium 4 on, and AMD processors from K8 (Athlon 64) on. This includes almost all Intel processors since 2000 and AMD processors since 2003. Altivec/VMX is supported on Motorola G4, IBM G5, and IBM PowerPC processors starting with the Power6, which includes almost all PowerPC-based desktop systems since 1999 and servers since 2007.[6]

[6] If your platform does not support either of these vector instruction sets, the configure script will stop with an error message.

Compiler: The source code conforms to ANSI C99 and POSIX standards. It should compile with any ANSI C99 compliant compiler, including the freely available GNU C compiler `gcc`. We test the code most frequently using the GNU `gcc`, Apple `llvm/clang`, and Intel `icc` compilers.[7]

[7] On OS/X, if you're compiling the source, make sure you have XCode installed so that you have a C compiler.

Libraries and other installation requirements: HMMER3 does not have any dependencies other than a C compiler. It does not require any additional libraries to be installed by you, other than standard ANSI C99 libraries that are already present on a system with a C99 compiler.

The HMMER distribution is bundled with a software library from our lab called Easel.[8] Bundling Easel instead of making it a separate installation requirement simplifies installation. Easel is also included in other software from our lab. For example, Infernal[9] bundles both HMMER and Easel. If you install the Easel miniapps, you probably only want to do that once, from the most recent version of HMMER, Infernal, or Easel itself, to avoid clobbering a newer version with an older one.

[8] bioeasel.org

[9] eddylab.org/infernal

Our configuration and compilation process uses standard UNIX utilities. Although these utilities are *supposed* to be available on all POSIX-compliant systems, there are always a few crufty old dinosaurs still running out there that do not support all the features

that our `./configure` script and Makefiles are expecting. We do aim to build cleanly on anything from supercomputers to Ebay'ed junk, but if you have an old system, you may want to hedge your bets and install up-to-date versions of GNU command line tools such as GNU make and GNU grep.

Running the test suite (and some of our development tools, if you delve deep into our codebase) requires Perl and Python to be installed. If you don't have them (which should be rare), `make check` won't work for you, but that's ok because `make` and `make install` will still work fine.

Compiling the user guide itself (this document) does require additional tools to be installed, including `rman` and some extra LaTeX packages, described below.

Multicore parallelization is default

HMMER supports multicore parallelization using POSIX threads. By default, the configure script will identify whether your platform supports POSIX threads (almost all platforms do), and it will automatically compile in multithreading support.

To disable multithreading at compile time, compile from source with the `--disable-threads` flag to `./configure`.

Multithreaded HMMER programs use master/worker parallelization, with `<n>` worker threads and one master thread. When HMMER is run on a machine with multiple available cores, the default number of worker threads is two[10]. You can control the number of cores each HMMER process will use for computation with the `--cpu <n>` command line option or the `HMMER_NCPU` environment variable.

[10] Set by a compile-time configuration option, `P7_NCPU`, in `src/p7_config.h.in`.

If you specify `--cpu 0`, a HMMER search program will run in serial-only mode, with no threading. We use this in debugging when we suspect something is awry with the parallel implementation, but it's not something you'd generally want to do in your work. Even with a single worker thread (`--cpu 1`), HMMER will be faster than serial-only mode, because the master thread handles input and output.

If you are running HMMER on a cluster that enforces policy on the number of cores a process can use, you may need to count both the workers and the master: you may need to tell your cluster management software that HMMER needs `<n>+1` cores.

MPI cluster parallelization is optional

MPI (Message Passing Interface) parallelization on clusters is supported in `hmmbuild` and all search programs except `nhmmer` and `nhmmscan`. To compile for MPI, you need to have an MPI library installed, such as OpenMPI.[11]

[11] open-mpi.org

MPI support is not enabled by default. To enable MPI support at compile time, add the `--enable-mpi` option to your `./configure` command.

To use MPI parallelization, each program that has an MPI-parallel mode has an `--mpi` command line option. This option activates a master/worker parallelization mode.

Without the `--mpi` option, if you run a program under `mpirun` or the equivalent on N nodes, you'll be running N duplicates, not a single MPI-enabled parallel search. Don't do that.

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

Using build directories

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

```
% mkdir my-hmmer-build
% cd my-hmmer-build
% ../configure
% make
```

This assumes you have a `make` that supports `VPATH`. If your system's `make` does not, you can install GNU make.

Makefile targets

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

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

`pdf` Compiles this user guide.

`install` Installs programs and man pages.

`uninstall` Removes programs and man pages from where `make install` put them.

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

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

Compiling the user guide

Compiling this User Guide from its source LaTeX requires LaTeX, of course, and also the `rman` program from `PolyGlotMan`.[12] It use a customized version of the Tufte-LaTeX book class[13] (which we include in our source code, so you don't have to install it), and the Tufte-LaTeX package depends on some optional LaTeX packages listed at the Tufte-LaTeX site. These packages are typically included in bundles in standard LaTeX distributions such as TeX Live.[14] You can probably identify a short list of basic plus extra LaTeX stuff you need to install on your machine. For example, on my Mac OS/X laptop, using the MacPorts package manager:[15]

[12] sourceforge.net/projects/polyglotman

[13] tufte-latex.github.io/tufte-latex

[14] www.tug.org/texlive

[15] www.macports.org

```
% sudo port install texlive
% sudo port install texlive-latex-extra
% sudo port install rman
```

Once you have these dependencies, doing:

```
% make pdf
```

in the top-level source directory builds `Userguide.pdf` in the subdirectory `documentation/userguide`.

What gets installed by `make install`, and where?

HMMER only installs programs and man pages. There are 18 programs in HMMER and 22 in Easel (the Easel "miniapps"), each with a man page.

Each program is free-standing. Programs don't depend on any details of where other files are installed, and they will run fine no matter where you copy them. Similarly the man pages can be read in any file location with `man full/path/to/manpage.man`. Nonetheless, it's most convenient if you put the programs in a directory that's in your `PATH` and the man pages in a standard man page directory, using `make install`.

The top-level Makefile has variables that specify the two directories where `make install` installs things:

Variable	What
bindir	programs
man1dir	man pages

These variables are constructed from others in accordance with GNU Coding Standards, as follows:

Variable	Default	`./configure` option
prefix	`/usr/local`	`--prefix`
exec_prefix	*prefix*	`--exec_prefix`
bindir	*exec_prefix*`/bin`	`--bindir`
datarootdir	*prefix*`/share`	`--datarootdir`
mandir	*datarootdir*`/man`	`--mandir`
man1dir	*mandir*`/man1`	`--man1dir`

You can change any of these defaults on the `./configure` command line using the corresponding option. The most commonly used option is `--prefix`. For example, if you want to install HMMER in a directory hierarchy all of its own, you might want to do something like:

```
% ./configure --prefix /usr/local/hmmer-3.2
```

That would keep HMMER out of your system-wide directories, which might be desirable. This is a simple way to install multiple versions of HMMER, for example, without having them clobber each other. Then you'd add `/usr/local/hmmer-3.2/bin` to your `PATH` and `/usr/local/hmmer-3.2/share/man` to your `MANPATH`.

Again, these variables only affect where `make install` copies stuff. HMMER and Easel programs have no pathnames compiled into them.

Installing both HMMER2 and HMMER3

HMMER3 and HMMER2 are distinct codebases that are generally incompatible with each other. The last release of HMMER2 was 2.3.2 in 2003. HMMER3 was first released in 2010.

HMMER3 is superior to HMMER2 in almost all respects. One exception is that HMMER2 is capable of global and glocal alignment, whereas HMMER3 programs generally only use local alignment. It turned out that some HMMER users need global/glocal alignment more than they want the speed and statistics, so HMMER2 still has users. I didn't anticipate this when I wrote H3. Unfortunately, the two packages have incompatible programs that have the same names, so installing them both can lead to problems.

HMMER3's speed depends on numerical properties that only hold for local alignment. Its statistics depend on a statistical conjecture that only holds well for local alignment. The internal HMMER3 API includes global and glocal alignment modes like HMMER2, but the programs don't use these modes.

Specifically, HMMER2 installs 9 programs, 6 of which have identical names with incompatible HMMER3 programs: `hmmalign`, `hmmbuild`, `hmmconvert`, `hmmemit`, `hmmfetch`, and `hmmsearch`.

One workaround is to install the two packages each in their own hierarchy, as above: `./configure -prefix=somewhere/hmmer-3.2` for HMMER3, and `./configure -prefix=somewhere/hmmer-2.3.2` for HMMER2. One set of programs could be in your `PATH`, and you could call the others using full pathnames.

Another workaround is simply to copy the HMMER2 programs to an installation directory while renaming them, bypassing `make install`. For example, something like:

```
% cp hmmalign /usr/local/bin/h2-hmmalign
% cp hmmconvert /usr/local/bin/h2-hmmconvert
...
```

and so on.

Seeing more output from `make`

By default, our `make` hides what's really going on with the compilation with a pretty wrapper that we stole from the source for `git`. If you want to see what the command lines really look like in all their ugly glory, pass a `V=1` option (V for "verbose") to `make`, as in:

```
% make V=1
```

Staged installations in a buildroot, for a packaging system

HMMER's `make install` supports staged installations, accepting the traditional `DESTDIR` variable that packagers use to specify a buildroot. For example, you can do:

```
% make DESTDIR=/rpm/tmp/buildroot install
```

Workarounds for unusual configure/compilation problems

Failure when trying to use a separate build directory. If you try to build in a build tree (other than the source tree) and you have any trouble in configuration or compilation, try just building in the source tree instead. Some `make` versions don't support the VPATH mechanism needed to use separate build trees. Another workaround is to install GNU make.

Configuration fails, complaining "no acceptable grep could be found". We've seen this happen on our Sun Sparc/Solaris machine. It's a known issue in GNU autoconf. You can either install GNU grep, or you can insist to `./configure` that the Solaris grep (or whatever grep you have) is ok by explicitly setting `GREP` to a path to one that works:

```
% ./configure GREP=/usr/xpg4/bin/grep
```

Many 'make check' tests fail. We have one report of a system that failed to link multithread-capable system C libraries correctly, and instead linked to one or more serial-only libraries.[16] We were unable

[16] If you're a pro: the telltale phenotype of this failure is to configure with debugging flags on and recompile. Run one of the failed unit test drivers (such as `easel/easel_utest`) yourself and let it dump core. Use a debugger to examine the stack trace in the core. If it failed in `__errno_location()`, then it's linked a non-thread-capable system C library.

to reproduce the problem, and are not sure what could possibly cause it. We optimistically believe it was a one-off messed-up system, not our fault, but then we often say things like that and they turn out to be wrong. If it does happen, it screws all kinds of things up with the multithreaded implementation. A workaround is to shut threading off:

```
% ./configure --disable-threads
```

This will compile code won't parallelize across multiple cores, of course, but it will still work fine on a single processor at a time (and MPI, if you build with MPI enabled).

Tutorial

First let's do something useful and see it work, then we'll do a complete walkthrough.

Tap, tap; is this thing on?

In the `tutorial` subdirectory, `globins4.sto` is an example of a basic Stockholm alignment file. `hmmbuild` builds a profile from an alignment:

```
% cd tutorial
% hmmbuild globins4.hmm globins4.sto
```

`hmmsearch` searches a profile against a sequence database. The file `tutorial/globins45.fa` is a small example of a FASTA file containing 45 globin sequences:

```
% hmmsearch globins4.hmm globins45.fa
```

This will print an output of the search results, with a table of significant hits followed by their alignments.

That's all you need to start using HMMER for work. You can build a profile of your favorite sequence alignment if you have one; you can also obtain alignments and profiles from Pfam.[1] You can obtain real sequence databases to search from NCBI[2] or UniProt[3]. You don't have to worry much about sequence file formats. HMMER can read most common alignment and sequence file formats automatically.

`hmmbuild` needs to be installed in your `PATH` to be able to just type the `hmmbuild` command like this. Otherwise you need to give a path to where `hmmbuild` is, which might be `src/hmmbuild`, if you're in the HMMER top level distribution directory. If you're new to how paths to programs and files work on the command line, skip ahead to running a HMMER program for some more detail.

[1] pfam.xfam.org

[2] ftp.ncbi.nih.gov/blast/db/FASTA/nr.gz

[3] www.uniprot.org/downloads

The programs in HMMER

In rough order of importance, the 18 HMMER programs are:

hmmbuild	build profile from input multiple alignment
hmmalign	make multiple sequence alignment using a profile
hmmsearch	search profile against sequence database
hmmscan	search sequence against profile database
hmmpress	prepare profile database for `hmmscan`
phmmer	search single sequence against sequence database
jackhmmer	iteratively search single sequence against database
nhmmer	search DNA query against DNA sequence database
nhmmscan	search DNA sequence against a DNA profile database
hmmfetch	retrieve profile(s) from a profile file
hmmstat	show summary statistics for a profile file
hmmemit	generate (sample) sequences from a profile
hmmlogo	produce a conservation logo graphic from a profile
hmmconvert	convert between different profile file formats
hmmpgmd	search daemon for the `hmmer.org` website
makehmmerdb	prepare an `nhmmer` binary database
hmmsim	collect score distributions on random sequences
alimask	add column mask to a multiple sequence alignment

The programs `hmmbuild`, `hmmsearch`, `hmmscan`, and `hmmalign` are the core functionality for protein domain analysis and annotation pipelines, for instance using profile databases like Pfam.

The `phmmer` and `jackhmmer` programs search a single protein sequence against a protein sequence database, akin to BLASTP and PSI-BLAST. (Internally, they just produce a profile from the query sequence, then run profile searches.)

`nhmmer` is the equivalent of `hmmsearch` and `phmmer`, but is capable of searching long, chromosome-size target DNA sequences. `nhmmscan` is the equivalent of `hmmscan`, capable of using chromosome-size DNA sequences as a query into a profile database.

`nhmmer` and `nhmmscan` were added in HMMER3.1.

Running a HMMER program

After you compile HMMER, these programs are in the `src/` subdirectory of the top-level HMMER directory. If you run them without arguments, they will give you a brief help message. In this chapter, I will assume that you have installed them (with `make install`, perhaps) so they're in your `PATH`. So if you type `hmmbuild` at the command line, you see:

If you run a HMMER program with a `-h` option and no arguments, it will give you a brief summary of its usage and options.

```
% hmmbuild
Incorrect number of command line arguments.
Usage: hmmbuild [-options] <hmmfile_out> <msafile>

where basic options are:
  -h     : show brief help on version and usage
  -n <s> : name the HMM <s>
  -o <f> : direct summary output to file <f>, not stdout
```

```
  -O <f> : resave annotated, possibly modified MSA to file <f>

To see more help on other available options, do:
  hmmbuild -h
```

If you haven't installed the HMMER programs, you need to specify both the program name and a path to it. This tutorial chapter assumes that you're in the `tutorial` subdirectory, where the tutorial example data files are. From `tutorial` , the relative path to the compiled programs is `../src/`. So instead of just typing `hmmbuild`, you could do:

The `%` represents your command prompt. It's not part of what you type.

```
% ../src/hmmbuild
```

Make sure you can run a HMMER program like this before moving on. If you are stuck getting something like `hmmbuild: command not found`, the unix shell isn't finding the program in your `PATH` or you're not giving a correct explicit path. Consult your shell's documentation, or a friendly local unix guru.

Files used in the tutorial

The subdirectory called `tutorial` in the HMMER distribution contains the files used in the tutorial. If you haven't already, change into that directory now.

```
% cd tutorial
```

If you do a `ls`, you'll see there are several example files in the `tutorial` directory:

`globins4.sto` An example alignment of four globin sequences, in Stockholm format. This alignment is a subset of a famous old published structural alignment from Don Bashford.[4]

[4] Donald Bashford, et al. Determinants of a protein fold: Unique features of the globin amino acid sequences. *J. Mol. Biol.*, 196:199–216, 1987

`globins45.fa` 45 unaligned globin sequences, in FASTA format.

`HBB_HUMAN` A FASTA file containing the sequence of human $\beta-$hemoglobin.

`fn3.sto` An example alignment of fibronectin type III domains. This is a Pfam fn3 seed alignment, in Stockholm format.

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

`7LESS_DROME` A FASTA file containing the sequence of the *Drosophila* Sevenless protein, a receptor tyrosine kinase whose extracellular region consists of an array of several fibronectin type III domains.

`MADE1.sto` An example DNA alignment, a subset of the Dfam seed alignment for the MADE1 transposable element family.

`dna_target.fa` A 330Kb sequence from human chromosome 1 in FASTA format, containing four MADE1 elements.

On sequence file formats, briefly

Input files for HMMER include unaligned sequence files and multiple sequence alignment files. Sequence files and alignment files can come in many different poorly standardized formats.

A commonly used format for unaligned sequence files and sequence databases is FASTA format. Several of the tutorial files give you examples (`globins45.fa`, for example).

HMMER's preferred alignment file format is Stockholm format, which is also the format that Pfam alignments are in. Stockholm allows detailed annotation of columns, residues, and sequences, and HMMER is built to use this annotation. Stockholm also allows a file to contain many alignments for many families (a multiple multiple alignment file?). `globins4.sto` is a simple example, and `fn3.sto` is an example with a lot of annotation markup.

Stockholm format was developed jointly with us by the Pfam curation team.

HMMER can read several other sequence and alignment file formats. By default, it autodetects what format an input file is in. Accepted unaligned sequence file formats include `fasta`, `uniprot`, `genbank`, `ddbj`, and `embl`. Accepted multiple alignment file formats include `stockholm`, `afa` (i.e. aligned FASTA), `clustal`, `clustallike` (MUSCLE, etc.), `a2m`, `phylip` (interleaved), `phylips` (sequential), `psiblast`, and `selex`. These formats are described in detail in a later chapter. Where applicable, the programs have command line options for asserting an input format and skipping autodetection, when you don't want to depend on it.

HMMER also automatically detects whether a sequence or alignment file contains nucleotide or protein sequence data. Like format autodetection, alphabet autodetection sometimes doesn't work on weird files. Where applicable, the programs have options (usually `--rna`, `--dna`, `--amino`) for asserting the input alphabet type.

For more information in HMMER input files and formats, see the formats chapter on page 199.

Searching a sequence database with a profile

Now let's go through the `hmmbuild`/`hmmsearch` example in a bit more detail.

Step 1: build a profile with `hmmbuild`

The file `globins4.sto` looks like this:

```
# STOCKHOLM 1.0

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

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

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

Most popular alignment formats are similar block-based formats. They can be turned into Stockholm format with a little editing or scripting. Don't forget the `# STOCKHOLM 1.0` line at the start of the alignment, nor the `//` at the end.

Stockholm alignments can be concatenated to create an alignment database flatfile containing many alignments. The Pfam database, for example, distributes a single file containing representative alignments for thousands of sequence families.

The Easel miniapps provide tools for manipulating alignment files, such as `esl-afetch` for extracting one alignment by name or accession from a Pfam file.

You ran `hmmbuild` to build a profile from that alignment:

```
% hmmbuild globins4.hmm globins4.sto
```

and you got some output that looks like:

```
# hmmbuild :: profile HMM construction from multiple sequence alignments
# HMMER 3.2 (June 2018); http://hmmer.org/
# Copyright (C) 2018 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# input alignment file:             globins4.sto
# output HMM file:                  globins4.hmm
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# idx name                  nseq  alen  mlen eff_nseq re/pos description
#---- -------------------- ----- ----- ----- -------- ------ -----------
1     globins4                 4   171   149     0.96  0.589

# CPU time: 0.07u 0.00s 00:00:00.07 Elapsed: 00:00:00.07
```

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

The information on these lines is almost self-explanatory. The `globins4` alignment consisted of 4 sequences with 171 aligned columns (`alen`). HMMER turned it into a profile of 149 consensus positions (`mlen`), which means it defined 22 gap-containing alignment columns to be insertions relative to consensus. The 4 sequences were only

counted as an "effective" total sequence number (`eff_nseq`) of 0.96, because they're fairly similar to each other.[5] The profile ended up with a relative entropy per position (`re/pos`; average score, or information content) of 0.589 bits.

[5] It's not unusual for this number to be less than 1. I'll explain why later.

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

```
HMMER3/f [3.2 | June 2018]
NAME  globins4
LENG  149
ALPH  amino
RF    no
MM    no
CONS  yes
CS    no
MAP   yes
DATE  Tue May 29 20:41:39 2018
NSEQ  4
EFFN  0.964844
CKSUM 2027839109
STATS LOCAL MSV       -9.9014  0.70957
STATS LOCAL VITERBI  -10.7224  0.70957
STATS LOCAL FORWARD   -4.1637  0.70957
HMM          A        C        D        E        F        G        H    ...      W        Y
            m->m     m->i     m->d     i->m     i->i     d->m     d->d
  COMPO   2.36553  4.52577  2.96709  2.70473  3.20818  3.02239  3.41069 ...   4.55393  3.62921
          2.68640  4.42247  2.77497  2.73145  3.46376  2.40504  3.72516 ...   4.58499  3.61525
          0.57544  1.78073  1.31293  1.75577  0.18968  0.00000        *
      1   1.70038  4.17733  3.76164  3.36686  3.72281  3.29583  4.27570 ...   5.32720  4.10031      9 v - - -
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494 ...   4.58477  3.61503
          0.03156  3.86736  4.58970  0.61958  0.77255  0.34406  1.23405
...
    149   2.92198  5.11574  3.28049  2.65489  4.47826  3.59727  2.51142 ...   5.42147  4.18835    165 k - - -
          2.68634  4.42241  2.77536  2.73098  3.46370  2.40469  3.72511 ...   4.58493  3.61418
          0.22163  1.61553        *  1.50361  0.25145  0.00000        *
//
```

The HMMER ASCII save file format is defined on page 202.

Step 2: search the sequence database with hmmsearch

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

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

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

Have a look at the output, `globins4.out`. The first section is the *header* that tells you what program you ran, on what, and with what options:

```
# hmmsearch :: search profile(s) against a sequence database
# HMMER 3.2 (June 2018); http://hmmer.org/
# Copyright (C) 2018 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query HMM file:                  globins4.hmm
# target sequence database:        uniprot_sprot.fasta
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       globins4  [M=149]
Scores for complete sequences (score includes all domains):
```

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

```
--- full sequence ---   --- best 1 domain ---    -#dom-
 E-value  score  bias    E-value  score  bias    exp  N  Sequence              Description
 ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
 6.7e-65  222.7   3.2    7.5e-65  222.6   3.2    1.0  1  sp|P02185|MYG_PHYCD   Myoglobin OS=Physeter catodon OX=9755
 3.5e-63  217.2   0.1    3.8e-63  217.0   0.1    1.0  1  sp|P02024|HBB_GORGO   Hemoglobin subunit beta OS=Gorilla gor
   5e-63  216.6   0.0    5.6e-63  216.5   0.0    1.0  1  sp|P68871|HBB_HUMAN   Hemoglobin subunit beta OS=Homo sapien
   5e-63  216.6   0.0    5.6e-63  216.5   0.0    1.0  1  sp|P68872|HBB_PANPA   Hemoglobin subunit beta OS=Pan paniscu
   5e-63  216.6   0.0    5.6e-63  216.5   0.0    1.0  1  sp|P68873|HBB_PANTR   Hemoglobin subunit beta OS=Pan troglod
 7.2e-63  216.1   3.0      8e-63  216.0   3.0    1.0  1  sp|P02177|MYG_ESCRO   Myoglobin OS=Eschrichtius robustus OX=
```

The last two columns, obviously, are the name of each target sequence and optional description. The description line usually gets truncated just to keep line lengths down to something reasonable. If you want the full description line, and don't mind long output line lengths, use the `--notextw` option.

The most important number here is the first one, the *sequence E-value*. The E-value is the expected number of false positives (non-homologous sequences) that scored this well or better. The E-value is a measure of statistical significance. The lower the E-value, the more significant the hit. I typically consider sequences with E-values $< 10^{-3}$ or so to be significant hits.

The E-value is based on the *sequence bit score*, the second number. This is the log-odds score for the complete sequence. Some people like to see a bit score instead of an E-value, because the bit score doesn't depend on the size of the sequence database, only on the profile and the target sequence. The E-value does depend on the size of the database you search: if you search a database ten times larger, you get ten times the number of false positives.

The next number, the *bias*, is a correction term for biased sequence composition that was applied to the sequence bit score. For instance, for the top hit `MYG_PHYCD` that scored 222.7 bits, the bias of 3.2 bits means that this sequence originally scored 225.9 bits, which was adjusted by the slight 3.2 bit biased-composition correction. The only time you really need to pay attention to the bias value is when it's large, on the same order of magnitude as the sequence bit score. Sometimes (rarely) the bias correction isn't aggressive enough, and allows a non-homolog to retain too much score. Conversely, the bias

The method that HMMER uses to compensate for biased composition remains unpublished, shamefully.

correction can be too aggressive sometimes, causing you to miss homologs. You can turn the biased-composition score correction off with the `--nonull2` option.[6]

[6] And if you're doing that, you may also want to set `--nobias`, to turn off another biased composition step called the bias filter, which affects which sequences get scored at all.

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

[7] This happens to be a Pfam seed alignment. It's a good example of an alignment with complex Stockholm annotation.

```
% hmmbuild fn3.hmm fn3.sto
% hmmsearch fn3.hmm 7LESS_DROME > fn3.out
```

In `fn3.out`, the sequence top hits list says:

```
--- full sequence ---   --- best 1 domain ---    -#dom-
 E-value  score  bias    E-value  score  bias    exp  N  Sequence    Description
 ------- ------ -----    ------- ------ -----   ---- --  --------    -----------
 6.2e-56  173.1   1.6    1.1e-16   47.3   0.9    9.5  9  7LESS_DROME  RecName: Full=Protein sevenless;          EC=2.7
```

OK, now let's pick up the explanation where we left off. The total sequence score of 173.1 sums up *all* the fibronectin III domains that were found in the `7LESS_DROME` sequence. The "single best dom" score and E-value are the bit score and E-value as if the target sequence only contained the single best-scoring domain, without this summation.

The idea is that we might be able to detect that a sequence is a member of a multidomain family because it contains multiple weakly-scoring domains, even if no single domain is solidly significant on its own. On the other hand, if the target sequence happened to be a piece of junk consisting of a set of identical internal repeats, and one of those repeats accidentially gives a weak hit to the query profile, all the repeats will sum up and the sequence score might look "significant".[8]

[8] Mathematically, alas, the correct answer: the null hypothesis we're testing against is that the sequence is a *random* sequence of some base composition, and a repetitive sequence isn't random.

So operationally:

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

OK, the sharp eyed reader asks, if that's so, then why in the globin4 output (all of which have only a single domain) do the full

sequence bit scores and best single domain bit scores not exactly agree? For example, the top ranked hit, MYG_PHYCD, has a full sequence score of 222.7 and a single best domain score of 222.6. What's going on? What's going on is that the position and alignment of that domain is uncertain – in this case, only very slightly so, but nonetheless uncertain. The full sequence score is summed over all possible alignments of the globin profile to the MYG_PHYCD sequence. When HMMER identifies domains, it identifies what it calls an **envelope** bounding where the domain's alignment most probably lies. The "single best dom" score is calculated after the domain envelope has been defined, and the summation is restricted only to the ensemble of possible alignments that lie within the envelope. The fact that the two scores are slightly different is therefore telling you that there's a small amount of probability (uncertainty) that the domain lies somewhat outside the envelope bounds that HMMER has selected.

The difference between envelopes and alignments comes up again below when we discuss the reported coordinates of domains and alignments in the next section of the output.

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

These two numbers should be about the same. Rarely, you might see that they're very different, and this would usually be a sign that the target sequence is so highly repetitive that it's confused the HMMER domain postprocessor.[9]

[9] Such sequences aren't likely to show up as significant homologs to any sensible query in the first place.

The sequence top hits output continues until it runs out of sequences to report. By default, the report includes all sequences with an E-value of 10.0 or less. It's showing you the top of the noise, so you can decide for yourself what's interesting or not: the default output is expected to contain about 10 false positives with E-values in the range of about 1-10.

Then comes the third output section, which starts with

```
Domain annotation for each sequence (and alignments):
```

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

```
>> 7LESS_DROME  RecName: Full=Protein sevenless;          EC=2.7.10.1;
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 ?   -1.5   0.0      0.18      0.18      60      72 ..     396     408 ..     395     410 .. 0.86
   2 !   40.8   0.0   1.2e-14   1.2e-14       1      83 [.     439     520 ..     439     521 .. 0.95
   3 !   14.8   0.0   1.5e-06   1.5e-06      12      84 ..     836     913 ..     826     914 .. 0.74
   4 !    5.0   0.0    0.0017    0.0017       9      36 ..    1209    1236 ..    1203    1258 .. 0.83
   5 !   22.4   0.0   6.7e-09   6.7e-09      13      79 ..    1313    1380 ..    1305    1385 .. 0.81
   6 ?    0.6   0.0      0.04      0.04      55      72 ..    1753    1769 ..    1720    1769 .. 0.87
   7 !   47.3   0.9   1.1e-16   1.1e-16       1      84 [.    1800    1890 ..    1800    1891 .. 0.91
   8 !   17.9   0.0   1.6e-07   1.6e-07       5      73 ..    1904    1966 ..    1901    1976 .. 0.91
   9 !   10.1   0.0   4.5e-05   4.5e-05       1      85 []    1994    2107 ..    1994    2107 .. 0.87
```

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

The `!` or `?` symbol indicates whether this domain does or does not satisfy both per-sequence and per-domain inclusion thresholds. Inclusion thresholds are used to determine what matches should be considered to be "true", as opposed to reporting thresholds that determine what matches will be reported.[10] By default, inclusion thresholds usually require a per-sequence E value of 0.01 or less and a per-domain conditional E-value of 0.01 or less (except jackhmmer, which requires a more stringent 0.001 for both), and reporting E-value thresholds are set to 10.0.

[10] The default reporting threshold of 10.0 is chosen so you get to see about ~10 insignificant hits at the top of the noise, so you can see what interesting sequences might be getting tickled by your search.

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

The first of the two E-values is the **conditional E-value**. It is an attempt to measure the statistical significance of each domain, *given that we've already decided that the target sequence overall is a true homolog*. It is the expected number of *additional* domains we'd find with a domain score this big in the set of sequences reported in the top hits list, if those sequences consisted only of random nonhomologous sequence outside the best region that sufficed to define them as homologs.

The conditional E-value is a weird statistic, and it's not clear I'm going to keep it.

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

The different between the two E-values is not apparent in the `7LESS_DROME` example because in both cases, the size of the search space as 1 sequence. There's a single sequence in the target sequence database (that's the size of the search space that the independent/best single domain E-value depends on). There's one sequence reported as a putative homolog in the sequence top hits list (that's the size of the search space that the conditional E-value depends on). A better example is to see what happens when we search UniProt (I used release 2018_02, which contains 556,825 sequences) with the `fn3` profile:

```
% hmmsearch fn3.hmm uniprot_sprot.fasta
```

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

```
>> sp|P13368|7LESS_DROME  Protein sevenless OS=Drosophila melanogaster OX=7227 GN=sev PE=1 SV=2
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 !   40.8   0.0   9.1e-12   6.4e-09       1      83 [.     439     520 ..     439     521 .. 0.95
   2 !   14.8   0.0    0.0012      0.85      12      84 ..     836     913 ..     826     914 .. 0.74
   3 ?    5.0   0.0       1.4   9.6e+02       9      36 ..    1209    1236 ..    1203    1258 .. 0.83
   4 !   22.4   0.0   5.2e-06    0.0037      13      79 ..    1313    1380 ..    1305    1385 .. 0.81
   5 !   47.3   0.9   8.4e-14     6e-11       1      84 [.    1800    1890 ..    1800    1891 .. 0.91
   6 !   17.9   0.0   0.00012     0.088       5      73 ..    1904    1966 ..    1901    1976 .. 0.91
   7 ?   10.1   0.0     0.035        25       1      85 []    1994    2107 ..    1994    2107 .. 0.87
```

Notice that *almost* everything's the same (it's the same target sequence, after all) *except* for what happens with E-values. The independent E-value is calculated assuming a search space of all 556,825 sequences. For example, look at the highest scoring domain (domain 5 here; domain 7 above). When we only looked at a single sequence, its score of 47.3 bits has an E-value of 1.1e-16. When we search a database of 556,825 sequences, a hit scoring 47.3 bits would be expected to happen 556,825 times as often: 1.1e-16 $\times$ 556,825 = 6.0e-11. In this UniProt search, 784 sequences were reported in the top hits list (with E-values $\leq$ 10). If we were to assume that all 784 are true homologs, x out the domain(s) that made us think that, and then went looking for *additional* domains in those 784 sequences, we'd be searching a smaller database of 784 sequences: the expected number of times we'd see a hit of 47.3 bits or better is now 1.1e-16 $\times$ 784 = 8.4e-14. That's where the conditional E-value (c-Evalue) is coming from.

If you calculate this yourself, you may see some small discrepancies due to floating point roundoff.

Notice that a couple of domains disappeared in the UniProt search, because now, in this larger search space size, they're not significant. Domain 1 (the one with the score of -1.5 bits) got a conditional E-value of 0.18 $\times$ 784 = 141, and domain 6 (with a bit score of 0.6) got a c-Evalue of 0.04 $\times$ 784= 31.4. These fail the default reporting threshold of 10.0. Also, the domains with scores of 5.0 and 10.1 shifted from being above to below the default inclusion thresholds, so now they're marked with ? instead of !.

Operationally:

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

E-value to decide the statistical significance of additional weak-scoring domains.

In the UniProt output, for example, we'd be pretty sure of four of the domains (1, 4, 5, and maybe 6), each of which has a strong enough independent E-value to declare 7LESS_DROME to be an fnIII-domain-containing protein. Domain 2 wouldn't be significant if it was all we saw in the sequence, but once we decide that 7LESS_DROME contains fn3 domains on the basis of the other hits, its conditional E-value indicates that it's probably an fn3 domain too. Domains 3 and 7 (the ones marked by ?) are too weak to be sure of, from this search alone, but would be something to pay attention to.

The next four columns give the endpoints of the reported local alignment with respect to both the query profile (`hmmfrom` and `hmm to`) and the target sequence (`alifrom` and `ali to`).

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

The next two columns (`envfrom` and `env to`) define the *envelope* of the domain's location on the target sequence. The envelope is almost always a little wider than what HMMER chooses to show as a reasonably confident alignment. As mentioned earlier, the envelope represents a subsequence that encompasses most of the posterior probability for a given homologous domain, even if precise endpoints are only fuzzily inferrable.

You'll notice that for higher scoring domains, the coordinates of the envelope and the inferred alignment will tend to be in tighter agreement, corresponding to sharper posterior probability defining the location of the homologous region.

Operationally, I recommend using the envelope coordinates to annotate domain locations on target sequences, not the alignment coordinates. Be aware that when two weaker-scoring domains are close to each other, envelope coordinates (and, rarely, sometimes even alignment coordinates) can and will overlap, corresponding to the overlapping uncertainty of where one domain ends and another begins.

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

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

```
FT   DOMAIN      440    533       Fibronectin type-III 1. {ECO:0000255|PROSITE-ProRule:PRU00316}.
FT   DOMAIN      824    924       Fibronectin type-III 2. {ECO:0000255|PROSITE-ProRule:PRU00316}.
FT   DOMAIN     1202   1290       Fibronectin type-III 3. {ECO:0000255|PROSITE-ProRule:PRU00316}.
```

```
FT   DOMAIN     1294   1397       Fibronectin type-III 4. {ECO:0000255|PROSITE-ProRule:PRU00316}.
FT   DOMAIN     1801   1901       Fibronectin type-III 5. {ECO:0000255|PROSITE-ProRule:PRU00316}.
FT   DOMAIN     1902   1988       Fibronectin type-III 6. {ECO:0000255|PROSITE-ProRule:PRU00316}.
FT   DOMAIN     1995   2117       Fibronectin type-III 7. {ECO:0000255|PROSITE-ProRule:PRU00316}.
```

These agree (modulo differences in start/endpoints) with the seven strongest domains identified by HMMER. The weaker partial domain hits (at 395-410 and 1720-1769) are also plausible homologies, given that the extracellular domain of Sevenless is pretty much just a big array of ∼100aa fibronectin repeats.

Under the domain table, an "optimal posterior accuracy" alignment[11] is computed within each domain's envelope and displayed. For example, (skipping domain 1 because it's weak and unconvincing), fibronectin III domain 2 in your `7LESS_DROME` output is shown as:

[11] I. Holmes. *Studies in Probabilistic Sequence Alignment and Evolution*. PhD thesis, University of Cambridge, 1998

```
== domain 2  score: 40.8 bits;  conditional E-value: 1.2e-14
                 TSBCEEEEEEEESSSEEEEEEE-CSSSSSTECEEEEEEEETTSSSTEEEEEEESTCSEEEEESSSTTEEEEEEEEEEETTEEEE CS
        fn3    1 saPsnlsvsevtstsltvsWeppkdgggpitgYeveyrekgseeewneftvprtttsvtltgLkpgteYevrVqavngggegp 83
                 saP   ++    ++ l v+W p +  +gpi+gY++++++++++  + e+ vp+   s+ +++L++gt+Y++ +  +n++gegp
7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHPGRFTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGEGP 520
                 5777788889999*************************9998.**********************************997 PP
```

The initial header line starts with a `==` as a little handle for a parsing script to grab hold of. I may put more information on that line eventually.

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

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

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

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

The bottom line represents the posterior probability (essentially the expected accuracy) of each aligned residue. A 0 means 0-5%, 1 means 5-15%, and so on; 9 means 85-95%, and a $*$ means 95-100% posterior probability. You can use these posterior probabilities to decide which parts of the alignment are well-determined or not. You'll often observe, for example, that expected alignment accuracy degrades around locations of insertion and deletion, which you'd intuitively expect.

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

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

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

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

```
Internal pipeline statistics summary:
-------------------------------------
Query model(s):                            1  (149 nodes)
Target sequences:                     556825  (199652254 residues searched)
Passed MSV filter:                     21395  (0.0384232); expected 11136.5 (0.02)
Passed bias filter:                    17546  (0.0315108); expected 11136.5 (0.02)
Passed Vit filter:                      2368  (0.00425268); expected 556.8 (0.001)
Passed Fwd filter:                      1127  (0.00202398); expected 5.6 (1e-05)
Initial search space (Z):              556825  [actual number of targets]
Domain search space  (domZ):             1126  [number of targets reported over threshold]
# CPU time: 2.57u 0.09s 00:00:02.65 Elapsed: 00:00:01.10
# Mc/sec: 26954.42
//
[ok]
```

This gives you some idea of what's going on in HMMER's acceleration pipeline. You've got one query profile, and the database has 556,825 target sequences. Each sequence goes through a gauntlet of three scoring algorithms called MSV, Viterbi, and Forward, in order of increasing sensitivity and increasing computational requirement.

MSV (the "Multi ungapped Segment Viterbi" algorithm) essentially calculates the HMMER equivalent of BLAST's sum score – an optimal sum of ungapped high-scoring alignment segments. Unlike BLAST, it does this calculation directly, without BLAST's word hit or hit extension step, using a SIMD vector-parallel algorithm. By default, HMMER is configured to allow sequences with a P-value of ≤ 0.02 through the MSV score filter.[12] Here, for this globin search, about 3.8% of the database got through the MSV filter.

[12] Thus, if the database contained no homologs and P-values were accurately calculated, the highest scoring 2% of the sequences will pass the filter.

A quick check is then done to see if the target sequence is "obviously" so biased in its composition that it's unlikely to be a true homolog. This is called the "bias filter". If you don't like it (it can occasionally be overaggressive) you can shut it off with the `--nobias` option. Here, 17546 sequences pass through the bias filter.

The Viterbi filter then calculates a gapped optimal alignment score. This is more sensitive than the MSV score, but the Viterbi filter is about four-fold slower than MSV. By default, HMMER lets sequences with a P-value of ≤ 0.001 through this stage. Here, because there's about a thousand true globin homologs in this database, more than

that gets through: 2368 sequences.

Then the full Forward score is calculated, which sums over all possible alignments of the profile to the target sequence. The default allows sequences with a P-value of $\leq 10^{-5}$ through: 1127 sequences pass.

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

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

Finally, it reports the speed of the search in units of Mc/sec (million dynamic programming cells per second), the CPU time, and the elapsed time. This search took about 1.1 seconds of elapsed (wall clock) time.

Single sequence protein queries using phmmer

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

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

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

```
% phmmer HBB_HUMAN uniprot_sprot.fasta
```

or

```
% phmmer HBB_HUMAN globins45.fa
```

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

Iterative protein searches using jackhmmer

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

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

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

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

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

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

```
% jackhmmer HBB_HUMAN uniprot_sprot.fasta
```

One difference from `phmmer` output you'll notice is that `jackhmmer` marks "new" sequences with a `+` and "lost" sequences with a `-`. New sequences are sequences that pass the inclusion threshold(s) in this round, but didn't in the round before. Lost sequences are the opposite: sequences that passed the inclusion threshold(s) in the previous round, but have now fallen beneath (yet are still in the reported hits – it's possible, though rare, to lose sequences utterly, if they no longer even pass the reporting threshold(s)). In the first round, everything above the inclusion thresholds is marked with a `+`, and nothing is marked with a `-`. For example, the top of this output looks like:

```
# jackhmmer :: iteratively search a protein sequence against a protein database
# HMMER 3.2 (June 2018); http://hmmer.org/
# Copyright (C) 2018 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query sequence file:             HBB_HUMAN
# target sequence database:        uniprot_sprot.fasta
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

Scores for complete sequences (score includes all domains):
   --- full sequence ---   --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence              Description
    ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
+   3.4e-98  330.5   0.6    3.8e-98  330.3   0.6    1.0  1  sp|P68871|HBB_HUMAN    Hemoglobin subunit beta OS=Homo sapien
+   3.4e-98  330.5   0.6    3.8e-98  330.3   0.6    1.0  1  sp|P68872|HBB_PANPA    Hemoglobin subunit beta OS=Pan paniscu
+   3.4e-98  330.5   0.6    3.8e-98  330.3   0.6    1.0  1  sp|P68873|HBB_PANTR    Hemoglobin subunit beta OS=Pan troglod
```

```
+   9.8e-98  329.0   0.7    1.1e-97  328.8   0.7    1.0  1  sp|P02024|HBB_GORGO    Hemoglobin subunit beta OS=Gorilla gor
+     3e-96  324.2   0.5    3.3e-96  324.0   0.5    1.0  1  sp|P02025|HBB_HYLLA    Hemoglobin subunit beta OS=Hylobates l
+     3e-95  320.9   0.6    3.3e-95  320.8   0.6    1.0  1  sp|P02032|HBB_SEMEN    Hemoglobin subunit beta OS=Semnopithec
...
```

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

```
+   0.00049   25.0   0.2    0.00057   24.8   0.2    1.0  1  sp|Q0KIY5|MYG_KOGBR    Myoglobin OS=Kogia breviceps OX=27615
+   0.00062   24.6   0.0    0.00073   24.4   0.0    1.0  1  sp|P14399|MYG_MUSAN    Myoglobin OS=Mustelus antarcticus OX=7
  ------ inclusion threshold ------
     0.0011   23.9   0.3      0.011   20.5   0.3    2.0  1  sp|P81044|HBAZ_MACEU   Hemoglobin subunit zeta (Fragments) OS
     0.0014   23.5   0.0     0.0018   23.2   0.0    1.1  1  sp|O80405|LGB3_PEA     Leghemoglobin Lb120-1 OS=Pisum sativum
```

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

```
@@ New targets included:   953
@@ New alignment includes: 954 subseqs (was 1), including original query
@@ Continuing to next round.

@@
@@ Round:                  2
@@ Included in MSA:        954 subsequences (query + 953 subseqs from 953 targets)
@@ Model size:             146 positions
@@
```

This (obviously) is telling you that the new alignment contains 954 sequences, your query plus 954 significant matches. For round two, it's built a new profile from this alignment. Now for round two, it fires off what's essentially an `hmmsearch` of the target database with this new profile:

```
Scores for complete sequences (score includes all domains):
   --- full sequence ---    --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Sequence              Description
    ------- ------ -----    ------- ------ -----   ---- --  --------              -----------
    4.7e-68  232.8   0.2    5.3e-68  232.6   0.2    1.0  1  sp|P02055|HBB_MELME    Hemoglobin subunit beta OS=Meles meles
    7.1e-68  232.2   0.4    7.9e-68  232.1   0.4    1.0  1  sp|P81042|HBE_MACEU    Hemoglobin subunit epsilon OS=Macropus
      9e-68  231.9   0.3      1e-67  231.7   0.3    1.0  1  sp|P15449|HBB_MELCA    Hemoglobin subunit beta OS=Mellivora c
    1.3e-67  231.4   0.2    1.4e-67  231.3   0.2    1.0  1  sp|P68046|HBB_ODORO    Hemoglobin subunit beta OS=Odobenus ro
...
```

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

```
...
    3.2e-30  110.1   0.0    3.5e-30  110.0   0.0    1.0  1  sp|Q9DEP1|MYG_PSEGE    Myoglobin OS=Pseudochaenichthys georgi
    4.2e-30  109.7   0.3    5.7e-30  109.3   0.3    1.2  1  sp|P21199|GLB3_MORMR   Globin-3 OS=Mordacia mordax OX=7755 PE
+   9.9e-30  108.5   0.2    1.1e-29  108.3   0.2    1.0  1  sp|P14397|MYG_GALGA    Myoglobin OS=Galeorhinus galeus OX=860
    1.1e-29  108.4   0.0    1.2e-29  108.2   0.0    1.0  1  sp|Q9DEP0|MYG_CRYAN    Myoglobin OS=Cryodraco antarcticus OX=
    1.6e-29  107.8   0.0    1.8e-29  107.7   0.0    1.0  1  sp|P02022|HBAM_LITCT   Hemoglobin heart muscle subunit alpha-
+   3.4e-29  106.8   0.1    3.8e-29  106.6   0.1    1.0  1  sp|P14398|MYG_HEMJP    Myoglobin OS=Hemitriakis japanica OX=3
    1.6e-28  104.6   0.0    1.9e-28  104.4   0.0    1.0  1  sp|P09106|HBAT_PAPAN   Hemoglobin subunit theta-1 OS=Papio an
    3.2e-28  103.6   0.0    3.9e-28  103.3   0.0    1.0  1  sp|P0C227|GLB_NERAL    Globin OS=Nerita albicilla OX=52928 PE
    3.9e-28  103.3   0.3    5.2e-28  102.9   0.3    1.0  1  sp|P80017|GLBD_MOLAR   Globin D, coelomic OS=Molpadia arenico
    5.9e-26   96.2   0.6      1e-25   95.5   0.6    1.4  1  sp|C0HJZ2|HBA_SOMMI    Hemoglobin subunit alpha (Fragments) O
    2.8e-25   94.1   0.2      3e-25   93.9   0.2    1.0  1  sp|P18979|HBA1_SAAHA   Hemoglobin subunit alpha-1 (Fragment)
+   3.6e-24   90.4   0.0    4.3e-24   90.2   0.0    1.0  1  sp|Q90W04|NGB_TETNG    Neuroglobin OS=Tetraodon nigroviridis
    6.9e-24   89.5   0.0    8.2e-24   89.3   0.0    1.0  1  sp|P59742|NGB1_ONCMY   Neuroglobin-1 OS=Oncorhynchus mykiss O
...
```

It's less usual to see sequences get lost (and marked with `-`), but it happens.

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

```
@@ New targets included:   172
@@ New alignment includes: 1126 subseqs (was 954), including original query
@@ Continuing to next round.

@@
@@ Round:                  3
@@ Included in MSA:        1126 subsequences (query + 1125 subseqs from 1125 targets)
@@ Model size:             146 positions
@@
```

Because new sequences were included, it keeps going to round three, and then again to round four and five. After round five, the search ends quietly because there's a default maximum of five iterations, and you get:

```
@@ New targets included:   2
@@ New alignment includes: 1168 subseqs (was 1167), including original query
//
[ok]
```

The `//` marks the end of the results for one query. You could search with more than one query in your input query sequence file.

There is an `[ok]` at the end of the search output as a signal that the search successfully completed. This might be useful if you're automating lots of searches and you want to be sure that they worked.

Searching a profile database with a query sequence

Rather than searching a single profile against a collection of sequences, you might want to wish to annotate a single sequence by searching it against a collection of profiles of different domains. `hmmscan` takes as input a query file containing one or more sequences to annotate, and a profile database to search them against. The profile database might be Pfam, SMART, or TIGRFams, for example, or another collection of your choice.

Either `hmmsearch` or `hmmscan` can compare a set of profiles to a set of sequences. Due to disk access patterns of the two tools, it is usually more efficient to use `hmmsearch`, unless the number of profiles greatly exceeds the number of sequences.

Step 1: create a profile database file

A profile "database" file is just a concatenation of individual profile files. To create a database file, you can either build individual profile files and concatenate them, or you can concatenate Stockholm alignments and use `hmmbuild` to build a profile database from them in one command.

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

```
% hmmbuild globins4.hmm globins4.sto
```

```
% hmmbuild fn3.hmm fn3.sto
% hmmbuild Pkinase.hmm Pkinase.sto
% cat globins4.hmm fn3.hmm Pkinase.hmm > minifam
```

We'll use `minifam` for our examples in just a bit, but first a few words on other ways to build profile databases, especially big ones.

The other way to do it is to start with an *alignment* database flatfile – a concatenation of many Stockholm files – and use `hmmbuild` to build a profile database file from it. For example, you could obtain the big `Pfam-A.seed` and/or `Pfam-A.full` Stockholm-format alignment flatfiles from Pfam. `hmmbuild` names each profile according to a `#=GF ID` annotation line in each Stockholm alignment. Normally the `ID` line is optional in Stockholm format, but `hmmbuild` has to name your new profile(s) somehow. For a single alignment, it will use your filename, or you can use the `hmmbuild -n <name>` option to provide a name yourself. For an alignment database, the only way `hmmbuild` can get a name for each alignment is from alignment annotation. Of alignment file formats, only Stockholm format provides a way to concatenate many alignments in the same file, with a name for each alignment. For example, from a Pfam seed alignment flatfile `Pfam-A.seed`, you can do:

For example, it won't work if you concatenate `globins4.sto` with other Stockholm files, because the simple little `globins4.sto` alignment doesn't have an `ID` line.

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

This would probably take a couple of hours to build all 20,000 profiles or so in Pfam. To speed the database construction process up, `hmmbuild` supports MPI parallelization. Running MPI programs can be a little arcane, so skip this bit if you're not in the mood.

As far as HMMER's concerned, all you have to do is add `--mpi` to the command line for `hmmbuild` to tell it to run in MPI master/worker mode across many cores and/or machines, assuming you've compiled support for MPI into it (see the installation instructions). You'll also need to know how to invoke an MPI job in your particular cluster environment, with your job scheduler and your MPI distribution.[13] In general, you will launch the parallel `hmmbuild` by using a command like `mpirun` or `srun` that manages the MPI environment for a specified number of processes. With the SGE (Sun Grid Engine) scheduler and Intel MPI, an example incantation for building `Pfam.hmm` from `Pfam-A.seed` in parallel across 128 processes:

[13] I can't really help you with this. Different sites have different cluster, scheduler, and MPI environments. Consult a local guru, as they say.

```
% qsub -N hmmbuild -j y -o errors.out -b y -cwd -V -pe impi 128 \
  'mpirun -np 128 ./hmmbuild -mpi Pfam.hmm Pfam-A.seed > hmmbuild.out'
```

or, an example SLURM incantation (on the `eddy` group partition on our Harvard cluster):

```
% sbatch -J hmmbuild -e hmmbuild.err -o hmmbuild.out -p eddy -n 128 -t 6-00:00 -mem-per-cpu=4000 \
   -wrap="srun -n 128 -mpi=pmi2 ./hmmbuild -mpi Pfam-A.hmm Pfam-A.seed"
```

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

Step 2: compress and index the flatfile with hmmpress

`hmmscan` has to read a lot of profiles in a hurry, and HMMER's text flatfiles are bulky. To accelerate this, `hmmscan` depends on binary compression and indexing of the flatfiles. First you compress and index your profile database with the `hmmpress` program:

```
% hmmpress minifam
```

This will produce:

```
Working...    done.
Pressed and indexed 3 HMMs (3 names and 2 accessions).
Models pressed into binary file:   minifam.h3m
SSI index for binary model file:   minifam.h3i
Profiles (MSV part) pressed into:  minifam.h3f
Profiles (remainder) pressed into: minifam.h3p
```

and you'll see these four new binary files in the directory.[14]

[14] Their format is "proprietary", an open source term of art that means both "I haven't found time to document them yet" and "I still might decide to change them arbitrarily without telling you".

Step 3: search the profile database with hmmscan

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

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

```
% hmmscan minifam 7LESS_DROME
```

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

```
# hmmscan :: search sequence(s) against a profile database
# HMMER 3.2 (June 2018); http://hmmer.org/
# Copyright (C) 2018 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# query sequence file:             7LESS_DROME
# target HMM database:             minifam
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Query:       7LESS_DROME  [L=2554]
Accession:   P13368
Description: RecName: Full=Protein sevenless;          EC=2.7.10.1;
Scores for complete sequence (score includes all domains):
   --- full sequence ---   --- best 1 domain ---    -#dom-
    E-value  score  bias    E-value  score  bias    exp  N  Model    Description
    ------- ------ -----    ------- ------ -----   ---- --  -------- -----------
    1.9e-55  173.1   1.6    3.2e-16   47.3   0.9    9.5  9  fn3       Fibronectin type III domain
    1.2e-40  127.2   0.0      2e-40  126.5   0.0    1.2  1  Pkinase   Protein kinase domain
```

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

The size of the search space for `hmmscan` is the number of profiles in the profile database (here, 3; for a Pfam search, on the order of

20000). In `hmmsearch`, the size of the search space is the number of sequences in the sequence database. This means that E-values may differ even for the same individual profile vs. sequence comparison, depending on how you do the search.

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

```
Domain annotation for each model (and alignments):
>> fn3  Fibronectin type III domain
   #    score  bias  c-Evalue  i-Evalue hmmfrom  hmm to    alifrom  ali to    envfrom  env to     acc
 ---   ------ ----- --------- --------- ------- -------    ------- -------    ------- -------    ----
   1 ?   -1.5   0.0      0.37      0.55      60      72 ..     396     408 ..     395     410 .. 0.86
   2 !   40.8   0.0   2.3e-14   3.5e-14       1      83 [.     439     520 ..     439     521 .. 0.95
   3 !   14.8   0.0   3.1e-06   4.6e-06      12      84 ..     836     913 ..     826     914 .. 0.74
   4 !    5.0   0.0    0.0035    0.0052       9      36 ..    1209    1236 ..    1203    1258 .. 0.83
   5 !   22.4   0.0   1.3e-08     2e-08      13      79 ..    1313    1380 ..    1305    1385 .. 0.81
   6 ?    0.6   0.0     0.079      0.12      55      72 ..    1753    1769 ..    1720    1769 .. 0.87
   7 !   47.3   0.9   2.2e-16   3.2e-16       1      84 [.    1800    1890 ..    1800    1891 .. 0.91
   8 !   17.9   0.0   3.2e-07   4.8e-07       5      73 ..    1904    1966 ..    1901    1976 .. 0.91
   9 !   10.1   0.0     9e-05   0.00014       1      85 []    1994    2107 ..    1994    2107 .. 0.87
```

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

```
  == domain 2  score: 40.8 bits;  conditional E-value: 2.3e-14
                  TSBCEEEEEEEESSSEEEEEE-CSSSSSTECEEEEEEEETTSSSTEEEEEEESTCSEEEEESSSTTEEEEEEEEEEETTEEEE CS
          fn3   1 saPsnlsvsevtstsltvsWeppkdgggpitgYeveyrekgseeewneftvprtttsvtltgLkpgteYevrVqavngggegp 83
                  saP   ++    ++ l v+W p +  +gpi+gY++++++++++  + e+ vp+   s+ +++L++gt+Y++ +  +n++gegp
  7LESS_DROME 439 SAPVIEHLMGLDDSHLAVHWHPGRFTNGPIEGYRLRLSSSEGNA-TSEQLVPAGRGSYIFSQLQAGTNYTLALSMINKQGEGP 520
                  5777788889999**************************9998.**********************************997 PP
```

You'd probably expect that except for the E-values (which depend on database search space sizes), you should get exactly the same scores, domain number, domain coordinates, and alignment every time you do a comparison of the same profile against the same sequence. Which is actually the case! But in fact, under the hood, it's actually not so obvious this should be, and HMMER is actually going out of its way to make it so. HMMER uses stochastic sampling algorithms to infer some parameters, and also to infer the exact domain number and domain boundaries in certain difficult cases. If HMMER ran its stochastic samples "properly", it would obtain different samples every time you ran a program, and all of you would complain to me that HMMER was weird and buggy because it gave different answers on the same problem. To suppress run-to-run variation, HMMER seeds its random number generator(s) *identically* every time you do a sequence comparison. If you're a stats expert, and you really want to see the proper stochastic variation that results from sampling algorithms, you can pass a command-line argument of `--seed 0` to programs that have this property (hmmbuild and the four search programs).

Creating multiple alignments with hmmalign

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

```
% hmmalign globins4.hmm globins45.fa
```

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

```
# STOCKHOLM 1.0

MYG_ESCGI          .-VLSDAEWQLVLNIWAKVEADVAGHGQDILIRLFKGHPETLEKFDKFKHLKTEAEMKASE ...
#=GR MYG_ESCGI  PP ..69*********************************************************** ...
MYG_HORSE          g--LSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASE ...
#=GR MYG_HORSE  PP 8..89********************************************************** ...
MYG_PROGU          g--LSDGEWQLVLNVWGKVEGDLSGHGQEVLIRLFKGHPETLEKFDKFKHLKAEDEMRASE ...
#=GR MYG_PROGU  PP 8..89********************************************************** ...
MYG_SAISC          g--LSDGEWQLVLNIWGKVEADIPSHGQEVLISLFKGHPETLEKFDKFKHLKSEDEMKASE ...
#=GR MYG_SAISC  PP 8..89********************************************************** ...
MYG_LYCPI          g--LSDGEWQIVLNIWGKVETDLAGHGQEVLIRLFKNHPETLDKFDKFKHLKTEDEMKGSE ...
#=GR MYG_LYCPI  PP 8..89********************************************************** ...
MYG_MOUSE          g--LSDGEWQLVLNVWGKVEADLAGHGQEVLIGLFKTHPETLDKFDKFKNLKSEEDMKGSE ...
...
```

and so on. (I've truncated long lines.)

First thing to notice here is that `hmmalign` uses both lower case and upper case residues, and it uses two different characters for gaps. This is because there are two different kinds of columns: "match" columns in which residues are assigned to match states and gaps are treated as deletions relative to consensus, and "insert" columns where residues are assigned to insert states and gaps in other sequences are just padding for the alignment to accomodate those insertions. In a match column, residues are upper case, and a '-' character means a deletion relative to the consensus. In an insert column, residues are lower case, and a '.' is padding. A '-' deletion has a cost: transition probabilities were assessed, penalizing the transition into and out of a deletion. A '.' pad has no cost per se; instead, the sequence(s) with insertions are paying transition probabilities into and out of their inserted residue.

This notation is only for your convenience in output files. You can see the structure of the profile reflected in the pattern of residues and gap characters . In input files, in most alignment formats[15] HMMER is case-insensitive, and it does not distinguish between different gap characters: '-' (dash), '.' (period), or even '_' (underscore) are accepted as gap characters.

Important: insertions relative to a profile are *unaligned*. Suppose one sequence has an insertion of length 10 and another has an insertion of length 2 in the same place in the profile. The alignment will have ten insert columns, to accomodate the longest insertion. The residues of the shorter insertion are thrown down in an arbitrary order. Notice that in the previous paragraph I oh-so-carefully

By default, `hmmalign` removes any columns that are all deletion characters, so the number of apparent match columns in a displayed alignment is $\leq$ the actual number of match states in the profile. To prevent this trimming and see columns for all match states, use the `--allcol` option. This can be helpful if you're writing a postprocessor that's trying to keep track of what columns are assigned to what match states in the profile.

[15] A2M format is an important exception!

By arbitrary HMMER convention, the insertion is divided in half; half is left-justified, and the other half is right-justified, leaving '.' characters in the middle.

said residues are "assigned" to a state, not "aligned" to a state. For match states, assigned and aligned are the same thing: a one-to-one correspondence between a residue and a consensus match state in the profile. But there may be one *or more* residues assigned to the same insert state.

Don't be confused by the unaligned nature of profile insertions. You're sure to see cases where lower-case inserted residues are "obviously misaligned". This is just because HMMER isn't trying to "align" them in the first place. It's assigning them to unaligned insertions.

Enough about the sequences in the alignment. Now, notice all those `PP` annotation lines. That's posterior probability annotation, as in the single sequence alignments that `hmmscan` and `hmmsearch` showed. This represents the confidence that each residue is assigned where it should be.

Again, that's "assigned", not "aligned". The posterior probability assigned to an inserted residue is the probability that it is assigned to the insert state corresponding to that column. Because the same insert state might correspond to more than one column, the probability on an insert residue is *not* the probability that it belongs in that particular column; again, if there's a choice of column for putting an inserted residue, that choice is arbitrary.

`hmmalign` currently has a, um, feature that you may dislike. Recall that HMMER only does local alignments. Here, we know that we've provided full length globin sequences, and `globins4` is a full length globin profile. We'd probably like `hmmalign` to produce a global alignment. It can't currently do that. If it doesn't quite manage to extend its local alignment to the full length of a target globin sequence, you'll get a weird-looking effect, as the nonmatching termini are pulled out to the left or right. For example, look at the N-terminal `g` in `MYG_HORSE` above. HMMER is about 80% confident that this residue is nonhomologous, though any sensible person would align it into the first globin consensus column.

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

```
# STOCKHOLM 1.0

MYG_ESCGI          .-VLSDAEWQLVLNIWAKVEADVAGHGQDILIRLFKGHPETLEKFDKFKHLKTEAEMKASE ...
#=GR MYG_ESCGI  PP ..69********************************************************* ...
MYG_HORSE          g--LSDGEWQQVLNVWGKVEADIAGHGQEVLIRLFTGHPETLEKFDKFKHLKTEAEMKASE ...
#=GR MYG_HORSE  PP 8..89******************************************************** ...
MYG_PROGU          g--LSDGEWQLVLNVWGKVEGDLSGHGQEVLIRLFKGHPETLEKFDKFKHLKAEDEMRASE ...
#=GR MYG_PROGU  PP 8..89******************************************************** ...
MYG_SAISC          g--LSDGEWQLVLNIWGKVEADIPSHGQEVLISLFKGHPETLEKFDKFKHLKSEDEMKASE ...
#=GR MYG_SAISC  PP 8..89******************************************************** ...
MYG_LYCPI          g--LSDGEWQIVLNIWGKVETDLAGHGQEVLIRLFKNHPETLDKFDKFKHLKTEDEMKGSE ...
#=GR MYG_LYCPI  PP 8..89******************************************************** ...
MYG_MOUSE          g--LSDGEWQLVLNVWGKVEADLAGHGQEVLIGLFKTHPETLDKFDKFKNLKSEEDMKGSE ...
...
```

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

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

Searching DNA sequences

HMMER was originally developed for protein sequence analysis. The `hmmsearch` and `hmmscan` programs assume that it's sensible to ask if the entire target sequence is homologous (or not) to a query profile. It makes sense to say "this sequence is a probable protein kinase" because we find a protein kinase domain in it. What if you want to use a DNA profile to search a very long (chromosome-sized) piece of DNA for homologous regions? We might want to identify Alu and L1 elements in human chromosome sequences, for example. It's not super useful to see the 24 chromosomes ranked by E-values in `hmmsearch` output – we're only interested in the element locations. Also, if we can avoid having to align the entire target chromosome sequence at once, we can scan the profile along the target sequence in a much more memory-efficient manner than `hmmsearch`/`hmmscan` would do.

The `nhmmer` and `nhmmscan` programs are designed for memory-efficient DNA profile searches of long DNA sequences. They were developed in concert with the Dfam database (dfam.org), which provides of alignments and profiles of many common DNA repeat elements for several important genomes. The alignment `tutorial/MADE1.sto` is a representative alignment of 100 human MADE1 transposable elements, a subset of the Dfam MADE1 alignment. We'll use the MADE1 alignment to show how `nhmmer`/`nhmmscan` work, which are similar to `hmmsearch`/`hmmscan`.

Step 1: build a profile with hmmbuild

`hmmbuild` works for both protein and DNA profiles, so:

```
% hmmbuild MADE1.hmm MADE1.sto
```

and you'll see some output that looks like:

```
# hmmbuild :: profile HMM construction from multiple sequence alignments
# HMMER 3.2 (June 2018); http://hmmer.org/
```

```
# Copyright (C) 2018 Howard Hughes Medical Institute.
# Freely distributed under the BSD open source license.
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# input alignment file:             MADE1.sto
# output HMM file:                  MADE1.hmm
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

# idx name                 nseq  alen  mlen     W eff_nseq re/pos description
#---- -------------------- ----- ----- ----- ----- -------- ------ -----------
1     MADE1                 100   304    80   473     3.55  0.709 MADE1 (MAriner Derived Element 1), a TcMar-Mariner ...

# CPU time: 0.01u 0.00s 00:00:00.01 Elapsed: 00:00:00.02
```

Notice the new output column with the header "W", which is only present when the input sequence alignment is DNA/RNA. This represents an upper bound on the length at which nhmmer expects to find an instance of the family. It is always larger than mlen, though the ratio of mlen to W depends on the observed insert rate in the seed alignment. This length is used deep in the acceleration pipeline, and modest changes are not expected to impact results, but larger values of W do lead to longer run time. The value can be overridden with the `--w_length` or `--w_beta` flags, at the risk of possibly missing instances of the family that happen to be longer than W due to plentiful insertions.

W is based on position-specific insert rates: only 10^{-7} of all sequences generated from the profile are expected to be longer than W.

Step 2: search the DNA sequence database with nhmmer

We'll use `dna_target.fa` as the target sequence database. It is a FASTA format file containing one 330Kb long DNA sequence extracted from human chromosome 1.

`nhmmer` accepts a target DNA sequence database in the same formats as hmmsearch (typically FASTA). For the query, it accepts either a profile file as produced above by hmmbuild, or a file containing either one DNA sequence or an alignment of multiple DNA sequences.

If a sequence or alignment is used as query input, `nhmmer` internally produces the profile for that alignment, then searches with that profile. The profile produced in this way is automatically saved to disk, and the default file name is chosen by appending ".hmm" to the name of the sequence file name. This can be overridden with the `--hmmout` flag.

Using default hmmbuild parameters; if you want more control, explicitly built the profile with hmmbuild.

To search `dna_target.fa` with our `MADE1.hmm` profile:

```
% nhmmer MADE1.hmm dna_target.fa
```

This output is largely similar to that of hmmsearch. The key difference is that each hit is not to a full sequence in the target database, but one local alignment of the profile to a subsequence of a target database sequence.

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

The second section is the *top hits* list. It is a list of ranked top hits

(sorted by E-value, most significant hit first), formatted much like the `hmmsearch` output *top hits* list:

```
  E-value  score  bias  Sequence        start    end  Description
  ------- ------ -----  --------        -----  -----  -----------
  1.3e-10   38.3   7.0  humanchr1_frag 302390 302466
  6.4e-08   29.7   8.2  humanchr1_frag 174456 174498
  8.6e-08   29.3   6.2  humanchr1_frag 302466 302389
  3.8e-06   24.0   7.0  humanchr1_frag 174493 174456
------ inclusion threshold ------
     0.69    7.1   7.8  humanchr1_frag 304073 304103
```

For each hit, the table shows its *E-value*, *bit score*, *bias*, *target sequence name* and *target sequence description*, much like `hmmsearch`.

The "start" and "end" columns are the coordinates in the target sequence where the hit is found. When the "end" is smaller than "start", this means the hit found on the reverse complement of the target database sequence.

`nhmmer` automatically searches both strands.

For example, note that the top hits here are coming in overlapping pairs, corresponding to the forward and reverse strands, like the hit to 302390..302466on the forward strand and a hit to 302466..302389 on the reverse strand. This is because the MADE1 DNA element is a near-perfect palindrome.

DNA elements that have a unique orientation will only hit on one strand. `nhmmer` treats the two strands independently. Palindromic elements will hit the same region on both strands and `nhmmer` will not filter the overlapping hits.

Then comes the third output section, which starts with

```
Annotation for each hit  (and alignments):
```

For each hit in the top hits list, there is a tabular line providing detailed information about the hit, followed by the alignment inferred for the hit. The first `MADE1` hit looks like:

```
>> humanchr1_frag
    score  bias    Evalue   hmmfrom    hmm to     alifrom    ali to      envfrom    env to       sq len      acc
    ------ ----- ---------   -------   -------    --------- ---------    --------- ---------    ---------    ----
 !   38.3   7.0   1.3e-10         4        80 .]    302390    302466 ..    302387    302466 ..    330000    0.90
```

All these pieces of information are as described for `hmmsearch`, plus a column for "sq len" that indicates the full length of the target sequence.

Under each one-line hit table is displayed the alignment inferred between the profile and the hit envelope. For example, the top hit from above is shown as:

```
  Alignment:
  score: 38.3 bits
                       xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx....xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx RF
           MADE1      4 ggttggtgcaaaagtaattgcggtttttgccattacttttaatggc....aaaaaccgcaattacttttgcaccaacctaa 80
                        ggt ggtgcaaaa  aattg ggtttttgccatt cttttaat gc    aaaa      a t ctttt caccaa ctaa
  humanchr1_frag 302390 GGTCGGTGCAAAATCAATTGTGGTTTTTGCCATTGCTTTTAATTGCttttAAAAG----TAATGCTTTTACACCAATCTAA 302466
                        89*****************************************9777666663....355789************996 PP
```

The alignment format is the same as for `hmmsearch`.

At the end of the output, you'll see summary statistics:

```
Internal pipeline statistics summary:
-------------------------------------
```

```
Query model(s):                            1  (80 nodes)
Target sequences:                          1  (660000 residues searched)
Residues passing SSV filter:           69202  (0.105); expected (0.02)
Residues passing bias filter:          57239  (0.0867); expected (0.02)
Residues passing Vit filter:            5181  (0.00785); expected (0.003)
Residues passing Fwd filter:            3187  (0.00483); expected (3e-05)
Total number of hits:                      6  (0.000498)
# CPU time: 0.03u 0.00s 00:00:00.03 Elapsed: 00:00:00.01
# Mc/sec: 3017.39
//
[ok]
```

This gives you some idea of what's going on in nhmmer's acceleration pipeline. You've got one query profile, and 660000 residues were searched (there are 330000 bases in the single sequence found in the file; the search includes the reverse complement, doubling the search space). The sequences in the database go through a gauntlet of three scoring algorithms called SSV, Viterbi, and Forward, in order of increasing sensitivity and increasing computational requirement.

SSV (the "Single ungapped Segment Viterbi" algorithm) as used in nhmmer is closely related to the MSV algorithm used in `hmmsearch`, in that it depends on ungapped alignment segments. The difference lies in how those alignments are used. Using MSV, a sequence is either rejected or accepted in its entirety. In the scanning-SSV filter of `nhmmer`, each sequence in the database is scanned for high-scoring ungapped alignment segments, and a window around each such segment is extracted (merging overlapping windows), and passed on to the next stage. By default, nhmmer is configured to allow sequence segments with a P-value of ≤ 0.02 through the SSV score filter.[16] Here, 69202 bases, or about 10.5% of the database, got through the SSV filter.

[16] Thus, if the database contained no homologs and P-values were accurately calculated, the highest scoring 2% of the sequence will pass the filter.

The "bias filter" is then applied, as in `hmmsearch`. Here, 57239 bases, roughly 8.7% of the database pass through the bias filter.

The Viterbi filter then calculates a gapped optimal alignment score for each window that survived the earlier stages. This score is a closer approximation than the SSV score of the final score that the window will achieve if it survives to final processing, but the Viterbi filter is about four-fold slower than SSV. By default, nhmmer lets windows with a P-value of ≤ 0.001 through this stage. Here, 5181 bases, about 0.8% of the database gets through.

Then the full Forward score is calculated, which sums over all possible alignments of the profile to the window. The default allows windows with a P-value of $\leq 10^{-5}$ through; 3187 bases passed.

All windows that make it through these filters are then subjected to a full probabilistic analysis using the HMM Forward/Backward algorithms, to identify hit envelopes, then determine posterior probabilities for each aligned residue, followed by optimal accuracy alignment. The results of this step are what you finally see on the output. The final number of hits and fractional coverage of the database is shown next. This is typically smaller than the fraction of the database

passing the Forward filter, as hit identification typically trims windows down to a smaller envelope.

Finally, nhmmer reports the speed of the search in units of Mc/sec (million dynamic programming cells per second), the CPU time, and the elapsed time.

`nhmmscan` is to `hmmscan` as `nhmmer` is to `hmmsearch`. There is not currently a iterative DNA search analog to `jackhmmer`.

The HMMER profile/sequence comparison pipeline

Now I'll briefly outline the processing pipeline for a single profile/sequence comparison. This should help give you a sense of what HMMER is doing under the hood, what sort of mistakes it may make (rarely, of course!), and what the various results in the output actually mean. I'll first describe the pipeline in the context of protein search (`phmmer`, `hmmsearch`, `hmmscan`, `jackhmmer`), then wrap back around to discuss the modified pipeline used in `nhmmer` and `nhmmscan`.

Code gurus, masochists: you can follow along in `src/p7_pipeline.c`.

In briefest outline, the comparison pipeline takes the following steps:

Null model. Calculate a score term for the "null hypothesis" (a probability model of *non*-homology). This score correction is used to turn all subsequent profile/sequence bit scores into a final log-odds bit score.

MSV filter. The main acceleration heuristic. The MSV ("Multiple Segment Viterbi") algorithm looks for one or more high-scoring *ungapped* alignments. If the MSV score passes a set threshold, the entire sequence passes on to the next pipeline step; else it is rejected.

Bias filter. A hack that reduces false positive MSV hits due to biased composition sequences. A two-state HMM is constructed from the mean residue composition of the profile and the standard residue composition of the null model, and used to score the sequence. The MSV bit score is corrected using this as a second null hypothesis. If the MSV score still passes the MSV threshold, the sequence passes on to the next step; else it is rejected. The bias filter score correction will also be applied to the Viterbi filter and Forward filter scores that follow.

Viterbi filter. A more stringent accelerated filter. An optimal (maximum likelihood) gapped alignment score is calculated. If this score passes a set threshold, the sequence passes to the next step; else it is rejected.

Forward filter/parser. The full likelihood of the profile/sequence comparison is evaluated, summed over the entire alignment ensemble, using the HMM Forward algorithm. This score is corrected to a bit score using the null model and bias filter scores. If the bit score passes a set threshold, the sequence passes on to the next step; else it is rejected.

Domain identification. Using the Forward parser results, now combined with a Backward parser, posterior probabilities of domain locations are calculated. A discrete set of putative domains (alignments) is identified by applying heuristics to posterior probabilities. This procedure identifies *envelopes*: subsequences on the target sequence which contain a lot of probability mass for a match to the profile.

Alignment. For each identified domain, a full Forward/Backward algorithm is performed. An *ad hoc* "null2" hypothesis is constructed for each domain's composition and used to calculate a biased composition score correction. A maximum expected accuracy (MEA) alignment is calculated. This identifies one MEA alignment within each envelope.

Storage. Now we have a *sequence score* (and P-value); the sequence contains one or more domains, each of which has a *domain score* (and P-value), and each domain has an MEA alignment annotated with per-residue posterior probabilities.

In more detail, each step is described below.

Null model

The "null model" calculates the probability that the target sequence is *not* homologous to the query profile. A HMMER bit score is the log of the ratio of the sequence's probability according to the profile (the homology hypothesis) over the null model probability (the non-homology hypothesis).

The null model is a one-state HMM configured to generate "random" sequences of the same mean length L as the target sequence, with each residue drawn from a background frequency distribution (a standard i.i.d. model: residues are treated as independent and identically distributed). Currently, this background frequency distribution is hardcoded as the mean residue frequencies in Swiss-Prot 50.8 (October 2006).

For technical reasons, HMMER incorporates the *residue emission* probabilities of the null model directly into the profile, by turning each emission probability in the profile into an odds ratio. The null

model score calculation therefore is only concerned with accounting for the remaining *transition* probabilities of the null model and toting them up into a bit score correction. The null model calculation is fast, because it only depends on the length of the target sequence, not its sequence.

MSV filter

The sequence is aligned to the profile using a specialized model that allows multiple high-scoring local ungapped segments to match. The optimal alignment score (Viterbi score) is calculated under this multisegment model, hence the term MSV, for "multi-segment Viterbi". This is HMMER's main speed heuristic.

The MSV score is comparable to BLAST's sum score (optimal sum of ungapped alignment segments). Roughly speaking, MSV is comparable to skipping the heuristic word hit and hit extension steps of the BLAST acceleration algorithm.

The MSV filter is very, very fast. In addition to avoiding indel calculations in the dynamic programming table, it uses reduced precision scores scaled to 8-bit integers, enabling acceleration via 16-way parallel SIMD vector instructions.

The MSV score is a true log-odds likelihood ratio, so it obeys conjectures about the expected score distribution[1] that allow immediate and accurate calculation of the statistical significance (P-value) of the MSV bit score.

[1] S. R. Eddy. A probabilistic model of local sequence alignment that simplifies statistical significance estimation. *PLOS Comput. Biol.*, 4:e1000069, 2008

By default, comparisons with a P-value of ≤ 0.02 pass this filter, meaning that about 2% of nonhomologous sequences are expected to pass. You can use the `--F1 <x>` option to change this threshold. For example, `--F1 0.05` would pass 5% of the comparisons, making a search more sensitive but slower. Setting the threshold to ≥ 1.0 (`--F1 99` for example) assures that all comparisons will pass. Shutting off the MSV filter may be worthwhile if you want to make sure you don't miss comparisons that have a lot of scattered insertions and deletions. Alternatively, the `--max` option causes the MSV filter step (and all other filter steps) to be bypassed.

The MSV bit score is calculated as a log-odds score using the null model for comparison. No correction for a biased composition or repetitive sequence is done at this stage. For comparisons involving biased sequences and/or profiles, more than 2% of comparisons will pass the MSV filter. At the end of search output, there is a line like:

```
Passed MSV filter:                 107917  (0.020272); expected 106468.8 (0.02)
```

which tells you how many and what fraction of comparisons passed the MSV filter, versus how many (and what fraction) were

expected.

Biased composition filter

It's possible for profiles and/or sequences to have biased residue compositions that result in "significant" log-odds bit scores not because the sequence matches the profile well, but because the sequence matches the null model badly.

HMMER uses fairly good methods to compensate its scores for biased composition, but these methods are computationally expensive and applied late in the pipeline (described below).

In a few cases, profiles and/or target sequences are sufficiently biased that too many comparisons pass the MSV filter, causing HMMER speed performance to be severely degraded. Although the final scores and E-values at the end of the pipeline will be calculated taking into account a "null2" model of biased composition and simple repetition, the null2 model is dependent on a full alignment ensemble calculation via the Forward/Backward algorithm, making it computationally complex, so it won't get calculated until the very end. The treatment of biased composition comparisons is an inadequately solved problem in HMMER. As a stopgap solution to rescuing most of the speed degradation while not sacrificing too much sensitivity, an *ad hoc* biased composition filtering step is applied to remove highly biased comparisons.

On the fly, a two-state HMM is constructed. One state emits residues from the background frequency distribution (same as the null1 model), and the other state emits residues from the mean residue composition of the profile (i.e. the expected composition of sequences generated by the core model, including match and insert states.[2] Thus if the profile is highly biased (cysteine-rich, for example; or highly hydrophobic with many transmembrane segments), this composition bias will be captured by this second state. This model's transitions are arbitrarily set such that state 1 emits an expected length of 400 at a time, and state 2 emits an expected length of M/8 at a time (for a profile of length M). An overall target sequence length distribution is set to a mean of L, identical to the null1 model.

[2] `p7_hmm.c:p7_hmm_SetComposition()`

The sequence is then rescored using this "bias filter model" in place of the null1 model, using the HMM Forward algorithm. (This replaces the null1 model score at all subsequent filter steps in the pipeline, until a final Forward score is calculated.) A new MSV bit score is obtained.

If the P-value of this still satisfies the MSV thresholds, the sequence passes the biased composition filter.

The `--F1 <x>` option controls the P-value threshold for passing the

MSV filter score, both before (with the simple null1 model) and after the bias composition filter is applied.

The `--max` option bypasses all filters in the pipeline, including the bias filter.

The `--nobias` option turns off (bypasses) the biased composition filter. The simple null model is used as a null hypothesis for MSV and in subsequent filter steps. The biased composition filter step compromises a small amount of sensitivity. Though it is good to have it on by default, you may want to shut it off if you know you will have no problem with biased composition hits.

At the end of a search output, you will see a line like:

```
Passed bias filter:                  105665  (0.019849); expected 106468.8 (0.02)
```

which tells you how many and what fraction of comparisons passed the biased composition filter, versus how many were expected. (If the filter was turned off, all comparisons pass.)

Viterbi filter

The sequence is now aligned to the profile using a fast Viterbi algorithm for optimal gapped alignment.

This Viterbi implementation is specialized for speed. It is implemented in 8-way parallel SIMD vector instructions, using reduced precision scores that have been scaled to 16-bit integers. Only one row of the dynamic programming matrix is stored, so the routine only recovers the score, not the optimal alignment itself. The reduced representation has limited range; local alignment scores will not underflow, but high scoring comparisons can overflow and return infinity, in which case they automatically pass the filter.

The final Viterbi filter bit score is then computed using the appropriate null model log likelihood (by default the biased composition filter model score, or if the biased filter is off, just the null model score). If the P-value of this score passes the Viterbi filter threshold, the sequence passes on to the next step of the pipeline.

The `--F2 <x>` option controls the P-value threshold for passing the Viterbi filter score. The default is 0.001. The `--max` option bypasses all filters in the pipeline.

At the end of a search output, you will see a line like:

```
Passed Vit filter:                     2207  (0.00443803); expected 497.3 (0.001)
```

which tells you how many and what fraction of comparisons passed the Viterbi filter, versus how many were expected.

Forward filter/parser

The sequence is now aligned to the profile using the full Forward algorithm, which calculates the likelihood of the target sequence given the profile, summed over the ensemble of all possible alignments.

This is a specialized time- and memory-efficient Forward implementation called the "Forward parser". It is implemented in 4-way parallel SIMD vector instructions, in full precision (32-bit floating point). It stores just enough information that, in combination with the results of the Backward parser (below), posterior probabilities of start and stop points of alignments (domains) can be calculated in the domain definition step (below), although the detailed alignments themselves cannot be.

The Forward filter bit score is calculated by correcting this score using the appropriate null model log likelihood (by default the biased composition filter model score, or if the biased filter is off, just the null model score). If the P-value of this bit score passes the Forward filter threshold, the sequence passes on to the next step of the pipeline.

The bias filter score has no further effect in the pipeline. It is only used in filter stages. It has *no* effect on final reported bit scores or P-values. Biased composition compensation for final bit scores is done by a more complex domain-specific algorithm, described below.

The `--F3 <x>` option controls the P-value threshold for passing the Forward filter score. The default is 1e-5. The `--max` option bypasses all filters in the pipeline.

At the end of a search output, you will see a line like:

```
Passed Fwd filter:                     1076  (0.00216371); expected 5.0 (1e-05)
```

which tells you how many and what fraction of comparisons passed the Forward filter, versus how many were expected.

Domain definition

A target sequence that reaches this point is very likely to contain one or more significant matches to the profile. These matches are referred to as "domains", since the main use of HMMER has historically been to match profile HMMs from protein domain databases like Pfam, and one of HMMER's strengths is to be able to cleanly parse a multidomain target sequence into its multiple nonoverlapping hits to the same domain model.

The domain definition step is essentially its own pipeline, with steps as follows:[3]

[3] `src/p7_domaindef.c`

Backward parser The counterpart of the Forward parser algorithm is calculated in an analogous time- and memory-efficient implementation. The Forward algorithm gives the likelihood of all *prefixes* of the target sequence, summed over their alignment ensemble, and the Backward algorithm gives the likelihood of all *suffixes*. For any given point of a possible model state/residue alignment, the product of the Forward and Backward likelihoods gives the likelihood of the entire alignment ensemble conditional on using that particular alignment point. Thus, we can calculate things like the posterior probability that an alignment starts or ends at a given position in the target sequence.

Domain decoding. The posterior decoding algorithm is applied, to calculate the posterior probability of alignment starts and ends (profile B and E state alignments) with respect to target sequence position.

The sum of the posterior probabilities of alignment starts (B states) over the entire target sequence is the *expected number of domains* in the sequence.

In a tabular output (`--tblout`) file, this number is in the column labeled `exp`.

Region identification. A heuristic is now applied to identify a *non-overlapping* set of "regions" that contain significant probability mass suggesting the presence of a match (alignment) to the profile.

For each region, the expected number of domains is calculated (again by posterior decoding on the Forward/Backward parser results). This number should be about 1: we expect each region to contain one local alignment to the profile.

In a tabular output (`--tblout`) file, the number of discrete regions identified by this posterior decoding step is in the column labeled `reg`. It ought to be almost the same as the expectation `exp`. If it is not, there may be something funny going on, like a tandem repetitive element in the target sequence (which can produce so many overlapping weak hits that the sequence appears to be a significant hit with lots of domains expected *somewhere*, but the probability is fuzzed out over the repetitive region and few or no good discrete alignment regions can be identified).

Envelope identification. Now, within each region, we will attempt to identify *envelopes*. An *envelope* is a subsequence of the target sequence that appears to contain alignment probability mass for a likely domain (one local alignment to the profile).

When the region contains $\simeq 1$ expected domain, envelope identification is already done: the region's start and end points are con-

verted directly to the envelope coordinates of a putative domain.

There are a few cases where the region appears to contain more than one expected domain -- where more than one domain is closely spaced on the target sequence and/or the domain scores are weak and the probability masses are ill-resolved from each other. These "multidomain regions", when they occur, are passed off to an even more *ad hoc* resolution algorithm called *stochastic traceback clustering*. In stochastic traceback clustering, we sample many alignments from the posterior alignment ensemble, cluster those alignments according to their overlap in start/end coordinates, and pick clusters that sum up to sufficiently high probability. Consensus start and end points are chosen for each cluster of sampled alignments. These start/end points define envelopes.

These envelopes identified by stochastic traceback clustering are *not* guaranteed to be nonoverlapping. It's possible that there are alternative "solutions" for parsing the sequence into domains, when the correct parsing is ambiguous. HMMER will report all high-likelihood solutions, not just a single nonoverlapping parse.

It's also possible (though rare) for stochastic clustering to identify *no* envelopes in the region.

In a tabular output (`--tblout`) file, the number of regions that had to be subjected to stochastic traceback clustering is given in the column labeled `clu`. This ought to be a small number (often it's zero). The number of envelopes identified by stochastic traceback clustering that overlap with other envelopes is in the column labeled `ov`. If this number is non-zero, you need to be careful when you interpret the details of alignments in the output, because HMMER is going to be showing overlapping alternative solutions. The total number of domain envelopes identified (either by the simple method or by stochastic traceback clustering) is in the column labeled `env`. It ought to be almost the same as the expectation and the number of regions.

Maximum expected accuracy alignment. Each envelope is now aligned to the profile using the full Forward/Backward algorithm. The profile is configured to "unihit" mode, so that the profile expects only one local alignment (domain) in the envelope (as opposed to multiple domains). Posterior decoding is used to calculate the posterior probability of every detailed alignment of profile state to sequence residue. The posterior decodings are used to extract a "maximum expected accuracy" alignment. Each aligned residue is annotated with its posterior probability in the Forward/Backward alignment ensemble.

Currently, the Forward, Backward, and posterior decoding calculations at this step are *not* memory efficient. They calculate matrices requiring roughly $36ML$ bytes, where M is the profile length and L is the length of the envelope subsequence. Usually in `hmmsearch` and

hmmscan, profiles and envelopes are small enough that this is not a problem. For example, a typical Pfam domain model is about 200 residues long, matching to individual target envelopes of about 200 residues each; this requires about 1.4 MB of memory in MEA alignment. However, in phmmer and jackhmmer programs, it's often going to be the case that you're aligning an entire query sequence to an entire target sequence in a single unresolved "domain" alignment. If this is titin (about 40,000 residues), it would require 57.6 GB of RAM. For this reason, currently, phmmer and jackhmmer can only handle query sequences of up to a few thousand residues. If you see a "fatal exception" error complaining about failure of a large memory allocation, you're almost certainly seeing a prohibitive memory requirement at this stage.[4]

[4] I know how to fix this with memory-efficient algorithms, and I'm working on it.

In a tabular output (--tblout) file, the number of domains in envelopes (before any significance thresholding) is in the column labeled dom. This will generally be the same as the number of envelopes.

Biased composition score correction ("null2") An *ad hoc* biased composition score correction is calculated for each envelope, using the posterior decoding. A corrected bit score and P-value for each envelope is calculated. These null2-corrected scores are subjected to the reporting and inclusion thresholds, at both the full sequence level and per-domain.

Modifications to the pipeline as used for DNA search

SSV, not MSV.

In the MSV filter, one or more high-scoring ungapped segments contribute to a score that, if sufficiently high, causes the entire sequence to be passed on to the next stage (the bias filter). This strategy won't work for long DNA sequences; it doesn't filter the human genome much to say "there's a hit on chromosome 1, now postprocess the whole thing". In the scanning-SSV ("Single ungapped Segment Viterbi") algorithm used in nhmmer and nhmmscan, each comparison between a query and target is scanned for high-scoring ungapped alignment segments, and a window around each such segment is extracted, merging overlapping windows. Each window is then passed on to the remaining filter cascade, where it is treated as described above for the most part. As with the MSV filter, the default P-value threshold is 0.02, and can be controlled with the --F1 flag.

The --max flag also controls the amount of the sequence database that passes the SSV filter, but instead of the threshold being set to 1.0,

as described for the protein pipeline, it is set to 0.4.

There are no domains, but there are envelopes

In HMMER's protein-search programs, multiple matches of the model to a target sequence are treated as domains contained within a single hit for that sequence. In the DNA-search programs, each match of the model to a subsequence is treated as an independent hit - there's no notion of a domain. This is largely a difference in reporting. Both pipelines rely on essentially the same envelope detection code; envelopes lead to domains in protein search, and hits in DNA search.

Biased composition.

DNA sequence is littered with regions containing tandem simple repeats or other low complexity sequence. Without accounting for such composition bias, we see many cases in which one part of a hit is obviously legitimate, and serves as the anchor for a neighboring alignment segment that is clearly low-complexity garbage, one form of a problem known as homologous overextension.[5]. The null2 method used in protein search delays score modification until after the alignment is complete, but we know that this kind of overextension can be (mostly) avoided if the model's log odds scores account for the composition bias of the target region while constructing the alignment. The DNA search pipeline therefore does just this: it modifies the scoring scheme for each target envelope as a function of that envelope's sequence composition, then builds the alignment according to that scheme.

[5] M. W. Gonzalez and W. R Pearson. Homologous over-extension: a challenge for iterative similarity searches. *Nucl. Acids Res.*, 38:2177–2189, 2010

Tabular output formats

The target hits table

The `--tblout` output option produces the *target hits table*. The target hits table consists of one line for each different query/target comparison that met the reporting thresholds, ranked by decreasing statistical significance (increasing E-value).

tblout fields for protein search programs In the protein search programs, each line consists of **18 space-delimited fields** followed by a free text target sequence description, as follows:

The `tblout` format is deliberately space-delimited (rather than tab-delimited) and justified into aligned columns, so these files are suitable both for automated parsing and for human examination. I feel that tab-delimited data files are difficult for humans to examine and spot check. For this reason, I think tab-delimited files are a minor evil in the world. Although I occasionally receive shrieks of outrage about this, I still stubbornly feel that space-delimited files are just as easily parsed as tab-delimited files.

(1) target name: The name of the target sequence or profile.

(2) accession: The accession of the target sequence or profile, or '-' if none.

(3) query name: The name of the query sequence or profile.

(4) accession: The accession of the query sequence or profile, or '-' if none.

(5) E-value (full sequence): The expectation value (statistical significance) of the target. This is a *per query* E-value; i.e. calculated as the expected number of false positives achieving this comparison's score for a *single* query against the Z sequences in the target dataset. If you search with multiple queries and if you want to control the *overall* false positive rate of that search rather than the false positive rate per query, you will want to multiply this per-query E-value by how many queries you're doing.

(6) score (full sequence): The score (in bits) for this target/query comparison. It includes the biased-composition correction (the "null2" model).

(7) Bias (full sequence): The biased-composition correction: the bit score difference contributed by the null2 model. High bias scores may be a red flag for a false positive, especially when the bias

score is as large or larger than the overall bit score. It is difficult to correct for all possible ways in which a nonrandom but non-homologous biological sequences can appear to be similar, such as short-period tandem repeats, so there are cases where the bias correction is not strong enough (creating false positives).

(8) E-value (best 1 domain): The E-value if only the single best-scoring domain envelope were found in the sequence, and none of the others. If this E-value isn't good, but the full sequence E-value is good, this is a potential red flag. Weak hits, none of which are good enough on their own, are summing up to lift the sequence up to a high score. Whether this is Good or Bad is not clear; the sequence may contain several weak homologous domains, or it might contain a repetitive sequence that is hitting by chance (i.e. once one repeat hits, all the repeats hit).

(9) score (best 1 domain): The bit score if only the single best-scoring domain envelope were found in the sequence, and none of the others. (Inclusive of the null2 bias correction.]

(10) bias (best 1 domain): The null2 bias correction that was applied to the bit score of the single best-scoring domain.

(11) exp: Expected number of domains, as calculated by posterior decoding on the mean number of begin states used in the alignment ensemble.

(12) reg: Number of discrete regions defined, as calculated by heuristics applied to posterior decoding of begin/end state positions in the alignment ensemble. The number of regions will generally be close to the expected number of domains. The more different the two numbers are, the less discrete the regions appear to be, in terms of probability mass. This usually means one of two things. On the one hand, weak homologous domains may be difficult for the heuristics to identify clearly. On the other hand, repetitive sequence may appear to have a high expected domain number (from lots of crappy possible alignments in the ensemble, no one of which is very convincing on its own, so no one region is discretely well-defined).

(13) clu: Number of regions that appeared to be multidomain, and therefore were passed to stochastic traceback clustering for further resolution down to one or more envelopes. This number is often zero.

(14) ov: For envelopes that were defined by stochastic traceback clustering, how many of them overlap other envelopes.

(15) env: The total number of envelopes defined, both by single envelope regions and by stochastic traceback clustering into one or more envelopes per region.

(16) dom: Number of domains defined. In general, this is the same as the number of envelopes: for each envelope, we find an MEA (maximum expected accuracy) alignment, which defines the endpoints of the alignable domain.

(17) rep: Number of domains satisfying reporting thresholds. If you've also saved a `--domtblout` file, there will be one line in it for each reported domain.

(18) inc: Number of domains satisfying inclusion thresholds.

(19) description of target: The remainder of the line is the target's description line, as free text.

tblout fields for DNA search programs In the DNA search programs, there is less concentration on domains, and more focus on presenting the hit ranges. Each line consists of **15 space-delimited fields** followed by a free text target sequence description, as follows:

(1) target name: The name of the target sequence or profile.

(2) accession: The accession of the target sequence or profile, or '-' if none.

(3) query name: The name of the query sequence or profile.

(4) accession: The accession of the query sequence or profile, or '-' if none.

(5) hmmfrom: The position in the hmm at which the hit starts.

(6) hmm to: The position in the hmm at which the hit ends.

(7) alifrom: The position in the target sequence at which the hit starts.

(8) ali to: The position in the target sequence at which the hit ends.

(9) envfrom: The position in the target sequence at which the surrounding envelope starts.

(10) env to: The position in the target sequence at which the surrounding envelope ends.

(11) sq len: The length of the target sequence..

(12) strand: The strand on which the hit was found ("-" when alifrom>ali to).

(13) E-value: The expectation value (statistical significance) of the target, as above.

(14) score (full sequence): The score (in bits) for this hit. It includes the biased-composition correction.

(15) Bias (full sequence): The biased-composition correction, as above

(16) description of target: The remainder of the line is the target's description line, as free text.

These tables are columnated neatly for human readability, but do not write parsers that rely on this columnation; rely on space-delimited fields. The pretty columnation assumes fixed maximum widths for each field. If a field exceeds its allotted width, it will still be fully represented and space-delimited, but the columnation will be disrupted on the rest of the row.

Note the use of target and query columns. A program like `hmmsearch` searches a query profile against a target sequence database. In an `hmmsearch` tblout file, the sequence (target) name is first, and the profile (query) name is second. A program like `hmmscan`, on the other hand, searches a query sequence against a target profile database. In a `hmmscan` tblout file, the profile name is first, and the sequence name is second. You might say, hey, wouldn't it be more consistent to put the profile name first and the sequence name second (or vice versa), so `hmmsearch` and `hmmscan` tblout files were identical? Well, first of all, they still wouldn't be identical, because the target database size used for E-value calculations is different (number of target sequences for `hmmsearch`, number of target profiles for `hmmscan`, and it's good not to forget this. Second, what about programs like `phmmer` where the query is a sequence and the targets are also sequences?

If the "domain number estimation" section of the protein table (exp, reg, clu, ov, env, dom, rep, inc) makes no sense to you, it may help to read the previous section of the manual, which describes the HMMER processing pipeline, including the steps that probabilistically define domain locations in a sequence.

The domain hits table (protein search only)

In protein search programs, the `--domtblout` option produces the *domain hits table*. There is one line for each domain. There may be more than one domain per sequence. The domain table has **22 whitespace-delimited fields** followed by a free text target sequence description, as follows:

(1) target name: The name of the target sequence or profile.

(2) target accession: Accession of the target sequence or profile, or '-' if none is available.

(3) tlen: Length of the target sequence or profile, in residues. This (together with the query length) is useful for interpreting where the domain coordinates (in subsequent columns) lie in the sequence.

(4) query name: Name of the query sequence or profile.

(5) accession: Accession of the target sequence or profile, or '-' if none is available.

(6) qlen: Length of the query sequence or profile, in residues.

(7) E-value: E-value of the overall sequence/profile comparison (including all domains).

(8) score: Bit score of the overall sequence/profile comparison (including all domains), inclusive of a null2 bias composition correction to the score.

(9) bias: The biased composition score correction that was applied to the bit score.

(10) #: This domain's number (1..ndom).

(11) of: The total number of domains reported in the sequence, ndom.

(12) c-Evalue: The "conditional E-value", a permissive measure of how reliable this particular domain may be. The conditional E-value is calculated on a smaller search space than the independent E-value. The conditional E-value uses the number of targets that pass the reporting thresholds. The null hypothesis test posed by the conditional E-value is as follows. Suppose that we believe that there is already sufficient evidence (from other domains) to identify the set of reported sequences as homologs of our query; now, how many *additional* domains would we expect to find with at least this particular domain's bit score, if the rest of those reported sequences were random nonhomologous sequence (i.e. outside the other domain(s) that were sufficient to identified them as homologs in the first place)?

(13) i-Evalue: The "independent E-value", the E-value that the sequence/profile comparison would have received if this were the only domain envelope found in it, excluding any others. This is a

stringent measure of how reliable this particular domain may be. The independent E-value uses the total number of targets in the target database.

(14) score: The bit score for this domain.

(15) bias: The biased composition (null2) score correction that was applied to the domain bit score.

(16) from (hmm coord): The start of the MEA alignment of this domain with respect to the profile, numbered 1..N for a profile of N consensus positions.

(17) to (hmm coord): The end of the MEA alignment of this domain with respect to the profile, numbered 1..N for a profile of N consensus positions.

(18) from (ali coord): The start of the MEA alignment of this domain with respect to the sequence, numbered 1..L for a sequence of L residues.

(19) to (ali coord): The end of the MEA alignment of this domain with respect to the sequence, numbered 1..L for a sequence of L residues.

(20) from (env coord): The start of the domain envelope on the sequence, numbered 1..L for a sequence of L residues. The *envelope* defines a subsequence for which their is substantial probability mass supporting a homologous domain, whether or not a single discrete alignment can be identified. The envelope may extend beyond the endpoints of the MEA alignment, and in fact often does, for weakly scoring domains.

(21) to (env coord): The end of the domain envelope on the sequence, numbered 1..L for a sequence of L residues.

(22) acc: The mean posterior probability of aligned residues in the MEA alignment; a measure of how reliable the overall alignment is (from 0 to 1, with 1.00 indicating a completely reliable alignment according to the model).

(23) description of target: The remainder of the line is the target's description line, as free text.

As with the target hits table (above), this table is columnated neatly for human readability, but you should not write parsers that rely on this columnation; parse based on space-delimited fields instead.

Manual pages for HMMER programs

alimask *- calculate and add column mask to a multiple sequence alignment*

Synopsis

alimask [*options*] *msafile postmsafile*

Description

`alimask` is used to apply a mask line to a multiple sequence alignment, based on provided alignment or model coordinates. When `hmmbuild` receives a masked alignment as input, it produces a profile model in which the emission probabilities at masked positions are set to match the background frequency, rather than being set based on observed frequencies in the alignment. Position-specific insertion and deletion rates are not altered, even in masked regions. `alimask` autodetects input format, and produces masked alignments in Stockholm format. *`msafile`* may contain only one sequence alignment.

A common motivation for masking a region in an alignment is that the region contains a simple tandem repeat that is observed to cause an unacceptably high rate of false positive hits.

In the simplest case, a mask range is given in coordinates relative to the input alignment, using `--alirange` <*s*>. However it is more often the case that the region to be masked has been identified in coordinates relative to the profile model (e.g. based on recognizing a simple repeat pattern in false hit alignments or in the HMM logo). Not all alignment columns are converted to match state positions in the profile (see the `--symfrac` flag for `hmmbuild` for discussion), so model positions do not necessarily match up to alignment column positions. To remove the burden of converting model positions to alignment positions, `alimask` accepts the mask range input in model coordinates as well, using `--modelrange` <*s*>. When using this flag, *`alimask`* determines which alignment positions would be identified by `hmmbuild` as match states, a process that requires that all `hmmbuild` flags impacting that decision be supplied to `alimask`. It is for this reason that many of the `hmmbuild` flags are also used by `alimask`.

Options

-h Help; print a brief reminder of command line usage and all available options.

-o <*f*> Direct the summary output to file <*f*>, rather than to stdout.

Options for Specifying Mask Range

A single mask range is given as a dash-separated pair, like `--modelrange 10-20` and multiple ranges may be submitted as a comma-separated list, `--modelrange 10-20,30-42`.

--modelrange <*s*> Supply the given range(s) in model coordinates.

--alirange <*s*> Supply the given range(s) in alignment coordinates.

--apendmask Add to the existing mask found with the alignment. The default is to overwrite any existing mask.

--model2ali <*s*> Rather than actually produce the masked alignment, simply print model range(s) corresponding to input alignment range(s).

--ali2model <*s*> Rather than actually produce the masked alignment, simply print alignment range(s) corresponding to input model range(s).

Options for Specifying the Alphabet

--amino Assert that sequences in *msafile* are protein, bypassing alphabet autodetection.

--dna Assert that sequences in *msafile* are DNA, bypassing alphabet autodetection.

--rna Assert that sequences in *msafile* are RNA, bypassing alphabet autodetection.

Options Controlling Profile Construction

These options control how consensus columns are defined in an alignment.

--fast Define consensus columns as those that have a fraction >= `symfrac` of residues as opposed to gaps. (See below for the `--symfrac` option.) This is the default.

--hand Define consensus columns in next profile using reference annotation to the multiple alignment. This allows you to define any consensus columns you like.

--symfrac <*x*> Define the residue fraction threshold necessary to define a consensus column when using the `--fast` option. The default is 0.5. The symbol fraction in each column is calculated after taking relative sequence weighting into account, and ignoring

gap characters corresponding to ends of sequence fragments (as opposed to internal insertions/deletions). Setting this to 0.0 means that every alignment column will be assigned as consensus, which may be useful in some cases. Setting it to 1.0 means that only columns that include 0 gaps (internal insertions/deletions) will be assigned as consensus.

--fragthresh <*x*> We only want to count terminal gaps as deletions if the aligned sequence is known to be full-length, not if it is a fragment (for instance, because only part of it was sequenced). HMMER uses a simple rule to infer fragments: if the sequence length L is less than or equal to a fraction <*x*> times the alignment length in columns, then the sequence is handled as a fragment. The default is 0.5. Setting `--fragthresh 0` will define no (nonempty) sequence as a fragment; you might want to do this if you know you've got a carefully curated alignment of full-length sequences. Setting `--fragthresh 1` will define all sequences as fragments; you might want to do this if you know your alignment is entirely composed of fragments, such as translated short reads in metagenomic shotgun data.

Options Controlling Relative Weights

HMMER uses an ad hoc sequence weighting algorithm to downweight closely related sequences and upweight distantly related ones. This has the effect of making models less biased by uneven phylogenetic representation. For example, two identical sequences would typically each receive half the weight that one sequence would. These options control which algorithm gets used.

--wpb Use the Henikoff position-based sequence weighting scheme [Henikoff and Henikoff, J. Mol. Biol. 243:574, 1994]. This is the default.

--wgsc Use the Gerstein/Sonnhammer/Chothia weighting algorithm [Gerstein et al, J. Mol. Biol. 235:1067, 1994].

--wblosum Use the same clustering scheme that was used to weight data in calculating BLOSUM subsitution matrices [Henikoff and Henikoff, Proc. Natl. Acad. Sci 89:10915, 1992]. Sequences are single-linkage clustered at an identity threshold (default 0.62; see `--wid`) and within each cluster of c sequences, each sequence gets relative weight 1/c.

--wnone No relative weights. All sequences are assigned uniform weight.

--wid <*x*> Sets the identity threshold used by single-linkage clustering when using `--wblosum`. Invalid with any other weighting

scheme. Default is 0.62.

Other Options

--informat <*s*> Assert that input *msafile* is in alignment format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`a2m` or `A2M` both work).

--outformat <*s*> Write the output *postmsafile* in alignment format <*s*>. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. The string <*s*> is case-insensitive (`a2m` or `A2M` both work). Default is `stockholm`.

--seed <*n*> Seed the random number generator with <*n*>, an integer >= 0. If <*n*> is nonzero, any stochastic simulations will be reproducible; the same command will give the same results. If <*n*> is 0, the random number generator is seeded arbitrarily, and stochastic simulations will vary from run to run of the same command. The default seed is 42.

hmmalign - *align sequences to a profile*

Synopsis

hmmalign [*options*] *hmmfile seqfile*

Description

Perform a multiple sequence alignment of all the sequences in *seqfile* by aligning them individually to the profile HMM in *hmmfile*. The new alignment is output to stdout.

The *hmmfile* should contain only a single profile. If it contains more, only the first profile in the file will be used.

Either *hmmfile* or *seqfile* (but not both) may be '-' (dash), which means reading this input from stdin rather than a file.

The sequences in *seqfile* are aligned in unihit local alignment mode. Therefore they should already be known to contain only a single domain (or a fragment of one). The optimal alignment may assign some residues as nonhomologous (N and C states), in which case these residues are still included in the resulting alignment, but shoved to the outer edges. To trim these unaligned nonhomologous residues from the result, see the `--trim` option.

Options

-h Help; print a brief reminder of command line usage and all available options.

-o <*f*> Direct the output alignment to file <*f*>, rather than to stdout.

--mapali <*f*> Merge the existing alignment in file <*f*> into the result, where <*f*> is exactly the same alignment that was used to build the model in *hmmfile*. This is done using a map of alignment columns to consensus profile positions that is stored in the *hmmfile*. The multiple alignment in <*f*> will be exactly reproduced in its consensus columns (as defined by the profile), but the displayed alignment in insert columns may be altered, because insertions relative to a profile are considered by convention to be unaligned data.

--trim Trim nonhomologous residues (assigned to N and C states in the optimal alignments) from the resulting multiple alignment output.

--amino Assert that sequences in *seqfile* are protein, bypassing alphabet autodetection.

--dna Assert that sequences in *seqfile* are DNA, bypassing alphabet autodetection.

--rna Assert that sequences in *seqfile* are RNA, bypassing alphabet autodetection.

--informat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--outformat <*s*> Write the output alignment in format <*s*>. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. The string <*s*> is case-insensitive (`a2m` or `A2M` both work). Default is `stockholm`.

hmmbuild *- construct profiles from multiple sequence alignments*

Synopsis

hmmbuild [*options*] *hmmfile msafile*

Description

For each multiple sequence alignment in *msafile* build a profile HMM and save it to a new file *hmmfile*.

msafile may be '-' (dash), which means reading this input from stdin rather than a file.

hmmfile may not be '-' (stdout), because sending the HMM file to stdout would conflict with the other text output of the program.

Options

-h Help; print a brief reminder of command line usage and all available options.

-n <*s*> Name the new profile <*s*>. The default is to use the name of the alignment (if one is present in the *msafile*, or, failing that, the name of the *hmmfile*. If *msafile* contains more than one alignment, `-n` doesn't work, and every alignment must have a name annotated in the *msafile* (as in Stockholm #=GF ID annotation).

-o <*f*> Direct the summary output to file <*f*>, rather than to stdout.

-O <*f*> After each model is constructed, resave annotated, possibly modified source alignments to a file <*f*> in Stockholm format. The alignments are annotated with a reference annotation line indicating which columns were assigned as consensus, and sequences are annotated with what relative sequence weights were assigned. Some residues of the alignment may have been shifted to accommodate restrictions of the Plan7 profile architecture, which disallows transitions between insert and delete states.

Options for Specifying the Alphabet

--amino Assert that sequences in *msafile* are protein, bypassing alphabet autodetection.

--dna Assert that sequences in *msafile* are DNA, bypassing alphabet autodetection.

--rna Assert that sequences in *msafile* are RNA, bypassing alphabet autodetection.

Options Controlling Profile Construction

These options control how consensus columns are defined in an alignment.

--fast Define consensus columns as those that have a fraction >= `symfrac` of residues as opposed to gaps. (See below for the `--symfrac` option.) This is the default.

--hand Define consensus columns in next profile using reference annotation to the multiple alignment. This allows you to define any consensus columns you like.

--symfrac <*x*> Define the residue fraction threshold necessary to define a consensus column when using the `--fast` option. The default is 0.5. The symbol fraction in each column is calculated after taking relative sequence weighting into account, and ignoring gap characters corresponding to ends of sequence fragments (as opposed to internal insertions/deletions). Setting this to 0.0 means that every alignment column will be assigned as consensus, which may be useful in some cases. Setting it to 1.0 means that only columns that include 0 gaps (internal insertions/deletions) will be assigned as consensus.

--fragthresh <*x*> We only want to count terminal gaps as deletions if the aligned sequence is known to be full-length, not if it is a fragment (for instance, because only part of it was sequenced). HMMER uses a simple rule to infer fragments: if the range of a sequence in the alignment (the number of alignment columns between the first and last positions of the sequence) is less than or equal to a fraction <*x*> times the alignment length in columns, then the sequence is handled as a fragment. The default is 0.5. Setting `--fragthresh 0` will define no (nonempty) sequence as a fragment; you might want to do this if you know you've got a carefully curated alignment of full-length sequences. Setting `--fragthresh 1` will define all sequences as fragments; you might want to do this if you know your alignment is entirely composed of fragments, such as translated short reads in metagenomic shotgun data.

Options Controlling Relative Weights

HMMER uses an ad hoc sequence weighting algorithm to downweight closely related sequences and upweight distantly related ones. This has the effect of making models less biased by uneven phylogenetic representation. For example, two identical sequences would typically each receive half the weight that one sequence would. These options control which algorithm gets used.

--wpb Use the Henikoff position-based sequence weighting scheme [Henikoff and Henikoff, J. Mol. Biol. 243:574, 1994]. This is

the default.

--wgsc Use the Gerstein/Sonnhammer/Chothia weighting algorithm [Gerstein et al, J. Mol. Biol. 235:1067, 1994].

--wblosum Use the same clustering scheme that was used to weight data in calculating BLOSUM subsitution matrices [Henikoff and Henikoff, Proc. Natl. Acad. Sci 89:10915, 1992]. Sequences are single-linkage clustered at an identity threshold (default 0.62; see `--wid`) and within each cluster of c sequences, each sequence gets relative weight 1/c.

--wnone No relative weights. All sequences are assigned uniform weight.

--wid <*x*> Sets the identity threshold used by single-linkage clustering when using `--wblosum`. Invalid with any other weighting scheme. Default is 0.62.

Options Controlling Effective Sequence Number

After relative weights are determined, they are normalized to sum to a total effective sequence number, *eff_nseq*. This number may be the actual number of sequences in the alignment, but it is almost always smaller than that. The default entropy weighting method (`--eent`) reduces the effective sequence number to reduce the information content (relative entropy, or average expected score on true homologs) per consensus position. The target relative entropy is controlled by a two-parameter function, where the two parameters are settable with `--ere` and `--esigma`.

--eent Adjust effective sequence number to achieve a specific relative entropy per position (see `--ere`). This is the default.

--eclust Set effective sequence number to the number of single-linkage clusters at a specific identity threshold (see `--eid`). This option is not recommended; it's for experiments evaluating how much better `--eent` is.

--enone Turn off effective sequence number determination and just use the actual number of sequences. One reason you might want to do this is to try to maximize the relative entropy/position of your model, which may be useful for short models.

--eset <*x*> Explicitly set the effective sequence number for all models to <*x*>.

--ere <*x*> Set the minimum relative entropy/position target to <*x*>. Requires `--eent`. Default depends on the sequence alphabet. For protein sequences, it is 0.59 bits/position; for nucleotide sequences, it is 0.45 bits/position.

--esigma <*x*> Sets the minimum relative entropy contributed by an entire model alignment, over its whole length. This has the effect of making short models have higher relative entropy per position than `--ere` alone would give. The default is 45.0 bits.

--eid <*x*> Sets the fractional pairwise identity cutoff used by single linkage clustering with the `--eclust` option. The default is 0.62.

Options Controlling Priors

By default, weighted counts are converted to mean posterior probability parameter estimates using mixture Dirichlet priors. Default mixture Dirichlet prior parameters for protein models and for nucleic acid (RNA and DNA) models are built in. The following options allow you to override the default priors.

--pnone Don't use any priors. Probability parameters will simply be the observed frequencies, after relative sequence weighting.

--plaplace Use a Laplace +1 prior in place of the default mixture Dirichlet prior.

Options Controlling Single Sequence Scoring

By default, if a query is a single sequence from a file in *fasta* format, `hmmbuild` constructs a search model from that sequence and a standard 20x20 substitution matrix for residue probabilities, along with two additional parameters for position-independent gap open and gap extend probabilities. These options allow the default single-sequence scoring parameters to be changed, and for single-sequence scoring options to be applied to a single sequence coming from an aligned format.

--singlemx If a single sequence query comes from a multiple sequence alignment file, such as in *stockholm* format, the search model is by default constructed as is typically done for multiple sequence alignments. This option forces `hmmbuild` to use the single-sequence method with substitution score matrix.

--mx <*s*> Obtain residue alignment probabilities from the built-in substitution matrix named <*s*>. Several standard matrices are built-in, and do not need to be read from files. The matrix name <*s*> can be PAM30, PAM70, PAM120, PAM240, BLOSUM45, BLOSUM50, BLOSUM62, BLOSUM80, BLOSUM90, or DNA1. Only one of the `--mx` and `--mxfile` options may be used.

--mxfile <*mxfile*> Obtain residue alignment probabilities from the substitution matrix in file <*mxfile*>. The default score matrix is BLOSUM62 for protein sequences, and DNA1 for nucleotide sequences (these matrices are internal to HMMER and do

not need to be available as a file). The format of a substitution matrix <*mxfile*> is the standard format accepted by BLAST, FASTA, and other sequence analysis software. See ftp.ncbi.nlm.nih.gov/blast/matrices/ for example files. (The only exception: we require matrices to be square, so for DNA, use files like NCBI's NUC.4.4, not NUC.4.2.)

--popen <*x*> Set the gap open probability for a single sequence query model to <*x*>. The default is 0.02. <*x*> must be >= 0 and < 0.5.

--pextend <*x*> Set the gap extend probability for a single sequence query model to <*x*>. The default is 0.4. <*x*> must be >= 0 and < 1.0.

Options Controlling E-value Calibration

The location parameters for the expected score distributions for MSV filter scores, Viterbi filter scores, and Forward scores require three short random sequence simulations.

--EmL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

--EmN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

--EvL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

--EvN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

--EfL <*n*> Sets the sequence length in simulation that estimates the location parameter tau for Forward E-values. Default is 100.

--EfN <*n*> Sets the number of sequences in simulation that estimates the location parameter tau for Forward E-values. Default is 200.

--Eft <*x*> Sets the tail mass fraction to fit in the simulation that estimates the location parameter tau for Forward evalues. Default is 0.04.

Other Options

--cpu <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this

number by setting an environment variable, *HMMER_NCPU*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

--informat <*s*> Assert that input *msafile* is in alignment format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`a2m` or `A2M` both work).

--seed <*n*> Seed the random number generator with <*n*>, an integer >= 0. If <*n*> is nonzero, any stochastic simulations will be reproducible; the same command will give the same results. If <*n*> is 0, the random number generator is seeded arbitrarily, and stochastic simulations will vary from run to run of the same command. The default seed is 42.

--w_beta <*x*> Window length tail mass. The upper bound, *W*, on the length at which nhmmer expects to find an instance of the model is set such that the fraction of all sequences generated by the model with length >= *W* is less than <*x*>. The default is 1e-7.

--w_length <*n*> Override the model instance length upper bound, *W*, which is otherwise controlled by `--w_beta`. It should be larger than the model length. The value of *W* is used deep in the acceleration pipeline, and modest changes are not expected to impact results (though larger values of *W* do lead to longer run time).

--mpi Run as a parallel MPI program. Each alignment is assigned to a MPI worker node for construction. (Therefore, the maximum parallelization cannot exceed the number of alignments in the input *msafile*.) This is useful when building large profile libraries. This option is only available if optional MPI capability was enabled at compile-time.

--stall For debugging MPI parallelization: arrest program execution immediately after start, and wait for a debugger to attach to the running process and release the arrest.

--maxinsertlen <*n*> Restrict insert length parameterization such that the expected insert length at each position of the model is no more than <*n*>.

hmmconvert - *convert profile file to various formats*

Synopsis

hmmconvert [*options*] *hmmfile*

Description

The hmmconvert utility converts an input profile file to different HMMER formats.

By default, the input profile can be in any HMMER format, including old/obsolete formats from HMMER2, ASCII or binary; the output profile is a current HMMER3 ASCII format.

hmmfile may be '-' (dash), which means reading this input from stdin rather than a file.

Options

-h Help; print a brief reminder of command line usage and all available options.

-a Output profiles in ASCII text format. This is the default.

-b Output profiles in binary format.

-2 Output in legacy HMMER2 ASCII text format, in ls (glocal) mode. This allows HMMER3 models to be converted back to a close approximation of HMMER2, for comparative studies.

--outfmt <*s*> Output in a HMMER3 ASCII text format other then the most current one. Valid choices for <*s*> are 3/a through 3/f. The current format is 3/f, and this is the default. The format 3/b was used in the official HMMER3 release, and the others were used in the various testing versions.

hmmemit - *sample sequences from a profile*

Synopsis

hmmemit [*options*] *hmmfile*

Description

The hmmemit program samples (emits) sequences from the profile HMM(s) in *hmmfile*, and writes them to output. Sampling sequences may be useful for a variety of purposes, including creating synthetic true positives for benchmarks or tests.

The default is to sample one unaligned sequence from the core probability model, which means that each sequence consists of one full-length domain. Alternatively, with the -c option, you can emit a simple majority-rule consensus sequence; or with the -a option, you can emit an alignment (in which case, you probably also want to set -N to something other than its default of 1 sequence per model).

As another option, with the -p option you can sample a sequence from a fully configured HMMER search profile. This means sampling a 'homologous sequence' by HMMER's definition, including nonhomologous flanking sequences, local alignments, and multiple domains per sequence, depending on the length model and alignment mode chosen for the profile.

The *hmmfile* may contain a library of HMMs, in which case each HMM will be used in turn.

hmmfile may be '-' (dash), which means reading this input from stdin rather than a file.

Common Options

-h Help; print a brief reminder of command line usage and all available options.

-o <*f*> Direct the output sequences to file <*f*>, rather than to stdout.

-N <*n*> Sample <*n*> sequences per model, rather than just one.

Options Controlling What to Emit

The default is to sample N sequences from the core model. Alternatively, you may choose one (and only one) of the following alternatives.

-a Emit an alignment for each HMM in the *hmmfile* rather than sampling unaligned sequences one at a time.

-c Emit a plurality-rule consensus sequence, instead of sampling a sequence from the profile HMM's probability distribution. The consensus sequence is formed by selecting the maximum probability residue at each match state.

-C Emit a fancier plurality-rule consensus sequence than the `-c` option. If the maximum probability residue has p < `minl` show it as a lower case 'any' residue (n or x); if p >= `minl` and < `minu` show it as a lower case residue; and if p >= `minu` show it as an upper case residue. The default settings of `minu` and `minl` are both 0.0, which means `-C` gives the same output as `-c` unless you also set `minu` and `minl` to what you want.

-p Sample unaligned sequences from the implicit search profile, not from the core model. The core model consists only of the homologous states (between the begin and end states of a HMMER Plan7 model). The profile includes the nonhomologous N, C, and J states, local/glocal and uni/multihit algorithm configuration, and the target length model. Therefore sequences sampled from a profile may include nonhomologous as well as homologous sequences, and may contain more than one homologous sequence segment. By default, the profile is in multihit local mode, and the target sequence length is configured for L=400.

Options Controlling Emission from Profiles

These options require that you have set the `-p` option.

-L <*n*> Configure the profile's target sequence length model to generate a mean length of approximately <n> rather than the default of 400.

--local Configure the profile for multihit local alignment.

--unilocal Configure the profile for unihit local alignment (Smith/Waterman).

--glocal Configure the profile for multihit glocal alignment.

--uniglocal Configure the profile for unihit glocal alignment.

Options Controlling Fancy Consensus Emission

These options require that you have set the `-C` option.

--minl <*x*> Sets the `minl` threshold for showing weakly conserved residues as lower case. (0 <= x <= 1)

--minu <*x*> Sets the `minu` threshold for showing strongly conserved residues as upper case. (0 <= x <= 1)

Other Options

--seed <*n*> Seed the random number generator with <*n*>, an integer >= 0. If <*n*> is nonzero, any stochastic simulations will be reproducible; the same command will give the same results. If <*n*>

is 0, the random number generator is seeded arbitrarily, and stochastic simulations will vary from run to run of the same command. The default is 0: use an arbitrary seed, so different `hmmemit` runs will generate different samples.

hmmfetch *- retrieve profiles from a file*

Synopsis

hmmfetch [*options*] *hmmfile key*
(retrieve HMM named *key*)

hmmfetch -f [*options*] *hmmfile keyfile*
(retrieve all HMMs listed in *keyfile*)

hmmfetch --index [*options*] *hmmfile*
(index *hmmfile* for fetching)

Description

Quickly retrieves one or more profile HMMs from an *hmmfile* (a large Pfam database, for example).

For maximum speed, the *hmmfile* should be indexed first, using `hmmfetch --index`. The index is a binary file named *hmmfile*.ssi. However, this is optional, and retrieval will still work from unindexed files, albeit much more slowly.

The default mode is to retrieve a single profile by name or accession, called the *key*. For example:

```
% hmmfetch Pfam-A.hmm Caudal_act
% hmmfetch Pfam-A.hmm PF00045
```

With the `-f` option, a *keyfile* containing a list of one or more keys is read instead. The first whitespace-delimited field on each non-blank non-comment line of the *keyfile* is used as a *key*, and any remaining data on the line is ignored. This allows a variety of whitespace delimited datafiles to be used as a *keyfile*.

When using `-f` and a *keyfile*, if `hmmfile` has been indexed, the keys are retrieved in the order they occur in the *keyfile*, but if `hmmfile` isn't indexed, keys are retrieved in the order they occur in the `hmmfile`. This is a side effect of an implementation that allows multiple keys to be retrieved even if the `hmmfile` is a nonrewindable stream, like a standard input pipe.

In normal use (without `--index` or `-f` options), *hmmfile* may be '-' (dash), which means reading input from stdin rather than a file. With the `--index` option, *hmmfile* may not be '-'; it does not make sense to index a standard input stream. With the `-f` option, either *hmmfile* or *keyfile* (but not both) may be '-'. It is often particularly useful to read *keyfile* from standard input, because this allows use to use arbitrary command line invocations to create a list of HMM names or accessions, then fetch them all to a new file, just with one command.

By default, fetched HMMs are printed to standard output in HMMER3 format.

Options

-h Help; print a brief reminder of command line usage and all available options.

-f The second commandline argument is a *keyfile* instead of a

single *key*. The first field on each line of the *keyfile* is used as a retrieval *key* (an HMM name or accession). Blank lines and comment lines (that start with a # character) are ignored.

-o <*f*> Output HMM(s) to file <*f*> instead of to standard output.

-O Output HMM(s) to individual file(s) named *key* instead of standard output. With the `-f` option, this can result in many files being created.

--index Instead of retrieving one or more profiles from *hmmfile*, index the *hmmfile* for future retrievals. This creates a *hmmfile*.ssi binary index file.

hmmlogo *- produce a conservation logo graphic from a profile*

Synopsis

hmmlogo [*options*] *hmmfile*

Description

`hmmlogo` computes letter height and indel parameters that can be used to produce a profile HMM logo. This tool is essentially a command-line interface for much of the data underlying the Skylign logo server (skylign.org). By default, `hmmlogo` prints out a table of per-position letter heights (dependent on the requested height method), then prints a table of per-position gap probabilities. In a typical logo, the total height of a stack of letters for one position depends on the information content of the position, and that stack height is subdivided according to the emission probabilities of the letters of the alphabet.

Options

-h	Help; print a brief reminder of command line usage and all available options.
--height_relent_all	Total height = relative entropy (aka information content); all letters are given a positive height. (default)
--height_relent_abovebg	Total height = relative entropy (aka information content); only letters with above-background probability are given positive height.
--height_score	Total height = sums of scores of positive-scoring letters; letter height depends on the score of that letter at that position. Only letters with above-background probability (positive score) are given positive height. (Note that only letter height is meaningful - stack height has no inherent meaning).
--no_indel	Don't print out the indel probability table.

hmmpgmd *- daemon for database search web services*

Synopsis

hmmpgmd [*options*]

Description

The `hmmpgmd` program is the daemon that we use internally for the hmmer.org web server. It essentially stands in front of the search programs `phmmer`, `hmmsearch`, and `hmmscan`.

To use `hmmpgmd`, first an instance must be started up as a master server, and provided with at least one sequence database (using the `--seqdb` flag) and/or an HMM database (using the `--hmmdb` flag). A sequence database must be in hmmpgmd format, which may be produced using `esl-reformat`. An HMM database is of the form produced by `hmmbuild`. The input database(s) will be loaded into memory by the master. When the master has finished loading the database(s), it prints the line: "Data loaded into memory. Master is ready."

Only after master is ready, one or more instances of hmmpgmd may be started as workers. These workers may be (and typically are) on different machines from the master, but must have access to the same database file(s) provided to the master, with the same path. As with the master, each worker loads the database(s) into memory, and indicates completion by printing: "Data loaded into memory. Worker is ready."

The master server and workers are expected to remain running. One or more clients then connect to the master and submit possibly many queries. The master distributes the work of a query among the workers, collects results, and merges them before responding to the client. Two example client programs are included in the HMMER src directory - the C program `hmmc2` and the perl script `hmmpgmd_client_example.pl`. These are intended as examples only, and should be extended as necessary to meet your needs.

A query is submitted to the master from the client as a character string. Queries may be the sort that would normally be handled by `phmmer` (protein sequence vs protein database), `hmmsearch` (protein HMM query vs protein database), or `hmmscan` (protein query vs protein HMM database).

The general form of a client query is to start with a single line of the form `@[options]`, followed by multiple lines of text representing either the query HMM or fasta-formatted sequence. The final line of each query is the separator `//`.

For example, to perform a `phmmer` type search of a sequence against a sequence database file, the first line is of the form `@--seqdb 1`, then the fasta-formatted query sequence starting with the header line `>sequence-name`, followed by one or more lines of sequence, and finally the closing `//`.

To perform an `hmmsearch` type search, the query sequence is replaced by the full text of a HMMER-format query HMM.

To perform an `hmmscan` type search, the text matches that of the `phmmer` type search, except that the first line changes to `@--hmmdb 1`.

In the hmmpgmd-formatted sequence database file, each sequence can be associated with one or more sub-databases. The `--seqdb` flag indicates which of these sub-databases will be queried. The HMM database format does not support sub-databases.

The result of each query is an undocumented data structure in binary format. In the future the data will be returned in a proper serialized structure, but for now, it requires meticulous unpacking within the client. The example clients show how this is done.

Options

`-h` Help; print a brief reminder of command line usage and all available options.

Expert Options

`--master` Run as the master server.

`--worker` <*s*> Run as a worker, connecting to the master server that is running on IP address <*s*>.

`--daemon` Run as a daemon using config file: /etc/hmmpgmd.conf

`--cport` <*n*> Port to use for communication between clients and the master server. The default is 51371.

`--wport` <*n*> Port to use for communication between workers and the master server. The default is 51372.

`--ccncts` <*n*> Maximum number of client connections to accept. The default is 16.

`--wcncts` <*n*> Maximum number of worker connections to accept. The default is 32.

`--pid` <*f*> Name of file into which the process id will be written.

`--seqdb` <*f*> Name of the file (in `hmmpgmd` format) containing protein sequences. The contents of this file will be cached for searches.

`--hmmdb` <*f*> Name of the file containing protein HMMs. The contents of this file will be cached for searches.

`--cpu` <*n*> Number of parallel threads to use (for `--worker`).

hmmpress - *prepare a profile database for hmmscan*

Synopsis

hmmpress [*options*] *hmmfile*

Description

Constructs binary compressed datafiles for hmmscan, starting from a profile database *hmmfile* in standard HMMER3 format. The hmmpress step is required for hmmscan to work.

Four files are created: *hmmfile*.h3m, *hmmfile*.h3i, *hmmfile*.h3f, and *hmmfile*.h3p. The *hmmfile*.h3m file contains the profile HMMs and their annotation in a binary format. The *hmmfile*.h3i file is an SSI index for the *hmmfile*.h3m file. The *hmmfile*.h3f file contains precomputed data structures for the fast heuristic filter (the MSV filter). The *hmmfile*.h3p file contains precomputed data structures for the rest of each profile.

hmmfile may not be '-' (dash); running hmmpress on a standard input stream rather than a file is not allowed.

Options

-h Help; print a brief reminder of command line usage and all available options.

-f Force; overwrites any previous hmmpress'ed datafiles. The default is to bitch about any existing files and ask you to delete them first.

hmmscan *- search sequence(s) against a profile database*

Synopsis

hmmscan [*options*] *hmmdb seqfile*

Description

`hmmscan` is used to search protein sequences against collections of protein profiles. For each sequence in *seqfile*, use that query sequence to search the target database of profiles in *hmmdb*, and output ranked lists of the profiles with the most significant matches to the sequence.

The *seqfile* may contain more than one query sequence. Each will be searched in turn against *hmmdb*.

The *hmmdb* needs to be press'ed using `hmmpress` before it can be searched with `hmmscan`. This creates four binary files, suffixed `.h3{fimp}`.

The query *seqfile* may be '-' (a dash character), in which case the query sequences are read from a stdin pipe instead of from a file. The *hmmdb* cannot be read from a stdin stream, because it needs to have those four auxiliary binary files generated by `hmmpress`.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The `--tblout` and `--domtblout` options save output in simple tabular formats that are concise and easier to parse. The `-o` option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

Options for Controlling Output

-o <*f*> Direct the main human-readable output to a file <*f*> instead of the default stdout.

--tblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target model found.

--domtblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-domain output, with one data line per homologous domain detected in a query sequence for each homologous model.

--pfamtblout <*f*> Save an especially succinct tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target model found.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw <*n*> Set the main output's line length limit to <*n*> characters per line. The default is 120.

Options for Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--domtblout`).

-E <*x*> In the per-target output, report target profiles with an E-value of <= <*x*>. The default is 10.0, meaning that on average, about 10 false positives will be reported per query, so you can see the top of the noise and decide for yourself if it's really noise.

-T <*x*> Instead of thresholding per-profile output on E-value, instead report target profiles with a bit score of >= <*x*>.

--domE <*x*> In the per-domain output, for target profiles that have already satisfied the per-profile reporting threshold, report individual domains with a conditional E-value of <= <*x*>. The default is 10.0. A conditional E-value means the expected number of additional false positive domains in the smaller search space of those comparisons that already satisfied the per-profile reporting threshold (and thus must have at least one homologous domain already).

--domT <*x*> Instead of thresholding per-domain output on E-value, instead report domains with a bit score of >= <*x*>.

Options for Inclusion Thresholds

Inclusion thresholds are stricter than reporting thresholds. Inclusion thresholds control which hits are considered to be reliable enough to be included in an output alignment or a subsequent search round. In `hmmscan`, which does not have any alignment output (like `hmmsearch` or `phmmer`) nor any iterative search steps (like `jackhmmer`), inclusion thresholds have little effect. They only affect what domains get marked as significant (!) or questionable (?) in domain output.

--incE <*x*> Use an E-value of <= <*x*> as the per-target inclusion threshold. The default is 0.01, meaning that on average, about 1

false positive would be expected in every 100 searches with different query sequences.

--incT <*x*> Instead of using E-values for setting the inclusion threshold, instead use a bit score of >= <*x*> as the per-target inclusion threshold. It would be unusual to use bit score thresholds with *hmmscan*, because you don't expect a single score threshold to work for different profiles; different profiles have slightly different expected score distributions.

--incdomE <*x*> Use a conditional E-value of <= <*x*> as the per-domain inclusion threshold, in targets that have already satisfied the overall per-target inclusion threshold. The default is 0.01.

--incdomT <*x*> Instead of using E-values, instead use a bit score of >= <*x*> as the per-domain inclusion threshold. As with `--incT` above, it would be unusual to use a single bit score threshold in `hmmscan`.

Options for Model-specific Score Thresholding

Curated profile databases may define specific bit score thresholds for each profile, superseding any thresholding based on statistical significance alone. To use these options, the profile must contain the appropriate (GA, TC, and/or NC) optional score threshold annotation; this is picked up by `hmmbuild` from Stockholm format alignment files. Each thresholding option has two scores: the per-sequence threshold <*x1*> and the per-domain threshold <*x2*>. These act as if `-T` <*x1*> `--incT` <*x1*> `--domT` <*x2*> `--incdomT` <*x2*> has been applied specifically using each model's curated thresholds.

--cut_ga Use the GA (gathering) bit scores in the model to set per-sequence (GA1) and per-domain (GA2) reporting and inclusion thresholds. GA thresholds are generally considered to be the reliable curated thresholds defining family membership; for example, in Pfam, these thresholds define what gets included in Pfam Full alignments based on searches with Pfam Seed models.

--cut_nc Use the NC (noise cutoff) bit score thresholds in the model to set per-sequence (NC1) and per-domain (NC2) reporting and inclusion thresholds. NC thresholds are generally considered to be the score of the highest-scoring known false positive.

--cut_tc Use the NC (trusted cutoff) bit score thresholds in the model to set per-sequence (TC1) and per-domain (TC2) reporting and inclusion thresholds. TC thresholds are generally considered to be the score of the lowest-scoring known true positive that is above all known false positives.

Control of the Acceleration Pipeline

HMMER3 searches are accelerated in a three-step filter pipeline: the MSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm. There is also a bias filter step between MSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

--max Turn off all filters, including the bias filter, and run full Forward/Backward postprocessing on every target. This increases sensitivity somewhat, at a large cost in speed.

--F1 <*x*> Set the P-value threshold for the MSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring nonhomologous targets are expected to pass the filter.

--F2 <*x*> Set the P-value threshold for the Viterbi filter step. The default is 0.001.

--F3 <*x*> Set the P-value threshold for the Forward filter step. The default is 1e-5.

--nobias Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Other Options

--nonull2 Turn off the null2 score corrections for biased composition.

-Z <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-sequence E-value calculations, rather than the actual number of targets seen.

--domZ <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-domain conditional E-value calculations, rather than the number of targets that passed the reporting thresholds.

--seed <*n*> Set the random number seed to <*n*>. Some steps in postprocessing require Monte Carlo simulation. The default is to use a fixed seed (42), so that results are exactly reproducible.

Any other positive integer will give different (but also reproducible) results. A choice of 0 uses an arbitrarily chosen seed.

--qformat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--cpu <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this number by setting an environment variable, *HMMER_NCPU*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

--stall For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: `(gdb) signal SIGCONT`) (Only available if optional MPI support was enabled at compile-time.)

--mpi Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time.

hmmsearch *- search profile(s) against a sequence database*

Synopsis

hmmsearch [*options*] *hmmfile seqdb*

Description

hmmsearch is used to search one or more profiles against a sequence database. For each profile in *hmmfile*, use that query profile to search the target database of sequences in *seqdb*, and output ranked lists of the sequences with the most significant matches to the profile. To build profiles from multiple alignments, see hmmbuild.

Either the query *hmmfile* or the target *seqdb* may be '-' (a dash character), in which case the query profile or target database input will be read from a stdin pipe instead of from a file. Only one input source can come through stdin, not both. An exception is that if the *hmmfile* contains more than one profile query, then *seqdb* cannot come from stdin, because we can't rewind the streaming target database to search it with another profile.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The --tblout and --domtblout options save output in simple tabular formats that are concise and easier to parse. The -o option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

Options for Controlling Output

-o <*f*> Direct the main human-readable output to a file <*f*> instead of the default stdout.

-A <*f*> Save a multiple alignment of all significant hits (those satisfying *inclusion thresholds*) to the file <*f*>.

--tblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target sequence found.

--domtblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-domain output, with one data line per homologous domain detected in a query sequence for each homologous model.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw <*n*> Set the main output's line length limit to <*n*> characters per line. The default is 120.

Options Controlling Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--domtblout`). Sequence hits and domain hits are ranked by statistical significance (E-value) and output is generated in two sections called per-target and per-domain output. In per-target output, by default, all sequence hits with an E-value <= 10 are reported. In the per-domain output, for each target that has passed per-target reporting thresholds, all domains satisfying per-domain reporting thresholds are reported. By default, these are domains with conditional E-values of <= 10. The following options allow you to change the default E-value reporting thresholds, or to use bit score thresholds instead.

-E <*x*> In the per-target output, report target sequences with an E-value of <= <*x*>. The default is 10.0, meaning that on average, about 10 false positives will be reported per query, so you can see the top of the noise and decide for yourself if it's really noise.

-T <*x*> Instead of thresholding per-profile output on E-value, instead report target sequences with a bit score of >= <*x*>.

--domE <*x*> In the per-domain output, for target sequences that have already satisfied the per-profile reporting threshold, report individual domains with a conditional E-value of <= <*x*>. The default is 10.0. A conditional E-value means the expected number of additional false positive domains in the smaller search space of those comparisons that already satisfied the per-target reporting threshold (and thus must have at least one homologous domain already).

--domT <*x*> Instead of thresholding per-domain output on E-value, instead report domains with a bit score of >= <*x*>.

Options for Inclusion Thresholds

Inclusion thresholds are stricter than reporting thresholds. Inclusion thresholds control which hits are considered to be reliable enough to be included in an output alignment or a subsequent search round, or marked as significant ("!") as opposed to questionable ("?") in domain output.

--incE <*x*> Use an E-value of <= <*x*> as the per-target inclusion threshold. The default is 0.01, meaning that on average, about 1 false positive would be expected in every 100 searches with different query sequences.

--incT <*x*> Instead of using E-values for setting the inclusion threshold, instead use a bit score of >= <*x*> as the per-target inclusion threshold. By default this option is unset.

--incdomE <*x*> Use a conditional E-value of <= <*x*> as the per-domain inclusion threshold, in targets that have already satisfied the overall per-target inclusion threshold. The default is 0.01.

--incdomT <*x*> Instead of using E-values, use a bit score of >= <*x*> as the per-domain inclusion threshold.

Options for Model-specific Score Thresholding

Curated profile databases may define specific bit score thresholds for each profile, superseding any thresholding based on statistical significance alone. To use these options, the profile must contain the appropriate (GA, TC, and/or NC) optional score threshold annotation; this is picked up by `hmmbuild` from Stockholm format alignment files. Each thresholding option has two scores: the per-sequence threshold <x1> and the per-domain threshold <x2> These act as if `-T` <*x1*> `--incT` <*x1*> `--domT` <*x2*> `--incdomT` <*x2*> has been applied specifically using each model's curated thresholds.

--cut_ga Use the GA (gathering) bit scores in the model to set per-sequence (GA1) and per-domain (GA2) reporting and inclusion thresholds. GA thresholds are generally considered to be the reliable curated thresholds defining family membership; for example, in Pfam, these thresholds define what gets included in Pfam Full alignments based on searches with Pfam Seed models.

--cut_nc Use the NC (noise cutoff) bit score thresholds in the model to set per-sequence (NC1) and per-domain (NC2) reporting and inclusion thresholds. NC thresholds are generally considered to be the score of the highest-scoring known false positive.

--cut_tc Use the TC (trusted cutoff) bit score thresholds in the model to set per-sequence (TC1) and per-domain (TC2) reporting and inclusion thresholds. TC thresholds are generally considered to be the score of the lowest-scoring known true positive that is above all known false positives.

Options Controlling the Acceleration Pipeline

HMMER3 searches are accelerated in a three-step filter pipeline: the MSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm. There is also a bias filter step

between MSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

--max Turn off all filters, including the bias filter, and run full Forward/Backward postprocessing on every target. This increases sensitivity somewhat, at a large cost in speed.

--F1 <*x*> Set the P-value threshold for the MSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring nonhomologous targets are expected to pass the filter.

--F2 <*x*> Set the P-value threshold for the Viterbi filter step. The default is 0.001.

--F3 <*x*> Set the P-value threshold for the Forward filter step. The default is 1e-5.

--nobias Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Other Options

--nonull2 Turn off the null2 score corrections for biased composition.

-Z <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-sequence E-value calculations, rather than the actual number of targets seen.

--domZ <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-domain conditional E-value calculations, rather than the number of targets that passed the reporting thresholds.

--seed <*n*> Set the random number seed to <*n*>. Some steps in postprocessing require Monte Carlo simulation. The default is to use a fixed seed (42), so that results are exactly reproducible. Any other positive integer will give different (but also reproducible) results. A choice of 0 uses a randomly chosen seed.

--tformat <*s*> Assert that target sequence file *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--cpu <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this number by setting an environment variable, *HMMER_NCPU*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

--stall For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: `(gdb) signal SIGCONT`) (Only available if optional MPI support was enabled at compile-time.)

--mpi Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time.

hmmsim *- collect profile score distributions on random sequences*

Synopsis

hmmsim [*options*] *hmmfile*

Description

The hmmsim program generates random sequences, scores them with the model(s) in *hmmfile*, and outputs various sorts of histograms, plots, and fitted distributions for the resulting scores.

hmmsim is not a mainstream part of the HMMER package and most users would have no reason to use it. It is used to develop and test the statistical methods used to determine P-values and E-values in HMMER3. For example, it was used to generate most of the results in a 2008 paper on H3's local alignment statistics (PLoS Comp Bio 4:e1000069, 2008; http://www.ploscompbiol.org/doi/pcbi.1000069).

Because it is a research testbed, you should not expect it to be as robust as other programs in the package. For example, options may interact in weird ways; we haven't tested nor tried to anticipate all different possible combinations.

The main task is to fit a maximum likelihood Gumbel distribution to Viterbi scores or an maximum likelihood exponential tail to high-scoring Forward scores, and to test that these fitted distributions obey the conjecture that lambda log_2 for both the Viterbi Gumbel and the Forward exponential tail.

The output is a table of numbers, one row for each model. Four different parametric fits to the score data are tested: (1) maximum likelihood fits to both location (mu/tau) and slope (lambda) parameters; (2) assuming lambda=log_2, maximum likelihood fit to the location parameter only; (3) same but assuming an edge-corrected lambda, using current procedures in H3 [Eddy, 2008]; and (4) using both parameters determined by H3's current procedures. The standard simple, quick and dirty statistic for goodness-of-fit is 'E@10', the calculated E-value of the 10th ranked top hit, which we expect to be about 10.

In detail, the columns of the output are:

name Name of the model.

tailp Fraction of the highest scores used to fit the distribution. For Viterbi, MSV, and Hybrid scores, this defaults to 1.0 (a Gumbel distribution is fitted to all the data). For Forward scores, this defaults to 0.02 (an exponential tail is fitted to the highest 2% scores).

mu/tau Location parameter for the maximum likelihood fit to the data.

lambda Slope parameter for the maximum likelihood fit to the data.

E@10 The E-value calculated for the 10th ranked high score ('E@10') using the ML mu/tau and lambda. By definition, this expected to be about 10, if E-value estimation were accurate.

mufix Location parameter, for a maximum likelihood fit with a known (fixed) slope parameter lambda of log_2 (0.693).

E@10fix The E-value calculated for the 10th ranked score using mufix and the expected lambda = log_2 = 0.693.

mufix2 Location parameter, for a maximum likelihood fit with an edge-effect-corrected lambda.

E@10fix2 The E-value calculated for the 10th ranked score using mufix2 and the edge-effect-corrected lambda.

pmu Location parameter as determined by H3's estimation procedures.

plambda Slope parameter as determined by H3's estimation procedures.

pE@10 The E-value calculated for the 10th ranked score using pmu, plambda.

At the end of this table, one more line is printed, starting with # and summarizing the overall CPU time used by the simulations.

Some of the optional output files are in xmgrace xy format. xmgrace is powerful and freely available graph-plotting software.

Options

-h Help; print a brief reminder of command line usage and all available options.

-a Collect expected Viterbi alignment length statistics from each simulated sequence. This only works with Viterbi scores (the default; see `--vit`). Two additional fields are printed in the output table for each model: the mean length of Viterbi alignments, and the standard deviation.

-v (Verbose). Print the scores too, one score per line.

-L <*n*> Set the length of the randomly sampled (nonhomologous) sequences to <*n*>. The default is 100.

-N <*n*> Set the number of randomly sampled sequences to <*n*>. The default is 1000.

--mpi Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time. It is parallelized at the level of sending one profile at a time to an MPI worker process, so parallelization only helps if you have more than one profile in the *`hmmfile`*, and you want to have at least as many profiles as MPI worker processes.

Options Controlling Output

-o <*f*> Save the main output table to a file <*f*> rather than sending it to stdout.

--afile <*f*> When collecting Viterbi alignment statistics (the `-a` option), for each sampled sequence, output two fields per line to a file <*f*>: the length of the optimal alignment, and the Viterbi bit score. Requires that the `-a` option is also used.

--efile <*f*> Output a rank vs. E-value plot in XMGRACE xy format to file <*f*>. The x-axis is the rank of this sequence, from highest score to lowest; the y-axis is the E-value calculated for this sequence. E-values are calculated using H3's default procedures (i.e. the pmu, plambda parameters in the output table). You expect a rough match between rank and E-value if E-values are accurately estimated.

--ffile <*f*> Output a "filter power" file to <*f*>: for each model, a line with three fields: model name, number of sequences passing the P-value threshold, and fraction of sequences passing the P-value threshold. See `--pthresh` for setting the P-value threshold, which defaults to 0.02 (the default MSV filter threshold in H3). The P-values are as determined by H3's default procedures (the pmu,plambda parameters in the output table). If all is well, you expect to see filter power equal to the predicted P-value setting of the threshold.

--pfile <*f*> Output cumulative survival plots (P(S>x)) to file <*f*> in XMGRACE xy format. There are three plots: (1) the observed score distribution; (2) the maximum likelihood fitted distribution; (3) a maximum likelihood fit to the location parameter (mu/tau) while assuming lambda=log_2.

--xfile <*f*> Output the bit scores as a binary array of double-precision floats (8 bytes per score) to file <*f*>. Programs like Easel's `esl-histplot` can read such binary files. This is useful when generating extremely large sample sizes.

Options Controlling Model Configuration (mode)

H3 only uses multihit local alignment (`--fs` mode), and this is where we believe the statistical fits. Unihit local alignment scores (Smith/Waterman; `--sw` mode) also obey our statistical conjectures. Glocal alignment statistics (either multihit or unihit) are still not adequately understood nor adequately fitted.

--fs Collect multihit local alignment scores. This is the default. "fs" comes from HMMER2's historical terminology for multihit local alignment as 'fragment search mode'.

--sw Collect unihit local alignment scores. The H3 J state is disabled. "sw" comes from HMMER2's historical terminology for unihit local alignment as 'Smith/Waterman search mode'.

--ls Collect multihit glocal alignment scores. In glocal (global/local) alignment, the entire model must align, to a subsequence of the target. The H3 local entry/exit transition probabilities are disabled. 'ls' comes from HMMER2's historical terminology for multihit local alignment as 'local search mode'.

--s Collect unihit glocal alignment scores. Both the H3 J state and local entry/exit transition probabilities are disabled. 's' comes from HMMER2's historical terminology for unihit glocal alignment.

Options Controlling Scoring Algorithm

--vit Collect Viterbi maximum likelihood alignment scores. This is the default.

--fwd Collect Forward log-odds likelihood scores, summed over alignment ensemble.

--hyb Collect 'Hybrid' scores, as described in papers by Yu and Hwa (for instance, Bioinformatics 18:864, 2002). These involve calculating a Forward matrix and taking the maximum cell value. The number itself is statistically somewhat unmotivated, but the distribution is expected be a well-behaved extreme value distribution (Gumbel).

--msv Collect MSV (multiple ungapped segment Viterbi) scores, using H3's main acceleration heuristic.

--fast For any of the above options, use H3's optimized production implementation (using SIMD vectorization). The default is to use the "generic" implementation (slow and non-vectorized). The optimized implementations sacrifice a small amount of numerical precision. This can introduce confounding noise into statistical simulations and fits, so when one gets superconcerned about exact details, it's better to be able to factor that source of noise out.

Options Controlling Fitted Tail Masses for Forward

In some experiments, it was useful to fit Forward scores to a range of different tail masses, rather than just one. These options provide a mechanism for fitting an evenly-spaced range of different tail masses. For each different tail mass, a line is generated in the output.

--tmin <*x*> Set the lower bound on the tail mass distribution. (The default is 0.02 for the default single tail mass.)

--tmax <*x*> Set the upper bound on the tail mass distribution. (The default is 0.02 for the default single tail mass.)

--tpoints <*n*> Set the number of tail masses to sample, starting from `--tmin` and ending at `--tmax`. (The default is 1, for the default 0.02 single tail mass.)

--tlinear Sample a range of tail masses with uniform linear spacing. The default is to use uniform logarithmic spacing.

Options Controlling H3 Parameter Estimation Methods

H3 uses three short random sequence simulations to estimating the location parameters for the expected score distributions for MSV scores, Viterbi scores, and Forward scores. These options allow these simulations to be modified.

--EmL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for MSV E-values. Default is 200.

--EmN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for MSV E-values. Default is 200.

--EvL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for Viterbi E-values. Default is 200.

--EvN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for Viterbi E-values. Default is 200.

--EfL <*n*> Sets the sequence length in simulation that estimates the location parameter tau for Forward E-values. Default is 100.

--EfN <*n*> Sets the number of sequences in simulation that estimates the location parameter tau for Forward E-values. Default is 200.

--Eft <*x*> Sets the tail mass fraction to fit in the simulation that estimates the location parameter tau for Forward evalues. Default is 0.04.

Debugging Options

--stall For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: *`(gdb) signal SIGCONT`*) (Only available if optional MPI support was enabled at compile-time.)

--seed <*n*> Set the random number seed to <*n*>. The default is 0, which makes the random number generator use an arbitrary seed, so that different runs of `hmmsim` will almost certainly generate

a different statistical sample. For debugging, it is useful to force reproducible results, by fixing a random number seed.

Experimental Options

These options were used in a small variety of different exploratory experiments.

--bgflat Set the background residue distribution to a uniform distribution, both for purposes of the null model used in calculating scores, and for generating the random sequences. The default is to use a standard amino acid background frequency distribution.

--bgcomp Set the background residue distribution to the mean composition of the profile. This was used in exploring some of the effects of biased composition.

--x-no-lengthmodel Turn the H3 target sequence length model off. Set the self-transitions for N,C,J and the null model to 350/351 instead; this emulates HMMER2. Not a good idea in general. This was used to demonstrate one of the main H2 vs. H3 differences.

--nu <*x*> Set the nu parameter for the MSV algorithm -- the expected number of ungapped local alignments per target sequence. The default is 2.0, corresponding to a E->J transition probability of 0.5. This was used to test whether varying nu has significant effect on result (it doesn't seem to, within reason). This option only works if `--msv` is selected (it only affects MSV), and it will not work with `--fast` (because the optimized implementations are hardwired to assume nu=2.0).

--pthresh <*x*> Set the filter P-value threshold to use in generating filter power files with `--ffile`. The default is 0.02 (which would be appropriate for testing MSV scores, since this is the default MSV filter threshold in H3's acceleration pipeline.) Other appropriate choices (matching defaults in the acceleration pipeline) would be 0.001 for Viterbi, and 1e-5 for Forward.

hmmstat - *summary statistics for a profile file*

Synopsis

hmmstat [*options*] *hmmfile*

Description

The `hmmstat` utility prints out a tabular file of summary statistics for each profile in *hmmfile*.

hmmfile may be '-' (a dash character), in which case profiles are read from a stdin pipe instead of from a file.

The columns are:

idx The index of this profile, numbering each on in the file starting from 1.

name The name of the profile.

accession The optional accession of the profile, or "-" if there is none.

nseq The number of sequences that the profile was estimated from.

eff_nseq The effective number of sequences that the profile was estimated from, after HMMER applied an effective sequence number calculation such as the default entropy weighting.

M The length of the model in consensus residues (match states).

relent Mean relative entropy per match state, in bits. This is the expected (mean) score per consensus position. This is what the default entropy-weighting method for effective sequence number estimation focuses on, so for default HMMER3 models, you expect this value to reflect the default target for entropy-weighting.

info Mean information content per match state, in bits. Probably not useful. Information content is a slightly different calculation than relative entropy.

p relE Mean positional relative entropy, in bits. This is a fancier version of the per-match-state relative entropy, taking into account the transition (insertion/deletion) probabilities; it may be a more accurate estimation of the average score contributed per model consensus position.

compKL Kullback-Leibler distance between the model's overall average residue composition and the default background frequency distribution. The higher this number, the more biased the residue composition of the profile is. Highly biased profiles can slow the HMMER3 acceleration pipeline, by causing too many nonhomologous sequences to pass the filters.

Options

-h Help; print a brief reminder of command line usage and all available options.

jackhmmer *- iteratively search sequence(s) against a sequence database*

Synopsis

jackhmmer [*options*] *seqfile seqdb*

Description

jackhmmer iteratively searches each query sequence in *seqfile* against the target sequence(s) in *seqdb*. The first iteration is identical to a phmmer search. For the next iteration, a multiple alignment of the query together with all target sequences satisfying inclusion thresholds is assembled, a profile is constructed from this alignment (identical to using hmmbuild on the alignment), and profile search of the *seqdb* is done (identical to an hmmsearch with the profile).

The query *seqfile* may be '-' (a dash character), in which case the query sequences are read from a stdin pipe instead of from a file. The *seqdb* cannot be read from a stdin stream, because jackhmmer needs to do multiple passes over the database.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The --tblout and --domtblout options save output in simple tabular formats that are concise and easier to parse. The -o option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

-N <*n*> Set the maximum number of iterations to <*n*>. The default is 5. If N=1, the result is equivalent to a phmmer search.

Options Controlling Output

By default, output for each iteration appears on stdout in a somewhat human readable, somewhat parseable format. These options allow redirecting that output or saving additional kinds of output to files, including checkpoint files for each iteration.

-o <*f*> Direct the human-readable output to a file <*f*>.

-A <*f*> After the final iteration, save an annotated multiple alignment of all hits satisfying inclusion thresholds (also including the original query) to <*f*> in Stockholm format.

--tblout <*f*> After the final iteration, save a tabular summary of top sequence hits to <*f*> in a readily parseable, columnar, whitespace-delimited format.

--domtblout <*f*> After the final iteration, save a tabular summary of top domain hits to <*f*> in a readily parseable, columnar, whitespace-delimited format.

--chkhmm *prefix* At the start of each iteration, checkpoint the query HMM, saving it to a file named *prefix-n*.hmm where *n* is the iteration number (from 1..N).

--chkali *prefix* At the end of each iteration, checkpoint an alignment of all domains satisfying inclusion thresholds (e.g. what will become the query HMM for the next iteration), saving it to a file named *prefix-n*.sto in Stockholm format, where *n* is the iteration number (from 1..N).

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw <*n*> Set the main output's line length limit to <*n*> characters per line. The default is 120.

Options Controlling Single Sequence Scoring (first Iteration)

By default, the first iteration uses a search model constructed from a single query sequence. This model is constructed using a standard 20x20 substitution matrix for residue probabilities, and two additional parameters for position-independent gap open and gap extend probabilities. These options allow the default single-sequence scoring parameters to be changed.

--popen <*x*> Set the gap open probability for a single sequence query model to <*x*>. The default is 0.02. <*x*> must be >= 0 and < 0.5.

--pextend <*x*> Set the gap extend probability for a single sequence query model to <*x*>. The default is 0.4. <*x*> must be >= 0 and < 1.0.

--mx <*s*> Obtain residue alignment probabilities from the built-in substitution matrix named <*s*>. Several standard matrices are built-in, and do not need to be read from files. The matrix name <*s*> can be PAM30, PAM70, PAM120, PAM240, BLOSUM45, BLOSUM50, BLOSUM62, BLOSUM80, or BLOSUM90. Only one of the --mx and --mxfile options may be used.

--mxfile *mxfile* Obtain residue alignment probabilities from the substitution matrix in file *mxfile*. The default score matrix is BLOSUM62 (this matrix is internal to HMMER and does not have to be

available as a file). The format of a substitution matrix *mxfile* is the standard format accepted by BLAST, FASTA, and other sequence analysis software. See `ftp.ncbi.nlm.nih.gov/blast/matrices/` for example files. (The only exception: we require matrices to be square, so for DNA, use files like NCBI's NUC.4.4, not NUC.4.2.)

Options Controlling Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--domtblout`). In each iteration, sequence hits and domain hits are ranked by statistical significance (E-value) and output is generated in two sections called per-target and per-domain output. In per-target output, by default, all sequence hits with an E-value <= 10 are reported. In the per-domain output, for each target that has passed per-target reporting thresholds, all domains satisfying per-domain reporting thresholds are reported. By default, these are domains with conditional E-values of <= 10. The following options allow you to change the default E-value reporting thresholds, or to use bit score thresholds instead.

`-E` <*x*> Report sequences with E-values <= <*x*> in per-sequence output. The default is 10.0.

`-T` <*x*> Use a bit score threshold for per-sequence output instead of an E-value threshold (any setting of `-E` is ignored). Report sequences with a bit score of >= <*x*>. By default this option is unset.

`-Z` <*x*> Declare the total size of the database to be <*x*> sequences, for purposes of E-value calculation. Normally E-values are calculated relative to the size of the database you actually searched (e.g. the number of sequences in *target_seqdb*). In some cases (for instance, if you've split your target sequence database into multiple files for parallelization of your search), you may know better what the actual size of your search space is.

`--domE` <*x*> Report domains with conditional E-values <= <*x*> in per-domain output, in addition to the top-scoring domain per significant sequence hit. The default is 10.0.

`--domT` <*x*> Use a bit score threshold for per-domain output instead of an E-value threshold (any setting of `--domT` is ignored). Report domains with a bit score of >= <*x*> in per-domain output, in addition to the top-scoring domain per significant sequence hit. By default this option is unset.

`--domZ` <*x*> Declare the number of significant sequences to be <*x*> sequences, for purposes of conditional E-value calculation for additional domain significance. Normally conditional

E-values are calculated relative to the number of sequences passing per-sequence reporting threshold.

Options Controlling Inclusion Thresholds

Inclusion thresholds control which hits are included in the multiple alignment and profile constructed for the next search iteration. By default, a sequence must have a per-sequence E-value of <= 0.001 (see `-E` option) to be included, and any additional domains in it besides the top-scoring one must have a conditional E-value of <= 0.001 (see `--domE` option). The difference between reporting thresholds and inclusion thresholds is that inclusion thresholds control which hits actually get used in the next iteration (or the final output multiple alignment if the `-A` option is used), whereas reporting thresholds control what you see in output. Reporting thresholds are generally more loose so you can see borderline hits in the top of the noise that might be of interest.

`--incE` <*x*> Include sequences with E-values <= <*x*> in subsequent iteration or final alignment output by `-A`. The default is 0.001.

`--incT` <*x*> Use a bit score threshold for per-sequence inclusion instead of an E-value threshold (any setting of `--incE` is ignored). Include sequences with a bit score of >= <*x*>. By default this option is unset.

`--incdomE` <*x*> Include domains with conditional E-values <= <*x*> in subsequent iteration or final alignment output by `-A`, in addition to the top-scoring domain per significant sequence hit. The default is 0.001.

`--incdomT` <*x*> Use a bit score threshold for per-domain inclusion instead of an E-value threshold (any setting of `--incT` is ignored). Include domains with a bit score of >= <*x*>. By default this option is unset.

Options Controlling Acceleration Heuristics

HMMER3 searches are accelerated in a three-step filter pipeline: the MSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm, slowest but most accurate. There is also a bias filter step between MSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Essentially the only free parameters that control HMMER's heuristic filters are the P-value thresholds controlling the expected fraction of nonhomologous sequences that pass the filters. Setting the default thresholds higher will pass a higher proportion of nonhomologous sequence, increasing sensitivity at the expense of speed; conversely, setting lower P-value thresholds will pass a smaller proportion, decreasing sensitivity and increasing speed. Setting a filter's P-value threshold to 1.0 means it will passing all sequences,

and effectively disables the filter. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

--max Maximum sensitivity. Turn off all filters, including the bias filter, and run full Forward/Backward postprocessing on every target. This increases sensitivity slightly, at a large cost in speed.

--F1 <*x*> First filter threshold; set the P-value threshold for the MSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring nonhomologous targets are expected to pass the filter.

--F2 <*x*> Second filter threshold; set the P-value threshold for the Viterbi filter step. The default is 0.001.

--F3 <*x*> Third filter threshold; set the P-value threshold for the Forward filter step. The default is 1e-5.

--nobias Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Options Controlling Profile Construction (later Iterations)

These options control how consensus columns are defined in multiple alignments when building profiles. By default, `jackhmmer` always includes your original query sequence in the alignment result at every iteration, and consensus positions are defined by that query sequence: that is, a default `jackhmmer` profile is always the same length as your original query, at every iteration.

--fast Define consensus columns as those that have a fraction >= `symfrac` of residues as opposed to gaps. (See below for the `--symfrac` option.) Although this is the default profile construction option elsewhere (in `hmmbuild`, in particular), it may have undesirable effects in `jackhmmer`, because a profile could iteratively walk in sequence space away from your original query, leaving few or no consensus columns corresponding to its residues.

--hand Define consensus columns in next profile using reference annotation to the multiple alignment. `jackhmmer` propagates

reference annotation from the previous profile to the multiple alignment, and thence to the next profile. This is the default.

--symfrac <*x*> Define the residue fraction threshold necessary to define a consensus column when using the `--fast` option. The default is 0.5. The symbol fraction in each column is calculated after taking relative sequence weighting into account, and ignoring gap characters corresponding to ends of sequence fragments (as opposed to internal insertions/deletions). Setting this to 0.0 means that every alignment column will be assigned as consensus, which may be useful in some cases. Setting it to 1.0 means that only columns that include 0 gaps (internal insertions/deletions) will be assigned as consensus.

--fragthresh <*x*> We only want to count terminal gaps as deletions if the aligned sequence is known to be full-length, not if it is a fragment (for instance, because only part of it was sequenced). HMMER uses a simple rule to infer fragments: if the sequence length L is less than or equal to a fraction <*x*> times the alignment length in columns, then the sequence is handled as a fragment. The default is 0.5. Setting `--fragthresh 0` will define no (nonempty) sequence as a fragment; you might want to do this if you know you've got a carefully curated alignment of full-length sequences. Setting `--fragthresh 1` will define all sequences as fragments; you might want to do this if you know your alignment is entirely composed of fragments, such as translated short reads in metagenomic shotgun data.

Options Controlling Relative Weights

Whenever a profile is built from a multiple alignment, HMMER uses an ad hoc sequence weighting algorithm to downweight closely related sequences and upweight distantly related ones. This has the effect of making models less biased by uneven phylogenetic representation. For example, two identical sequences would typically each receive half the weight that one sequence would (and this is why `jackhmmer` isn't concerned about always including your original query sequence in each iteration's alignment, even if it finds it again in the database you're searching). These options control which algorithm gets used.

--wpb Use the Henikoff position-based sequence weighting scheme [Henikoff and Henikoff, J. Mol. Biol. 243:574, 1994]. This is the default.

--wgsc Use the Gerstein/Sonnhammer/Chothia weighting algorithm [Gerstein et al, J. Mol. Biol. 235:1067, 1994].

--wblosum Use the same clustering scheme that was used to weight data

in calculating BLOSUM subsitution matrices [Henikoff and Henikoff, Proc. Natl. Acad. Sci 89:10915, 1992]. Sequences are single-linkage clustered at an identity threshold (default 0.62; see `--wid`) and within each cluster of c sequences, each sequence gets relative weight 1/c.

`--wnone` No relative weights. All sequences are assigned uniform weight.

`--wid` <*x*> Sets the identity threshold used by single-linkage clustering when using `--wblosum`. Invalid with any other weighting scheme. Default is 0.62.

Options Controlling Effective Sequence Number

After relative weights are determined, they are normalized to sum to a total effective sequence number, *`eff_nseq`*. This number may be the actual number of sequences in the alignment, but it is almost always smaller than that. The default entropy weighting method (`--eent`) reduces the effective sequence number to reduce the information content (relative entropy, or average expected score on true homologs) per consensus position. The target relative entropy is controlled by a two-parameter function, where the two parameters are settable with `--ere` and `--esigma`.

`--eent` Adjust effective sequence number to achieve a specific relative entropy per position (see `--ere`). This is the default.

`--eclust` Set effective sequence number to the number of single-linkage clusters at a specific identity threshold (see `--eid`). This option is not recommended; it's for experiments evaluating how much better `--eent` is.

`--enone` Turn off effective sequence number determination and just use the actual number of sequences. One reason you might want to do this is to try to maximize the relative entropy/position of your model, which may be useful for short models.

`--eset` <*x*> Explicitly set the effective sequence number for all models to <*x*>.

`--ere` <*x*> Set the minimum relative entropy/position target to <*x*>. Requires `--eent`. Default depends on the sequence alphabet; for protein sequences, it is 0.59 bits/position.

`--esigma` <*x*> Sets the minimum relative entropy contributed by an entire model alignment, over its whole length. This has the effect of making short models have higher relative entropy per position than `--ere` alone would give. The default is 45.0 bits.

`--eid` <*x*> Sets the fractional pairwise identity cutoff used by single linkage clustering with the `--eclust` option. The default is 0.62.

Options Controlling Priors

In profile construction, by default, weighted counts are converted to mean posterior probability parameter estimates using mixture Dirichlet priors. Default mixture Dirichlet prior parameters for protein models and for nucleic acid (RNA and DNA) models are built in. The following options allow you to override the default priors.

--pnone Don't use any priors. Probability parameters will simply be the observed frequencies, after relative sequence weighting.

--plaplace Use a Laplace +1 prior in place of the default mixture Dirichlet prior.

Options Controlling E-value Calibration

Estimating the location parameters for the expected score distributions for MSV filter scores, Viterbi filter scores, and Forward scores requires three short random sequence simulations.

--EmL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

--EmN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

--EvL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

--EvN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

--EfL <*n*> Sets the sequence length in simulation that estimates the location parameter tau for Forward E-values. Default is 100.

--EfN <*n*> Sets the number of sequences in simulation that estimates the location parameter tau for Forward E-values. Default is 200.

--Eft <*x*> Sets the tail mass fraction to fit in the simulation that estimates the location parameter tau for Forward evalues. Default is 0.04.

Other Options

--nonull2 Turn off the null2 score corrections for biased composition.

-Z <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-sequence E-value calculations, rather than the actual number of targets seen.

--domZ <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-domain conditional E-value calculations, rather than the number of targets that passed the reporting thresholds.

--seed <*n*> Seed the random number generator with <*n*>, an integer >= 0. If <*n*> is >0, any stochastic simulations will be reproducible; the same command will give the same results. If <*n*> is 0, the random number generator is seeded arbitrarily, and stochastic simulations will vary from run to run of the same command. The default seed is 42.

--qformat <*s*> Assert that input query *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--tformat <*s*> Assert that the input target sequence *seqdb* is in format <*s*>. See `--qformat` above for accepted choices for <*s*>.

--cpu <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this number by setting an environment variable, *HMMER_NCPU*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

--stall For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: `(gdb) signal SIGCONT`) (Only available if optional MPI support was enabled at compile-time.)

--mpi Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time.

makehmmerdb *- build nhmmer database from a sequence file*

Synopsis

makehmmerdb [*options*] *seqfile binaryfile*

Description

`makehmmerdb` is used to create a binary file from a DNA sequence file. This binary file may be used as a target database for the DNA search tool `nhmmer`. Using default settings in `nhmmer`, this yields a roughly 10-fold acceleration with small loss of sensitivity on benchmarks.

Options

-h Help; print a brief reminder of command line usage and all available options.

Other Options

--informat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--bin_length <*n*> Bin length. The binary file depends on a data structure called the FM index, which organizes a permuted copy of the sequence in bins of length <*n*>. Longer bin length will lead to smaller files (because data is captured about each bin) and possibly slower query time. The default is 256. Much more than 512 may lead to notable reduction in speed.

--sa_freq <*n*> Suffix array sample rate. The FM index structure also samples from the underlying suffix array for the sequence database. More frequent sampling (smaller value for <*n*>) will yield larger file size and faster search (until file size becomes large enough to cause I/O to be a bottleneck). The default value is 8. Must be a power of 2.

--block_size <*n*> The input sequence is broken into blocks of size <*n*> million letters. An FM index is built for each block, rather than building an FM index for the entire sequence database. Default is 50. Larger blocks do not seem to yield substantial speed increase.

nhmmer *- search DNA queries against a DNA sequence database*

Synopsis

nhmmer [*options*] *queryfile seqdb*

Description

`nhmmer` is used to search one or more nucleotide queries against a nucleotide sequence database. For each query in *queryfile*, use that query to search the target database of sequences in *seqdb*, and output a ranked list of the hits with the most significant matches to the query. A query may be either a profile model built using `hmmbuild`, a sequence alignment, or a single sequence. Sequence based queries can be in a number of formats (see `--qformat`), and can typically be autodetected. Note that only Stockholm format supports queries made up of more than one sequence alignment.

Either the query *queryfile* or the target *seqdb* may be '-' (a dash character), in which case the query file or target database input will be read from a <stdin> pipe instead of from a file. Only one input source can come through <stdin>, not both. If the *queryfile* contains more than one query, then *seqdb* cannot come from stdin, because we can't rewind the streaming target database to search it with another profile.

If the query is sequence-based, a new file containing the HMM(s) built from the input(s) in *queryfile* may optionally be produced, with the filename set using the `--hmmout` flag.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The `--tblout` option saves output in a simple tabular format that is concise and easier to parse. The `-o` option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

Options for Controlling Output

-o <*f*> Direct the main human-readable output to a file <*f*> instead of the default stdout.

-A <*f*> Save a multiple alignment of all significant hits (those satisfying "inclusion thresholds") to the file <*f*>.

--tblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target sequence found.

--dfamtblout <*f*> Save a tabular (space-delimited) file summarizing the per-hit output, similar to `--tblout` but more succinct.

--aliscoresout <*f*> Save to file a list of per-position scores for each hit. This is useful, for example, in identifying regions of high score density for use in resolving overlapping hits from different models.

--hmmout <*f*> If *queryfile* is sequence-based, write the internally-computed HMM(s) to file <*f*>.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw <*n*> Set the main output's line length limit to <*n*> characters per line. The default is 120.

Options Controlling Single Sequence Scoring

By default, if a query is a single sequence from a file in fasta format, `nhmmer` uses a search model constructed from that sequence and a standard 20x20 substitution matrix for residue probabilities, along with two additional parameters for position-independent gap open and gap extend probabilities. These options allow the default single-sequence scoring parameters to be changed, and for single-sequence scoring options to be applied to a single sequence coming from an aligned format.

--singlemx If a single sequence query comes from a multiple sequence alignment file, such as in Stockholm format, the search model is by default constructed as is typically done for multiple sequence alignments. This option forces `nhmmer` to use the single-sequence method with substitution score matrix.

--mxfile<*mxfile* Obtain residue alignment probabilities from the substitution matrix in file *mxfile*. The default score matrix is DNA1 (this matrix is internal to HMMER and does not have to be available as a file). The format of a substitution matrix *mxfile* is the standard format accepted by BLAST, FASTA, and other sequence analysis software. See `ftp.ncbi.nlm.nih.gov/blast/matrices/` for example files. (The only exception: we require matrices to be square, so for DNA, use files like NCBI's NUC.4.4, not NUC.4.2.)

--popen <*x*> Set the gap open probability for a single sequence query model to <*x*>. The default is 0.02. <*x*> must be >= 0 and < 0.5.

--pextend <*x*> Set the gap extend probability for a single sequence query model to <*x*>. The default is 0.4. <*x*> must be >= 0 and < 1.0.

Options Controlling Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--dfamtblout`). Hits are ranked by statistical significance (E-value).

-E <*x*> Report target sequences with an E-value of <= <*x*>. The default is 10.0, meaning that on average, about 10 false positives will be reported per query, so you can see the top of the noise and decide for yourself if it's really noise.

-T <*x*> Instead of thresholding output on E-value, instead report target sequences with a bit score of >= <*x*>.

Options for Inclusion Thresholds

Inclusion thresholds are stricter than reporting thresholds. Inclusion thresholds control which hits are considered to be reliable enough to be included in an output alignment or a subsequent search round, or marked as significant ("!") as opposed to questionable ("?") in hit output.

--incE <*x*> Use an E-value of <= <*x*> as the inclusion threshold. The default is 0.01, meaning that on average, about 1 false positive would be expected in every 100 searches with different query sequences.

--incT <*x*> Instead of using E-values for setting the inclusion threshold, use a bit score of >= <*x*> as the inclusion threshold. By default this option is unset.

Options for Model-specific Score Thresholding

Curated profile databases may define specific bit score thresholds for each profile, superseding any thresholding based on statistical significance alone. To use these options, the profile must contain the appropriate (GA, TC, and/or NC) optional score threshold annotation; this is picked up by `hmmbuild` from Stockholm format alignment files. For a nucleotide model, each thresholding option has a single per-hit threshold <x> This acts as if `-T` <*x*> `--incT` <*x*> has been applied specifically using each model's curated thresholds.

--cut_ga Use the GA (gathering) bit score threshold in the model to set per-hit reporting and inclusion thresholds. GA thresholds are generally considered to be the reliable curated thresholds defining family membership; for example, in Dfam, these thresholds are applied when annotating a genome with a

model of a family known to be found in that organism. They may allow for minimal expected false discovery rate.

--cut_nc Use the NC (noise cutoff) bit score threshold in the model to set per-hit reporting and inclusion thresholds. NC thresholds are less stringent than GA; in the context of Pfam, they are generally used to store the score of the highest-scoring known false positive.

--cut_tc Use the TC (trusted cutoff) bit score threshold in the model to set per-hit reporting and inclusion thresholds. TC thresholds are more stringent than GA, and are generally considered to be the score of the lowest-scoring known true positive that is above all known false positives; for example, in Dfam, these thresholds are applied when annotating a genome with a model of a family not known to be found in that organism.

Options Controlling the Acceleration Pipeline

HMMER3 searches are accelerated in a three-step filter pipeline: the scanning-SSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm. There is also a bias filter step between SSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

--max Turn off (nearly) all filters, including the bias filter, and run full Forward/Backward postprocessing on most of the target sequence. In contrast to `phmmer` and `hmmsearch`, where this flag really does turn off the filters entirely, the `--max` flag in `nhmmer` sets the scanning-SSV filter threshold to 0.4, not 1.0. Use of this flag increases sensitivity somewhat, at a large cost in speed.

--F1 <*x*> Set the P-value threshold for the SSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring non-homologous targets are expected to pass the filter.

--F2 <*x*> Set the P-value threshold for the Viterbi filter step. The default is 0.001.

--F3 <*x*> Set the P-value threshold for the Forward filter step. The default is 1e-5.

--nobias Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of

hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Options for Specifying the Alphabet

--amino Assert that sequences in *msafile* are protein, bypassing alphabet autodetection.

--dna Assert that sequences in *msafile* are DNA, bypassing alphabet autodetection.

--rna Assert that sequences in *msafile* are RNA, bypassing alphabet autodetection.

Options Controlling Seed Search Heuristic

When searching with `nhmmer`, one may optionally precompute a binary version of the target database, using `makehmmerdb`, then search against that database. Using default settings, this yields a roughly 10-fold acceleration with small loss of sensitivity on benchmarks. This is achieved using a heuristic method that searches for seeds (ungapped alignments) around which full processing is done. This is essentially a replacement to the SSV stage. (This method has been extensively tested, but should still be treated as somewhat experimental.) The following options only impact `nhmmer` if the value of `--tformat` is `hmmerdb`. Changing parameters for this seed-finding step will impact both speed and sensitivity - typically faster search leads to lower sensitivity.

--seed_max_depth <*n*> The seed step requires that a seed reach a specified bit score in length no longer than <*n*>. By default, this value is 15. Longer seeds allow a greater chance of meeting the bit score threshold, leading to diminished filtering (greater sensitivity, slower run time).

--seed_sc_thresh <*x*> The seed must reach score <*x*> (in bits). The default is 15.0 bits. A higher threshold increases filtering stringency, leading to faster run times and lower sensitivity.

--seed_sc_density <*x*> Either all prefixes or all suffixes of a seed must have bit density (bits per aligned position) of at least <*x*>. The default is 0.8 bits/position. An increase in the density requirement leads to increased filtering stringency, thus faster run times and lower sensitivity.

--seed_drop_max_len <*n*> A seed may not have a run of length <*n*> in which the score drops by `--seed_drop_lim` or more. Basically, this prunes seeds that go through long slightly-negative seed extensions. The default is 4. Increasing the limit causes (slightly) diminished

filtering efficiency, thus slower run times and higher sensitivity. (minor tuning option)

--seed_drop_lim <*x*> In a seed, there may be no run of length `--seed_drop_max_len` in which the score drops by `--seed_drop_lim`. The default is 0.3 bits. Larger numbers mean less filtering. (minor tuning option)

--seed_req_pos <*n*> A seed must contain a run of at least <*n*> positive-scoring matches. The default is 5. Larger values mean increased filtering. (minor tuning option)

--seed_ssv_length <*n*> After finding a short seed, an ungapped alignment is extended in both directions in an attempt to meet the `--F1` score threshold. The window through which this ungapped alignment extends is length <*n*>. The default is 70. Decreasing this value slightly reduces run time, at a small risk of reduced sensitivity. (minor tuning option)

Other Options

--qhmm Assert that the input *queryfile* contains one or more profile HMMs, as built by `hmmbuild`.

--qfasta Assert that the input *queryfile* contains one or more unaligned sequences, stored in fasta format.

--qmsa Assert that the input *queryfile* contains one or more sequence alignments. The format of the file may be specified with the `--qformat` flag.

--qformat <*s*> Assert that input *queryfile* is a sequence file (unaligned or aligned), in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*>

--tformat <*s*> Assert that target sequence database *seqdb* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`, `ncbi`, `fmindex`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work). The format `ncbi` indicates that the database file is a binary file produced using `makeblastdb`. The format `fmindex` indicates that the database file is a binary file produced using `makehmmerdb`.

--nonull2 Turn off the null2 score corrections for biased composition.

-Z <*x*> For the purposes of per-hit E-value calculations, Assert that the total size of the target database is <*x*> million nucleotides, rather than the actual number of targets seen.

--seed <*n*> Set the random number seed to <*n*>. Some steps in postprocessing require Monte Carlo simulation. The default is to use a fixed seed (42), so that results are exactly reproducible. Any other positive integer will give different (but also reproducible) results. A choice of 0 uses a randomly chosen seed.

--w_beta <*x*> Window length tail mass. The upper bound, *W*, on the length at which nhmmer expects to find an instance of the model is set such that the fraction of all sequences generated by the model with length >= W is less than <*x*>. The default is 1e-7. This flag may be used to override the value of *W* established for the model by `hmmbuild`, or when the query is sequence-based.

--w_length <*n*> Override the model instance length upper bound, W, which is otherwise controlled by `--w_beta`. It should be larger than the model length. The value of W is used deep in the acceleration pipeline, and modest changes are not expected to impact results (though larger values of W do lead to longer run time). This flag may be used to override the value of W established for the model by `hmmbuild`, or when the query is sequence-based.

--watson Only search the top strand. By default both the query sequence and its reverse-complement are searched.

--crick Only search the bottom (reverse-complement) strand. By default both the query sequence and its reverse-complement are searched.

--cpu <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this number by setting an environment variable, *HMMER_NCPU*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

--stall For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: `(gdb) signal SIGCONT`) (Only available if optional MPI support was enabled at compile-time.)

--mpi Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time.

nhmmscan - *search DNA sequence(s) against a DNA profile database*

Synopsis

nhmmscan [*options*] *hmmdb seqfile*

Description

nhmmscan is used to search nucleotide sequences against collections of nucleotide profiles. For each sequence in *seqfile*, use that query sequence to search the target database of profiles in *hmmdb*, and output ranked lists of the profiles with the most significant matches to the sequence.

The *seqfile* may contain more than one query sequence. It can be in FASTA format, or several other common sequence file formats (genbank, embl, and uniprot, among others), or in alignment file formats (stockholm, aligned fasta, and others). See the *--qformat* option for a complete list.

The *hmmdb* needs to be press'ed using hmmpress before it can be searched with nhmmscan. This creates four binary files, suffixed .h3{fimp}.

The query *seqfile* may be '-' (a dash character), in which case the query sequences are read from a stdin pipe instead of from a file. The *hmmdb* cannot be read from a stdin stream, because it needs to have the four auxiliary binary files generated by hmmpress.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The --tblout option saves output in a simple tabular format that is concise and easier to parse. The -o option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

Options for Controlling Output

-o <*f*> Direct the main human-readable output to a file <*f*> instead of the default stdout.

--tblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-hit output, with one data line per homologous target model hit found.

--dfamtblout <*f*> Save a tabular (space-delimited) file summarizing the per-hit output, similar to --tblout but more succinct.

--aliscoresout <*f*> Save to file a list of per-position scores for each hit. This is useful, for example, in identifying regions of high score density for use in resolving overlapping hits from different models.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw <*n*> Set the main output's line length limit to <*n*> characters per line. The default is 120.

Options for Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--dfamtblout`). Hits are ranked by statistical significance (E-value).

-E <*x*> Report target profiles with an E-value of <= <*x*>. The default is 10.0, meaning that on average, about 10 false positives will be reported per query, so you can see the top of the noise and decide for yourself if it's really noise.

-T <*x*> Instead of thresholding output on E-value, instead report target profiles with a bit score of >= <*x*>.

Options for Inclusion Thresholds

Inclusion thresholds are stricter than reporting thresholds. Inclusion thresholds control which hits are considered to be reliable enough to be included in an output alignment or a subsequent search round. In `nhmmscan`, which does not have any alignment output (like `nhmmer`), inclusion thresholds have little effect. They only affect what hits get marked as significant (!) or questionable (?) in hit output.

--incE <*x*> Use an E-value of <= <*x*> as the inclusion threshold. The default is 0.01, meaning that on average, about 1 false positive would be expected in every 100 searches with different query sequences.

--incT <*x*> Instead of using E-values for setting the inclusion threshold, use a bit score of >= <*x*> as the inclusion threshold. It would be unusual to use bit score thresholds with `hmmscan`, because you don't expect a single score threshold to work for different profiles; different profiles have slightly different expected score distributions.

Options for Model-specific Score Thresholding

Curated profile databases may define specific bit score thresholds for each profile, superseding any thresholding based on statistical significance alone. To use these op-

tions, the profile must contain the appropriate (GA, TC, and/or NC) optional score threshold annotation; this is picked up by `hmmbuild` from Stockholm format alignment files. For a nucleotide model, each thresholding option has a single per-hit threshold <x> This acts as if `-T` <*x*> `--incT` <*x*> has been applied specifically using each model's curated thresholds.

`--cut_ga` Use the GA (gathering) bit score threshold in the model to set per-hit reporting and inclusion thresholds. GA thresholds are generally considered to be the reliable curated thresholds defining family membership; for example, in Dfam, these thresholds are applied when annotating a genome with a model of a family known to be found in that organism. They may allow for minimal expected false discovery rate.

`--cut_nc` Use the NC (noise cutoff) bit score threshold in the model to set per-hit reporting and inclusion thresholds. NC thresholds are less stringent than GA; in the context of Pfam, they are generally used to store the score of the highest-scoring known false positive.

`--cut_tc` Use the TC (trusted cutoff) bit score threshold in the model to set per-hit reporting and inclusion thresholds. TC thresholds are more stringent than GA, and are generally considered to be the score of the lowest-scoring known true positive that is above all known false positives; for example, in Dfam, these thresholds are applied when annotating a genome with a model of a family not known to be found in that organism.

Control of the Acceleration Pipeline

HMMER3 searches are accelerated in a three-step filter pipeline: the scanning-SSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm. There is also a bias filter step between SSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

`--max` Turn off (nearly) all filters, including the bias filter, and run full Forward/Backward postprocessing on most of the target sequence. In contrast to `hmmscan`, where this flag really does turn off the filters entirely, the `--max` flag in `nhmmscan` sets the scanning-SSV filter threshold to 0.4, not 1.0. Use of this flag increases sensitivity somewhat, at a large cost in speed.

`--F1` <*x*> Set the P-value threshold for the MSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring

nonhomologous targets are expected to pass the filter.

--F2 <*x*> Set the P-value threshold for the Viterbi filter step. The default is 0.001.

--F3 <*x*> Set the P-value threshold for the Forward filter step. The default is 1e-5.

--nobias Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Other Options

--nonull2 Turn off the null2 score corrections for biased composition.

-Z <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-sequence E-value calculations, rather than the actual number of targets seen.

--seed <*n*> Set the random number seed to <*n*>. Some steps in postprocessing require Monte Carlo simulation. The default is to use a fixed seed (42), so that results are exactly reproducible. Any other positive integer will give different (but also reproducible) results. A choice of 0 uses an arbitrarily chosen seed.

--qformat <*s*> Assert that input query *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--w_beta <*x*> Window length tail mass. The upper bound, W, on the length at which nhmmer expects to find an instance of the model is set such that the fraction of all sequences generated by the model with length >= W is less than <*x*>. The default is 1e-7. This flag may be used to override the value of W established for the model by `hmmbuild`.

--w_length <*n*> Override the model instance length upper bound, W, which is otherwise controlled by `--w_beta`. It should be larger than

the model length. The value of W is used deep in the acceleration pipeline, and modest changes are not expected to impact results (though larger values of W do lead to longer run time). This flag may be used to override the value of W established for the model by `hmmbuild`.

`--watson` Only search the top strand. By default both the query sequence and its reverse-complement are searched.

`--crick` Only search the bottom (reverse-complement) strand. By default both the query sequence and its reverse-complement are searched.

`--cpu` <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this number by setting an environment variable, *`HMMER_NCPU`*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

`--stall` For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: `(gdb) signal SIGCONT`) (Only available if optional MPI support was enabled at compile-time.)

`--mpi` Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time.

phmmer *- search protein sequence(s) against a protein sequence database*

Synopsis

phmmer [*options*] *seqfile seqdb*

Description

phmmer is used to search one or more query protein sequences against a protein sequence database. For each query sequence in *seqfile*, use that sequence to search the target database of sequences in *seqdb*, and output ranked lists of the sequences with the most significant matches to the query.

Either the query *seqfile* or the target *seqdb* may be '-' (a dash character), in which case the query sequences or target database input will be read from a <stdin> pipe instead of from a file. Only one input source can come through <stdin>, not both. An exception is that if the *seqfile* contains more than one query sequence, then *seqdb* cannot come from <stdin>, because we can't rewind the streaming target database to search it with another query.

The output format is designed to be human-readable, but is often so voluminous that reading it is impractical, and parsing it is a pain. The --tblout and --domtblout options save output in simple tabular formats that are concise and easier to parse. The -o option allows redirecting the main output, including throwing it away in /dev/null.

Options

-h Help; print a brief reminder of command line usage and all available options.

Options for Controlling Output

-o <*f*> Direct the main human-readable output to a file <*f*> instead of the default stdout.

-A <*f*> Save a multiple alignment of all significant hits (those satisfying inclusion thresholds) to the file <*f*> in Stockholm format.

--tblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-target output, with one data line per homologous target sequence found.

--domtblout <*f*> Save a simple tabular (space-delimited) file summarizing the per-domain output, with one data line per homologous domain detected in a query sequence for each homologous model.

--acc Use accessions instead of names in the main output, where available for profiles and/or sequences.

--noali Omit the alignment section from the main output. This can greatly reduce the output volume.

--notextw Unlimit the length of each line in the main output. The default is a limit of 120 characters per line, which helps in displaying the output cleanly on terminals and in editors, but can truncate target profile description lines.

--textw <*n*> Set the main output's line length limit to <*n*> characters per line. The default is 120.

Options Controlling Scoring System

The probability model in `phmmer` is constructed by inferring residue probabilities from a standard 20x20 substitution score matrix, plus two additional parameters for position-independent gap open and gap extend probabilities.

--popen <*x*> Set the gap open probability for a single sequence query model to <*x*>. The default is 0.02. <*x*> must be >= 0 and < 0.5.

--pextend <*x*> Set the gap extend probability for a single sequence query model to <*x*>. The default is 0.4. <*x*> must be >= 0 and < 1.0.

--mx <*s*> Obtain residue alignment probabilities from the built-in substitution matrix named <*s*>. Several standard matrices are built-in, and do not need to be read from files. The matrix name <*s*> can be PAM30, PAM70, PAM120, PAM240, BLOSUM45, BLOSUM50, BLOSUM62, BLOSUM80, or BLOSUM90. Only one of the `--mx` and `--mxfile` options may be used.

--mxfile *mxfile* Obtain residue alignment probabilities from the substitution matrix in file *mxfile*. The default score matrix is BLOSUM62 (this matrix is internal to HMMER and does not have to be available as a file). The format of a substitution matrix *mxfile* is the standard format accepted by BLAST, FASTA, and other sequence analysis software. See ftp.ncbi.nlm.nih.gov/blast/matrices/ for example files. (The only exception: we require matrices to be square, so for DNA, use files like NCBI's NUC.4.4, not NUC.4.2.)

Options Controlling Reporting Thresholds

Reporting thresholds control which hits are reported in output files (the main output, `--tblout`, and `--domtblout`). Sequence hits and domain hits are ranked by statistical significance (E-value) and output is generated in two sections called per-target and per-domain output. In per-target output, by default, all sequence hits with an E-value

<= 10 are reported. In the per-domain output, for each target that has passed per-target reporting thresholds, all domains satisfying per-domain reporting thresholds are reported. By default, these are domains with conditional E-values of <= 10. The following options allow you to change the default E-value reporting thresholds, or to use bit score thresholds instead.

-E <*x*> In the per-target output, report target sequences with an E-value of <= <*x*>. The default is 10.0, meaning that on average, about 10 false positives will be reported per query, so you can see the top of the noise and decide for yourself if it's really noise.

-T <*x*> Instead of thresholding per-profile output on E-value, instead report target sequences with a bit score of >= <*x*>.

--domE <*x*> In the per-domain output, for target sequences that have already satisfied the per-profile reporting threshold, report individual domains with a conditional E-value of <= <*x*>. The default is 10.0. A conditional E-value means the expected number of additional false positive domains in the smaller search space of those comparisons that already satisfied the per-target reporting threshold (and thus must have at least one homologous domain already).

--domT <*x*> Instead of thresholding per-domain output on E-value, instead report domains with a bit score of >= <*x*>.

Options Controlling Inclusion Thresholds

Inclusion thresholds are stricter than reporting thresholds. They control which hits are included in any output multiple alignment (the `-A` option) and which domains are marked as significant ("!") as opposed to questionable ("?") in domain output.

--incE <*x*> Use an E-value of <= <*x*> as the per-target inclusion threshold. The default is 0.01, meaning that on average, about 1 false positive would be expected in every 100 searches with different query sequences.

--incT <*x*> Instead of using E-values for setting the inclusion threshold, instead use a bit score of >= <*x*> as the per-target inclusion threshold. By default this option is unset.

--incdomE <*x*> Use a conditional E-value of <= <*x*> as the per-domain inclusion threshold, in targets that have already satisfied the overall per-target inclusion threshold. The default is 0.01.

--incdomT <*x*> Instead of using E-values, use a bit score of >= <*x*> as the per-domain inclusion threshold. By default this option is unset.

Options Controlling the Acceleration Pipeline

HMMER3 searches are accelerated in a three-step filter pipeline: the MSV filter, the Viterbi filter, and the Forward filter. The first filter is the fastest and most approximate; the last is the full Forward scoring algorithm, slowest but most accurate. There is also a bias filter step between MSV and Viterbi. Targets that pass all the steps in the acceleration pipeline are then subjected to postprocessing -- domain identification and scoring using the Forward/Backward algorithm. Essentially the only free parameters that control HMMER's heuristic filters are the P-value thresholds controlling the expected fraction of nonhomologous sequences that pass the filters. Setting the default thresholds higher will pass a higher proportion of nonhomologous sequence, increasing sensitivity at the expense of speed; conversely, setting lower P-value thresholds will pass a smaller proportion, decreasing sensitivity and increasing speed. Setting a filter's P-value threshold to 1.0 means it will passing all sequences, and effectively disables the filter. Changing filter thresholds only removes or includes targets from consideration; changing filter thresholds does not alter bit scores, E-values, or alignments, all of which are determined solely in postprocessing.

`--max` Maximum sensitivity. Turn off all filters, including the bias filter, and run full Forward/Backward postprocessing on every target. This increases sensitivity slightly, at a large cost in speed.

`--F1` *<x>* First filter threshold; set the P-value threshold for the MSV filter step. The default is 0.02, meaning that roughly 2% of the highest scoring nonhomologous targets are expected to pass the filter.

`--F2` *<x>* Second filter threshold; set the P-value threshold for the Viterbi filter step. The default is 0.001.

`--F3` *<x>* Third filter threshold; set the P-value threshold for the Forward filter step. The default is 1e-5.

`--nobias` Turn off the bias filter. This increases sensitivity somewhat, but can come at a high cost in speed, especially if the query has biased residue composition (such as a repetitive sequence region, or if it is a membrane protein with large regions of hydrophobicity). Without the bias filter, too many sequences may pass the filter with biased queries, leading to slower than expected performance as the computationally intensive Forward/Backward algorithms shoulder an abnormally heavy load.

Options Controlling E-value Calibration

Estimating the location parameters for the expected score distributions for MSV filter scores, Viterbi filter scores, and Forward scores requires three short random sequence simulations.

--EmL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

--EmN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for MSV filter E-values. Default is 200.

--EvL <*n*> Sets the sequence length in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

--EvN <*n*> Sets the number of sequences in simulation that estimates the location parameter mu for Viterbi filter E-values. Default is 200.

--EfL <*n*> Sets the sequence length in simulation that estimates the location parameter tau for Forward E-values. Default is 100.

--EfN <*n*> Sets the number of sequences in simulation that estimates the location parameter tau for Forward E-values. Default is 200.

--Eft <*x*> Sets the tail mass fraction to fit in the simulation that estimates the location parameter tau for Forward evalues. Default is 0.04.

Other Options

--nonull2 Turn off the null2 score corrections for biased composition.

-Z <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-sequence E-value calculations, rather than the actual number of targets seen.

--domZ <*x*> Assert that the total number of targets in your searches is <*x*>, for the purposes of per-domain conditional E-value calculations, rather than the number of targets that passed the reporting thresholds.

--seed <*n*> Seed the random number generator with <*n*>, an integer >= 0. If <*n*> is >0, any stochastic simulations will be reproducible; the same command will give the same results. If <*n*> is 0, the random number generator is seeded arbitrarily, and stochastic simulations will vary from run to run of the same command. The default seed is 42.

--qformat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive

(`fasta` or `FASTA` both work). `--tformat` <*s*> Assert that target sequence database *seqdb* is in format <*s*>, bypassing format autodetection. See `--qformat` above for list of accepted format codes for <*s*>.

`--cpu` <*n*> Set the number of parallel worker threads to <*n*>. On multicore machines, the default is 2. You can also control this number by setting an environment variable, *HMMER_NCPU*. There is also a master thread, so the actual number of threads that HMMER spawns is <*n*>+1. This option is not available if HMMER was compiled with POSIX threads support turned off.

`--stall` For debugging the MPI master/worker version: pause after start, to enable the developer to attach debuggers to the running master and worker(s) processes. Send SIGCONT signal to release the pause. (Under gdb: `(gdb) signal SIGCONT`) (Only available if optional MPI support was enabled at compile-time.)

`--mpi` Run under MPI control with master/worker parallelization (using `mpirun`, for example, or equivalent). Only available if optional MPI support was enabled at compile-time.

Manual pages for Easel miniapps

esl-afetch *- retrieve alignments from a multi-MSA database*

Synopsis

esl-afetch [*options*] *msafile key*
(single MSA retrieval)
esl-afetch -f [*options*] *msafile keyfile*
(multiple MSA retrieval, from a file of keys)
esl-afetch --index *msafile*
(index an MSA file for retrieval)

Description

`esl-afetch` retrieves the alignment named *key* from an alignment database in file *msafile*. The *msafile* is a "multiple multiple alignment" file in Stockholm (e.g. native Pfam or Rfam) format. The *key* is either the name (ID) of the alignment, or its accession number (AC).

Alternatively, `esl-afetch -f` provides the ability to fetch many alignments at once. The `-f` option has it interpret the second argument as a *keyfile*, a file consisting of one name or accession per line.

The *msafile* should first be SSI indexed with `esl-afetch --index` for efficient retrieval. An SSI index is not required, but without one alignment retrieval may be painfully slow.

Options

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

-f Interpret the second argument as a *keyfile* instead of as just one *key*. The *keyfile* contains one name or accession per line. This option doesn't work with the `--index` option.

-o <*f*> Output retrieved alignments to a file <*f*> instead of to stdout.

-O Output retrieved alignment to a file named *key*. This is a convenience for saving some typing: instead of

```
% esl-afetch -o RRM_1 msafile RRM_1
```

you can just type

```
% esl-afetch -O msafile RRM_1
```

The `-O` option only works if you're retrieving a single alignment; it is incompatible with `-f`.

`--index` Instead of retrieving a *`key`*, the special command `esl-afetch --index` *`msafile`* produces an SSI index of the names and accessions of the alignments in the *`msafile`*. Indexing should be done once on the *`msafile`* to prepare it for all future fetches.

esl-alimanip - *manipulate a multiple sequence alignment*

Synopsis

esl-alimanip [*options*] *msafile*

Description

esl-alimanip can manipulate the multiple sequence alignment(s) in *msafile* in various ways. Options exist to remove specific sequences, reorder sequences, designate reference columns using Stockholm "#=GC RF" markup, and add annotation that numbers columns.

The alignments can be of protein or DNA/RNA sequences. All alignments in the same *msafile* must be either protein or DNA/RNA. The alphabet will be autodetected unless one of the options --amino, --dna, or --rna are given.

Options

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

-o <*f*> Save the resulting, modified alignment in Stockholm format to a file <*f*>. The default is to write it to standard output.

--informat <*s*> Assert that *msafile* is in alignment format <*s*>, bypassing format autodetection. Common choices for <*s*> include: stockholm, a2m, afa, psiblast, clustal, phylip. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (a2m or A2M both work).

--outformat <*s*> Write the output in alignment format <*s*>. Common choices for <*s*> include: stockholm, a2m, afa, psiblast, clustal, phylip. The string <*s*> is case-insensitive (a2m or A2M both work). Default is stockholm.

--devhelp Print help, as with -h, but also include undocumented developer options. These options are not listed below, are under development or experimental, and are not guaranteed to even work correctly. Use developer options at your own risk. The only resources for understanding what they actually do are the brief one-line description printed when --devhelp is enabled, and the source code.

Expert Options

--lnfract <*x*> Remove any sequences with length less than <*x*> fraction the length of the median length sequence in the alignment.

--lxfract <*x*> Remove any sequences with length more than <*x*> fraction the length of the median length sequence in the alignment.

--lmin <*n*> Remove any sequences with length less than <*n*> residues.

--lmax <*n*> Remove any sequences with length more than <*n*> residues.

--rfnfract <*x*> Remove any sequences with nongap RF length less than <*x*> fraction the nongap RF length of the alignment.

--detrunc <*n*> Remove any sequences that have all gaps in the first <*n*> non-gap #=GC RF columns or the last <*n*> non-gap #=GC RF columns.

--xambig <*n*> Remove any sequences that has more than <*n*> ambiguous (degenerate) residues.

--seq-r <*f*> Remove any sequences with names listed in file <*f*>. Sequence names listed in <*f*> can be separated by tabs, new lines, or spaces. The file must be in Stockholm format for this option to work.

--seq-k <*f*> Keep only sequences with names listed in file <*f*>. Sequence names listed in <*f*> can be separated by tabs, new lines, or spaces. By default, the kept sequences will remain in the original order they appeared in *msafile*, but the order from <*f*> will be used if the `--k-reorder` option is enabled. The file must be in Stockholm format for this option to work.

--small With `--seq-k` or `--seq-r`, operate in small memory mode. The alignment(s) will not be stored in memory, thus `--seq-k` and `--seq-r` will be able to work on very large alignments regardless of the amount of available RAM. The alignment file must be in Pfam format and `--informat pfam` and one of `--amino`, `--dna`, or `--rna` must be given as well.

--k-reorder With `--seq-k` <*f*>, reorder the kept sequences in the output alignment to the order from the list file <*f*>.

--seq-ins <*n*> Keep only sequences that have at least 1 inserted residue after nongap RF position <*n*>.

--seq-ni <*n*> With `--seq-ins` require at least <*n*> inserted residues in a sequence for it to be kept.

--seq-xi <*n*> With `--seq-ins` allow at most <*n*> inserted residues in a sequence for it to be kept.

--trim <*f*> File <*f*> is an unaligned FASTA file containing truncated versions of each sequence in the *msafile*. Trim the sequences in the alignment to match their truncated versions in <*f*>. If the alignment output format is Stockholm (the default output format), all per-column (GC) and per-residue (GR) annotation will be removed from the alignment when `--trim`

is used. However, if `--t-keeprf` is also used, the reference annotation (GC RF) will be kept.

`--t-keeprf` Specify that the 'trimmed' alignment maintain the original reference (GC RF) annotation. Only works in combination with `--trim`.

`--minpp` <*x*> Replace all residues in the alignments for which the posterior probability annotation (#=GR PP) is less than <*x*> with gaps. The PP annotation for these residues is also converted to gaps. <*x*> must be greater than 0.0 and less than or equal to 0.95.

`--tree` <*f*> Reorder sequences by tree order. Perform single linkage clustering on the sequences in the alignment based on sequence identity given the alignment to define a 'tree' of the sequences. The sequences in the alignment are reordered according to the tree, which groups similar sequences together. The tree is output in Newick format to <*f*>.

`--reorder` <*f*> Reorder sequences to the order listed in file <*f*>. Each sequence in the alignment must be listed in <*f*>. Use `--k-reorder` to reorder only a subset of sequences to a subset alignment file. The file must be in Stockholm format for this option to work.

`--mask2rf` <*f*> Read in the 'mask' file <*f*> and use it to define new #=GC RF annotation for the alignment. <*f*> must be a single line, with exactly <*alen*> or <*rflen*> characters, either the full alignment length or the number of nongap #=GC RF characters, respectively. Each character must be either a '1' or a '0'. The new #=GC RF markup will contain an 'x' for each column that is a '1' in lane mask file, and a '.' for each column that is a '0'. If the mask is of length <*rflen*> then it is interpreted as applying to only nongap RF characters in the existing RF annotation, all gap RF characters will remain gaps and nongap RF characters will be redefined as above.

`--m-keeprf` With `--mask2rf`, do not overwrite existing nongap RF characters that are included by the input mask as 'x', leave them as the character they are.

`--num-all` Add annotation to the alignment numbering all of the columns in the alignment.

`--num-rf` Add annotation to the alignment numbering the non-gap (non '.') #=GC RF columns of the alignment.

`--rm-gc` <*s*> Remove certain types of #=GC annotation from the alignment. <*s*> must be one of: `RF`, `SS_cons`, `SA_cons`, `PP_cons`.

--sindi Annotate individual secondary structures for each sequence by imposing the consensus secondary structure defined by the #=GC SS_cons annotation.

--post2pp Update Infernal's cmalign 0.72-1.0.2 posterior probability "POST" annotation to "PP" annotation, which is read by other miniapps, including `esl-alimask` and `esl-alistat`.

--amino Assert that the *`msafile`* contains protein sequences.

--dna Assert that the *`msafile`* contains DNA sequences.

--rna Assert that the *`msafile`* contains RNA sequences.

esl-alimap - *map two alignments to each other*

Synopsis

esl-alimap [*options*] *msafile1 msafile2*

Description

esl-alimap is a highly specialized application that determines the optimal alignment mapping of columns between two alignments of the same sequences. An alignment mapping defines for each column in alignment 1 a matching column in alignment 2. The number of residues in the aligned sequences that are in common between the two matched columns are considered 'shared' by those two columns.

For example, if the nth residue of sequence i occurs in alignment 1 column x and alignment 2 column y, then only a mapping of alignment 1 and 2 that includes column x mapping to column y would correctly map and share the residue.

The optimal mapping of the two alignments is the mapping which maximizes the sum of shared residues between all pairs of matching columns. The fraction of total residues that are shared is reported as the coverage in the esl-alimap output.

Only the first alignments in *msafile1* and *msafile2* will be mapped to each other. If the files contain more than one alignment, all alignments after the first will be ignored.

The two alignments (one from each file) must contain exactly the same sequences (if they were unaligned, they'd be identical) in precisely the same order. They must also be in Stockholm format.

The output of esl-alimap differs depending on whether one or both of the alignments contain reference (#=GC RF) annotation. If so, the coverage for residues from nongap RF positions will be reported separately from the total coverage.

esl-alimap uses a dynamic programming algorithm to compute the optimal mapping. The algorithm is similar to the Needleman-Wunsch-Sellers algorithm but the scores used at each step of the recursion are not residue-residue comparison scores but rather the number of residues shared between two columns. The `--mask-a2a <f>`, `--mask-a2rf <f>`, `--mask-rf2a <f>`, and `--mask-rf2rf <f>` options create 'mask' files that pertain to the optimal mapping in slightly different ways. A mask file consists of a single line, of only '0' and '1' characters. These denote which positions of the alignment from *msafile1* map to positions of the alignment from *msafile2* as described below for each of the four respective masking options. These masks can be used to extract only those columns of the *msafile1* alignment that optimally map to columns of the *msafile2* alignment using the esl-alimask miniapp. To extract the corresponding set of columns from *msafile2* (that optimally map to columns of the alignment from *msafile1*), it is necessary to rerun the program with the order of the two msafiles reversed, save new masks, and use esl-alimask again.

Options

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

-q Be quiet; don't print information the optimal mapping of each column, only report coverage and potentially save masks to optional output files.

--mask-a2a <*f*> Save a mask of 'o's and '1's to file <*f*>. A '1' at position x means that position x of the alignment from *msafile1* maps to an alignment position in the alignment from *msafile2* in the optimal map.

--mask-a2rf <*f*> Save a mask of 'o's and '1's to file <*f*>. A '1' at position x means that position x of the alignment from *msafile1* maps to a nongap RF position in the alignment from *msafile2* in the optimal map.

--mask-rf2a <*f*> Save a mask of 'o's and '1's to file <*f*>. A '1' at position x means that nongap RF position x of the alignment from *msafile1* maps to an alignment position in the alignment from *msafile2* in the optimal map.

--mask-rf2rf <*f*> Save a mask of 'o's and '1's to file <*f*>. A '1' at position x means that nongap RF position x of the alignment from *msafile1* maps to a nongap RF position in the alignment from *msafile2* in the optimal map.

--submap <*f*> Specify that all of the columns from the alignment from *msafile1* exist identically (contain the same residues from all sequences) in the alignment from *msafile2*. This makes the task of mapping trivial. However, not all columns of *msafile1* must exist in *msafile2*. Save the mask to file <*f*>. A '1' at position x of the mask means that position x of the alignment from *msafile1* is the same as position y of *msafile2*, where y is the number of '1's that occur at positions <= x in the mask.

--amino Assert that *msafile1* and *msafile2* contain protein sequences.

--dna Assert that *msafile1* and *msafile2* contain DNA sequences.

--rna Assert that the *msafile1* and *msafile2* contain RNA sequences.

esl-alimask - *remove columns from a multiple sequence alignment*

Synopsis

esl-alimask [*options*] *msafile maskfile*
(remove columns based on a mask in an input file)
esl-alimask -t [*options*] *msafile coords*
(remove a contiguous set of columns at the start and end of an alignment)
esl-alimask -g [*options*] *msafile*
(remove columns based on their frequency of gaps)
esl-alimask -p [*options*] *msafile*
(remove columns based on their posterior probability annotation)
esl-alimask --rf-is-mask [*options*] *msafile*
(only remove columns that are gaps in the RF annotation)
The `-g` and `-p` options may be used in combination.

Description

`esl-alimask` reads a single input alignment, removes some columns from it (i.e. masks it), and outputs the masked alignment.

`esl-alimask` can be run in several different modes.

`esl-alimask` runs in "mask file mode" by default when two command-line arguments (*msafile* and *maskfile*) are supplied. In this mode, a bit-vector mask in the *maskfile* defines which columns to keep/remove. The mask is a string that may only contain the characters '0' and '1'. A '0' at position x of the mask indicates that column x is excluded by the mask and should be removed during masking. A '1' at position x of the mask indicates that column x is included by the mask and should not be removed during masking. All lines in the *maskfile* that begin with '#' are considered comment lines and are ignored. All non-whitespace characters in non-comment lines are considered to be part of the mask. The length of the mask must equal either the total number of columns in the (first) alignment in *msafile*, or the number of columns that are not gaps in the RF annotation of that alignment. The latter case is only valid if *msafile* is in Stockholm format and contains '#=GC RF' annotation. If the mask length is equal to the non-gap RF length, all gap RF columns will automatically be removed.

`esl-alimask` runs in "truncation mode" if the `-t` option is used along with two command line arguments (*msafile* and *coords*). In this mode, the alignment will be truncated by removing a contiguous set of columns from the beginning and end of the alignment. The second command line argument is the *coords* string, that specifies what range of columns to keep in the alignment, all columns outside of this range will be removed. The *coords* string consists of start and end coordinates separated by any nonnumeric, nonwhitespace character or characters you like; for example, `23..100`, `23/100`, or `23-100` all work. To keep all alignment columns beginning at 23 until the end of the alignment, you can omit the end; for example, `23:` would work. If the `--t-rf` option is used in combination with `-t`, the coordinates in *coords* are interpreted as

non-gap RF column coordinates. For example, with `--t-rf`, a *coords* string of `23-100` would remove all columns before the 23rd non-gap residue in the "#=GC RF" annotation and after the 100th non-gap RF residue.

`esl-alimask` runs in "RF mask" mode if the `--rf-is-mask` option is enabled. In this mode, the alignment must be in Stockholm format and contain '#=GC RF' annotation. `esl-alimask` will simply remove all columns that are gaps in the RF annotation.

`esl-alimask` runs in "gap frequency mode" if `-g` is enabled. In this mode columns for which greater than <*f*> fraction of the aligned sequences have gap residues will be removed. By default, <*f*> is 0.5, but this value can be changed to <*f*> with the `--gapthresh` <*f*> option. In this mode, if the alignment is in Stockholm format and has RF annotation, then all columns that are gaps in the RF annotation will automatically be removed, unless `--saveins` is enabled.

`esl-alimask` runs in "posterior probability mode" if `-p` is enabled. In this mode, masking is based on posterior probability annotation, and the input alignment must be in Stockholm format and contain '#=GR PP' (posterior probability) annotation for all sequences. As a special case, if `-p` is used in combination with `--ppcons`, then the input alignment need not have '#=GR PP' annotation, but must contain '#=GC PP_cons' (posterior probability consensus) annotation.

Characters in Stockholm alignment posterior probability annotation (both '#=GR PP' and '#=GC PP_cons') can have 12 possible values: the ten digits '0-9', '*', and '.'. If '.', the position must correspond to a gap in the sequence (for '#=GR PP') or in the RF annotation (for '#=GC PP_cons'). A value of '0' indicates a posterior probability of between 0.0 and 0.05, '1' indicates between 0.05 and 0.15, '2' indicates between 0.15 and 0.25 and so on up to '9' which indicates between 0.85 and 0.95. A value of '*' indicates a posterior probability of between 0.95 and 1.0. Higher posterior probabilities correspond to greater confidence that the aligned residue belongs where it appears in the alignment.

When `-p` is enabled with `--ppcons` <*x*>, columns which have a consensus posterior probability of less than <*x*> will be removed during masking, and all other columns will not be removed.

When `-p` is enabled without `--ppcons`, the number of each possible PP value in each column is counted. If <*x*> fraction of the sequences that contain aligned residues (i.e. do not contain gaps) in a column have a posterior probability greater than or equal to <*y*>, then that column will not be removed during masking. All columns that do not meet this criterion will be removed. By default, the values of both <*x*> and <*y*> are 0.95, but they can be changed with the `--pfract` <*x*> and `--pthresh` <*y*> options, respectively.

In posterior probability mode, all columns that have 0 residues (i.e. that are 100% gaps) will be automatically removed, unless the `--pallgapok` option is enabled, in which case such columns will not be removed.

Importantly, during posterior probability masking, unless `--pavg` is used, PP annotation values are always considered to be the minimum numerical value in their corresponding range. For example, a PP '9' character is converted to a numerical posterior probability of 0.85. If `--pavg` is used, PP annotation values are considered

to be the average numerical value in their range. For example, a PP '9' character is converted to a numerical posterior probability of 0.90.

In posterior probability mode, if the alignment is in Stockholm format and has RF annotation, then all columns that are gaps in the RF annotation will automatically be removed, unless `--saveins` is enabled.

A single run of `esl-alimask` can perform both gap frequency-based masking and posterior probability-based masking if both the `-g` and `-p` options are enabled. In this case, a gap frequency-based mask and a posterior probability-based mask are independently computed. These two masks are combined to create the final mask using a logical 'and' operation. Any column that is to be removed by either the gap or PP mask will be removed by the final mask.

With the `--small` option, `esl-alimask` will operate in memory saving mode and the required RAM for the masking will be minimal (usually less than a Mb) and independent of the alignment size. To use `--small`, the alignment alphabet must be specified with either `--amino`, `--dna`, or `--rna`, and the alignment must be in Pfam format (non-interleaved, 1 line/sequence Stockholm format). Pfam format is the default output format of INFERNAL's `cmalign` program. Without `--small` the required RAM will be equal to roughly the size of the first input alignment (the size of the alignment file itself if it only contains one alignment).

Output

By default, `esl-alimask` will print only the masked alignment to stdout and then exit. If the `-o` <*f*> option is used, the alignment will be saved to file <*f*> , and information on the number of columns kept and removed will be printed to stdout. If `-q` is used in combination with `-o`, nothing is printed to stdout.

The mask(s) computed by `esl-alimask` when the `-t`, `-p`, `-g`, or `--rf-is-mask` options are used can be saved to output files using the options `--fmask-rf` <*f*>, `--fmask-all` <*f*>, `--gmask-rf` <*f*>, `--gmask-all` <*f*>, `--pmask-rf` <*f*>, and `--pmask-all` <*f*>. In all cases, <*f*> will contain a single line, a bit vector of length <*n*>, where <*n*> is the either the total number of columns in the alignment (for the options suffixed with 'all') or the number of non-gap columns in the RF annotation (for the options suffixed with 'rf'). The mask will be a string of '0' and '1' characters: a '0' at position x in the mask indicates column x was removed (excluded) by the mask, and a '1' at position x indicates column x was kept (included) by the mask. For the 'rf' suffixed options, the mask only applies to non-gap RF columns. The options beginning with 'f' will save the 'final' mask used to keep/remove columns from the alignment. The options beginning with 'g' save the masks based on gap frequency and require `-g`. The options beginning with 'p' save the masks based on posterior probabilities and require `-p`.

Options

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

-o <*f*> Output the final, masked alignment to file <*f*> instead of to stdout. When this option is used, information about the number of columns kept/removed is printed to stdout.

-q Be quiet; do not print anything to stdout. This option can only be used in combination with the `-o` option.

--small Operate in memory saving mode. Required RAM will be independent of the size of the input alignment to mask, instead of roughly the size of the input alignment. When enabled, the alignment must be in Pfam Stockholm (non-interleaved 1 line/seq) format (see `esl-reformat`) and the output alignment will be in Pfam format.

--informat <*s*> Assert that input *msafile* is in alignment format <*s*>. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`a2m` or `A2M` both work). Default is `stockholm` format, unless `--small` is used, in which case `pfam` format (non-interleaved Stockholm) is assumed.

--outformat <*s*> Write the output *msafile* in alignment format <*s*>. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. The string <*s*> is case-insensitive (`a2m` or `A2M` both work). Default is `stockholm`, unless `--small` is enabled, in which case `pfam` (noninterleaved Stockholm) is the default output format.

--fmask-rf <*f*> Save the non-gap RF-length final mask used to mask the alignment to file <*f*>. The input alignment must be in Stockholm format and contain '#=GC RF' annotation for this option to be valid. See the OUTPUT section above for more details on output mask files.

--fmask-all <*f*> Save the full alignment-length final mask used to mask the alignment to file <*f*>. See the OUTPUT section above for more details on output mask files.

--amino Specify that the input alignment is a protein alignment. By default, `esl-alimask` will try to autodetect the alphabet, but if the alignment is sufficiently small it may be ambiguous. This option defines the alphabet as protein. Importantly, if `--small` is enabled, the alphabet must be specified with either `--amino`, `--dna`, or `--rna`.

--dna Specify that the input alignment is a DNA alignment.

--rna Specify that the input alignment is an RNA alignment.

--t-rf With `-t`, specify that the start and end coordinates defined in the second command line argument *coords* correspond to

non-gap RF coordinates. To use this option, the alignment must be in Stockholm format and have "#=GC RF" annotation. See the DESCRIPTION section for an example of using the `--t-rf` option.

`--t-rmins` With `-t`, specify that all columns that are gaps in the reference (RF) annotation in between the specified start and end coordinates be removed. By default, these columns will be kept. To use this option, the alignment must be in Stockholm format and have "#=GC RF" annotation.

`--gapthresh` <*x*> With `-g`, specify that a column is kept (included by mask) if no more than <*f*> fraction of sequences in the alignment have a gap ('.', '-', or '_') at that position. All other columns are removed (excluded by mask). By default, <*x*> is 0.5.

`--gmask-rf` <*f*> Save the non-gap RF-length gap frequency-based mask used to mask the alignment to file <*f*>. The input alignment must be in Stockholm format and contain '#=GC RF' annotation for this option to be valid. See the OUTPUT section above for more details on output mask files.

`--gmask-all` <*f*> Save the full alignment-length gap frequency-based mask used to mask the alignment to file <*f*>. See the OUTPUT section above for more details on output mask files.

`--pfract` <*x*> With `-p`, specify that a column is kept (included by mask) if the fraction of sequences with a non-gap residue in that column with a posterior probability of at least <*y*> (from `--pthresh` <*y*>) is <*x*> or greater. All other columns are removed (excluded by mask). By default <*x*> is 0.95.

`--pthresh` <*y*> With `-p`, specify that a column is kept (included by mask) if <*x*> (from `--pfract` <*x*>) fraction of sequences with a non-gap residue in that column have a posterior probability of at least <*y*>. All other columns are removed (excluded by mask). By default <*y*> is 0.95. See the DESCRIPTION section for more on posterior probability (PP) masking. Due to the granularity of the PP annotation, different <*y*> values within a range covered by a single PP character will be have the same effect on masking. For example, using `--pthresh 0.86` will have the same effect as using `--pthresh 0.94`.

`--pavg` <*x*> With `-p`, specify that a column is kept (included by mask) if the average posterior probability of non-gap residues in that column is at least <*x*>. See the DESCRIPTION section for more on posterior probability (PP) masking.

`--ppcons` <*x*> With `-p`, use the '#=GC PP_cons' annotation to define which columns to keep/remove. A column is kept (included by

mask) if the PP_cons value for that column is <*x*> or greater. Otherwise it is removed.

--pallgapok With `-p`, do not automatically remove any columns that are 100% gaps (i.e. contain 0 aligned residues). By default, such columns will be removed.

--pmask-rf <*f*> Save the non-gap RF-length posterior probability-based mask used to mask the alignment to file <*f*>. The input alignment must be in Stockholm format and contain '#=GC RF' annotation for this option to be valid. See the OUTPUT section above for more details on output mask files.

--pmask-all <*f*> Save the full alignment-length posterior probability-based mask used to mask the alignment to file <*f*>. See the OUTPUT section above for more details on output mask files.

--keepins If `-p` and/or `-g` is enabled and the alignment is in Stockholm or Pfam format and has '#=GC RF' annotation, then allow columns that are gaps in the RF annotation to possibly be kept. By default, all gap RF columns would be removed automatically, but with this option enabled gap and non-gap RF columns are treated identically. To automatically remove all gap RF columns when using a *maskfile* , then define the mask in *maskfile* as having length equal to the non-gap RF length in the alignment. To automatically remove all gap RF columns when using `-t`, use the `--t-rmins` option.

esl-alimerge - *merge alignments based on their reference (RF) annotation*

Synopsis

esl-alimerge [*options*] *alifile1 alifile2*
(merge two alignment files)
esl-alimerge --list [*options*] *listfile*
(merge many alignment files listed in a file)

Description

`esl-alimerge` reads more than one input alignments, merges them into a single alignment and outputs it.

The input alignments must all be in Stockholm format. All alignments must have reference ('#=GC RF') annotation. Further, the RF annotation must be identical in all alignments once gap characters in the RF annotation ('.','-','_') have been removed. This requirement allows alignments with different numbers of total columns to be merged together based on consistent RF annotation, such as alignments created by successive runs of the `cmalign` program of the INFERNAL package using the same CM. Columns which have a gap character in the RF annotation are called 'insert' columns.

All sequence data in all input alignments will be included in the output alignment regardless of the output format (see `--outformat` option below). However, sequences in the merged alignment will usually contain more gaps ('.') than they did in their respective input alignments. This is because `esl-alimerge` must add 100% gap columns to each individual input alignment so that insert columns in the other input alignments can be accomodated in the merged alignment.

If the output format is Stockholm or Pfam, annotation will be transferred from the input alignments to the merged alignment as follows. All per-sequence ('#=GS') and per-residue ('#=GR') annotation is transferred. Per-file ('#=GF') annotation is transferred if it is present and identical in all alignments. Per-column ('#=GC') annotation is transferred if it is present and identical in all alignments once all insert positions have been removed and the '#=GC' annotation includes zero non-gap characters in insert columns.

With the `--list` <*f*> option, <*f*> is a file listing alignment files to merge. In the list file, blank lines and lines that start with '#' (comments) are ignored. Each data line contains a single word: the name of an alignment file to be merged. All alignments in each file will be merged.

With the `--small` option, `esl-alimerge` will operate in memory saving mode and the required RAM for the merge will be minimal (should be only a few Mb) and independent of the alignment sizes. To use `--small`, all alignments must be in Pfam format (non-interleaved, 1 line/sequence Stockholm format). You can reformat alignments to Pfam using the `esl-reformat` Easel miniapp. Without `--small` the required RAM will be equal to roughly the size of the final merged alignment file which will necessarily be at least the summed size of all of the input alignment files to be merged and some-

times several times larger. If you're merging large alignments or you're experiencing very slow performance of `esl-alimerge`, try reformatting to Pfam and using `--small`.

Options

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

-o <*f*> Output merged alignment to file <*f*> instead of to stdout.

-v Be verbose; print information on the size of the alignments being merged, and the annotation transferred to the merged alignment to stdout. This option can only be used in combination with the `-o` option (so that the printed info doesn't corrupt the output alignment file).

--small Operate in memory saving mode. Required RAM will be independent of the sizes of the alignments to merge, instead of roughly the size of the eventual merged alignment. When enabled, all alignments must be in Pfam Stockholm (non-interleaved `1` line/seq) format; see `esl-reformat`(1). The output alignment will be in Pfam format.

--rfonly Only include columns that are not gaps in the GC RF annotation in the merged alignment.

--outformat <*s*> Write the output alignment in format <*s*>. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. The string <*s*> is case-insensitive (`a2m` or `A2M` both work). Default is `stockholm`.

--rna Specify that the input alignments are RNA alignments. By default `esl-alimerge` will try to autodetect the alphabet, but if the alignment is sufficiently small it may be ambiguous. This option defines the alphabet as RNA.

--dna Specify that the input alignments are DNA alignments.

--amino Specify that the input alignments are protein alignments.

esl-alipid *- calculate pairwise percent identities for all sequence*

pairs in an MSA

Synopsis

esl-alipid [*options*] *msafile*

Description

esl-alistat calculates the pairwise percent identity of each sequence pair in in the MSA(s) in *msafile*. For each sequence pair, it outputs a line of <*sqname1*> <sqname2> <pid> <nid> <n> where <*pid*> is the percent identity, <*nid*> is the number of identical aligned pairs, and <*n*> is the denominator used for the calculation: the shorter of the two (unaligned) sequence lengths.

If *msafile* is - (a single dash), alignment input is read from stdin.

Only canonical residues are counted toward <*nid*> and <*n*>. Degenerate residue codes are not counted.

Options

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

--informat <*s*> Assert that input *msafile* is in alignment format <*s*>, bypassing format autodetection. Common choices for <*s*> include: stockholm, a2m, afa, psiblast, clustal, phylip. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (a2m or A2M both work).

--amino Assert that the *msafile* contains protein sequences.

--dna Assert that the *msafile* contains DNA sequences.

--rna Assert that the *msafile* contains RNA sequences.

esl-alirev - *reverse complement a multiple alignment*

Synopsis

esl-alirev [*options*] *msafile*

Description

esl-alirev reads the multiple alignment in *msafile* and outputs its reverse complement to stdout.

An example of where you might need to do this is when you've downloaded a chunk of multiway genomic alignment from one of the genome browsers, but your RNA of interest is on the opposite strand.

Any per-column and per-residue annotation lines are reversed as well, including Stockholm format and old SELEX format annotations. Annotations that Easel recognizes as secondary structure annotation (a consensus structure line, individual secondary structure lines) will be "reverse complemented" to preserve proper bracketing orders: for example, ...<<<...>>> is reverse complemented to <<<...>>>..., not simply reversed to >>>...<<<..., which would be wrong.

If *msafile* is - (a single dash), alignment input is read from stdin.

By default the output alignment is written in the same format as the input alignment. See the --outformat option to use a different output format.

Because the alignment is parsed into Easel's digital internal representation, the output alignment may differ in certain details from the original alignment; these details should be inconsequential but may catch your eye. One is that if you have a reference annotation line, Easel's output will put consensus residues in upper case, nonconsensus (inserted) residues in lower case. Another is that the headers for some formats, such as Clustal format, are written with an arbitrary version number - so you may find yourself revcomping an alignment in "MUSCLE (3.7) multiple sequence alignment" format and it could come out claiming to be a "CLUSTAL 2.1 multiple sequence alignment", just because Easel writes all of its Clustal format alignment files with that header.

The *msafile* must contain nucleic acid sequences (DNA or RNA). The alphabet will be autodetected by default. See the --dna or --rna options to assert an alphabet.

Options

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

--informat <*s*> Assert that input *msafile* is in alignment format <*s*>, bypassing format autodetection. Common choices for <*s*> include: stockholm, a2m, afa, psiblast, clustal, phylip. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (a2m or A2M both work).

--outformat <*s*> Write the output alignment in alignment format <*s*>. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. The string <*s*> is case-insensitive (`a2m` or `A2M` both work). Default is to use same format as the input *`msafile`*.

--dna Assert that the *`msafile`* contains DNA sequences.

--rna Assert that the *`msafile`* contains RNA sequences.

esl-alistat - *summarize a multiple sequence alignment file*

Synopsis

esl-alistat [*options*] *msafile*

Description

`esl-alistat` summarizes the contents of the multiple sequence alignment(s) in *msafile*, such as the alignment name, format, alignment length (number of aligned columns), number of sequences, average pairwise % identity, and mean, smallest, and largest raw (unaligned) lengths of the sequences.

If *msafile* is - (a single dash), multiple alignment input is read from stdin.

The `--list`, `--icinfo`, `--rinfo`, `--pcinfo`, `--psinfo`, `--cinfo`, `--bpinfo`, and `--iinfo` options allow dumping various statistics on the alignment to optional output files as described for each of those options below.

The `--small` option allows summarizing alignments without storing them in memory and can be useful for large alignment files with sizes that approach or exceed the amount of available RAM. When `--small` is used, `esl-alistat` will print fewer statistics on the alignment, omitting data on the smallest and largest sequences and the average identity of the alignment. `--small` only works on Pfam formatted alignments (a special type of non-interleaved Stockholm alignment in which each sequence occurs on a single line) and `--informat pfam` must be given with `--small`. Further, when `--small` is used, the alphabet must be specified with `--amino`, `--dna`, or `--rna`.

Options

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

-1 Use a tabular output format with one line of statistics per alignment in *msafile*. This is most useful when *msafile* contains many different alignments (such as a Pfam database in Stockholm format).

Expert Options

--informat <*s*> Assert that input *msafile* is in alignment format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`a2m` or `A2M` both work).

--amino Assert that the *msafile* contains protein sequences.

--dna Assert that the *msafile* contains DNA sequences.

--rna Assert that the *msafile* contains RNA sequences.

--small Operate in small memory mode for Pfam formatted alignments. `--informat pfam` and one of `--amino`, `--dna`, or `--rna` must be given as well.

--list <*f*> List the names of all sequences in all alignments in `msafile` to file <*f*>. Each sequence name is written on its own line.

--icinfo <*f*> Dump the information content per position in tabular format to file <*f*>. Lines prefixed with "#" are comment lines, which explain the meanings of each of the tab-delimited fields.

--rinfo <*f*> Dump information on the frequency of gaps versus nongap residues per position in tabular format to file <*f*>. Lines prefixed with "#" are comment lines, which explain the meanings of each of the tab-delimited fields.

--pcinfo <*f*> Dump per column information on posterior probabilities in tabular format to file <*f*>. Lines prefixed with "#" are comment lines, which explain the meanings of each of the tab-delimited fields.

--psinfo <*f*> Dump per sequence information on posterior probabilities in tabular format to file <*f*>. Lines prefixed with "#" are comment lines, which explain the meanings of each of the tab-delimited fields.

--iinfo <*f*> Dump information on inserted residues in tabular format to file <*f*>. Insert columns of the alignment are those that are gaps in the reference (#=GC RF) annotation. This option only works if the input file is in Stockholm format with reference annotation. Lines prefixed with "#" are comment lines, which explain the meanings of each of the tab-delimited fields.

--cinfo <*f*> Dump per-column residue counts to file <*f*>. If used in combination with `--noambig` ambiguous (degenerate) residues will be ignored and not counted. Otherwise, they will be marginalized. For example, in an RNA sequence file, a 'N' will be counted as 0.25 'A', 0.25 'C', 0.25 'G', and 0.25 'U'.

--noambig With `--cinfo`, do not count ambiguous (degenerate) residues.

--bpinfo Dump per-column basepair counts to file <*f*>. Counts appear for each basepair in the consensus secondary structure (annotated as "#=GC SS_cons"). Only basepairs from sequences for which both paired positions are canonical residues will be counted. That is, any basepair that is a gap or an ambiguous (degenerate) residue at either position of the pair is ignored and not counted.

--weight With `--icinfo`, `--rinfo`, `--pcinfo`, `--iinfo`, `--cinfo`, and `--bpinfo`, weight counts based on #=GS WT annotation in the input

msafile. A residue or basepair from a sequence with a weight of <*x*> will be considered <*x*> counts. By default, raw, unweighted counts are reported; corresponding to each sequence having an equal weight of 1.

esl-compalign - *compare two multiple sequence alignments*

Synopsis

esl-compalign [*options*] *trusted_file test_file*

Description

esl-compalign evaluates the accuracy of a predicted multiple sequence alignment with respect to a trusted alignment of the same sequences.

The *trusted_file* and *test_file* must contain the same number of alignments. Each predicted alignment in the *test_file* will be compared against a single trusted alignment from the *trusted_file*. The first alignments in each file correspond to each other and will be compared, the second alignment in each file correspond to each other and will be compared, and so on. Each corresponding pair of alignments must contain the same sequences (i.e. if they were unaligned they would be identical) in the same order in both files. Further, both alignment files must be in Stockholm format and contain 'reference' annotation, which appears as "#=GC RF" per-column markup for each alignment. The number of nongap (non '.' characters) in the reference (RF) annotation must be identical between all corresponding alignments in the two files.

esl-compalign reads an alignment from each file, and compares them based on their 'reference' annotation. The number of correctly predicted residues for each sequence is computed as follows. A residue that is in the Nth nongap RF column in the trusted alignment must also appear in the Nth nongap RF column in the predicted alignment to be counted as 'correct', otherwise it is 'incorrect'. A residue that appears in a gap RF column in the trusted alignment between nongap RF columns N and N+1 must also appear in a nongap RF column in the predicted alignment between nongap RF columns N and N+1 to be counted as 'correct', otherwise it is incorrect.

The default output of esl-compalign lists each sequence and the number of correctly and incorrectly predicted residues for that sequence. These counts are broken down into counts for residues in the predicted alignments that occur in 'match' columns and 'insert' columns. A 'match' column is one for which the RF annotation does not contain a gap. An 'insert' column is one for which the RF annotation does contain a gap.

Options

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

-c Print per-column statistics instead of per-sequence statistics.

-p Print statistics on accuracy versus posterior probability values. The *test_file* must be annotated with posterior probabilities (#=GR PP) for this option to work.

Expert Options

--p-mask <*f*> This option may only be used in combination with the `-p` option. Read a "mask" from file <*f*>. The mask file must consist of a single line, of only '0' and '1' characters. There must be exactly RFLEN characters where RFLEN is the number of nongap characters in the RF annotation of all alignments in both *trusted_file* and *test_file*. Positions of the mask that are '1' characters indicate that the corresponding nongap RF position is included by the mask. The posterior probability accuracy statistics for match columns will only pertain to positions that are included by the mask, those that are excluded will be ignored from the accuracy calculation. `--c2dfile` <*f*> Save a 'draw file' to file <*f*> which can be read into the `esl-ssdraw` miniapp. This draw file will define two postscript pages for `esl-ssdraw`. The first page will depict the frequency of errors per match position and frequency of gaps per match position, indicated by magenta and yellow, respectively. The darker magenta, the more errors and the darker yellow, the more gaps. The second page will depict the frequency of errors in insert positions in shades of magenta, the darker the magenta the more errors in inserts after each position. See `esl-ssdraw` documentation for more information on these diagrams.

--amino Assert that *trusted_file* and *test_file* contain protein sequences.

--dna Assert that *trusted_file* and *test_file* contain DNA sequences.

--rna Assert that the *trusted_file* and *test_file* contain RNA sequences.

esl-compstruct - *calculate accuracy of RNA secondary structure predictions*

Synopsis

esl-compstruct [*options*] *trusted_file test_file*

Description

esl-compstruct evaluates the accuracy of RNA secondary structure predictions on a per-base-pair basis. The *trusted_file* contains one or more sequences with trusted (known) RNA secondary structure annotation. The *test_file* contains the same sequences, in the same order, with predicted RNA secondary structure annotation. esl-compstruct reads the structures and compares them, and calculates both the sensitivity (the number of true base pairs that are correctly predicted) and the positive predictive value (PPV; the number of predicted base pairs that are true). Results are reported for each individual sequence, and in summary for all sequences together.

Both files must contain secondary structure annotation in WUSS notation. Only SELEX and Stockholm formats support structure markup at present.

The default definition of a correctly predicted base pair is that a true pair (i,j) must exactly match a predicted pair (i,j).

Mathews and colleagues (Mathews et al., JMB 288:911-940, 1999) use a more relaxed definition. Mathews defines "correct" as follows: a true pair (i,j) is correctly predicted if any of the following pairs are predicted: (i,j), (i+1,j), (i-1,j), (i,j+1), or (i,j-1). This rule allows for "slipped helices" off by one base. The -m option activates this rule for both sensitivity and for specificity. For specificity, the rule is reversed: predicted pair (i,j) is considered to be true if the true structure contains one of the five pairs (i,j), (i+1,j), (i-1,j), (i,j+1), or (i,j-1).

Options

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

-m Use the Mathews relaxed accuracy rule (see above), instead of requiring exact prediction of base pairs.

-p Count pseudoknotted base pairs towards the accuracy, in either trusted or predicted structures. By default, pseudoknots are ignored.

Normally, only the *trusted_file* would have pseudoknot annotation, since most RNA secondary structure prediction programs do not predict pseudoknots. Using the -p option allows you to penalize the prediction program for not predicting known pseudoknots. In a case where both the *trusted_file* and the *test_file* have pseudoknot annotation, the -p option lets you count pseudoknots in evaluating the prediction accuracy. Beware, however,

the case where you use a pseudoknot-capable prediction program to generate the *test_file*, but the *trusted_file* does not have pseudoknot annotation; in this case, `-p` will penalize any predicted pseudoknots when it calculates specificity, even if they're right, because they don't appear in the trusted annotation. This is probably not what you'd want to do.

Expert Options

`--quiet` Don't print any verbose header information. (Used by regression test scripts, for example, to suppress version/date information.)

esl-construct - *describe or create a consensus secondary structure*

Synopsis

esl-construct [*options*] *msafile*

Description

esl-construct reports information on existing consensus secondary structure annotation of an alignment or derives new consensus secondary structures based on structure annotation for individual aligned sequences.

The alignment file must contain either individual sequence secondary structure annotation (Stockholm #=GR SS), consensus secondary structure annotation (Stockohlm #=GC SS_cons), or both. All structure annotation must be in WUSS notation (Vienna dot paranetheses notation will be correctly interpreted). At present, the alignment file must be in Stockholm format and contain RNA or DNA sequences.

By default, esl-construct generates lists the sequences in the alignment that have structure annotation and the number of basepairs in those structures. If the alignment also contains consensus structure annotation, the default output will list how many of the individual basepairs overlap with the consensus basepairs and how many conflict with a consensus basepair.

For the purposes of this miniapp, a basepair 'conflict' exists between two basepairs in different structures, one between columns i and j and the other between columns k and l, if (i == k and j != l) or (j == l and i != k).

esl-construct can also be used to derive a new consensus structure based on structure annotation for individual sequences in the alignment by using any of the following options: -x, -r, -c, --indi <*s*>, --ffreq <*x*>, --fmin. These are described below. All of these options require the -o <*f*> option be used as well to specify that a new alignment file <*f*> be created. Differences between the new alignment(s) and the input alignment(s) will be limited to the the consensus secondary structure (#=GC SS_cons) annotation and possibly reference (#=GC RF) annotation.

Options

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

-a List all alignment positions that are involved in at least one conflicting basepair in at least one sequence to the screen, and then exit.

-v Be verbose; with no other options, list individual sequence basepair conflicts as well as summary statistics.

-x Compute a new consensus structure as the maximally sized set of basepairs (greatest number of basepairs) chosen from all individual structures that contains o conflicts. Output

the alignment with the new SS_cons annotation. This option must be used in combination with the `-o` option.

`-r` Remove any consensus basepairs that conflict with >= 1 individual basepair and output the alignment with the new SS_cons annotation. This option must be used in combination with the `-o` option.

`-c` Define a new consensus secondary structure as the individual structure annotation that has the maximum number of consistent basepairs with the existing consensus secondary structure annotation. This option must be used in combination with the `-o` option.

`--rfc` With `-c`, set the reference annotation (#=GC RF) as the sequence whose individual structure becomes the consensus structure.

`--indi` <*s*> Define a new consensus secondary structure as the individual structure annotation from sequence named <*s*>. This option must be used in combination with the `-o` option.

`--rfindi` With `--indi` <*s*>, set the reference annotation (#=GC RF) as the sequence named <s>.

`--ffreq` <*x*> Define a new consensus structure as the set of basepairs between columns i:j that are paired in more than <*x*> fraction of the individual sequence structures. This option must be used in combination with the `-o` option.

`--fmin` Same as `--ffreq` <*x*> except find the maximal <*x*> that gives a consistent consensus structure. A consistent structure has each base (alignment position) as a member of at most 1 basepair.

`-o` <*s*>, Output the alignment(s) with new consensus structure annotation to file <*f*>.

`--pfam` With `-o`, specify that the alignment output format be Pfam format, a special type of non-interleaved Stockholm on which each sequence appears on a single line.

`-l` <*f*> Create a new file <*f*> that lists the sequences that have at least one basepair that conflicts with a consensus basepair.

`--lmax` <*n*> With `-l`, only list sequences that have more than <*n*> basepairs that conflict with the consensus structure to the list file.

esl-histplot - *collate data histogram, output xmgrace datafile*

Synopsis

esl-histplot [*options*] *datafile*

Description

esl-histplot summarizes numerical data in the input file *datafile*.

One real-numbered value is taken from each line of the input file. Each line is split into whitespace-delimited fields, and one field is converted to data. By default this is the first field; this can be changed by the -f option.

Default output is a survival plot (Prob(value > x)) in xmgrace XY data format, to stdout. Output may be directed to a file with the -o option.

If *datafile* is - (a single dash), input lines are read from stdin instead of opening a file.

Options

-f <*n*> Read data from whitespace-delimited field <*n*> on each line, instead of the first field. Fields are numbered starting from 1.

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

-o <*f*> Send output to file <*f*> instead of stdout.

esl-mask *- mask sequence residues with X's (or other characters)*

Synopsis

esl-mask [*options*] *seqfile maskfile*

Description

esl-mask reads lines from *maskfile* that give start/end coordinates for regions in each sequence in *seqfile*, masks these residues (changes them to X's), and outputs the masked sequence.

The *maskfile* is a space-delimited file. Blank lines and lines that start with '#' (comments) are ignored. Each data line contains at least three fields: *seqname*, *start*, and *end*. The *seqname* is the name of a sequence in the *seqfile*, and *start* and *end* are coordinates defining a region in that sequence. The coordinates are indexed <1..L> with respect to a sequence of length <L>.

By default, the sequence names must appear in exactly the same order and number as the sequences in the *seqfile*. This is easy to enforce, because the format of *maskfile* is also legal as a list of names for esl-sfetch, so you can always fetch a temporary sequence file with esl-sfetch and pipe that to esl-mask. (Alternatively, see the -R option for fetching from an SSI-indexed *seqfile*.)

The default is to mask the region indicated by <*start*>..<*end*>. Alternatively, everything but this region can be masked; see the -r reverse masking option.

The default is to mask residues by converting them to X's. Any other masking character can be chosen (see -m option), or alternatively, masked residues can be lowercased (see -l option).

Options

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

-l Lowercase; mask by converting masked characters to lower case and unmasked characters to upper case.

-m <*c*> Mask by converting masked residues to <*c*> instead of the default X.

-o <*f*> Send output to file <*f*> instead of stdout.

-r Reverse mask; mask everything outside the region *start..end*, as opposed to the default of masking that region.

-R Random access; fetch sequences from *seqfile* rather than requiring that sequence names in *maskfile* and *seqfile* come in exactly the same order and number. The *seqfile* must be SSI indexed (see esl-sfetch --index.)

-x <*n*> Extend all masked regions by up to <n> residues on each side. For normal masking, this means masking <*start*>-

<n>..<end>+<n>. For reverse masking, this means masking 1..<start>-1+<n> and <end>+1-<n>..L in a sequence of length L.

--informat <s> Assert that input *seqfile* is in format <s>, bypassing format autodetection. Common choices for <s> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <s> is case-insensitive (`fasta` or `FASTA` both work).

esl-reformat - *convert sequence file formats*

Synopsis

esl-reformat [*options*] *format seqfile*

Description

esl-reformat reads the sequence file *seqfile* in any supported format, reformats it into a new format specified by *format*, then outputs the reformatted text.

The *format* argument must (case-insensitively) match a supported sequence file format. Common choices for *format* include: fasta, embl, genbank. If *seqfile* is an alignment file, alignment output formats also work. Common choices include: stockholm, a2m, afa, psiblast, clustal, phylip. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (fasta or FASTA both work).

Unaligned format files cannot be reformatted to aligned formats. However, aligned formats can be reformatted to unaligned formats, in which case gap characters are simply stripped out.

Options

-d DNA; convert U's to T's, to make sure a nucleic acid sequence is shown as DNA not RNA. See -r.

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

-l Lowercase; convert all sequence residues to lower case. See -u.

-n For DNA/RNA sequences, converts any character that's not unambiguous RNA/DNA (e.g. ACGTU/acgtu) to an N. Used to convert IUPAC ambiguity codes to N's, for software that can't handle all IUPAC codes (some public RNA folding codes, for example). If the file is an alignment, gap characters are also left unchanged. If sequences are not nucleic acid sequences, this option will corrupt the data in a predictable fashion.

-o <*f*> Send output to file <*f*> instead of stdout.

-r RNA; convert T's to U's, to make sure a nucleic acid sequence is shown as RNA not DNA. See -d.

-u Uppercase; convert all sequence residues to upper case. See -l.

-x For DNA sequences, convert non-IUPAC characters (such as X's) to N's. This is for compatibility with benighted peo-

ple who insist on using X instead of the IUPAC ambiguity character N. (X is for ambiguity in an amino acid residue).

Warning: like the `-n` option, the code doesn't check that you are actually giving it DNA. It simply literally just converts non-IUPAC DNA symbols to N. So if you accidentally give it protein sequence, it will happily convert most every amino acid residue to an N.

Expert Options

`--gapsym` <*c*> Convert all gap characters to <*c*>. Used to prepare alignment files for programs with strict requirements for gap symbols. Only makes sense if the input *`seqfile`* is an alignment.

`--informat` <*s*> Assert that input *`seqfile`* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

`--mingap` If *`seqfile`* is an alignment, remove any columns that contain 100% gap or missing data characters, minimizing the overall length of the alignment. (Often useful if you've extracted a subset of aligned sequences from a larger alignment.)

`--keeprf` When used in combination with `--mingap`, never remove a column that is not a gap in the reference (#=GC RF) annotation, even if the column contains 100% gap characters in all aligned sequences. By default with `--mingap`, nongap RF columns that are 100% gaps in all sequences are removed.

`--nogap` Remove any aligned columns that contain any gap or missing data symbols at all. Useful as a prelude to phylogenetic analyses, where you only want to analyze columns containing 100% residues, so you want to strip out any columns with gaps in them. Only makes sense if the file is an alignment file.

`--wussify` Convert RNA secondary structure annotation strings (both consensus and individual) from old "KHS" format, ><, to the new WUSS notation, <>. If the notation is already in WUSS format, this option will screw it up, without warning. Only SELEX and Stockholm format files have secondary structure markup at present.

`--dewuss` Convert RNA secondary structure annotation strings from the new WUSS notation, <>, back to the old KHS format,

>><. If the annotation is already in KHS, this option will corrupt it, without warning. Only SELEX and Stockholm format files have secondary structure markup.

--fullwuss Convert RNA secondary structure annotation strings from simple (input) WUSS notation to full (output) WUSS notation.

--replace <*s*> <*s*> must be in the format <*s1*>:<*s2*> with equal numbers of characters in <*s1*> and <*s2*> separated by a ":" symbol. Each character from <*s1*> in the input file will be replaced by its counterpart (at the same position) from <*s2*>. Note that special characters in <*s*> (such as " ") may need to be prefixed by a "\" character.

--small Operate in small memory mode for input alignment files in Pfam format. If not used, each alignment is stored in memory so the required memory will be roughly the size of the largest alignment in the input file. With `--small`, input alignments are not stored in memory. This option only works in combination with `--informat pfam` and output format *pfam* or *afa*.

esl-selectn - *select random subset of lines from file*

Synopsis

esl-selectn [*options*] *nlines filename*

Description

esl-selectn selects *nlines* lines at random from file *filename* and outputs them on *stdout*.

If *filename* is - (a single dash), input is read from stdin.

Uses an efficient reservoir sampling algorithm that only requires only a single pass through *filename*, and memory storage proportional to *nlines* (and importantly, not to the size of the file *filename* itself). esl-selectn can therefore be used to create large scale statistical sampling experiments, especially in combination with other Easel miniapplications.

Options

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

--seed <*d*> Set the random number seed to <*d*>, an integer greater than 0. The default is to use the current value of time(). (As the return value of time() is likely to be in units of seconds, two calls to esl-selectn within the same second will generate exactly the same sample; this may not be what you want.)

esl-seqrange - *determine a range of sequences for one of many parallel*

processes

Synopsis

esl-sfetch [*options*] *seqfile procidx nproc*

Description

esl-seqrange reads an SSI-indexed *seqfile* and determines the range of sequence indices in that file that process number *procidx* out of *nproc* total processes should operate on during a parallel processing of *seqfile*.

The *seqfile* must be indexed first using esl-sfetch --index *seqfile*. This creates an SSI index file *seqfile*.ssi. An SSI file is required in order for esl-seqrange to work.

Sequence index ranges are calculated using a simple rule: the number of sequences for each process should be identical, or as close as possible to identical, across all processes. The lengths of the sequences are not considered (even though they probably should be).

Options

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

--informat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: fasta, embl, genbank. Alignment formats also work; common choices include: stockholm, a2m, afa, psiblast, clustal, phylip. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (fasta or FASTA both work).

esl-seqstat - *summarize contents of a sequence file*

Synopsis

esl-seqstat [*options*] *seqfile*

Description

esl-seqstat summarizes the contents of the *seqfile*. It prints the format, alphabet type, number of sequences, total number of residues, and the mean, smallest, and largest sequence length.

If *seqfile* is - (a single dash), sequence input is read from stdin.

Options

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

-a Additionally show a summary statistic line showing the name, length, and description of each individual sequence. Each of these lines is prefixed by an = character, in order to allow these lines to be easily grepped out of the output.

-c Additionally print the residue composition of the sequence file.

Expert Options

--informat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: fasta, embl, genbank. Alignment formats also work; common choices include: stockholm, a2m, afa, psiblast, clustal, phylip. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (fasta or FASTA both work).

--amino Assert that the *seqfile* contains protein sequences.

--dna Assert that the *seqfile* contains DNA sequences.

--rna Assert that the *seqfile* contains RNA sequences.

esl-sfetch - *retrieve (sub-)sequences from a sequence file*

Synopsis

esl-sfetch [*options*] *seqfile key*
(retrieve a single sequence by key)
esl-sfetch -c *from..to* [*options*] *seqfile key*
(retrieve a single subsequence by key and coords)
esl-sfetch -f [*options*] *seqfile keyfile*
(retrieve multiple sequences using a file of keys)
esl-sfetch -Cf [*options*] *seqfile subseq-coord-file*
(retrieve multiple subsequences using file of keys and coords)
esl-afetch --index *msafile*
(index a sequence file for retrievals)

Description

`esl-sfetch` retrieves one or more sequences or subsequences from *seqfile*.

The *seqfile* must be indexed using `esl-sfetch --index` *seqfile*. This creates an SSI index file *seqfile*.ssi.

To retrieve a single complete sequence, do `esl-sfetch` *seqfile key*, where *key* is the name or accession of the desired sequence.

To retrieve a single subsequence rather than a complete sequence, use the `-c` *start..end* option to provide *start* and *end* coordinates. The *start* and *end* coordinates are provided as one string, separated by any nonnumeric, nonwhitespace character or characters you like; see the `-c` option below for more details.

To retrieve more than one complete sequence at once, you may use the `-f` option, and the second command line argument will specify the name of a *keyfile* that contains a list of names or accessions, one per line; the first whitespace-delimited field on each line of this file is parsed as the name/accession.

To retrieve more than one subsequence at once, use the `-C` option in addition to `-f`, and now the second argument is parsed as a list of subsequence coordinate lines. See the `-C` option below for more details, including the format of these lines.

In DNA/RNA files, you may extract (sub-)sequences in reverse complement orientation in two different ways: either by providing a *from* coordinate that is greater than *to*, or by providing the *-r* option.

When the `-f` option is used to do multiple (sub-)sequence retrieval, the file argument may be - (a single dash), in which case the list of names/accessions (or subsequence coordinate lines) is read from standard input. However, because a standard input stream can't be SSI indexed, (sub-)sequence retrieval from stdin may be slow.

Options

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

-c *coords* Retrieve a subsequence with start and end coordinates specified by the *coords* string. This string consists of start and end coordinates separated by any nonnumeric, nonwhitespace character or characters you like; for example, `-c 23..100`, `-c 23/100`, or `-c 23-100` all work. To retrieve a suffix of a subsequence, you can omit the *end*; for example, `-c 23:` would work. To specify reverse complement (for DNA/RNA sequence), you can specify *from* greater than *to*; for example, `-c 100..23` retrieves the reverse complement strand from 100 to 23.

-f Interpret the second argument as a *keyfile* instead of as just one *key*. The first whitespace-limited field on each line of *keyfile* is interpreted as a name or accession to be fetched. This option doesn't work with the `--index` option. Any other fields on a line after the first one are ignored. Blank lines and lines beginning with # are ignored.

-o <*f*> Output retrieved sequences to a file <*f*> instead of to stdout.

-n <*s*> Rename the retrieved (sub-)sequence <*s*>. Incompatible with `-f`.

-r Reverse complement the retrieved (sub-)sequence. Only accepted for DNA/RNA sequences.

-C Multiple subsequence retrieval mode, with `-f` option (required). Specifies that the second command line argument is to be parsed as a subsequence coordinate file, consisting of lines containing four whitespace-delimited fields: *new_name*, *from*, *to*, *name/accession*. For each such line, sequence *name/accession* is found, a subsequence *from*..*to* is extracted, and the subsequence is renamed *new_name* before being output. Any other fields after the first four are ignored. Blank lines and lines beginning with # are ignored.

-O Output retrieved sequence to a file named *key*. This is a convenience for saving some typing: instead of

```
% esl-sfetch -o SRPA_HUMAN swissprot SRPA_HUMAN
```

you can just type

```
% esl-sfetch -O swissprot SRPA_HUMAN
```

The `-O` option only works if you're retrieving a single alignment; it is incompatible with `-f`.

--index Instead of retrieving a *key*, the special command `esl-sfetch --index` *seqfile* produces an SSI index of the names and accessions of the alignments in the *seqfile*. Indexing should be done once on the *seqfile* to prepare it for all future fetches.

Expert Options

`--informat` *<s>* Assert that *`seqfile`* is in format *<s>*, bypassing format autodetection. Common choices for *<s>* include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string *<s>* is case-insensitive (`fasta` or `FASTA` both work).

esl-shuffle - *shuffling sequences or generating random ones*

Synopsis

esl-shuffle [*options*] *seqfile*
(shuffle sequences)
esl-shuffle -G [*options*]
(generate random sequences)
esl-shuffle -A [*options*] *msafile*
(shuffle multiple sequence alignments columnwise)
esl-shuffle -Q [*options*] *qrna-alignment-file*
(shuffle QRNA pairwise alignments)

Description

`esl-shuffle` is capable of four different modes of operation.

By default, `esl-shuffle` reads individual sequences from *seqfile*, shuffles them, and outputs the shuffled sequences. By default, shuffling is done by preserving monoresidue composition; other options are listed below.

With the `-G` option, `esl-shuffle` generates some number of random sequences of some length in some alphabet. The `-N` option controls the number (default is 1), the `-L` option controls the length (default is 0), and the `--amino`, `--dna`, and `--rna` options control the alphabet.

With the `-A` option, `esl-shuffle` reads one or more multiple alignments from *msafile* and shuffles them columnwise.

Finally, the `-Q` option is for shuffling pairwise alignments in QRNA input files. A QRNA input file is a quasi-FASTA file, where each successive pair of sequences is interpreted as a pairwise alignment; sequences may contain gap characters (period, dash, or underscore: .-_) and these pairs of sequences must have exactly the same aligned length.

General Options

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

-o <*f*> Direct output to a file named <*f*> rather than to stdout.

-N <*n*> Generate <*n*> sequences, or <*n*> perform independent shuffles per input sequence or alignment.

-L <*n*> Generate sequences of length <*n*>, or truncate output shuffled sequences or alignments to a length of <*n*>.

Sequence Shuffling Options

These options only apply in default (sequence shuffling) mode. They are mutually exclusive.

-m Monoresidue shuffling (the default): preserve monoresidue composition exactly. Uses the Fisher/Yates algorithm (aka Knuth's "Algorithm P").

-d Diresidue shuffling; preserve diresidue composition exactly. Uses the Altschul/Erickson algorithm (Altschul and Erickson, 1986). A more efficient algorithm (Kandel and Winkler 1996) is known but has not yet been implemented in Easel.

-0 0th order Markov generation: generate a sequence of the same length with the same 0th order Markov frequencies. Such a sequence will approximately preserve the monoresidue composition of the input.

-1 1st order Markov generation: generate a sequence of the same length with the same 1st order Markov frequencies. Such a sequence will approximately preserve the diresidue composition of the input.

-r Reversal; reverse each input.

-w <*n*> Regionally shuffle the input in nonoverlapping windows of size <*n*> residues, preserving exact monoresidue composition in each window.

Multiple Alignment Shuffling Options

-b Sample columns with replacement, in order to generate a bootstrap-resampled alignment dataset.

Sequence Generation Options

One of these must be selected, if `-G` is used.

--amino Generate amino acid sequences.

--dna Generate DNA sequences.

--rna Generate RNA sequences.

Expert Options

--informat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--seed <*n*> Specify the seed for the random number generator, where the seed <*n*> is an integer greater than zero. This can be used

to make the results of `esl-shuffle` reproducible. If <*n*> is 0, the random number generator is seeded arbitrarily and stochastic simulations will vary from run to run. Arbitrary seeding (0) is the default.

esl-ssdraw - *create postscript secondary structure diagrams*

Synopsis

esl-ssdraw [*options*] *msafile postscript_template postscript_output_file*

Description

esl-ssdraw reads an existing template consensus secondary structure diagram from *postscript_template* and creates new postscript diagrams including the template structure but with positions colored differently based on alignment statistics such as frequency of gaps per position, average posterior probability per position or information content per position. Additionally, all or some of the aligned sequences can be drawn separately, with nucleotides or posterior probabilities mapped onto the corresponding positions of the consensus structure.

The alignment must be in Stockholm format with per-column reference annotation (#=GC RF). The sequences in the alignment must be RNA or DNA sequences. The *postscript_template* file must contain one page that includes <rflen> consensus nucleotides (positions), where <rflen> is the number of nongap characters in the reference (RF) annotation of the first alignment in *msafile*. The specific format required in the *postscript_template* is described below in the INPUT section. Postscript diagrams will only be created for the first alignment in *msafile*.

Output

By default (if run with zero command line options), esl-ssdraw will create a six or seven page *postscript_output_file*, with each page displaying a different alignment statistic. These pages display the alignment consensus sequence, information content per position, mutual information per position, frequency of inserts per position, average length of inserts per position, frequency of deletions (gaps) per position, and average posterior probability per position (if posterior probabilites exist in the alignment) If -d is enabled, all of these pages plus additional ones, such as individual sequences (see discussion of --indi below) will be drawn. These pages can be selected to be drawn individually by using the command line options --cons, --info, --mutinfo, --ifreq, --iavglen, --dall, and --prob. The calculation of the statistics for each of these options is discussed below in the description for each option. Importantly, only so-called 'consensus' positions of the alignment will be drawn. A consensus position is one that is a nongap nucleotide in the 'reference' annotation of the Stockholm alignment (#=GC RF) read from *msafile*.

By default, a consensus sequence for the input alignment will be calculated and displayed on the alignment statistic diagrams. The consensus sequence is defined as the most common nucleotide at each consensus position of the alignment. The consensus sequence will not be displayed if the --no-cnt option is used. The --cthresh, --cambig, and --athresh options affect the definition of the consensus sequence as explained below in the descriptions for those options.

If the `--tabfile <f>` option is used, a tab-delimited text file <f> will be created that includes per-position lists of the numerical values for each of the calculated statistics that were drawn to *postscript_output_file*. Comment lines in <f> are prefixed with a '#' character and explain the meaning of each of the tab-delimited columns and how each of the statistics was calculated. If `--indi` is used, `esl-ssdraw` will create diagrams showing each sequence in the alignment on a separate page, with aligned nucleotides in their corresponding position in the structure diagram. By default, basepaired nucleotides will be colored based on their basepair type: either Watson-Crick (A:U, U:A, C:G, or G:C), G:U or U:G, or non-canonical (the other ten possible basepairs). This coloring can be turned off with the `--no-bp` option. Also by default, nucleotides that differ from the most common nucleotide at each aligned consensus position will be outlined. If the most common nucleotide occurs in more than 75% of sequences that do not have a gap at that position, the outline will be bold. Outlining can be turned off with the `--no-ol` option.

With `--indi`, if the alignment contains posterior probability annotation (#=GR PP), the *postscript_output_file* will contain an additional page for each sequence drawn with positions colored by the posterior probability of each aligned nucleotide. No posterior probability pages will be drawn if the `--no-pp` option is used.

`esl-ssdraw` can also be used to draw 'mask' diagrams which color positions of the structure one of two colors depending on if they are included or excluded by a mask. This is enabled with the `--mask-col <f>` option. <f> must contain a single line of <rflen> characters, where <rflen> is the the number of nongap RF characters in the alignment. The line must contain only '0' and '1' characters. A '0' at position <x> of the string indicates position <x> is excluded from the mask, and a '1' indicates position <x> is included by the mask. A page comparing the overlap of the <f> mask from `--mask-col` and another mask in <f2> will be created if the `--mask-diff <f2>` option is used.

If the `--mask <f>` option is used, positions excluded by the mask in <f> will be drawn differently (as open circles by default) than positions included by the mask. The style of the masked positions can be modified with the `--mask-u`, `--mask-x`, and `--mask-a` options.

Finally, two different types of input files can be used to customize output diagrams using the `--dfile` and `--efile` options, as described below.

Input

The *postscript_template_file* is a postscript file that must be in a very specific format in order for `esl-ssdraw` to work. The specifics of the format, described below, are likely to change in future versions of `esl-ssdraw`. The *postscript_output_file* files generated by `esl-ssdraw` will not be valid *postscript_template_file* format (i.e. an output file from `esl-ssdraw` cannot be used as an *postscript_template_file* in a subsequent run of the program).

An example *postscript_template_file* ('trna-ssdraw.ps') is included with the Easel distribution in the 'testsuite/' subdirectory of the top-level 'easel' directory.

The *postscript_template_file* is a valid postscript file. It includes postscript commands for drawing a secondary structure. The commands specify x and y coordinates for placing each nucleotide on the page. The *postscript_template_file* might also contain commands for drawing lines connecting basepaired positions and tick marks indicating every tenth position, though these are not required, as explained below.

If you are unfamiliar with the postscript language, it may be useful for you to know that a postscript page is, by default, 612 points wide and 792 points tall. The (0,0) coordinate of a postscript file is at the bottom left corner of the page, (0,792) is the top left, (612,0) is the bottom right, and (612,792) is the top right. `esl-ssdraw` uses 8 point by 8 point cells for drawing positions of the consensus secondary structure. The 'scale' section of the *postscript_template_file* allows for different 'zoom levels', as described below. Also, it is important to know that postscript lines beginning with '%' are considered comments and do not include postscript commands.

An `esl-ssdraw` *postscript_template_file* contains n >= 1 pages, each specifying a consensus secondary structure diagram. Each page is delimited by a 'showpage' line in an 'ignore' section (as described below). `esl-ssdraw` will read all pages of the *postscript_template_file* and then choose the appropriate one that corresponds with the alignment in *msafile* based on the consensus (nongap RF) length of the alignment. For an alignment of consensus length <rflen>, the first page of *postscript_template_file* that has a structure diagram with consensus length <rflen> will be used as the template structure for the alignment.

Each page of *postscript_template_file* contains blocks of text organized into seven different possible sections. Each section must begin with a single line '% begin <sectionname>' and end with a single line '% end <sectionname>' and have n >= 1 lines in between. On the begin and end lines, there must be at least one space between the '%' and the 'begin' or 'end'. <sectionname> must be one of the following: 'modelname', 'legend', 'scale', 'regurgitate', 'ignore', 'text positiontext', 'text nucleotides', 'lines positionticks', or 'lines bpconnects'. The n >=1 lines in between the begin and end lines of each section must be in a specific format that differs for each section as described below.

Importantly, each page must end with an 'ignore' section that includes a single line 'showpage' between the begin and end lines. This lets `esl-ssdraw` know that a page has ended and another might follow.

Each page of a *postscript_template_file* must include a single 'modelname' section. This section must include exactly one line in between its begin and end lines. This line must begin with a '%' character followed by a single space. The remainder of the line will be parsed as the model name and will appear on each page of `postscript_output_file` in the header section. If the name is more than 16 characters, it will be truncated in the output.

Each page of a *postscript_template_file* must include a single 'legend' section. This section must include exactly one line in between its begin and end lines. This line must be formatted as '% <d1> <f1> <f2> <d2> <f3>', where <d1> is an integer specifying the consensus position with relation to which the legend will be placed; <f1> and <f2> specify the x and y axis offsets for the top left corner of the legend

relative to the x and y position of consensus position <d1>; <d2> specifies the size of a cell in the legend and <f3> specifies how many extra points should be between the right hand edge of the legend and the end of the page. the offset of the right hand end of the legend . For example, the line '% 34 -40. -30. 12 0.' specfies that the legend be placed 40 points to the left and 30 points below the 34th consensus position, that cells appearing in the legend be squares of size 12 points by 12 points, and that the right hand side of the legend flush against the right hand edge of the printable page.

Each page of a *postscript_template_file* must include a single 'scale' section. This section must include exactly one line in between its begin and end lines. This line must be formatted as '<f1> <f2> scale', where <f1> and <f2> are both positive real numbers that are identical, for example '1.7 1.7 scale' is valid, but '1.7 2.7 scale' is not. This line is a valid postscript command which specifies the scale or zoom level on the pages in the output. If <f1> and <f2> are '1.0' the default scale is used for which the total size of the page is 612 points wide and 792 points tall. A scale of 2.0 will reduce this to 306 points wide by 396 points tall. A scale of 0.5 will increase it to 1224 points wide by 1584 points tall. A single cell corresponding to one position of the secondary structure is 8 points by 8 points. For larger RNAs, a scale of less than 1.0 is appropriate (for example, SSU rRNA models (about 1500 nt) use a scale of about 0.6), and for smaller RNAs, a scale of more than 1.0 might be desirable (tRNA (about 70 nt) uses a scale of 1.7). The best way to determine the exact scale to use is trial and error.

Each page of a *postscript_template_file* can include n >= 0 'regurgitate' sections. These sections can include any number of lines. The text in this section will not be parsed by `esl-ssdraw` but will be included in each page of *postscript_output_file*. The format of the lines in this section must therefore be valid postscript commands. An example of content that might be in a regurgitate section are commands to draw lines and text annotating the anticodon on a tRNA secondary structure diagram.

Each page of a *postscript_template_file* must include at least 1 'ignore' section. One of these sections must include a single line that reads 'showpage'. This section should be placed at the end of each page of the template file. Other ignore sections can include any number of lines. The text in these section will not be parsed by `esl-ssdraw` nor will it be included in each page of *postscript_output_file*. An ignore section can contain comments or postscript commands that draw features of the *postscript_template_file* that are unwanted in the *postscript_output_file*.

Each page of a *postscript_template_file* must include a single 'text nucleotides' section. This section must include exactly <rflen> lines, indicating that the consensus secondary structure has exactly <rflen> nucleotide positions. Each line must be of the format '(<c>) <x> <y> moveto show' where <c> is a nucleotide (this can be any character actually), and <x> and <y> are the coordinates specifying the location of the nucleotide on the page, they should be positive real numbers. The best way to determine what these coordinates should be is manually by trial and error, by inspecting the resulting structure as you add each nucleotide. Note that `esl-ssdraw` will color an 8 point by 8 point cell for each position, so nucleotides should be placed about 8 points apart from each other.

Each page of a *postscript_template_file* may or may not include a single 'text positiontext' section. This section can include n >= 1 lines, each specifying text to be placed next to specific positions of the structure, for example, to number them. Each line must be of the format '(<s>) <x> <y> moveto show' where <s> is a string of text to place at coordinates (<x>,<y>) of the postscript page. Currently, the best way to determine what these coordinates is manually by trial and error, by inspecting the resulting diagram as you add each line.

Each page of a *postscript_template_file* may or may not include a single 'lines positionticks' section. This section can include n >= 1 lines, each specifying the location of a tick mark on the diagram. Each line must be of the format '<x1> <y1> <x2> <y2> moveto show'. A tick mark (line of width 2.0) will be drawn from point (<x1>,<y1>) to point (<x2>,<y2>) on each page of *postscript_output_file*. Currently, the best way to determine what these coordinates should be is manually by trial and error, by inspecting the resulting diagram as you add each line.

Each page of a *postscript_template_file* may or may not include a single 'lines bpconnects' section. This section must include <nbp> lines, where <nbp> is the number of basepairs in the consensus structure of the input *msafile* annotated as #=GC SS_cons. Each line should connect two basepaired positions in the consensus structure diagram. Each line must be of the format '<x1> <y1> <x2> <y2> moveto show'. A line will be drawn from point (<x1>,<y1>) to point (<x2>,<y2>) on each page of *postscript_output_file*. Currently, the best way to determine what these coordinates should be is manually by trial and error, by inspecting the resulting diagram as you add each line.

Required Memory

The memory required by `esl-ssdraw` will be equal to roughly the larger of 2 Mb and the size of the first alignment in *msafile*. If the `--small` option is used, the memory required will be independent of the alignment size. To use `--small` the alignment must be in Pfam format, a non-interleaved (1 line/seq) version of Stockholm format. If the `--indi` option is used, the required memory may exceed the size of the alignment by up to ten-fold, and the output `postscript_output_file` may be up to 50 times larger than the `msafile`.

Options

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

-d Draw the default set of alignment summary diagrams: consensus sequence, information content, mutual information, insert frequency, average insert length, deletion frequency, and average posterior probability (if posterior probability annotation exists in the alignment). These diagrams are also drawn by default (if zero command line options are used),

but using the `-d` option allows the user to add additional pages, such as individual aligned sequences with `--indi`.

`--mask` *<f>* Read the mask from file *<f>*, and draw positions differently in *postscript_output_file* depending on whether they are included or excluded by the mask. *<f>* must contain a single line of length <rflen> with only '0' and '1' characters. <rflen> is the number of nongap characters in the reference (#=GC RF) annotation of the first alignment in *msafile* A '0' at position <x> of the mask indicates position <x> is excluded by the mask, and a '1' indicates that position <x> is included by the mask.

`--small` Operate in memory saving mode. Without `--indi`, required RAM will be independent of the size of the alignment in *msafile*. With `--indi`, the required RAM will be roughly ten times the size of the alignment in *msafile*. For `--small` to work, the alignment must be in Pfam Stockholm (non-interleaved 1 line/seq) format.

`--rf` Add a page to *postscript_output_file* showing the reference sequence from the #=GC RF annotation in *msafile*. By default, basepaired nucleotides will be colored based on what type of basepair they are. To turn this off, use `--no-bp`. This page is drawn by default (if zero command-line options are used).

`--info` Add a page to *postscript_output_file* with consensus (nongap RF) positions colored based on their information content from the alignment. Information content is calculated as 2.0 - H, where H = sum_x p_x log_2 p_x for x in {A,C,G,U}. This page is drawn by default (if zero command-line options are used).

`--mutinfo` Add a page to *postscript_output_file* with basepaired consensus (nongap RF) positions colored based on the amount of mutual information they have in the alignment. Mutual information is sum_{x,y} p_{x,y} log_2 ((p_x * p_y) / p_{x,y}, where x and y are the four possible bases A,C,G,U. p_x is the fractions of aligned sequences that have nucleotide x of in the left half (5' half) of the basepair. p_y is the fraction of aligned sequences that have nucleotide y in the position corresponding to the right half (3' half) of the basepair. And p_{x,y} is the fraction of aligned sequences that have basepair x:y. For all p_x, p_y and p{x,y} only sequences that that have a nongap nucleotide at both the left and right half of the basepair are counted. This page is drawn by default (if zero command-line options are used).

--ifreq Add a page to *postscript_output_file* with each consensus (nongap RF) position colored based on the fraction of sequences that span each position that have at least 1 inserted nucleotide after the position. A sequence s spans consensus position x that is actual alignment position a if s has at least one nongap nucleotide aligned to a position b <= a and at least one nongap nucleotide aligned to a consensus position c >= a. This page is drawn by default (if zero command-line options are used).

--iavglen Add a page to *postscript_output_file* with each consensus (nongap RF) position colored based on average length of insertions that occur after it. The average is calculated as the total number of inserted nucleotides after position x, divided by the number of sequences that have at least 1 inserted nucleotide after position x (so the minimum possible average insert length is 1.0).

--dall Add a page to *postscript_output_file* with each consensus (nongap RF) position colored based on the fraction of sequences that have a gap (delete) at the position. This page is drawn by default (if zero command-line options are used).

--dint Add a page to *postscript_output_file* with each consensus (nongap RF) position colored based on the fraction of sequences that have an internal gap (delete) at the position. An internal gap in a sequence is one that occurs after (5′ of) the sequence's first aligned nucleotide and after (3′ of) the sequence's final aligned nucleotide. This page is drawn by default (if zero command-line options are used).

--prob Add a page to *postscript_output_file* with positions colored based on average posterior probability (PP). The alignment must contain #=GR PP annotation for all sequences. PP annotation is converted to numerical PP values as follows: '*' = 0.975, '9' = 0.90, '8' = 0.80, '7' = 0.70, '6' = 0.60, '5' = 0.50, '4' = 0.40, '3' = 0.30, '2' = 0.20, '1' = 0.10, '0' = 0.025. This page is drawn by default (if zero command-line options are used).

--span Add a page to *postscript_output_file* with consensus (nongap RF) positions colored based on the fraction of sequences that 'span' the position. A sequence s spans consensus position x that is actual alignment position a if s has at least one nongap nucleotide aligned to a position b <= a and at least one nongap nucleotide aligned to a consensus position c >= a. This page is drawn by default (if zero command-line options are used).

Options for Drawing Individual Aligned Sequences

--indi Add a page displaying the aligned nucleotides in their corresponding consensus positions of the structure diagram for each aligned sequence in the alignment. By default, basepaired nucleotides will be colored based on what type of basepair they are. To turn this off, use `--no-bp`. If posterior probability information (#=GR PP) exists in the alignment, one additional page per sequence will be drawn displaying the posterior probabilities.

-f With `--indi`, force `esl-ssdraw` to create a diagram, even if it is predicted to be large (> 100 Mb). By default, if the predicted size exceeds 100 Mb, `esl-ssdraw` will fail with a warning.

Options for Omitting Parts of the Diagrams

--no-leg Omit the legend on all pages of *postscript_output_file*.

--no-head Omit the header on all pages of *postscript_output_file*.

--no-foot Omit the footer on all pages of *postscript_output_file*.

Options for Simple Two-color Mask Diagrams

--mask-col With `--mask`, *postscript_output_file* will contain exactly 1 page showing positions included by the mask as black squares, and positions excluded as pink squares.

--mask-diff <*f*> With `--mask` <*f2*> and `mask-col`, *postscript_output_file* will contain one additional page comparing the mask from <*f*> and the mask from <*f2*>. Positions will be colored based on whether they are included by one mask and not the other, excluded by both masks, and included by both masks.

Expert Options for Controlling Individual Sequence Diagrams

--no-pp When used in combination with `--indi`, do not draw posterior probability structure diagrams for each sequence, even if the alignment has PP annotation.

--no-bp Do not color basepaired nucleotides based on their basepair type.

--no-ol When used in combination with `--indi`, do not outline nucleotides that differ from the majority rule consensus nucleotide given the alignment.

--no-ntpp When used in combination with `--indi`, do not draw nucleotides on the individual sequence posterior probability diagrams.

Expert Options Related to Consensus Sequence Definition

--no-cnt Do not draw consensus nucleotides on alignment statistic diagrams (such as information content diagrams). By default, the consensus nucleotide is defined as the most frequent nucleotide in the alignment at the corresponding position. Consensus nucleotides that occur in at least <*x*> fraction of the aligned sequences (that do not contain a gap at the position) are capitalized. By default <*x*> is 0.75, but can be changed with the `--cthresh` <*x*> option.

--cthresh <*x*> Specify the threshold for capitalizing consensus nucleotides defined by the majority rule (i.e. when `--cambig` is not enabled) as <*x*>.

--cambig Change how consensus nucleotides are calculated from majority rule to the least ambiguous IUPAC nucleotide that represents at least <*x*> fraction of the nongap nucleotides at each consensus position. By default <*x*> is 0.9, but can be changed with the `--athresh` <*x*> option.

--athresh <*x*> With `--cambig`, specify the threshold for defining consensus nucleotides is the least ambiguous IUPAC nucleotide that represents at least <*x*> fraction of the nongap nucleotides at each position.

Expert Options Controlling Style of Masking Positions

--mask-u With `--mask`, change the style of masked columns to squares.

--mask-x With `--mask`, change the style of masked columns to x's.

--mask-a With `--mask` and `--mask-u` or `--mask-x` draw the alternative style of square or 'x' masks.

Expert Options Related to Input Files

--dfile <*f*> Read the 'draw file' <*f*> which specifies numerical values for each consensus position in one or more postscript pages. For each page, the draw file must include <rflen>+3 lines (<rflen> is defined in the DESCRIPTION section). The first three lines are special. The following <rflen> 'value lines' each must contain a single number, the numerical value for the corresponding position. The first of the three special lines defines the 'description' for the page. This should be text that describes what the numerical values refer to for the page. The maximum allowable length is roughly 50 characters (the exact maximum length depends on the template

file and the program will report an informative error message upon execution if it is exceeded). The second special line defines the 'legend header' line that which will appear immediately above the legend. It has a maximum allowable length of about 30 characters. The third special line per page must contain exactly 7 numbers, which must be in increasing order, each separated by a space. These numbers define the numerical ranges for the six different colors used to draw the consensus positions on the page. The first number defines the minimum value for the first color (blue) and must be less than or equal to the minimum value from the value lines. The second number defines the minimum value for the second color (turquoise). The third, fourth, fifth and sixth numbers define the minimum values for the third, fourth, fifth and sixth colors (light green, yellow, orange, red), and the seventh final number defines the maximum value for red and must be equal to or greater than the maximum value from the value lines. After the <rflen> value lines, there must exist a special line with only '//', signifying the end of a page. The draw file <*f*> must end with this special '//' line, even if it only includes a single page. A draw file specifying <n> pages should include exactly <n> * (<rflen> + 4) lines.

--efile <*f*> Read the 'expert draw file' <*f*> which specifies the colors and nucleotides to draw on each consensus position in one or more postscript pages. Unlike with the `--dfile` option, no legend will be drawn when `--efile` is used. For each page, the draw file must include <rflen> lines, each with four or five tab-delimited tokens. The first four tokens on line <x> specify the color to paint position <x> and must be real numbers between 0 and 1. The four numbers specify the cyan, magenta, yellow and black values, respectively, in the CMYK color scheme for the postscript file. The fifth token on line <x> specifies which nucleotide to write on position <x> (on top of the colored background). If the fifth token does not exist, no nucleotide will be written. After the <rflen> lines, there must exist a special line with only '//', signifying the end of a page. The expert draw file <*f*> must end with this special '//' line, even if it only includes a single page. A expert draw file specifying <n> pages should include exactly <n> * (<rflen> + 1) lines.

--ifile <*f*> Read insert information from the file <*f*>, which may have been created with INFERNAL's `cmalign`(1) program. The insert information in *`msafile`* will be ignored and the information

from <f> will supersede it. Inserts are columns that are gaps in the reference (#=GC RF) annotation.

esl-translate - *translate DNA sequence in six frames into individual*

ORFs

Synopsis

esl-translate [*options*] *seqfile*

Description

Given a *seqfile* containing DNA or RNA sequences, esl-translate outputs a six-frame translation of them as individual open reading frames in FASTA format.

By default, only open reading frames greater than 20aa are reported. This minimum ORF length can be changed with the -l option.

By default, no specific initiation codon is required, and any amino acid can start an open reading frame. This is so esl-translate may be used on sequence fragments, eukaryotic genes with introns, or other cases where we do not want to assume that ORFs are complete coding regions. This behavior can be changed. With the -m option, ORFs start with an initiator AUG Met. With the -M option, ORFs start with any of the initiation codons allowed by the genetic code. For example, the "standard" code (NCBI transl_table 1) allows AUG, CUG, and UUG as initiators. When -m or -M are used, an initiator is always translated to Met (even if the initiator is something like UUG or CUG that doesn't encode Met as an elongator).

If *seqfile* is - (a single dash), input is read from the stdin pipe. This (combined with the output being a standard FASTA file) allows esl-translate to be used in command line incantations. If *seqfile* ends in .gz, it is assumed to be a gzip-compressed file, and Easel will try to read it as a stream from gunzip -c.

Output Format

The output FASTA name/description line contains information about the source and coordinates of each ORF. Each ORF is named orf1, etc., with numbering starting from 1, in order of their start position on the top strand followed by the bottom strand. The rest of the FASTA name/desc line contains 4 additional fields, followed by the description of the source sequence:

source=<*s*>	<*s*> is the name of the source DNA/RNA sequence.
coords=*start*..*end*	Coords, 1..L, for the translated ORF in a source DNA sequence of length L. If start is greater than end, the ORF is on the bottom (reverse complement) strand. The start is the first nucleotide of the first codon; the end is the last nucleotide of the last codon. The stop codon is not included in the coordinates (unlike in CDS annotation in GenBank, for example.)
length=<*n*>	Length of the ORF in amino acids.

frame=<*n*> Which frame the ORF is in. Frames 1..3 are the top strand; 4..6 are the bottom strand. Frame 1 starts at nucleotide 1. Frame 4 starts at nucleotide L.

Alternative Genetic Codes

By default, the "standard" genetic code is used (NCBI transl_table 1). Any NCBI genetic code transl_table can be selected with the `-c` option, as follows:

1 Standard

2 Vertebrate mitochondrial

3 Yeast mitochondrial

4 Mold, protozoan, coelenterate mitochondrial; Mycoplasma/Spiroplasma

5 Invertebrate mitochondrial

6 Ciliate, dasycladacean, Hexamita nuclear

9 Echinoderm and flatworm mitochondrial

10 Euplotid nuclear

11 Bacterial, archaeal; and plant plastid

12 Alternative yeast

13 Ascidian mitochondrial

14 Alternative flatworm mitochondrial

16 Chlorophycean mitochondrial

21 Trematode mitochondrial

22 Scenedesmus obliquus mitochondrial

23 Thraustochytrium mitochondrial

24 Pterobranchia mitochondrial

25 Candidate Division SR1 and Gracilibacteria

As of this writing, more information about the genetic codes in the NCBI translation tables is at *http://www.ncbi.nlm.nih.gov/Taxonomy/* at a link titled *Genetic* codes.

Iupac Degeneracy Codes in Dna

DNA sequences may contain IUPAC degeneracy codes, such as N, R, Y, etc. If all codons consistent with a degenerate codon translate to the same amino acid (or to a stop), that translation is done; otherwise, the codon is translated as X (even if one or more compatible codons are stops). For example, in the standard code, UAR translates to * (stop), GGN translates to G (glycine), NNN translates to X, and UGR translates to X (it could be either a UGA stop or a UGG Trp).

Degenerate initiation codons are handled essentially the same. If all codons consistent with the degenerate codon are legal initiators, then the codon is allowed to

initiate a new ORF. Stop codons are never a legal initiator (not only with `-m` or `-M` but also with the default of allowing any amino acid to initiate), so degenerate codons consistent with a stop cannot be initiators. For example, NNN cannot initiate an ORF, nor can UGR -- even though they translate to X. This means that we don't translate long stretches of N's as long ORFs of X's, which is probably a feature, given the prevalence of artificial runs of N's in genome sequence assemblies.

Degenerate DNA codons are not translated to degenerate amino acids other than X, even when that is possible. For example, SAR and MUH are decoded as X, not Z (Q|E) and J (I|L). The extra complexity needed for a degenerate to degenerate translation doesn't seem worthwhile.

Options

-h Print brief help. Includes version number and summary of all options. Also includes a list of the available NCBI transl_tables and their numerical codes, for the `-c` option.

-c <*id*> Choose alternative genetic code <*id*> where <*id*> is the numerical code of one of the NCBI transl_tables.

-l <*n*> Set the minimum reported ORF length to <*n*> aa.

-m Require ORFs to start with an initiator codon AUG (Met).

-M Require ORFs to start with an initiator codon, as specified by the allowed initiator codons in the NCBI transl_table. In the default Standard code, AUG, CUG, and UUG are allowed as initiators. An initiation codon is always translated as Met, even if it does not normally encode Met as an elongator.

-W Use a memory-efficient windowed sequence reader. The default is to read entire DNA sequences into memory, which may become memory limited for some very large eukaryotic chromosomes. The windowed reader cannot reverse complement a nonrewindable input stream, so either *seqfile* must be a file, or you must use *--watson* to limit translation to the top strand.

--informat <*s*> Assert that input *seqfile* is in format <*s*>, bypassing format autodetection. Common choices for <*s*> include: `fasta`, `embl`, `genbank`. Alignment formats also work; common choices include: `stockholm`, `a2m`, `afa`, `psiblast`, `clustal`, `phylip`. For more information, and for codes for some less common formats, see main documentation. The string <*s*> is case-insensitive (`fasta` or `FASTA` both work).

--watson Only translate the top strand.

--crick Only translate the bottom strand.

esl-weight - *calculate sequence weights in MSA(s)*

Synopsis

esl-weight [*options*] *msafile*

Description

esl-weight calculates individual sequence weights for each alignment in *msafile* and outputs a new multiple sequence alignment file in Stockholm format with the weights annotated in Stockholm-format #=GS *seqname* WT *weight* lines. The default weighting algorithm is the Gerstein/Sonnhammer/Chothia algorithm.

If *msafile* is - (a single dash), MSA input is read from stdin.

Options

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

-g Use the Gerstein/Sonnhammer/Chothia weighting algorithm; this is the default.

-p Use the Henikoff position-based weighting algorithm. This is faster and more memory efficient than the default.

-b "BLOSUM weights": use approximately the same rule used in constructing the BLOSUM score matrices. This involves single-linkage clustering at some fractional identity threshold (default 0.62; see --id option), then for each cluster, splitting a total weight of one uniformly amongst all sequences in the cluster.

Expert Options

--id <*x*> Sets the fractional identity threshold used by the BLOSUM weighting rule (option -b; required), to a number 0<=x<=1. Default is 0.62.

--amino Assert that the *msafile* contains protein sequences.

--dna Assert that the *msafile* contains DNA sequences.

--rna Assert that the *msafile* contains RNA sequences.

Input files and formats

Reading from files, compressed files, and pipes

Generally, HMMER programs read their sequence and/or profile input from files. Unix power users often find it convenient to string an incantation of commands together with pipes.[1] For example, you might extract a subset of query sequences from a larger file using a one-liner combination of scripting commands (python, perl, awk, whatever). To facilitate the use of HMMER programs in such incantations, you can almost always use an argument of '-' (dash) in place of a filename, and the program will take its input from a standard input pipe instead of opening a file.

[1] Indeed, such wizardly incantations are a point of pride.

For example, the following three commands are equivalent, and give essentially identical output:

```
% hmmsearch globins4.hmm uniprot_sprot.fasta
% cat globins4.hmm | hmmsearch - uniprot_sprot.fasta
% cat uniprot_sprot.fasta | hmmsearch globins4.hmm -
```

Most Easel "miniapp" programs share the same pipe-reading ability.

Because the programs for profile HMM fetching (`hmmfetch`) and sequence fetching (`esl-sfetch`) can fetch any number of profiles or sequences by names/accessions given in a list, *and* these programs can also read these lists from a stdin pipe, you can craft incantations that generate subsets of queries or targets on the fly. For example:

```
% esl-sfetch -index uniprot_sprot.fasta
% cat mytargs.list | esl-sfetch -f uniprot_sprot.fasta - | hmmsearch globins4.hmm -
```

This takes a list of sequence names/accessions in `mytargets.list`, fetches them one by one from UniProt (note that we index the UniProt file first, for fast retrieval; and note that `esl-sfetch` is reading its `<namefile>` list of names/accessions through a pipe using the '-' argument), and pipes them to an `hmmsearch`. It should be obvious from this that we can replace the `cat mytargets.list` with *any* incantation that generates a list of sequence names/accessions (including SQL

database queries).

Ditto for piping subsets of profiles. Supposing you have a copy of Pfam in Pfam-A.hmm:

```
% hmmfetch -index Pfam-A.hmm
% cat myqueries.list | hmmfetch -f Pfam.hmm - | hmmsearch - uniprot_sprot.fasta
```

This takes a list of query profile names/accessions in `myqueries.list`, fetches them one by one from Pfam, and does an hmmsearch with each of them against UniProt. As above, the `cat myqueries.list` part can be replaced by any suitable incantation that generates a list of profile names/accessions.

There are three kinds of cases where using '-' is restricted or doesn't work. A fairly obvious restriction is that you can only use one '-' per command; you can't do a `hmmsearch - -` that tries to read both profile queries and sequence targets through the same stdin pipe. Second, another case is when an input file must be obligately associated with additional, separately generated auxiliary files, so reading data from a single stream using '-' doesn't work because the auxiliary files aren't present (in this case, using '-' will be prohibited by the program). An example is `hmmscan`, which needs its `<hmmfile>` argument to be associated with four auxiliary files named `<hmmfile>.h3{mifp}` that `hmmpress` creates, so `hmmscan` does not permit a '-' for its `<hmmfile>` argument. Finally, when a command would require multiple passes over an input file, the command will generally abort after the first pass if you are trying to read that file through a standard input pipe (pipes are nonrewindable in general; a few HMMER or Easel programs will buffer input streams to make multiple passes possible, but this is not usually the case). An example would be trying to search a file containing multiple profile queries against a streamed target database:

```
% cat myqueries.list | hmmfetch -f Pfam.hmm > many.hmms
% cat mytargets.list | esl-sfetch -f uniprot_sprot.fasta - | hmmsearch many.hmms -
```

This will fail. Unfortunately the above business about how it will "generally abort after the first pass" means it fails weirdly. The first query profile search will succeed, and its output will appear; then an error message will be generated when `hmmsearch` sees the *second* profile query and oops, suddenly realizes it is unable to rewind the target sequence database stream. This is inherent in how it reads the profile HMM query file sequentially as a stream (which is what's allowing it to read input from stdin pipes in the first place), one model at a time: it doesn't see there's more than one query model in the file until it gets to the second model.

This case isn't too restricting because the same end goal can be

achieved by reordering the commands. In cases where you want to do multiple queries against multiple targets, you always want to be reading the *queries* from a stdin pipe, not the targets:

```
% cat mytargets.list | esl-sfetch -f uniprot_sprot.fasta > mytarget.seqs
% cat myqueries.list | hmmfetch -f Pfam.hmm - | hmmsearch - mytarget.seqs
```

So in this multiple queries/multiple targets case of using stdin pipes, you just have to know, for any given program, which file it considers to be queries and which it considers to be targets. (That is, the logic in searching many queries against many targets is "For each query: search the target database; then rewind the target database to the beginning.") For `hmmsearch`, the profiles are queries and sequences are targets. For `hmmscan`, the reverse.

In general, HMMER and Easel programs document in their man page whether (and which) command line arguments can be replaced by '-'. You can always check by trial and error, too. The worst that can happen is a "Failed to open file -" error message, if the program can't read from pipes.

.gz compressed files

In general, HMMER programs and Easel miniapps can also read `.gz` compressed files; they will uncompress them on the fly. You need to have `gunzip` installed on your system for this to work.

HMMER profile HMM files

A HMMER profile file looks like this, with ...'s marking elisions made for clarity and space:[2]

[2] This is the `globins4.hmm` profile from the tutorial.

```
HMMER3/f [3.2 | June 2018]
NAME  globins4
LENG  149
ALPH  amino
RF    no
MM    no
CONS  yes
CS    no
MAP   yes
DATE  Tue May 29 20:41:39 2018
NSEQ  4
EFFN  0.964844
CKSUM 2027839109
STATS LOCAL MSV       -9.9014  0.70957
STATS LOCAL VITERBI  -10.7224  0.70957
STATS LOCAL FORWARD   -4.1637  0.70957
HMM          A        C        D        E        F        G        H    ...      W        Y
            m->m     m->i     m->d     i->m     i->i     d->m     d->d
  COMPO   2.36553  4.52577  2.96709  2.70473  3.20818  3.02239  3.41069 ...   4.55393  3.62921
          2.68640  4.42247  2.77497  2.73145  3.46376  2.40504  3.72516 ...   4.58499  3.61525
          0.57544  1.78073  1.31293  1.75577  0.18968  0.00000        *
      1   1.70038  4.17733  3.76164  3.36686  3.72281  3.29583  4.27570 ...   5.32720  4.10031      9 v - - -
          2.68618  4.42225  2.77519  2.73123  3.46354  2.40513  3.72494 ...   4.58477  3.61503
          0.03156  3.86736  4.58970  0.61958  0.77255  0.34406  1.23405
...
    149   2.92198  5.11574  3.28049  2.65489  4.47826  3.59727  2.51142 ...   5.42147  4.18835    165 k - - -
          2.68634  4.42241  2.77536  2.73098  3.46370  2.40469  3.72511 ...   4.58493  3.61418
          0.22163  1.61553        *  1.50361  0.25145  0.00000        *
//
```

A profile file consists of one or more profiles. Each profile starts with a format version identifier (here, `HMMER3/f`) and ends with `//` on a line by itself. The format version identifier allows backward compatibility as the HMMER software evolves: it tells the parser this file is from HMMER3's save file format version f.[3] The closing `//` allows multiple profiles to be concatenated.

[3] HMMER 3.0 used 3/b format. HMMER 3.1 and 3.2 use 3/f format. Some alpha test versions of 3.0 used 3/a format. Internal development versions of 3.1 used 3/c, 3/d, and 3/e formats.

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

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

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

A more detailed description of the format follows.

header section

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

HMMER3/f Unique identifier for the save file format version; the `/f` means that this is HMMER3 profile file format version f. When HMMER3 changes its save file format, the revision code advances. This way, parsers can be backwards compatible. The remainder of the line after the `HMMER3/f` tag is free text that is ignored by the parser. HMMER currently writes its version number and release date in brackets here, e.g. `[3.2 | June 2018]` in this example. **Mandatory.**

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

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

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

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

MAXL <d> Max instance length; `<d>`, a positive nonzero integer, is the upper bound on the length at which and instance of the model is expected to be found. Used only by nhmmer and nhmmscan. **Optional.**

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

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

MM <s> Model masked flag; <s> is either no or yes (case insensitive). If yes, the model mask annotation character field for each match state in the main model (see below) is valid; if no, these characters are ignored. Indicates that the profile model was created such that emission probabilities at masked positions are set to match the background frequency, rather than being set based on observed frequencies in the alignment. Position-specific insertion and deletion rates are not altered, even in masked regions. **Optional**; assumed to be no if not present.

CONS <s> Consensus residue annotation flag; <s> is either no or yes (case insensitive). If yes, the consensus residue field for each match state in the main model (see below) is valid. If no, these characters are ignored. Consensus residue annotation is determined when models are built. For models of single sequences, the consensus is the same as the query sequence. For models of multiple alignments, the consensus is the maximum likelihood residue at each position. Upper case indicates that the model's emission probability for the consensus residue is $\geq$ an arbitrary threshold (0.5 for protein models, 0.9 for DNA/RNA models).

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

MAP <s> Map annotation flag; <s> is either no or yes (case insensitive). If set to yes, the map annotation field in the main model (see below) is valid; if no, that field will be ignored. The HMM/alignment map annotates each match state with the index of the alignment column from which it came. It can be used for quickly mapping any subsequent HMM alignment

back to the original multiple alignment, via the model. **Optional**; assumed to be no if not present.

DATE <s> Date the model was constructed; <s> is a free text date string. This field is only used for logging purposes.[4] **Optional.**

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

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

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

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

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

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

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

STATS <s1> <s2> <f1> <f2> Statistical parameters needed for E-value calculations. <s1> is the model's alignment mode configuration: currently only LOCAL is recognized. <s2> is the name of the score distribution: currently MSV, VITERBI, and FORWARD are recognized. <f1> and <f2> are two real-valued parameters controlling location and slope of each distribution, respectively; μ and λ for Gumbel distributions for MSV and Viterbi scores, and τ and λ for exponential tails for Forward scores. λ values must be positive. All three lines or none of them must be present: when all

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

three are present, the model is considered to be calibrated for E-value statistics. **Optional.**

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

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

main model section

All the remaining fields are **mandatory**.

The first two lines in the main model section are atypical.[5] They contain information for the core model's BEGIN node. This is stored as model node 0, and match state 0 is treated as the BEGIN state. The begin state is mute, so there are no match emission probabilities. The first line is the insert 0 emissions. The second line contains the transitions from the begin state and insert state 0. These seven numbers are: $B \to M_1$, $B \to I_0$, $B \to D_1$; $I_0 \to M_1$, $I_0 \to I_0$; then a 0.0 and a '*', because by convention, nonexistent transitions from the nonexistent delete state 0 are set to $\log 1 = 0$ and $\log 0 = -\infty =$ '*'.

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

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

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

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

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

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

The next field is the MM mask value for this node. If MM was yes in the header, then this is a single 'm' character, indicating that the position was identified as a masked position during model construction; otherwise, this field is '-'.

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

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

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

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

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

Stockholm, the recommended multiple sequence alignment format

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

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

```
# STOCKHOLM 1.0

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

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

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

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

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

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

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

syntax of Stockholm markup

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

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

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

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

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

semantics of Stockholm markup

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

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

HMMER uses Stockholm markup to pick up a variety of information from the Pfam multiple alignment database. The Pfam consor-

tium 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.

MM Model mask line. An 'm' indicates that the column lies within a masked range, so that `hmmbuild` should produce emissions matching the background for a match state corresponding to that alignment column. Otherwise, a '.' is used.

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

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

recognized #=GR annotations

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

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

PP Posterior probability for an aligned residue. This represents the probability that this residue is assigned to the HMM state corresponding to this alignment column, as opposed to some other state. (Note that a residue can be confidently *unaligned*: a residue in an insert state or flanking N or C state may have high posterior probability.) The posterior probability is encoded as 11 possible

characters `0-9*+`: $0.0 \leq p < 0.05$ is coded as 0, $0.05 \leq p < 0.15$ is coded as 1, (... and so on ...), $0.85 \leq p < 0.95$ is coded as 9, and $0.95 \leq p \leq 1.0$ is coded as '*'. Gap characters appear in the PP line where no residue has been assigned.

A2M multiple alignment format

HMMER's Easel library routines are capable of writing alignments in UC Santa Cruz "A2M" (alignment to model) format, the native input format for the UCSC SAM profile HMM software package.

To select A2M format, use the format code `a2m`: for example, to reformat a Stockholm alignment to A2M:

```
esl-reformat a2m myali.sto
```

Easel currently does not read A2M format, and it currently only writes in what UCSC calls "dotless" A2M format.

The most official documentation for A2M format appears to be at http://compbio.soe.ucsc.edu/a2m-desc.html. You may refer to that document if anything in the brief description below is unclear.

An example A2M file

This alignment:

```
seq1  ACDEF...GHIKLMNPQTVWY
seq2  ACDEF...GHIKLMNPQTVWY
seq3  ---EFmnrGHIKLMNPQT---
```

is encoded in A2M format as:

```
>seq1  Sequence 1 description
ACDEFGHIKLMNPQTVWY
>seq2  Sequence 2 description
ACDEFGHIKLMNPQTVWY
>seq3  Sequence 3 description
---EFmnrGHIKLMNPQT---
```

A2M format looks a lot like aligned FASTA format. A crucial difference is that the aligned sequences in a "dotless" A2M file do not necessarily all have the same number of characters. The format distinguishes between "consensus columns" (where residues are in upper case and gaps are a dash, '-') and "insert columns" (where residues are in lower case and gaps are dots, '.', that aren't explicitly shown in the format – hence "dotless" A2M). The position and number of gaps in insert columns (dots) is implicit in this representation. An advantage of this format is its compactness.

This representation only works if all insertions relative to consensus are considered to be unaligned characters. That is how insertions are handled by profile HMM implementations like SAM and HMMER, and profile SCFG implementations like Infernal.

Thus every sequence must have the same number of consensus columns (upper case letters plus '-' characters), and may have additional lower case letters for insertions.

Legal characters

A2M (and SAM) do not support some special characters such as '*' (not-a-residue) or '~' (missing data). Easel outputs these characters as gaps: either '-' in a consensus column, or nothing in an insert column.

The SAM software parses only a subset of legal ambiguity codes for amino acids and nucleotides. For amino acids, it only reads {BXZ} in addition to the 20 standard one-letter codes. For nucleotides, it only reads {NRY} in addition to {ACGTU}. With one crucial exception, it treats all other letters as X or N.

The crucial exception is 'O'. SAM reads an 'O' as the position of a "free insertion module" (FIM), a concept specific to SAM-style profile HMMs. This has no impact on nucleic acid sequences, where 'O' is not a legal character. But in amino acid sequences, 'O' means pyrrolysine, one of the unusual genetically-encoded amino acids. This means that A2M format alignments must not contain pyrrolysine residues, lest they be read as FIMs. For this reason, Easel converts 'O' residues to 'X' when it writes an amino acid alignment in A2M format.

Determining consensus columns

Writing A2M format requires knowing which alignment columns are supposed to be considered consensus and which are considered inserts. If the alignment was produced by HMMER or Infernal, then the alignment has so-called "reference annotation" (what appears as a `#=GC RF` line in Stockholm format) marking the consensus columns.

Often, an alignment has no reference annotation; for example, if it has been read from an alignment format that has no reference annotation line (only Stockholm and SELEX formats support reference annotation). In this case, Easel internally generates a "reasonable" guess at consensus columns, using essentially the same procedure that HMMER's `hmmbuild` program uses by default: sequence fragments (sequences $<$50% of the mean sequence length in the alignment overall) are ignored, and for the remaining sequences, any column containing $\geq$ 50% residues is considered to be a consensus column.

hmmpgmd sequence database format

The hmmpgmd sequence database format closely resembles the FASTA format, with slight modification to support use within HMMER's `hmmpgmd` daemon.

The `hmmpgmd` program enables search of one or more sequence databases (e.g. NR, SwissProt, UniProt) within a single instance, having loaded a single sequence file into memory. Because the set of sequences found within the various databases may overlap, the hmmpgmd format allows each sequence to be stored once, and includes a small piece of metadata that indicates, for each sequence, the list of source databases in which the sequence is found. When a search is performed in `hmmpgmd`, a single target database is specified, and search is restricted to sequences belonging to that database.

Furthermore, because a single sequence might be found multiple times within a single sequence database, `hmmpgmd` is designed to compute E-values not just on the total number of non-redundant sequences within a database, but on the total number of sequences in the original (possibly redundant) database, provided those redundant counts are given in the hmmpgmd-formatted file.

The `hmmpgmd` file begins with a single line containing various counts describing the contents of the file, of the form

`#res_cnt seq_cnt db_cnt cnt_1 fullcnt_1 cnt_2 fullcnt_2 ... date_stamp`

Fields in header line

`res_cnt` Number of residues in the sequence file.

`seq_cnt` Number of sequences in the sequence file.

`db_cnt` Number of databases in the sequence file.

`cnt_i` Of the sequnces in the file, the number that belong to database `i`. Note that if the file contains only a single database, this will match `seq_cnt`.

`fullcnt_i` For database `i`, the number of sequences that should be used in computing E-values. If redundant sequences were collapsed out of the original database, this may be larger than `cnt_i`.

FASTA-like sequence format

In the main body of the sequence file, database sequences are stored sequentially, with each entry consisting of a one-line FASTA-like header followed by one or more lines containing the amino acid sequence, like

```
>1 100
ACDEFGHIKLMNPQTVWY
>2 010
ACDKLMNPQTVWYEFGHI
>3 111
EFMNRGHIKLMNPQT
```

Note that the per-entry header line is made up of two parts. The first part is a numeric, incrementing sequence identifier (the i'th entry has the identifier `i`). The second part is a bit string indicating database membership. In this example, sequence 1 is found in database 1, sequence 2 is found in database 2, and sequence 3 found in databases 1, 2, and 3. The number of bits in each bit string should match `db_cnt`.

Because `hmmpgmd` accepts only numeric sequence identifiers, it is necessary to keep track of the mapping between each numeric sequence identifier and the corresponding meta data (e.g. name, accession, description) external to the hmmpgmd-format file, and post-process `hmmpgmd` seach results to apply those fields to the target sequence information. Depending on the size of the map list, this might be easily acheived with a simple perl array, or might require a more heavy-duty mapping backend such as mongodb (http://www.mongodb.org).

Creating a file in hmmpgmd format

The HMMER-Easel tool `esl-reformat` is able to convert a file in unaligned fasta format into an hmmpgmd format file, such as

```
esl-reformat -id_map mydb.hmmpgmd.map hmmpgmd mydb.fasta > mydb.hmmpgmd
```

The optional `-id_map` flag captures sequence name and description information into a simple tabular file, to be used for mapping those values back to `hmmpgmd` query hits.

Score matrix files

Profile HMMs can be built from single sequences, not just from multiple sequence alignments. For example, the `phmmer` and `jackhmmer` search programs take a single sequence query as an input, and `nhmmer` can as well. For single sequence queries, the probability parameters for residue alignments are derived from a substitution score matrix, such as BLOSUM62. Scores are converted to probabilities as described by Altschul (1991).[6]. The scores can be arbitrary, but they must satisfy a couple of conditions so they can be interpreted as implicit log-odds probabilities: there must be at least one positive score, and the expected score (on nonhomologous alignments) must be negative.

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

The default score matrix for protein alignments is BLOSUM62; for DNA, a matrix we call DNA1. Using the `-mx` option (for programs that can use score matrices), you can choose instead one of several alternative protein score matrices: PAM30, PAM70, PAM120, PAM240, BLOSUM45, BLOSUM50, BLOSUM80, and BLOSUM90. For example, you could use `-mx BLOSUM50`.

The `-mxfile` option allows you to provide a score matrix as a file. You can download many standard score matrice files from NCBI at ftp://ftp.ncbi.nlm.nih.gov/blast/matrices/. HMMER can read *almost* any of these files, with one exception: because it requires the score matrix to be symmetrical (same number of residues in rows and columns), the NCBI `NUC.4.4` matrix for DNA works, but the alternative short-form format `NUC.4.2` does not work. The scores in these two files are identical, so just use `NUC.4.4`, and files like it.

Here's a simple example file:

```
# My score matrix
#
   A  T  G  C
A  1 -3 -3 -3
T -3  1 -3 -3
G -3 -3  1 -3
C -3 -3 -3  1
```

In more detail, the rules for the format are:

- Blank lines are ignored.
- A `#` indicates a comment. Anything after it is ignored. Lines that start with `#` are ignored like blank lines.
- The first data line is a header line, labeling each column with n single residue characters (case-insensitive). A nucleic matrix has $4 \leq n \leq 16$: at least the four canonical residues `ACGT` (or `U`, for you RNA zealots), and it may also contain any or all ambiguity codes `RYMKSWHBVDN*`. A protein matrix has $20 \leq n27$; it must contain at least

the 20 canonical residues ACDEFGHIKLMNPQRSTVWY, and may contain any or all ambiguity codes BJZOU. These residues can be in any order, but the rows must be in the same order as the columns.

- The next n data lines are the rows of a square $n \times n$ score matrix:
  - Fields in the row are whitespace-delimited (tabs or spaces).
  - Optionally, each row can start with its single residue label. Therefore each row has either $n + 1$ fields if there is a leading label, or n fields if not. Rows and columns are in the same order.
  - Each score is an integer. Plus signs are optional.
- The file may not contain any additional lines (other than comments or blank lines).

HMMER only uses the scores for canonical residues, and uses them to calculate probabilities. If scores for ambiguous residue codes are provided, HMMER ignores them; it has its own logic for dealing with the probability of ambiguous residues, given the probability of canonical residues.

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 enlightened support, though they thought they had funded me to study the genetics of neural development in *C. elegans*.

The first public release of HMMER (1.8) was in April 1995, shortly after I moved to the Department of Genetics at 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 escaped St. Louis and make it to some genome centers, but 1.9 was never supported. HMMER 1.9 burned down and sank into the swamp in 1996.

HMMER 2 was a nearly complete rewrite, based on the new Plan 7 model architecture, 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. I also 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. Much of HMMER2 was written in coffee shops, airport lounges, transoceanic flights, and Graeme Mitchison's kitchen. The source code still contains a disjointed record of where and when various bits were written.

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

If the world worked as I hoped, the combination of our 1998 book *Biological Sequence Analysis* and the existence of HMMER2 as a proof

of principle would have motivated the widespread adoption of probabilistic modeling methods for sequence database searching. We would declare Victory! and move on. Indeed, probabilistic modeling did become important in the field, and the other authors of *Biological Sequence Analysis* did move on. Richard Durbin moved on to human genomics; Anders Krogh moved on to pioneer a number of other probabilistic approaches for other biological sequence analysis problems; Graeme Mitchison moved on to quantum computing; I moved on to noncoding structural RNAs.

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

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

Coicidentally, an embedded systems engineer named Michael Farrar contacted me in January 2007. As a side hobby, Farrar has developed an efficient new "striped" method for using SIMD vector instructions to accelerate Smith/Waterman sequence alignment. He had used it to accelerate Bill Pearson's SSEARCH program by 10-20x, and wanted to know if his ideas could be applied in HMMER. He published a short Bioinformatics paper later in 2007 on the SSEARCH work, as a solo author with no academic affiliation. In December 2007, working from Michael's description, I implemented striped SIMD vectorization for HMMER, and one pleasant day, I realized how to do the fast filter we now call the SSV and MSV filters. Michael and I started corresponding frequently by email. We met for coffee at the Starbucks on Church Street in Cambridge in early 2008, and I started trying to recruit him to Janelia Farm. We negotiated off and on for a year, and he joined the group in June 2009. HMMER3.0 was first released in March 2010.

HMMER is still my baby, but it is also the work of several people who have come through my lab and other collaborators, including contributions from Bill Arndt, Dawn Brooks, Nick Carter, Sergi Castellano, Alex Coventry, Michael Farrar, Rob Finn, Ian Holmes, Steve Johnson, Bjarne Knudsen, Diana Kolbe, Eric Nawrocki, Elena Rivas, Walt Shands, and Travis Wheeler.[7]

[7] I write "I" in this guide, but a few parts of it were first written by Travis. I think there's probably some stuff that was first written by Ewan Birney in here too.

I thank Scott Yockel, James Cuff, and the Harvard Odyssey team for our computing environment at Harvard, and Goran Ceric and his team for our previous environment at Janelia Farm. Without the skills of the teams at our high-performance computing centers, we would be nowhere. HMMER testing can spin up hundreds or thousands of processors at a time, an unearthly amount of computing power.

In the olden days, 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 in Copenhagen) and Kimmen Sjölander, was 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 not in my earshot, that HMMER started as simply a re-implementation of their original ideas, just to teach myself what HMMs were.

In many places, I've reimplemented algorithms described in the literature. These are too numerous to thank here. The original references are given in 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 Apple OS/X and GNU/Linux machines, but is tested on a variety of hardware. Over the years, Compaq, IBM, Intel, Sun Microsystems, Silicon Graphics, Hewlett-Packard, Paracel, and nVidia have provided generous hardware support that makes this possible. I'm endebted to the free software community for the development tools I use: an incomplete list includes GNU gcc, gdb, emacs, and autoconf; valgrind; Subversion and Git; Perl and Python; LaTeX; PolyglotMan; and the UNIX and Linux operating systems.

Finally, I'd like to cryptically thank Dave "Mr. Frog" Pare and Tom "Chainsaw" Ruschak for an unrelated open source software product that was historically instrumental in HMMER's development, for reasons that are best not discussed while sober.