INFERNAL User's Guide

Sequence analysis using profiles of RNA secondary structure consensus

http://infernal.janelia.org/
Version 1.0; January 2009

Eddy lab
HHMI Janelia Farm
19700 Helix Drive
Ashburn VA 20147
http://selab.janelia.org/

Contents

1 Introduction

INFERNAL is a software package that allows you to make consensus RNA secondary structure profiles, and use them to search nucleic acid sequence databases for homologous RNAs, or to create new structure-based multiple sequence alignments.

To make a profile, you need to have a multiple sequence alignment of an RNA sequence family, and the alignment must be annotated with a consensus RNA secondary structure. The program **`cmbuild`** takes an annotated multiple alignment as input, and outputs a profile.

You can then use that profile to search a sequence database for homologs, using the program **`cmsearch`**.

You can also use the profile to align a set of unaligned sequences to the profile, producing a structural alignment, using the program **`cmalign`**. This allows you to build hand-curated representative alignments of RNA sequence families, then use a profile to automatically align any number of sequences to that profile. This seed alignment/full alignment strategy combines the strength of stable, carefully human-curated alignments with the power of automated updating of complete alignments as sequence databases grow. This is the strategy used to maintain the RFAM database of RNA multiple alignments and profiles.

INFERNAL is comparable to HMMER (http://hmmer.janelia.org). The HMMER software package builds profile hidden Markov models (profile HMMs) of multiple sequence alignments. Profile HMMs capture only primary sequence consensus features. INFERNAL models are profile stochastic context-free grammars (profile SCFGs). Profile SCFGs include both sequence and RNA secondary structure consensus information.

INFERNAL is slow and CPU-intensive. You will probably need a large number of CPUs in order to use it for serious work.

2 Installation

Quick installation instructions

Download the source tarball (**infernal.tar.gz**) from ftp://selab.janelia.org/pub/software/infernal/ or http://infernal.janelia.org

Unpack the software:

```
> tar xvf infernal.tar.gz
```

Go into the newly created top-level directory (named either **infernal**, or **infernal-xx** where **xx** is a release number:

```
> cd infernal
```

Configure for your system, and build the programs:

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

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

```
> make check
```

The programs are now in the **src/** subdirectory. The user's guide (this document) is in the **documentation/userguide** subdirectory. The man pages are in the **documentation/manpages** subdirectory. You can manually move or copy all of these to appropriate locations if you want. You will want the programs to be in your $PATH.

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

```
> make install
```

That's all. More complete instructions follow, including how to change the default installation directories for **make install**.

More detailed installation notes

INFERNAL is distributed as ANSI C source code. It is designed to be built and used on UNIX platforms. It is developed on Intel GNU/Linux systems, and intermittently tested on a variety of other UNIX platforms. It is not currently tested on either Microsoft Windows or Apple OS/X, but it should work there; it should be possible to build it on any platform with an ANSI C compiler. The software itself is vanilla POSIX-compliant ANSI C. You may need to work around the configuration scripts and Makefiles to get it built on a non-UNIX platform.

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

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

All customizations can and should be done at the **./configure** command line, unless you're a guru delving into the details of the source code.

setting installation targets

The most important options are those that let you set the installation directories for **make install** to be appropriate to your system. What you need to know is that INFERNAL installs only two types of files: programs and man pages. It installs the programs in **--bindir** (which defaults to **/usr/local/bin**), and the man pages in the **man1** subdirectory of **--mandir** (default **/usr/local/man**). Thus, say you want **make**

install to install programs in **/usr/bioprogs/bin/** and man pages in **/usr/share/man/man1**; you would configure with:

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

That's really all you need to know, since INFERNAL installs so few files. But just so you know; GNU configure is very flexible, and has shortcuts that accomodates several standard conventions for where programs get installed. One common strategy is to install all files under one directory, like the default **/usr/local**. To change this prefix to something else, say **/usr/mylocal/** (so that programs go in **/usr/mylocal/bin** and man pages in **/usr/mylocal/man/man1**, you can use the **--prefix** option:

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

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

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

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

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

setting compiler and compiler flags

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

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

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

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

which is the one-line shorthand for:

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

If you are using a non-GNU compiler, you will almost certainly want to set **CFLAGS** to some sensible optimization flags for your platform and compiler. The **-g** default generated unoptimized code. At a minimum, turn on your compiler's default optimizations with **CFLAGS=-O**.

turning on Message Passing Interface (MPI) support

INFERNAL includes four programs **cmsearch**, **cmcalibrate**, **cmalign** and **cmscore** that optionally use MPI parallelization by invoking the **--mpi** option. To enable the option to use MPI in these four executables, add **--enable-mpi** to the configuration command:

```
> ./configure --enable-mpi
```

To run a program in MPI mode, you must run them in an MPI environment with **mpirun** or **mpiexec**, with the **--mpi** option enabled. For example, in our LAM environment:

```
> mpirun C cmsearch --mpi query.cm target.fa
```

Other environments besides LAM MPI should work also, but may require different command syntax.

No longer supported: rigorous filters

Previous versions of INFERNAL included programs by Zasha Weinberg that implement rigorous filtering. The 1.0 release does not include these programs. If you'd like to use them you can either download Zasha's own implementation in RAVENNA from http://bliss.biology.yale.edu/ zasha/ravenna/ download an older 0.x version of INFERNAL, or try to modify this version to work with rigorous filters (the code is still included in `rigfilters/`).

Example configuration

The Intel GNU/Linux version installed at Janelia Farm is configured as follows:

```
> env CFLAGS="-O3" ./configure --enable-mpi --enable-lfs --prefix=/usr/local/infernal-1
```

3 Tutorial

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

The programs in INFERNAL

There are seven programs supported in the INFERNAL 1.0 package:

cmalign	Align sequences to an existing model.
cmbuild	Build a model from a multiple sequence alignment.
cmcalibrate	Determine expectation value scores (E-values) for more sensitive searches and appropriate HMM filter score cutoffs for faster searches.
cmemit	Emit sequences probabilistically from a model.
cmscore	Test the efficacy of different alignment algorithms. (Mainly useful for development and testing).
cmsearch	Search a sequence database for matches to a model.
cmstat	Report statistics on a model.

Files used in the tutorial

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

trna.5.sto A multiple alignment of five tRNA sequences. This file is a simple example of *Stockholm format* that INFERNAL uses for structurally-annotated alignments.

tosearch.300Kb.db A 300,000 nt sequence "database" that contains a tRNA. The file is in FASTA format, which INFERNAL uses for unaligned sequence data.

toalign.3.fa Three tRNA sequences in unaligned FASTA format.

toalign.1trunc.fa A truncated tRNA sequence, actually the first sequence from **toalign.3.fa** with some 5' and 3' residues removed to demonstrate alignment of truncated sequences.

toalign.1.fa The first tRNA sequence from toalign.3.fa.

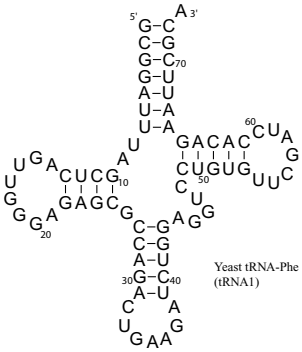
my.c.cm A calibrated version of a model built from **trna.5.sto**, included to save time.

Create a new directory that you can work in, and copy all the files in **tutorial** there. I'll assume for the following examples that you've installed the INFERNAL programs in your path; if not, you'll need to give a complete path name to the programs (e.g. something like **/usr/people/nawrocki/infernal-1/src/cmbuild** instead of just **cmbuild**).

Format of a simple input RNA alignment file

Look at the alignment file **`trna.5.sto`** in the **`intro/`** subdirectory of the INFERNAL distribution. It is shown below, with a secondary structure of the first sequence shown to the right for reference (yeast Phe tRNA, labeled as "tRNA1" in the file):

```
# STOCKHOLM 1.0

tRNA1             GCGGAUUUAGCUCAGUUGGG.AGAGCGCCAGACUGAAGAUCUGGAGGUCC
tRNA2             UCCGAUAUAGUGUAAC.GGCUAUCACAUCACGCUUUCACCGUGGAGA.CC
tRNA3             UCCGUGAUAGUUUAAU.GGUCAGAAUGGGCGCUUGUCGCGUGCCAGA.UC
tRNA4             GCUCGUAUGGCGCAGU.GGU.AGCGCAGCAGAUUGCAAAUCUGUUGGUCC
tRNA5             GGGCACAUGGCGCAGUUGGU.AGCGCGCUUCCCUUGCAAGGAAGAGGUCA
#=GC SS_cons      <<<<<<<..<<<<.........>>>>.<<<<<.......>>>>>.....<

tRNA1             UGUGUUCGAUCCACAGAAUUCGCA
tRNA2             GGGGUUCGACUCCCCGUAUCGGAG
tRNA3             GGGGUUCAAUUCCCCGUCGCGGAG
tRNA4             UUAGUUCGAUCCUGAGUGCGAGCU
tRNA5             UCGGUUCGAUUCCGGUUGCGUCCA
#=GC SS_cons      <<<<.......>>>>>>>>>>>>.
//
```



This is a simple example of a multiple RNA sequence alignment with secondary structure annotation, in *Stockholm format*. Stockholm format, the native alignment format used by HMMER and INFERNAL and the PFAM and RFAM databases, is documented in detail later in the guide.

For now, what you need to know about the key features of the input file is:

- The alignment is in an interleaved format, like other common alignment file formats such as CLUSTALW. Lines consist of a name, followed by an aligned sequence; long alignments are split into blocks separated by blank lines.
- Each sequence must have a unique name that has zero spaces in it. (This is important!)
- For residues, any one-letter IUPAC nucleotide code is accepted, including ambiguous nucleotides. Case is ignored; residues may be either upper or lower case.
- Gaps are indicated by the characters ., _, -, or ~. (Blank space is not allowed.)
- A special line starting with `#=GC SS_cons` indicates the secondary structure consensus. Gap characters annotate unpaired (single-stranded) columns. Base pairs are indicated by any of the following pairs: `<>`, `()`, `[]`, or `[]`. No pseudoknots are allowed; the open/close-brackets notation is only unambiguous for strictly nested base-pairing interactions.
- The file begins with the special tag line `# STOCKHOLM 1.0`, and ends with `//`.

Building a model with `cmbuild`

To build a model from this alignment, do:

```
> cmbuild my.cm trna.5.sto
```

Almost instantly, **`cmbuild`** reads in the alignment, constructs a model, and saves that model to the new file **`my.cm`**. It is a convention to use the **`.cm`** suffix for model files; CM stands for "covariance model", another name for the profile SCFG architecture used by INFERNAL (Eddy and Durbin, 1994).

The output from **cmbuild** contains information about the size of your input alignment (in aligned columns and # of sequences), and some statistics describing the model that was constructed. You don't need to understand this to use the model, so for now we'll skip describing the output, and revisit it in the "Profile SCFG construction" section.

The result, the model file in **my.cm** is a text file. You can look at it (e.g. **> more my.cm**) if you like, but it isn't really designed to be human-interpretable. You can treat **.cm** files as compiled models of your RNA alignment.

▷ ***Can I build a model from a single sequence?*** *Yes. But a structure for the sequence must still be supplied. With single sequences, you can also build a RSEARCH (Klein and Eddy, 2003) CM using the* **--rsearch** *option to* **cmbuild***. There's more on this in a later section.*

▷ ***Can I build a model from unaligned sequences?*** *In principle, CMs can be trained from unaligned sequences; however, this functionality is not yet implemented in* INFERNAL*. I recommend CLUSTALW as an excellent, freely available multiple sequence alignment program. The original* **covet** *CM training program from COVE, the predecessor of* INFERNAL *is also still available by ftp.*

Calibrate the model with cmcalibrate

This step is optional, but we strongly recommend it because it will increase the sensitivity of your database search and potentially make it much faster.

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

Additionally, for some searches with some models it is possible to use an HMM filter to accelerate the search at a very small (theoretical) cost to sensitivity. Besides calibrating E-values, the **cmcalibrate** program determines when this acceleration is possible, and a "calibrated" model will automatically employ the HMM filter during search.

If a non-calibrated model is used to search a database, E-values will not be calculated, and a default HMM filter will be used at an unknown cost to sensitivity. There's an example of this in section 6.

Importantly, if you're not going to use a model for searching, there is no need to calibrate it. For example, if you are only going to build alignments with a model of a large family like small subunit ribosomal RNA, don't waste time calibrating it. **cmsearch** is the only INFERNAL program that uses E-values and HMM filters, so if you won't use it, don't calibrate your model.

▷ ***Do I really have to calibrate my model?*** *No, but we recommend it in most situations. Importantly, if you're not going to search with your model, then don't calibrate it (see above). If you are going to search with your model, you still are not required to calibrate your model. If you choose not to calibrate, you'll have to settle for default filter threshold cutoffs which will compromise sensitivity to an unknown degree and possibly not accelerate your search as much as possible. An example of searching with non-calibrated models is in section 6.*

CM calibration takes a long time, but it only has to be done once per model, and can potentially save a lot of time during database searches. The amount of time the calibration takes varies widely, but depends mainly on the size of the RNA family being modeled. So you can know what kind of a wait you're in for, the **cmcalibrate** has a **--forecast <n>** option which reports an estimate of the running time (**<n>** is the number of processors you'll use for calibration, for now **<n>** is 1, but if you're using MPI it could be higher). To estimate the time required for calibration of your tRNA model, type:

```
> cmcalibrate --forecast 1 my.cm
```

The program will dump about fifty lines to standard output but don't worry about any of it now except the line with "all" in the "stage" column towards the end:

```
# all        -    -    -          -      -       01:24:22
```

This line gives the total predicted time of this run of **cmcalibrate**, about 1 hour and twenty minutes. So that you don't have to wait an hour to do this step, we've included the file **my.c.cm**, a calibrated version of **my.cm**. The **my.c.cm** file was created with the same **cmbuild** command you just performed (except **my.c.cm** was used as the output file instead of **my.cm**) and then calibrated with:

```
> cmcalibrate my.c.cm
```

When **cmcalibrate** finished, the **my.c.cm** file had been updated to include information about E-values and HMM filter thresholds. More detail on what **cmcalibrate** does can be found in sections 5 and 6. To make things simpler for our tutorial, copy over the **my.cm** file you just made with the calibrated version:

```
> cp my.c.cm my.cm.
```

Searching a sequence database with cmsearch

You can use your model to search for new homologues of your RNA family. The file **tosearch.300Kb.db** contains an example sequence "database": one 300,000 nt sequence, with yeast tRNA-Phe embedded at position 101...173. The **cmsearch** also has a **--forecast** option to predict running times, which is useful if you're searching large database files. Since this database is relatively small, we'll just do the search:

```
> cmsearch my.cm tosearch.300Kb.db
```

First, the program will print a header and "Pre-search info" to the screen. This output is explained in detail later in section 6, don't worry about it right now. The search should take about 2 minutes.

cmsearch now searches both strands of each sequence in the target database, and returns alignments for high scoring hits. In this case 3 hits are returned. Look at the first hit:

```
CM: trna.5-1
>example

  Plus strand results:

 Query = 1 - 72, Target = 101 - 173
 Score = 78.06, E = 3.133e-21, P = 2.906e-26, GC =  53

          (((((((,,<<<<___.____>>>>,<<<<<_______>>>>>,,,,,<<<<<_______
        1 gCcgacAUaGcgcAgU.GGuAgcgCgccagccUgucAagcuggAGgUCCgggGUUCGAUu 59
          GC:+A::UAGC:CAGU GG AG:GCGCCAG:CUG+++A:CUGGAGGUCC:G:GUUCGAU
      101 GCGGAUUUAGCUCAGUuGGGAGAGCGCCAGACUGAAGAUCUGGAGGUCCUGUGUUCGAUC 160

          >>>>>)))))))):
       60 CcccGUgucgGca 72
          C:C:G::U+:GCA
      161 CACAGAAUUCGCA 173
```

The first line gives the name of the CM (this can be defined in the input Stockholm alignment file or as an option to **cmbuild**, as described later). Next comes the results section, the name of each target sequence in the target database is given starting with a >, in this case there is only one: **example**. Next, all the hits to the top (Watson) strand of **example** are given, in this example there is a single hit from position 101 to 173 with a score of 78.06 bits. The E-value of this hit is $3.133e-21$, this is the number hits we expect to find with a bit score of 78.06 or better if we were searching a database of random sequences of total length 300,000. So this is a really good hit. Bit scores and E-values are discussed in more detail in section 5.

The alignment is shown in a BLAST-like format, augmented by secondary structure annotation.

The top line shows the predicted secondary structure of the target sequence. The format is a little fancier and more informative than the simple least-common-denominator format we used in the input alignment file. It's designed to make it easier to see the secondary structure by eye. The format is described in detail later; for now, here's all you need to know. Base pairs in simple stem loops are annotated with `<>` characters. Base pairs enclosing multifurcations (multiple stem loops) are annotated with `()`, such as the tRNA acceptor stem in this example. In more complicated structures, `[]` and `{}` annotations also show up, to reflect deeper nestings of multifurcations. For single stranded residues, _ characters mark hairpin loops; `-` characters mark interior loops and bulges; `,` characters mark single-stranded residues in multifurcation loops; and `:` characters mark single stranded residues external to any secondary structure. Insertions relative to this consensus are annotated by a `.` character.

The second line shows that consensus of the query model. The highest scoring residue sequence is shown. Upper case residues are highly conserved. Lower case residues are weakly conserved or unconserved.

The third line shows where the alignment score is coming from. For a consensus base pair, if the observed pair is the highest-scoring possible pair according to the consensus, both residues are shown in upper case; if a pair has a score of ≥ 0, both residues are annotated by : characters (indicating an acceptable compensatory base pair); else, there is a space, indicating that a negative contribution of this pair to the alignment score. For a single-stranded consensus residue, if the observed residue is the highest scoring possibility, the residue is shown in upper case; if the observed residue has a score of ≥ 0, a + character is shown; else there is a space, indicating a negative contribution to the alignment score.

Finally, the fourth line is the target sequence.

After the alignment is post-search information. This is explained later in section 6.

▷ ***How come the dbsize reported by `cmsearch` is twice the length of my database?*** *This is because by default* **`cmsearch`** *searches both strands of the database, so according to the program your database is twice the size you might think it is. You may have noticed this in the tutorial example, the 300 Kb database is reported as 600 Kb in the beginning of the* **`cmsearch`** *output. You can optionally search only the top strand or the bottom strand of your target database with the* **`--toponly`** *and* **`--bottomonly`** *options to* **`cmsearch`** *respectively.*

Creating new multiple alignments with `cmalign`

You can also use a model to structurally align any number of new RNA sequences to your consensus structure. This is how the RFAM database is constructed: we start with a "seed" alignment, build a CM of it, and use that CM to align all known members of the sequence family and create a "full" alignment. This allows us to maintain representative seed alignments that are stable and small enough to be human-curated, while still being able to automatically incorporate and align all homologues detected in the rapidly growing public sequence databases.

An example of three unaligned tRNA sequences are in the file **toalign.3.fa**. The first two sequences are real tRNAs. The third sequence, tRNA8 was created by deleting some residues out of the middle of the tRNA1 sequence from the file **trna.5.sto**. (tRNA8 will be used a little later to demonstrate local alignment.)

To align these sequences to the model we made in **my.cm**, do:

```
> cmalign my.cm toalign.3.fa
```

The output of cmalign is described later in detail. For now let's only look at the alignment:

```
# STOCKHOLM 1.0
#=GF AU Infernal 1.0

tRNA6         GUCCCGCUGGUGUAAU.GGAuAGCAUACGAUCCUUCUAAGUUUGCGG-UC
tRNA7         ACUUUUAAAGGAUAGU.AGUuUAUCCGUUGGUCUUAGGAACCAAAAA--A
tRNA8         GCGGAUUUAGCUCAGUuGGG.AGAGCGC------CAGAC----GAGGUCC
#=GC SS_cons ((((((,,<<<<___.___._>>>>,<<<<<_______>>>>>,,,,,<
#=GC RF       gCcgacAUaGcgcAgU.GGu.AgcgCgccagccUgucAagcuggAGgUCC

tRNA6         CUGGUUCGAUCCCAGGGCGGGAUA
tRNA7         UUGGUGCAACUCCAAAUAAAAGUA
tRNA8         UGUGUUCGAUCCACAGAAUUCGCA
#=GC SS_cons <<<<_______>>>>>)))))))):
#=GC RF       gggGUUCGAUuCcccGUgucgGca
//
```

In the aligned sequences, a `.` character indicates an inserted column relative to consensus; the `.` character is an alignment pad. A `-` character is a deletion relative to consensus.

The symbols in the consensus secondary structure annotation line have the same meaning that they did in a pairwise alignment from **cmsearch**.

The `#=GC RF` line is *reference annotation*. Non-gap characters in this line mark consensus columns; **cmalign** uses the residues of the consensus sequence here, with upper case denoting strongly conserved residues, and lower case denoting weakly conserved residues. Gap characters (specifically, the `.` pads) mark insertions relative to consensus. As described below, **cmbuild** is capable of reading these RF lines, so you can specify which columns are consensus and which are inserts (otherwise, **cmbuild** makes an automated guess, based on the frequency of gaps in each column).

If you want to save the alignment to a file, you can use the **-o** option:

```
> cmalign -o my.sto my.cm toalign.3.fa
```

Local versus glocal alignment in `cmsearch` and `cmalign`

The programs **cmsearch** and **cmalign** can be run in two different modes. Glocal alignment requires that the entire model match a subsequence of the target (global with respect to the query model, local with respect to the target sequence). Local alignment allows only part of the model to match a subsequence of the target. Local alignment is useful when a homologous RNA structure has undergone enough changes that parts of the its structure cannot be aligned to the full consensus model. Empirically, local alignment is often a more sensitive search strategy so by default **cmsearch** uses local alignment. Glocal mode can be turned on in **cmsearch** using the **-g** option. Conversely, **cmalign** uses glocal alignment by default, and the local alignment mode can be switched on using the the **-l** option.

First let's look at an example of local alignment in **cmsearch**:

```
> cmsearch my.cm toalign.3.fa
```

Look at the alignment for the target sequence tRNA8 (the last alignment in the output):

```
>tRNA8

  Plus strand results:

 Query = 1 - 72, Target = 1 - 63
 Score = 54.39, E = 4.664e-17, P = 6.303e-19, GC =  56

          (((((((,,<<<<___.____>>>>,<~~~~~~>,,,,,<<<<<_______>>>>>))))
        1 gCcgacAUaGcgcAgU.GGuAgcgCgc*[15]*gAGgUCCgggGUUCGAUuCcccGUguc 68
          GC:+A::UAGC:CAGU GG AG:GCGC      GAGGUCC:G:GUUCGAU C:C:G::U+
        1 GCGGAUUUAGCUCAGUuGGGAGAGCGC*[ 5]*GAGGUCCUGUGUUCGAUCCACAGAAUU 59

          ))):
       69 gGca 72
          :GCA
       60 CGCA 63
```

The `*[15]*` and `*[5]*` in the query and target, respectively, indicate that 15 consensus residues and 5 target residues were left unaligned; the target does not appear to have the consensus structure in this region. (Not surprising, since I made the tRNA8 example sequence by deleting part of the anticodon stem.) The structure annotation line is marked with `~~~~~~` to indicate the gap in the alignment, and to distinguish local alignment induced gaps from normal insertions (which are marked with `.` characters).

You can activate local alignment in **`cmalign`** with the **`-l`** option:

```
> cmalign -l my.cm toalign.3.fa
```

This results in the following alignment: [1]

```
# STOCKHOLM 1.0
#=GF AU Infernal 1.0

tRNA6        GUCCCGCUGGUGUAAU.GGAuAGCAUACGAUCCUUCUAA.....GUUUGC
tRNA7        ACUUUUAAAGGAUAGU.AGUuUAUCCGUUGGUCUUAGGA.....ACCAAA
tRNA8        GCGGAUUUAGCUCAGUuGGG.AGAGCGC-----------cagac----GA
#=GC SS_cons (((((((,,<<<<___.___._>>>>,<<<<<_______~~~~~>>>>>,
#=GC RF      gCcgacAUaGcgcAgU.GGu.AgcgCgccagccUgucAa~~~~~gcuggA

tRNA6        GG-UCCUGGUUCGAUCCCAGGGCGGGAUA
tRNA7        AA--AUUGGUGCAACUCCAAAUAAAAGUA
tRNA8        GGUCCUGUGUUCGAUCCACAGAAUUCGCA
#=GC SS_cons ,,,,<<<<<_______>>>>>)))))))):
#=GC RF      GgUCCgggGUUCGAUuCcccGUgucgGca
//
```

Note how the local alignment is represented for tRNA8. The deleted consensus columns are marked by - characters. The unaligned "insertion" is shown in its own columns; those columns are again marked with ~ characters in the consensus secondary structure annotation and the reference (RF) annotation lines.

[1] The discontinuity of structural local alignment presents a quandary for representing multiple alignments. On the one hand, you might not want to even show the unaligned target residues in the gap (e.g., cagac) – they aren't aligned to the model. On the other hand, you sort of expect that if you pull an RNA sequence out of a multiple alignment, it represents a true subsequence of a larger sequence, not a concatenation of disjoint subsequences – you'd at least like some indication of where some residues have gone missing. One option would be to leave a *[5]* in the gap, as in the pairwise representation; but one of the nice properties of Stockholm format is that it's easy to interconvert it to other alignment formats just by stripping off everything by the name/sequence part of the alignment, and sticking non-sequence characters like *[5]* in the alignment would prevent that.

Now you have successfully built a CM, calibrated it and used it search for and align new sequences using INFERNAL programs. These programs have some useful options that you haven't seen yet, which we discuss and in some cases demonstrate next. Before you start using INFERNAL, we strongly urge you to at least skim this part.

Important options to `cmbuild`

using optional annotation to completely specify model architecture

`cmbuild` needs to know two things to convert your alignment into a profile SCFG.

First, it needs to know the consensus secondary structure. It reads this from the `#=GC SS_cons` line, as described above. This annotation is mandatory.

It also needs to know which columns are consensus, and which columns are insertions relative to consensus. By default, it will determine this by a simple rule: if a column contains more than a certain fraction of gap characters (by default $>$50%, but this can be changed with the **`--gapthresh`** option), the column is called an insertion. This may not be what you want; for instance, maybe you are trying to iteratively build models based on larger and larger numbers of sequences (based on an RFAM seed, say), but you don't want the curated consensus model architecture to change just because you added some new sequences to the alignment.

You can optionally override that default and specify the complete architecture of the model, using both a `#=GC SS_cons` structure annotation line and a `#=GC RF` reference column annotation line. To do this, you use the **`--rf`** option to **`cmbuild`**.

For example, if **`trna.5.sto`** had `#=GC RF` annotation, to build a model from it called **`second.cm`** ith architecture dictated by the RF annotation, you would do:

```
> cmbuild --rf second.cm trna.5.sto
```

Since **`cmalign`** leaves an RF line on the alignments it generates, the **`--rf`** option allows you to propagate your consensus structure into new, larger alignments. The RF line is also handy when you want the model's coordinate system to be the same as a canonical, well-studied single sequence: you can simply use that sequence as the RF line, or manually create any consensus coordinate system you like. (This is the origin of RF as the "reference line", e.g. giving a reference coordinate system.) The only thing that matters in the RF line is nongap versus gap characters: the line can be as simple as x's marking consensus columns, `.`'s for insert columns.

creating multiple models from a single alignment

`cmbuild` has the ability to cluster and partition the input training alignment into several alignments based on sequence identity and build a separate CM from each of those alignments. This can be viewed as splitting the input alignment of a single RNA family into "subfamilies" based on sequence identity, each cluster being a subfamily of sequences more similar to each other than to those in other clusters. Performing homology search with these several models collectively may be a more sensitive search strategy than a single search with one model built from the entire alignment (although it will take longer). The most important options affecting this behavior are the **`--cmaxid <f>`** and **`--ctarget <n>`** options. With **`--cmaxid <f>`** the clusters are defined such that no two sequences in *different* clusters are more than **`<f>`** fractionally identical. With **`--ctarget <n>`**, a fractional identity cutoff is found that partitions the alignment into

exactly **<n>** clusters. The **--cdump <f>** option can be used in combination with either of these options to cause **cmbuild** to dump each cluster alignment to the file **<f>**. Let's try the **--cmaxid** option:

```
> cmbuild --cmaxid 0.6 --cdump my.cmaxid60.sto my.cmaxid60.cm trna.5.sto
```

```
# Alignment split into 3 clusters; each will be used to train a CM.
# Maximum identity b/t any 2 seqs in different clusters: 0.60
#
#                                                                        rel entropy
#                                                                       ------------
#  aln  cm idx  name            nseq  eff_nseq    alen   clen   bps  bifs     CM    HMM
# ----  ------  ----------  --------  --------  ------  -----  ----  ----  -----  -----
     1       1  trna.5-1.1         1      1.00      74     73    21     2  0.693  0.409
     1       2  trna.5-1.2         2      1.31      74     73    21     2  0.696  0.395
     1       3  trna.5-1.3         2      1.41      74     72    21     2  0.704  0.407
```

In this case, the input alignment was split into three clusters, of 1 sequence, 2 sequences and 2 sequences. The file **my.cmaxid60.sto** includes these three alignments. The CM file **my.cmaxid60.cm** is a database of three CMs, one built from each of these three alignments.

The **--corig** option in combination with the **--ctarget** or **--cmaxid** options will cause **cmbuild** to build one extra model, the original default model from the entire alignment, and add it to the end of the cluster models. The **--call** option can be used instead of the **--ctarget** or **--cmaxid** options to build a separate model for each sequence in the training alignment.

refining the training alignment prior to building a model

Constructing structural RNA alignments is not easy. **cmbuild** offers one experimental option to potentially help automate this procedure. The **--refine <f>** option will cause **cmbuild** to go through a two step iterative procedure to realign the sequences in the input alignment before building the model. First, a model is built from the initial given alignment. This model is then used to optimally align all the sequences, giving a new alignment. The new alignment is used to build a new model and the sequences are realigned to the new model. This two step build/align procedure repeats until convergence, when two successive iterations yield identical (or very nearly identical) alignments. The final alignment is used to build a model which is saved to the CM file and saved to file **<f>**. Importantly, the **--refine** option will not change the consensus structure of the initial alignment, so this is not a structure prediction tool. The **--gibbs** option can be used in combination with **--refine** to modify it's behavior. Instead of choosing the optimal alignment during alignment refinement, with **--gibbs** an alignment is sampled from the posterior distribution of alignments given the current model.

building RSEARCH models

The RSEARCH program (Klein and Eddy, 2003) implements a special case of homology search with covariance models in which only one training sequence/structure is known. RSEARCH had forked from INFERNAL a few years ago, but has been reintegrated. The main difference between RSEARCH and INFERNAL is the determination of the model scoring parameters. Whereas INFERNAL uses mean posterior estimates (see the "parameterization" subsection of section 4), RSEARCH uses a RIBOSUM scoring matrix to determine emission scores. You can build CMs parameterized with RIBOSUM scores using the **--rsearch <f>** option where **<f>** is the RIBOSUM matrix file to use . (The matrix files are in the **/matrices** subdirectory of INFERNAL). For more on how these matrices were created see (Klein and Eddy, 2003). Importantly, you can only use the **--rsearch** option if the input training alignment has only 1 sequence, or if you use it in

combination with the **`--call`** option which causes **`cmbuild`** to build a separate model from each sequence in the input alignment file.

Here's an example of building five **`rsearch`** CMs for tRNA:

```
> cmbuild -F --rsearch  /infernal/matrices/RIBOSUM85-60.mat --call my.rsearch.cm
trna.5.sto
```

```
# Alignment split into 5 clusters; each comprised of exactly 1 sequence
#
#                                                                          rel entropy
#                                                                         ------------
#  aln  cm idx  name            nseq  eff_nseq    alen   clen   bps  bifs     CM    HMM
# ----  ------  ----------  --------  --------  ------  -----  ----  ----  -----  -----
     1       1  trna.5-1.1         1      1.00      74     73    21     2  1.022  0.739
     1       2  trna.5-1.2         1      1.00      74     72    21     2  1.033  0.758
     1       3  trna.5-1.3         1      1.00      74     72    21     2  1.035  0.777
     1       4  trna.5-1.4         1      1.00      74     72    21     2  1.016  0.741
     1       5  trna.5-1.5         1      1.00      74     73    21     2  1.014  0.752
```

▷ ***I used the `cmbuild --rsearch` option and tested my results against the* RSEARCH *program, and the scores didn't match. Why?*** *INFERNAL can't build models quite exactly the same as RSEARCH does. The main reason is because INFERNAL uses probabilistic transition scores while RSEARCH does not. This difference leads to large differences in the bit scores from the programs, but the E-values of those scores should be similar (see section 5 for more on bit scores and E-values.)*

Important options to `cmalign`

including a fixed alignment within the output alignment

When aligning sequences to a model, **`cmalign`** allows you to include the initial training alignment used to build the model within its output alignment. This could be useful if you're updating the training alignment with new homologs, or just to easily see how new sequences align in the context of the original alignment. To turn on this behavior, use the **`--withali <f>`** option to **`cmalign`**, where **`<f>`** is the existing fixed alignment to include. Note, if you used the **`--rf`** or **`--gapthresh <x>`** options to **`cmbuild`** when you built the model, you must use those same options to **`cmalign`**. Here is an example:

```
> cmalign --withali trna.5.sto my.cm toalign.3.fa
```

```
# STOCKHOLM 1.0
#=GF AU Infernal 1.0

tRNA1        GCGGAUUUAGCUCAGUuGGG.AGAGCGCCAGACUGAAGAUCUGGAGGUCC
tRNA2        UCCGAUAUAGUGUAAC.GGCuAUCACAUCACGCUUUCACCGUGGAGA-CC
tRNA3        UCCGUGAUAGUUUAAU.GGUcAGAAUGGGCGCUUGUCGCGUGCCAGA-UC
tRNA4        GCUCGUAUGGCGCAGU.GGU.AGCGCAGCAGAUUGCAAAUCUGUUGGUCC
tRNA5        GGGCACAUGGCGCAGUuGGU.AGCGCGCUUCCCUUGCAAGGAAGAGGUCA
tRNA6        GUCCCGCUGGUGUAAU.GGAuAGCAUACGAUCCUUCUAAGUUUGCGG-UC
tRNA7        ACUUUUAAAGGAUAGU.AGUuUAUCCGUUGGUCUUAGGAACCAAAAA--A
tRNA8        GCGGAUUUAGCUCAGUuGGG.AGAGCGC------CAGAC----GAGGUCC
#=GC SS_cons (((((((,,<<<<___.___._>>>>,<<<<<_______>>>>>,,,,,<
#=GC RF      gCcgacAUaGcgcAgU.GGu.AgcgCgccagccUgucAagcuggAGgUCC

tRNA1        UGUGUUCGAUCCACAGAAUUCGCA
tRNA2        GGGGUUCGACUCCCCGUAUCGGAG
tRNA3        GGGGUUCAAUUCCCCGUCGCGGAG
tRNA4        UUAGUUCGAUCCUGAGUGCGAGCU
tRNA5        UCGGUUCGAUUCCGGUUGCGUCCA
tRNA6        CUGGUUCGAUCCCAGGGCGGGAUA
tRNA7        UUGGUGCAACUCCAAAUAAAAGUA
tRNA8        UGUGUUCGAUCCACAGAAUUCGCA
#=GC SS_cons <<<<_______>>>>>)))))))):
#=GC RF      gggGUUCGAUuCcccGUgucgGca
//
```

The top five sequences are from the training alignment **`trna.5.sto`** and the bottom three sequence are from **`toalign.3.fa`**.

▷ ***How come my fixed alignment didn't stay fixed in the `cmalign` output?*** *It is possible for the* ***`--withali`*** *alignment to change slightly when it is output from* ***`cmalign`****, but only residues that are insertions (i.e. present in non-consensus columns) should move around. This is a feature, not a bug, of the program; it is impossible for* ***`cmalign`*** *to always keep inserted residues in one sequence fixed with respect to inserted residues in another sequence. For more information on insertion columns versus consensus columns see the* ***`cmbuild`*** *section.*

▷ ***I'd like to use the `--withali` option to `cmalign` but my training alignment is hundreds of sequences deep which makes the cmalign output difficult to read, is there a way to include a subset of the training alignment in the output alignment?*** *Yes. You can remove as many sequences as you like from the training alignment if it you add* `#=GC RF` *annotation that defines the consensus columns the same way they were defined in* ***`cmbuild`*** *using the full alignment. Also, in this case you must specify the* ***`--rf`*** *option to* ***`cmalign`****.*

aligning truncated sequences

By default, the **cmalign** program assumes that the target sequences it is aligning are full length as they appear in their genomic context. However some sequences you want to align may be truncated, i.e. have missing residues at the 5' and/or 3' end . For example, sequences from shotgun sequencing projects or 16S SSU ribosomal RNA sequences from PCR based environmental surveys are often truncated. If you think the sequences you're aligning are potentially truncated, you should use the **--sub** option to **cmalign**. Without this option, truncated sequences are often terribly misaligned.

Here is an example with a single, artificially truncated tRNA sequence. The file **toalign.1trunc.fa** includes a truncated version the sequence in **toalign.1.fa** (residues 1-20 and 57-72 were removed) that was aligned as the example above for the **--withali** option. If we align the truncated sequence **without** the **--sub** option:

```
> cmalign --withali trna.5.sto my.cm toalign.1trunc.fa
```

```
# STOCKHOLM 1.0
#=GF AU Infernal 1.0

tRNA1        GCGGAUUUAGCUCAGUuGGG.AGAGCGCCAGACUGAAGAUCUGGAGGUCC
tRNA2        UCCGAUAUAGUGUAAC.GGCuAUCACAUCACGCUUUCACCGUGGAGA-CC
tRNA3        UCCGUGAUAGUUUAAU.GGUcAGAAUGGGCGCUUGUCGCGUGCCAGA-UC
tRNA4        GCUCGUAUGGCGCAGU.GGU.AGCGCAGCAGAUUGCAAAUCUGUUGGUCC
tRNA5        GGGCACAUGGCGCAGUuGGU.AGCGCGCUUCCCUUGCAAGGAAGAGGUCA
tRNA6-trunc  AG--------------.--C.A---UACGAUCCUUCUAAGUUUGCGGUCC
#=GC SS_cons (((((((,,<<<<___.___._>>>>,<<<<<_______>>>>>,,,,,<
#=GC RF      gCcgacAUaGcgcAgU.GGu.AgcgCgccagccUgucAagcuggAGgUCC

tRNA1        UGUGUUCGAUCCACAGAAUUCGCA
tRNA2        GGGGUUCGACUCCCCGUAUCGGAG
tRNA3        GGGGUUCAAUUCCCCGUCGCGGAG
tRNA4        UUAGUUCGAUCCUGAGUGCGAGCU
tRNA5        UCGGUUCGAUUCCGGUUGCGUCCA
tRNA6-trunc  U-------------GG-----UUC
#=GC SS_cons <<<<_______>>>>>)))))))):
#=GC RF      gggGUUCGAUuCcccGUgucgGca
//
```

We know the alignment of the truncated sequence **tRNA6-trunc** is wrong because we saw the correct alignment of the full tRNA6 sequence in the example above. Particularly striking is the misalignment of the 3' end (**UGGUUC**) which you'd think based on the other sequences would be easily alignable.

Here's what happens with the **--sub** option turned on:

```
> cmalign --sub --withali trna.5.sto my.cm toalign.1trunc.fa
```

```
# STOCKHOLM 1.0
#=GF AU Infernal 1.0

tRNA1        GCGGAUUUAGCUCAGUuGGG.AGAGCGCCAGACUGAAGAUCUGGAGGUCC
tRNA2        UCCGAUAUAGUGUAAC.GGCuAUCACAUCACGCUUUCACCGUGGAGA-CC
tRNA3        UCCGUGAUAGUUUAAU.GGUcAGAAUGGGCGCUUGUCGCGUGCCAGA-UC
tRNA4        GCUCGUAUGGCGCAGU.GGU.AGCGCAGCAGAUUGCAAAUCUGUUGGUCC
tRNA5        GGGCACAUGGCGCAGUuGGU.AGCGCGCUUCCCUUGCAAGGAAGAGGUCA
tRNA6-trunc  ----------------.---.AGCAUACGAUCCUUCUAAGUUUGCGG-UC
#=GC SS_cons (((((((,,<<<<___.___._>>>>,<<<<<_______>>>>>,,,,,<
#=GC RF      gCcgacAUaGcgcAgU.GGu.AgcgCgccagccUgucAagcuggAGgUCC

tRNA1        UGUGUUCGAUCCACAGAAUUCGCA
tRNA2        GGGGUUCGACUCCCCGUAUCGGAG
tRNA3        GGGGUUCAAUUCCCCGUCGCGGAG
tRNA4        UUAGUUCGAUCCUGAGUGCGAGCU
tRNA5        UCGGUUCGAUUCCGGUUGCGUCCA
tRNA6-trunc  CUGGUUC-----------------
#=GC SS_cons <<<<_______>>>>>)))))))):
#=GC RF      gggGUUCGAUuCcccGUgucgGca
//
```

This is much better. Briefly, the **`--sub`** option uses an HMM to predict where the start and end points of the alignment are and then restructures the model to only align between that start and end. This is a hack that usually works, but there's more well-principled ways of doing this, which we're currently working on.

alignment confidence estimates

`cmalign` has the ability to estimate the confidence of the alignment at each residue for each sequence. These confidence estimates are actually posterior probabilities, which can be added to the output alignment using the **`-p`** option. We'll demonstrate this with alignment of a single sequence:

```
> cmalign -p my.cm toalign.1.fa
```

```
# STOCKHOLM 1.0
#=GF AU Infernal 1.0

tRNA6              GUCCCGCUGGUGUAAUGGAuAGCAUACGAUCCUUCUAAGUUUGCGG-UCC
#=GR tRNA6 POSTX.  99999999999999998877999999999999999999999999999-899
#=GR tRNA6 POST.X  99999999999999991199999999999999999999999999999-977
#=GC SS_cons       (((((((,,<<<<_______._>>>>,<<<<<_______>>>>>,,,,,<<
#=GC RF            gCcgacAUaGcgcAgUGGu.AgcgCgccagccUgucAagcuggAGgUCCg

tRNA6              UGGUUCGAUCCCAGGGCGGGAUA
#=GR tRNA6 POSTX.  99999999999999999999999
#=GR tRNA6 POST.X  89999999999999999999999
#=GC SS_cons       <<<_______>>>>>)))))))):
#=GC RF            ggGUUCGAUuCcccGUgucgGca
//
```

The **`-p`** option adds `#=GR tRNA6 POSTX.` and `#=GR tRNA6 POST.X` markup to the output stockholm alignment. The `POSTX.` row indicates the 'tens' place of the confidence estimate while `POST.X` row indicates the 'ones' place. So the confidence estimate for the final **`C`** in the first row of the alignment to two significant digits is 97%. This means that if you sampled alignments from the posterior distribution of all possible alignments of this sequence to the model, about 97% of the time that **`C`** would appear in that position

of the alignment. One special case: if the posterior probability is “very nearly” 100% (it’s difficult to be more precise on the exact percentage due to numerical precision issues) the annotated posterior values will be “*” characters in both the tens and one places.

Putting it all together: an example of iterative search

Now that you've seen some useful features of **cmbuild** and **cmalign**, we're ready to go through an example of an iterative search with a CM with real sequence data. Iterative search is a powerful technique for finding homologs consisting of three main steps:

1. Build and calibrate a model from current alignment of homologs.
2. Search genomes (or databases) for new homologs.
3. Add the new homologs to the alignment.
4. Go back to step 1.

You can iterate over these steps as long as you'd like or until until you stop finding new homologs in step 2. The example we present here starts with only 1 training sequence/structure in the first iteration. In this case, with few sequences in the training alignment, it's smart to carefully select the genomes you search in step 2, picking ones that are rather closely related to the organisms represented in your alignment. Then, if you're able to find homologs in these closely related genomes, you can build a new alignment in step 3 which, because it's built from a deeper alignment, will have a more knowledge of the sequence divergence of the family than the initial model. You can then search genomes a bit further away on the tree of life in the next iteration. Then realign any new homologs you find and build a new model, and so on.

files used in the iterative search example

Let's work through an example of iterative search with a Purine riboswitch model. Here's a list of the files we'll use, from the **/tutorial** subdirectory of INFERNAL:

purine.1.sto A Stockholm alignment file with a single Purine riboswitch sequence and structure.

purine.1.c.cm A calibrated version of a model built from **purine.1.sto**, included to save time.

T.tengcongensis.genome.fa : the 2.5 Mb genome of the bacteria *Thermoanaerobacter tengcongensis*, in FASTA format, downloaded from the NCBI CoreNucleotide database (accession: NC_003869).

C.psychrerythraea.genome.fa : the 5 Mb genome of the bacteria *Colwellia psychrereythraea*, in FASTA format, downloaded from the NCBI CoreNucleotide database (accession: CP000083).

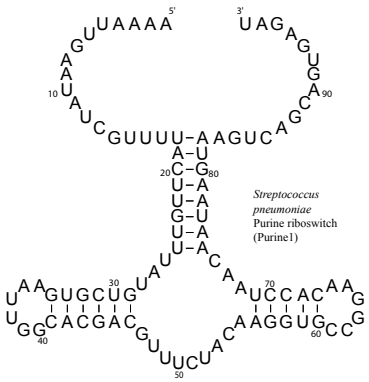
purine.teng.fa A putative Purine riboswitch in FASTA format.

purine.psych.fa A different putative Purine riboswitch in FASTA format.

iteration 1, step 1: build and calibrate a model

Look at the **purine.1.sto** stockholm file. It contains exactly one sequence and predicted structure of a Purine riboswitch from the genome of *Streptococcus pneumoniae*, a member of the Firmicutes division of the Bacterial domain. The structure from the `#=GC SS_cons` annotation for the sequence is shown on the right. (This sequence is part of the Rfam 8.1 Purine RF00167 "full" alignment).

```
# STOCKHOLM 1.0
#=GF ID    Purine
#=GF AU    Boese B, Barrick JE, Breaker RR

Purine1          AAAAUUGAAUAUCGUUUUACUUGUUUAUGUCGUGAAUUGGCACGACGUUU
#=GC SS_cons     .................<<<<<<<<...<<<<<<.....>>>>>>....

Purine1          CUACAAGGUGCCGGAACACCUAACAAUAAGUAAGUCAGCAGUGAGAU
#=GC SS_cons     ....<<<<<<.....>>>>>>..>>>>>>>>..............
//
```



First we build the model:

```
> cmbuild purine.1.cm purine.1.sto
```

Now we want to calibrate it. As before, we can use **`--forecast`** to see the predicted running time:

```
> cmcalibrate --forecast 1 purine.1.cm
```

It should take about two hours. Feel free to calibrate the model yourself if you want to, but to save time we've included **`purine.1.c.cm`** a calibrated version of the single sequence Purine model. To use our calibrated model, copy it over the model you just built with:

```
> cp purine.1.c.cm purine.1.cm
```

Now we're ready to search genomes. As mentioned earlier, at early stages of iterative search if you've built a model from very few training sequences you should carefully pick target genomes to search that are not too evolutionarily distant from the genomes of your training sequences. In this case let's search another Firmicutes bacteria, *Thermoanerobacteria tengcongensis*, the genome of which is in **`T.tengcongensis.genome.fa`**:

iteration 1, step 2: search a genome

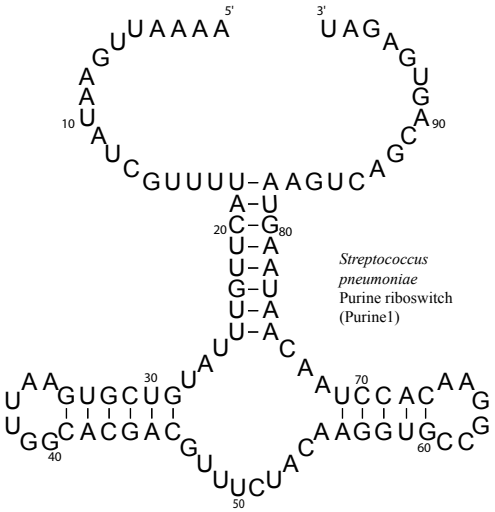
```
> cmsearch purine.1.cm T.tengcongensis.genome.fa
```

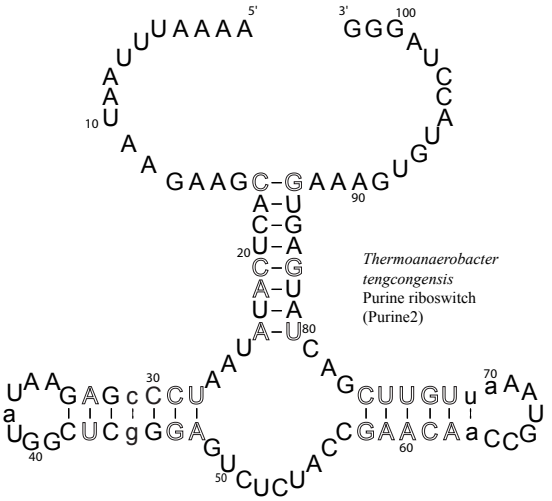
This will take about twenty minutes for **`cmsearch`** to search the 5 Mb genome. (If you don't want to

wait, please continue reading). Let's look at the first hit:

```
>gi|20806542|ref|NC_003869.1|

  Plus strand results:

 Query = 1 - 97, Target = 586366 - 586467
 Score = 35.67, E = 3.979e-06, P = 3.121e-12, GC =  40

           :::::::::::::::::(((((((,,,<<<<<<._____..>>>>>>,,,,,,,,<<<
         1 AAAAUUGAAUAUCGUUUuaCuuguuUAUGuCGuG.AAUUGG..CaCGaCGUUUCUACaaG 57
           AAAAUU AAUA  G   :ACU::U:UA ::C::: AAU  G  :::G::GU UCUAC:::
    586366 AAAAUUUAAUAA-GAAGCACUCAUAUAAUCCCGAgAAUAUGgcUCGGGAGUCUCUACCGA 586424

           <<<._____..>>>>>>,,)))))))):::::::::::::::
        58 GuG.CCGGAA..CaCCuaACaauaaGuaAGUCAGCAGUGAGAU 97
           ::: CCG AA  ::::::AC:A::AGU:A    G A   AG
    586425 ACAaCCGUAAauUGUUCGACUAUGAGUGAAAGUGUACCUAGGG 586467
```

The E-value of this hit is 3.979e-06, which means we'd expect about 0.000003979 hits of this score in searching a 5 Mb database of random sequence. Obviously an E-value by itself doesn't prove this is a homologous sequence, but this is a good hit, and it warrants closer scrutiny. Let's look at the structure more closely and how it relates to our initial training sequence. As we saw earlier in the tutorial, the **cmsearch** output is showing us where the high score is coming from. Included below is another view of the secondary structures. The figure on the left is the structure of the *Streptococcus pneumoniae* Purine riboswitch that we built our model from (this exact figure is also shown above). The figure on the right is our putative homolog from *Thermoanaerobacter tengcongensis*. Base-paired residues that are different from the training sequence are indicated as hollow outlined letters. Insertions relative to the consensus model are in lowercase (these residues are also lower case in the **cmsearch** alignment above). Notice that all the base-paired residues that are different between the sequences are putative *compensatory mutations* from one Watson-Crick (A-U, U-A, C-G, G-C) or U-G/G-U base-pair to another. You can also see this above in the **cmsearch** output. This is very strong evidence that these two sequences are homologous as they share strong structural similarity despite weak conservation at the primary sequence level (55% identity).





iteration 1, step 3: add new homolog to training alignment

Let's assume that you've convinced yourself our putative homolog is a real Purine riboswitch. Now we can use knowledge of this new homolog to increase our power in the search for new ones. First we need to add our new sequence to our initial training alignment. This requires extracting the hit from the genome. We've already done this for you, the single hit from the *Thermoanerobacteria* genome is in unaligned FASTA format in the file **purine.2.fa**. Let's align it to our training sequence, and output both sequences aligned together using the **--withali** option. We'll save the output alignment to **purine.2.sto**:

```
cmalign -o purine.2.sto --withali purine.1.sto purine.1.cm purine.teng.fa
```

Now we've completed one round of the iterative search strategy listed above and we're ready to do another round, armed with an alignment of two examples of the Purine riboswitch. First, we build a new model:

iteration 2, step 1: build and calibrate a new model

```
> cmbuild purine.2.cm purine.2.sto
```

Once again the calibration step is a bottleneck. We've provided a calibrated file so you can proceed with this tutorial in **purine.2.c.cm**, copy this file over your own model:

```
> cp purine.2.c.cm purine.2.cm
```

iteration 2, step 2: search a genome

Now we're at the search stage. In the previous iteration we searched a Firmicutes genome mainly because we only had one training sequence, from Firmicutes. Now we have two training sequences, both from Firmicutes, but actually they're pretty divergent at the sequence level. Let's do a more ambitious search this time, and look for Purine riboswitch homologs in the genome of *Colwellia psychrerythraea*, a member of the Bacterial γ-proteobacteria division:

```
> cmsearch purine.2.cm C.psychrerythraea.genome.fa
```

This search will take about 50 minutes. Two hits are found. Let's look at the first, highest-scoring hit:

```
>gi|71143482|gb|CP000083.1|

  Plus strand results:

 Query = 19 - 87, Target = 1401709 - 1401775
 Score = 39.23, E = 2.291e-06, P = 8.005e-13, GC =  45

            (((((((,,,<<<-<<<_______>>>->>>,,,,,,,,<<<<<<_________>>>>>>
         19 ACucauaUAagcCcGaGAAUAUGGCuCgGgcGUuUCUACcgggcgACCGuAAAucgcccg 78
            ::UC:UAUAA:CCC: : AUAUGG: :GGG:GU+UCUACC:GG:  C  UAA   :CC:G
  1401709 CUUCGUAUAACCCCAGUGAUAUGGAUUGGGGGUCUCUACCAGGAACCAAUAA--AUCCUG 1401766

            ,,)))))))
         79 ACuaugaGU 87
            A UA:GA::
  1401767 AUUACGAAG 1401775
```

This hit looks very promising. As before, even with a good E-value (this one is 2.291e-06), a hit is not necessarily a real homolog, but this hit is definitely worth looking at more closely. The **cmsearch**

alignment is showing us that the loop regions have high primary sequence conservation and the stems exhibit compensatory mutations. The figure below on the left shows the sequence and structure of the putative homolog. To me (and hopefully to you) this hit is convincing.

iteration 2, step 3: add new homolog to training alignment

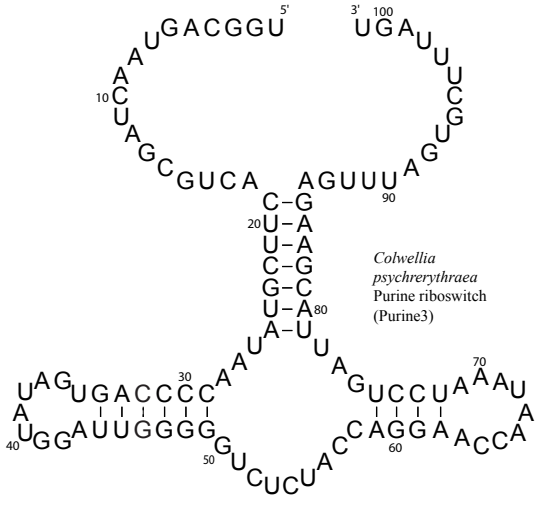
To save time we've extracted this sequence from the genome and saved it as **`purine.psych.fa`**. Let's align it our model:

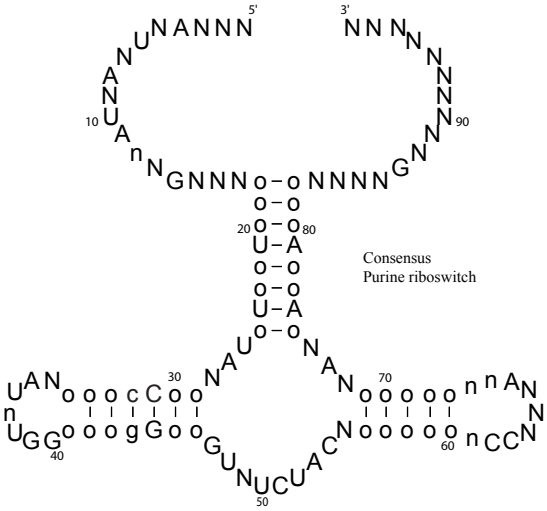
```
> cmalign -o purine.3.sto --withali purine.2.sto purine.2.cm purine.psych.fa
```

This alignment is depicted in the structure on the right below. Each column of the alignment is represented by a residue. Lowercase residues indicate positions where are least one sequence is a gap. For single-stranded residues: Ns denote any column that does not have identical residues in all three sequences, and non-Ns indicate all three sequence are identical. Any base-pair for which at least two distinct Watson-Crick or U-G/G-U pairs are present are indicated by open circles. Notice that 16 of the 20 base pairs show compensatory mutations between at least two of our three sequences.

Now, in theory we're ready for another round of iterative search. However, this is as far as we'll go with the guided tutorial.

This example was contrived to showcase the power of CMs to detect homologous structural RNAs with very little primary sequence conservation. The iterative approach is a powerful one, and we'd love to be able to automate it, but we haven't tried. Mainly because we're seriously impeded by the incredibly slow calibration step. As E-values for CMs become more completely understood, we hope to be able to streamline calibration and implement automated iterative search routines.





Parallelizing search, alignment and calibration with Message Passing Interface (MPI)

As mentioned in the Installation section, four of the seven INFERNAL programs can be run in parallel using MPI: **`cmalign`**, **`cmcalibrate`**, **`cmscore`** and **`cmsearch`**. These programs must be run using **`mpirun`**. These MPI programs are under current development, and we have only tested them using the LAM and OpenMPI implementations of MPI. Here are example runs using LAM:

```
> mpirun C cmsearch --mpi query.cm target.fa

> mpirun C cmcalibrate --mpi query.cm

> mpirun C cmalign --mpi query.cm target.fa

> mpirun C cmscore --mpi query.cm
```

Getting more information

For a quick refresher on the command line usage of any program and its commonly used options, just type the name of the program with no other arguments: e.g.

```
> cmemit
```

and you'll get a brief help:

```
> cmemit
Incorrect number of command line arguments.
Usage: cmemit [-options] <cmfile> <sequence output file>

  where basic options are:
  -h        : show brief help on version and usage
  -n <n>    : generate <n> sequences  [10]  (n>0)
  -u        : write generated sequences as unaligned FASTA  [default]
  -a        : write generated sequences as a STOCKHOLM alignment
  -c        : generate a single "consensus" sequence only
  -l        : local; emit from a locally configured model
  -s <n>    : set random number generator seed to <n>  (n>0)
  --devhelp : show list of otherwise undocumented developer options

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

To see more help on other available options, do **`cmemit -h`**

```
> cmemit -h
```

```
# cmemit :: generate sequences from a covariance model
# INFERNAL 1.0 (June 2008)
# Copyright (C) 2001-2008 HHMI Janelia Farm Research Campus
# Freely distributed under the GNU General Public License (GPL)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Usage: cmemit [-options] <cmfile> <sequence output file>

where general options are:
  -h        : show brief help on version and usage
  -n <n>    : generate <n> sequences  [10]  (n>0)
  -u        : write generated sequences as unaligned FASTA  [default]
  -a        : write generated sequences as a STOCKHOLM alignment
  -c        : generate a single "consensus" sequence only
  -l        : local; emit from a locally configured model
  -s <n>    : set random number generator seed to <n>  (n>0)
  --devhelp : show list of otherwise undocumented developer options

miscellaneous output options are:
  --rna       : output alignment as RNA sequence data  [default]
  --dna       : output alignment as DNA (not RNA) sequence data
```

```
  --tfile <f> : dump parsetrees to file <f>

expert options:
  --exp <x>   : exponentiate CM probabilities by <x> before emitting  (x>0)
  --begin <n> : truncate alignment, begin at match column <n>  (n>=1)
  --end <n>   : truncate alignment,   end at match column <n>  (n>=1)
```

More detailed information on usage and command line options is available in UNIX manual pages. If they have been installed for your system, you can see this information with, e.g.:

```
> man cmalign
```

Copies of the man pages are also provided at the end of this guide.

4 Profile SCFG construction: the `cmbuild` program

INFERNAL builds a model of consensus RNA secondary structure using a formalism called a *covariance model* (CM), which is a type of *profile stochastic context-free grammar* (profile SCFG) (Eddy and Durbin, 1994; Durbin et al., 1998; Eddy, 2002).

What follows is a technical description of what a CM is, how it corresponds to a known RNA secondary structure, and how it is built and parameterized.[2] You certainly don't have to understand the technical details of CMs to understand **`cmbuild`** or INFERNAL, but it will probably help to at least skim this part. After that is a description of what the **`cmbuild`** program does to build a CM from an input RNA multiple alignment, and how to control the behavior of the program.

Technical description of a covariance model

Definition of a stochastic context free grammar

A stochastic context free grammar (SCFG) consists of the following:

- M different nonterminals (here called *states*). I will use capital letters to refer to specific nonterminals; V and Y will be used to refer generically to unspecified nonterminals.
- K different terminal symbols (e.g. the observable alphabet, a,c,g,u for RNA). I will use small letters a, b to refer generically to terminal symbols.
- a number of *production rules* of the form: $V \rightarrow \gamma$, where γ can be any string of nonterminal and/or terminal symbols, including (as a special case) the empty string ϵ.
- Each production rule is associated with a probability, such that the sum of the production probabilities for any given nonterminal V is equal to 1.

SCFG productions allowed in CMs

A CM is a specific, repetitive SCFG architecture consisting of groups of model states that are associated with base pairs and single-stranded positions in an RNA secondary structure consensus. A CM has seven types of states and production rules:

State type	Description	Production	Emission	Transition
P	(pair emitting)	$P \rightarrow aYb$	$e_v(a,b)$	$t_v(Y)$
L	(left emitting)	$L \rightarrow aY$	$e_v(a)$	$t_v(Y)$
R	(right emitting)	$R \rightarrow Ya$	$e_v(a)$	$t_v(Y)$
B	(bifurcation)	$B \rightarrow SS$	1	1
D	(delete)	$D \rightarrow Y$	1	$t_v(Y)$
S	(start)	$S \rightarrow Y$	1	$t_v(Y)$
E	(end)	$E \rightarrow \epsilon$	1	1

Each overall production probability is the independent product of an emission probability e_v and a transition probability t_v, both of which are position-dependent parameters that depend on the state v (analogous

[2] Much of this text is taken from (Eddy, 2002).

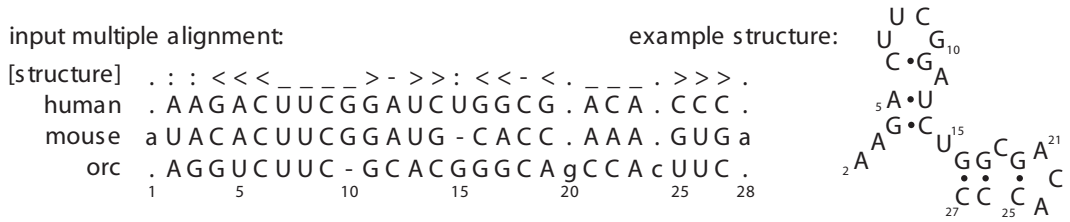


Figure 1: **An example RNA sequence family.** Left: a toy multiple alignment of three sequences, with 28 total columns, 24 of which will be modeled as consensus positions. The [structure] line annotates the consensus secondary structure in WUSS notation. Right: the secondary structure of the "human" sequence.

to hidden Markov models). For example, a particular pair (P) state v produces two correlated letters a and b (e.g. one of 16 possible base pairs) with probability $e_v(a, b)$ and transits to one of several possible new states Y of various types with probability $t_v(Y)$. A bifurcation (B) state splits into two new start (S) states with probability 1. The E state is a special case ϵ production that terminates a derivation.

A CM consists of many states of these seven basic types, each with its own emission and transition probability distributions, and its own set of states that it can transition to. Consensus base pairs will be modeled by P states, consensus single stranded residues by L and R states, insertions relative to the consensus by more L and R states, deletions relative to consensus by D states, and the branching topology of the RNA secondary structure by B, S, and E states. The procedure for starting from an input multiple alignment and determining how many states, what types of states, and how they are interconnected by transition probabilities is described next.

From consensus structural alignment to guide tree

Figure 1 shows an example input file: a multiple sequence alignment of homologous RNAs, with a line in WUSS notation that describes the consensus RNA secondary structure. The first step of building a CM is to produce a binary *guide tree* of *nodes* representing the consensus secondary structure. The guide tree is a parse tree for the consensus structure, with nodes as nonterminals and alignment columns as terminals.

The guide tree has eight types of nodes:

Node	Description	Main state type
MATP	(pair)	P
MATL	(single strand, left)	L
MATR	(single strand, right)	R
BIF	(bifurcation)	B
ROOT	(root)	S
BEGL	(begin, left)	S
BEGR	(begin, right)	S
END	(end)	E

These consensus node types correspond closely with the CM's final state types. Each node will eventually contain one or more states. The guide tree deals with the consensus structure. For individual sequences, we will need to deal with insertions and deletions with respect to this consensus. The guide tree is the skeleton on which we will organize the CM. For example, a MATP node will contain a P-type state to model a

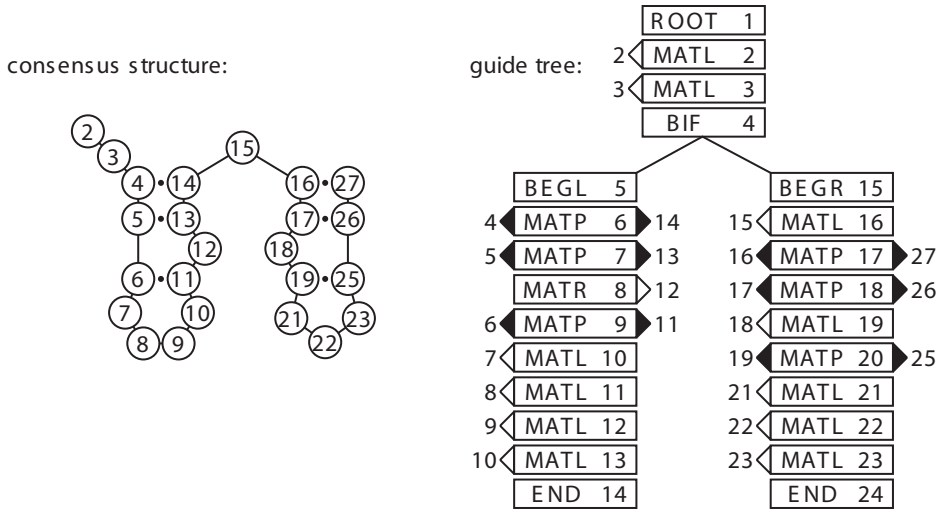


Figure 2: **The structural alignment is converted to a guide tree.** Left: the consensus secondary structure is derived from the annotated alignment in Figure 1. Numbers in the circles indicate alignment column coordinates: e.g. column 4 base pairs with column 14, and so on. Right: the CM guide tree corresponding to this consensus structure. The nodes of the tree are numbered 1..24 in preorder traversal (see text). MATP, MATL, and MATR nodes are associated with the columns they generate: e.g., node 6 is a MATP (pair) node that is associated with the base-paired columns 4 and 14.

consensus base pair; but it will also contain several other states to model infrequent insertions and deletions at or adjacent to this pair.

The input alignment is first used to construct a consensus secondary structure (Figure 2) that defines which aligned columns will be ignored as non-consensus (and later modeled as insertions relative to the consensus), and which consensus alignment columns are base-paired to each other. For the purposes of this description, I assume that both the structural annotation and the labeling of insert versus consensus columns is given in the input file, as shown in the alignment in Figure 1, where both are are indicated by the WUSS notation in the [structure] line (where, e.g., insert columns are marked with `.`). (In practice, **`cmbuild`** does need secondary structure annotation, but it doesn't require insert/consensus annotation or full WUSS notation in its input alignment files; this would require a lot of manual annotation. More on this later.)

Given the consensus structure, consensus base pairs are assigned to MATP nodes and consensus unpaired columns are assigned to MATL or MATR nodes. One ROOT node is used at the head of the tree. Multifurcation loops and/or multiple stems are dealt with by assigning one or more BIF nodes that branch to subtrees starting with BEGL or BEGR head nodes. (ROOT, BEGL, and BEGR start nodes are labeled differently because they will be expanded to different groups of states; this has to do with avoiding ambiguous parse trees for individual sequences, as described below.) Alignment columns that are considered to be insertions relative to the consensus structure are ignored at this stage.

In general there will be more than one possible guide tree for any given consensus structure. Almost all of this ambiguity is eliminated by three conventions: (1) MATL nodes are always used instead of MATR nodes where possible, for instance in hairpin loops; (2) in describing interior loops, MATL nodes are used before MATR nodes; and (3) BIF nodes are only invoked where necessary to explain branching secondary structure stems (as opposed to unnecessarily bifurcating in single stranded sequence). One source of ambiguity remains. In invoking a bifurcation to explain alignment columns $i..j$ by two substructures on columns

$i..k$ and $k+1..j$, there will be more than one possible choice of k if $i..j$ is a multifurcation loop containing three or more stems. The choice of k impacts the performance of the divide and conquer algorithm; for optimal time performance, we will want bifurcations to split into roughly equal sized alignment problems, so I choose the k that makes $i..k$ and $k+1..j$ as close to the same length as possible.

The result of this procedure is the guide tree. The nodes of the guide tree are numbered in preorder traversal (e.g. a recursion of "number the current node, visit its left child, visit its right child": thus parent nodes always have lower indices than their children). The guide tree corresponding to the input multiple alignment in Figure 1 is shown in Figure 2.

From guide tree to covariance model

A CM must deal with insertions and deletions in individual sequences relative to the consensus structure. For example, for a consensus base pair, either partner may be deleted leaving a single unpaired residue, or the pair may be entirely deleted; additionally, there may be inserted nonconsensus residues between this pair and the next pair in the stem. Accordingly, each node in the master tree is expanded into one or more *states* in the CM as follows:

Node	States	total # states	# of split states	# of insert states
MATP	[MP ML MR D] IL IR	6	4	2
MATL	[ML D] IL	3	2	1
MATR	[MR D] IR	3	2	1
BIF	[B]	1	1	0
ROOT	[S] IL IR	3	1	2
BEGL	[S]	1	1	0
BEGR	[S] IL	2	1	1
END	[E]	1	1	0

Here we distinguish between consensus ("M", for "match") states and insert ("I") states. ML and IL, for example, are both L type states with L type productions, but they will have slightly different properties, as described below.

The states are grouped into a *split set* of 1-4 states (shown in brackets above) and an *insert set* of 0-2 insert states. The split set includes the main consensus state, which by convention is first. One and only one of the states in the split set must be visited in every parse tree (and this fact will be exploited by the divide and conquer algorithm). The insert state(s) are not obligately visited, and they have self-transitions, so they will be visited zero or more times in any given parse tree.

State transitions are then assigned as follows. For bifurcation nodes, the B state makes obligate transitions to the S states of the child BEGL and BEGR nodes. For other nodes, each state in a split set has a possible transition to every insert state in the *same* node, and to every state in the split set of the *next* node. An IL state makes a transition to itself, to the IR state in the same node (if present), and to every state in the split set of the next node. An IR state makes a transition to itself and to every state in the split set of the next node.

There is one exception to this arrangement of transitions: insert states that are immediately before an END node are effectively *detached* from the model by making transitions into them impossible. This inelegant solution was imposed on the CM model building procedure to fix a design flaw that allowed an

ambiguity in the determination of a parsetree given a structure. The detachment of these special insert states removes this ambiguity.

This arrangement of transitions guarantees that (given the guide tree) there is unambiguously one and only one parse tree for any given individual structure. This is important. The algorithm will find a maximum likelihood parse tree for a given sequence, and we wish to interpret this result as a maximum likelihood structure, so there must be a one to one relationship between parse trees and secondary structures (Giegerich, 2000).

The final CM is an array of M states, connected as a directed graph by transitions $t_v(y)$ (or probability 1 transitions $v \rightarrow (y, z)$ for bifurcations) with the states numbered such that $(y, z) \geq v$. There are no cycles in the directed graph other than cycles of length one (e.g. the self-transitions of the insert states). We can think of the CM as an array of states in which all transition dependencies run in one direction; we can do an iterative dynamic programming calculation through the model states starting with the last numbered end state M and ending in the root state 1. An example CM, corresponding to the input alignment of Figure 1, is shown in Figure 3.

As a convenient side effect of the construction procedure, it is guaranteed that the transitions from any state are to a *contiguous* set of child states, so the transitions for state v may be kept as an offset and a count. For example, in Figure 3, state 12 (an MP) connects to states 16, 17, 18, 19, 20, and 21. We can store this as an offset of 4 to the first connected state, and a total count of 6 connected states. We know that the offset is the distance to the next non-split state in the current node; we also know that the count is equal to the number of insert states in the current node, plus the number of split set states in the next node. These properties make establishing the connectivity of the CM trivial. Similarly, all the parents of any given state are also contiguously numbered, and can be determined analogously. We are also guaranteed that the states in a split set are numbered contiguously. This contiguity is exploited by the divide and conquer implementation.

Parameterization

Using the guide tree and the final CM, each individual sequence in the input multiple alignment can be converted unambiguously to a CM parse tree, as shown in Figure 4. Weighted counts for observed state transitions and singlet/pair emissions are then collected from these parse trees. These counts are converted to transition and emission probabilities, as maximum *a posteriori* estimates using mixture Dirichlet priors.

Comparison to profile HMMs

The relationship between an SCFG and a covariance model is analogous to the relationship of hidden Markov models (HMMs) and profile HMMs for modeling multiple sequence alignments (Krogh et al., 1994; Durbin et al., 1998; Eddy, 1998). A comparison may be instructive to readers familiar with profile HMMs. A profile HMM is a repetitive HMM architecture that associates each consensus column of a multiple alignment with a single type of model node – a MATL node, in the above notation. Each node contains a "match", "delete", and "insert" HMM state – ML, IL, and D states, in the above notation. The profile HMM also has special begin and end states. Profile HMMs could therefore be thought of as a special case of CMs. An unstructured RNA multiple alignment would be modeled by a guide tree of all MATL nodes, and converted to an unbifurcated CM that would essentially be identical to a profile HMM. (The only difference is trivial; the CM root node includes a IR state, whereas the start node of a profile HMM does not.) All the other node types (especially MATP, MATR, and BIF) and state types (e.g. MP, MR, IR, and B) are SCFG augmentations necessary to extend profile HMMs to deal with RNA secondary structure.

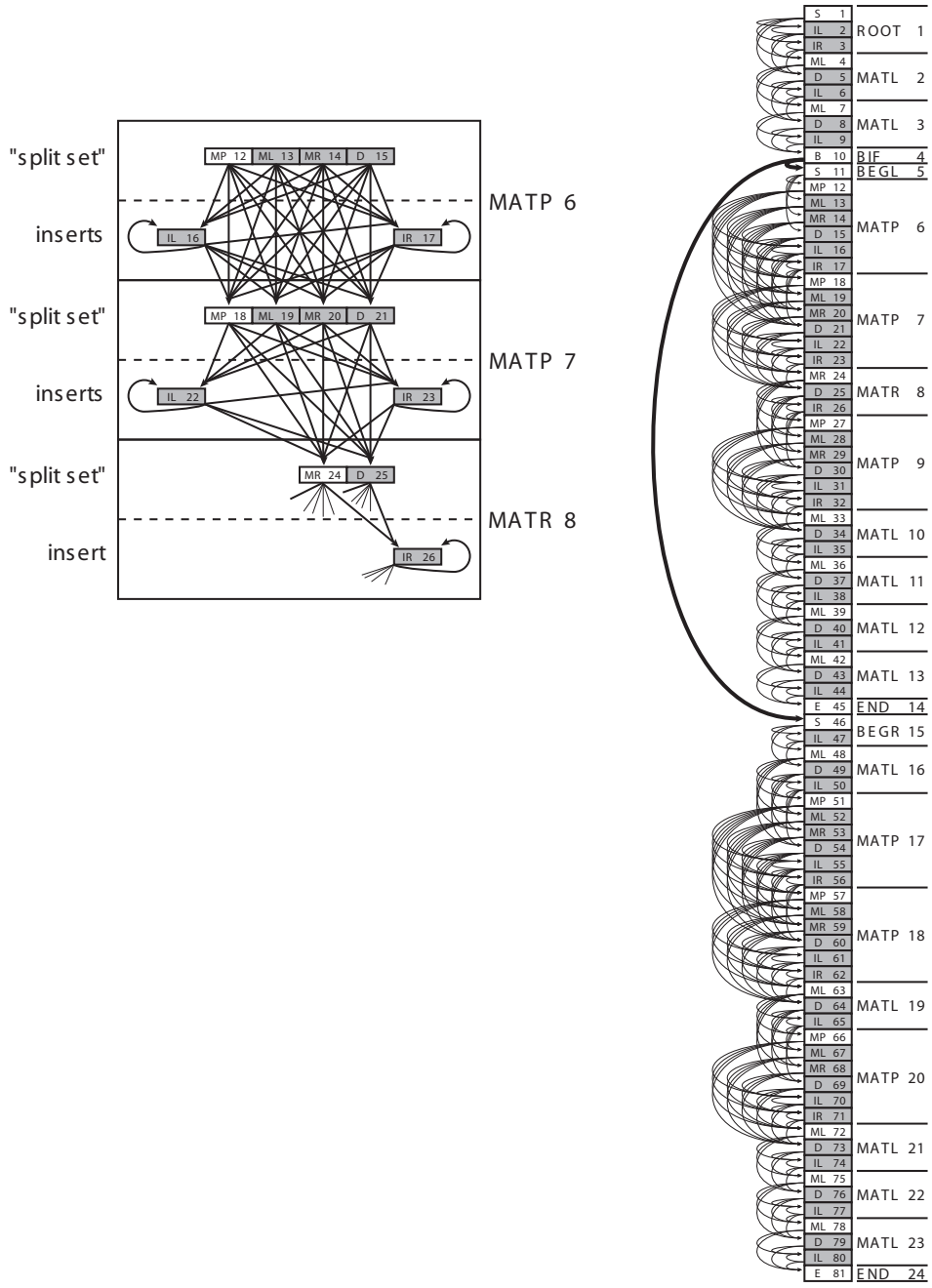


Figure 3: **A complete covariance model.** Right: the CM corresponding to the alignment in Figure 1. The model has 81 states (boxes, stacked in a vertical array). Each state is associated with one of the 24 nodes of the guide tree (text to the right of the state array). States corresponding to the consensus are in white. States responsible for insertions and deletions are gray. The transitions from bifurcation state B10 to start states S11 and S46 are in bold because they are special: they are an obligate (probability 1) bifurcation. All other transitions (thin arrows) are associated with transition probabilities. Emission probability distributions are not represented in the figure. Left: the states are also arranged according to the guide tree. A blow up of part of the model corresponding to nodes 6, 7, and 8 shows more clearly the logic of the connectivity of transition probabilities (see main text), and also shows why any parse tree must transit through one and only one state in each "split set".

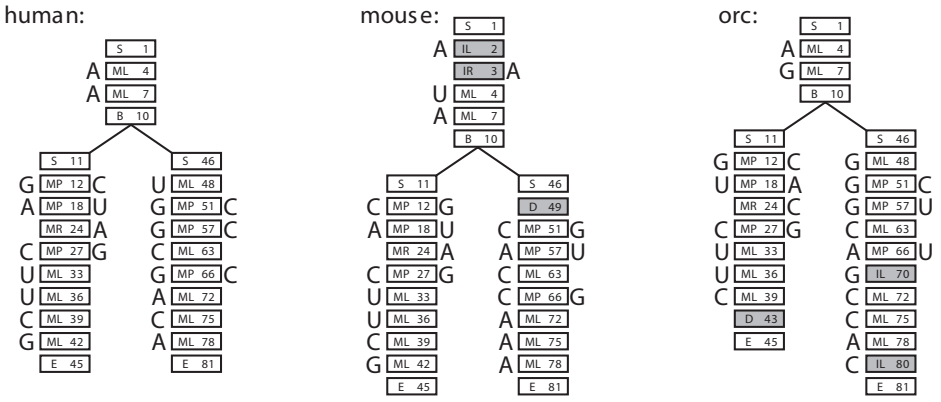


Figure 4: **Example parse trees.** Parse trees are shown for the three sequences/structures from Figure 1, given the CM in Figure 3. For each sequence, each residue must be associated with a state in the parse tree. (The sequences can be read off its parse tree by starting at the upper left and reading counterclockwise around the edge of parse tree.) Each parse tree corresponds directly to a secondary structure – base pairs are pairs of residues aligned to MP states. A collection of parse trees also corresponds to a multiple alignment, by aligning residues that are associated with the same state – for example, all three trees have a residue aligned to state ML4, so these three residues would be aligned together. Insertions and deletions relative to the consensus use nonconsensus states, shown in gray.

The `cmbuild` program, step by step

The **`cmbuild`** command line syntax is:

```
> cmbuild <options> [cmfile] [alifile]
```

where **`[alifile]`** is the name of the input alignment file, and **`[cmfile]`** is the name of the output CM file. What follows describes the steps that **`cmbuild`** goes through, and the most important options that can be chosen to affect its behavior.

Alignment input file

The input alignment file must be in Stockholm format, and it must have a consensus secondary structure annotation line (`#=GC SS_cons`).

The program is actually capable of reading many common multiple alignment formats (ClustalW, PHYLIP, GCG MSF, and others) but no other format currently supports consensus RNA secondary structure annotation. This may change in the future, either when other formats allow structure annotation, or when **`cmbuild`** is capable of inferring consensus structure from the alignment by automated comparative analysis, as the earlier COVE suite was capable of (Eddy and Durbin, 1994).

If the file does not exist, is not readable, or is not in a recognized format, the program exits with a "Alignment file doesn't exist or is not readable" error. If the file does not have consensus secondary structure annotation, the program exits with a "no consensus structure annotation" error. This includes all non-Stockholm alignment files.

Parsing secondary structure annotation

The structure annotation line only needs to indicate which columns are base paired to which. It does not have to be in full WUSS notation. Even if it is, the details of the notation are largely ignored. Nested pairs

of <>, (), [], or {} symbols are interpreted as base paired columns. All other columns marked with the symbols :,_-.~ are interpreted as single stranded columns.

A simple minimal annotation is therefore to use <> symbols to mark base pairs and . for single stranded columns.

If a secondary structure annotation line is in WUSS notation and it contains valid pseudoknot annotation (e.g. additional non-nested stems marked with AAA,aaa or BBB,bbb, etc.), this annotation is ignored because INFERNAL cannot handle pseudoknots. Internally, these columns are treated as if they were marked with . symbols.

> ▷ ***How should I choose to annotate pseudoknots?*** INFERNAL *can only deal with nested base pairs. If there is a pseudoknot, you have to make a choice of which stem to annotate as normal nested structure (thus including it in the model) and which stem to call additional "pseudoknotted" structure (thus ignoring it in the model). For example, for a simple two-stem pseudoknot, should you annotate it as* AAAA.<<<<aaaa....>>>>*, or* <<<<.AAAA>>>>....aaaa*? From an RNA structure viewpoint, which stem I label as the pseudoknotted one is an arbitrary choice; but since one of the stems in the pseudoknot will have to be modeled as a single stranded region by* INFERNAL*, the choice makes a slight difference in the performance of your model. You want your model to capture as much information content as possible. Thus, since the information content of the model is a sum of the sequence conservation plus the additional information contributed by pairwise correlations in base-paired positions, you should tend to annotate the shorter stem as the "pseudoknot" (modeling as many base pairs as possible), and you should also annotate the stem with the more conserved primary sequence as the "pseudoknot" (if one stem is more conserved at the sequence level, you won't lose as much by modeling that one as primary sequence consensus only).*

If (aside from any ignored pseudoknot annotation) the structure annotation line contains characters other than <>()[]{}:_-,.~ then those characters are ignored (treated as .) and a warning is printed.

If, after this "data cleaning", the structure annotation is inconsistent with a secondary structure (for example, if the number of < and > characters isn't the same), then the program exits with a "failed to parse consensus structure annotation" error.

Sequence weighting

By default, the input sequences are weighted in two ways to compensate for biased sampling (phylogenetic correlations). Relative sequence weights are calculated by the Gerstein/Chothia/Sonnhammer method (Gerstein et al., 1994). (The **--wgsc** option forces GSC weights, but is redundant since that's the default.) To turn relative weighting off (e.g. set all weights to 1.0), use the **--wnone** option.

Some alignment file formats allow relative sequence weights to be given in the file. This includes Stockholm format, which has #=GS WT weight annotations. Normally **cmbuild** ignores any such input weights. The **--wgiven** option tells **cmbuild** to use them. This lets you set the weights with any external procedure you like; for example, the **weight** utility program in SQUID implements some common weighting algorithms, including the fast $O(N)$ Henikoff position-based weights (Henikoff and Henikoff, 1994).

Absolute weights (the "effective sequence number") is calculate by "entropy weighting" (Karplus et al., 1998). This sets the balance between the prior and the data, and affects the information content of the model. Entropy weighting reduces the effective sequence number (the total sum of the weights) and increases the entropy (degrading the information content) of the model until a threshold is reached. The default entropy is 1.41 bits per position (roughly 0.59 bits of information, relative to uniform base composition). This threshold can be changed with the **--ere <x>** option. Entropy weighting may be turned off entirely with the **--enone** option.

Architecture construction

The CM architecture is now constructed from your input alignment and your secondary structure annotation, as described in the previous section.

The program needs to determine which columns are consensus (match) columns, and which are insert columns. (Remember that although WUSS notation allows insertions to be annotated in the secondary structure line, **cmbuild** is only paying attention to annotated base pairs.) By default, it does this by a simple rule based on the frequency of gaps in a column. If the frequency of gaps is greater than a threshold, the column is considered to be an insertion.

The threshold defaults to 0.5. It can be changed to another number <x> (from 0 to 1.0) by the **--gapthresh <x>** option. The higher the number, the more columns are included in the model. At **--gapthresh 1.0**, all the columns are considered to be part of the consensus. At **--gapthresh 0.0**, only columns with no gaps are.

You can also manually specify which columns are consensus versus insert by including reference coordinate annotation (e.g. a `#=GC RF` line, in Stockholm format) and using the **--rf** option. Any columns marked by non-gap symbols become consensus columns. (The simplest thing to do is mark consensus columns with x's, and insert columns with `.`'s. Remember that spaces aren't allowed in alignments in Stockholm format.) If you set the **--rf** option but your file doesn't have reference coordinate annotation, the program exits with an error.

Parameterization

Weighted observed emission and transition counts are then collected from the alignment data. These count vectors c are then converted to estimated probabilities p using mixture Dirichlet priors. The default mixture priors are described in (Nawrocki and Eddy, 2007). You can provide your own prior as a file, using the **--prior <f>** option.

Naming the model

Each CM gets a name. Stockholm format allows the alignment to have a name, provided in the `#=GF ID` tag. If this name is provided, it is used as the CM name.

Stockholm format allows more than one alignment per file, and **cmbuild** supports this: CM files can contain more than one model, and if you say e.g. **cmbuild Rfam.cm Rfam.sto** where `Rfam.sto` contains a whole database of alignments, **cmbuild** will create a database of CMs in the **Rfam.cm** file, one per alignment.

If a name or names are not provided in the Stockholm `#=GF ID` annotation, the name given to each CM is the input filename plus a "-" character and an integer indicating the position of that alignment within the alignment file. For example, if you build a model from a single alignment in alignment file **RNaseP.sto**, the model will be named RNaseP-1.

If the **--cmaxid, --ctarget** or **--call** options are used to split each input alignment into several alignments (see the Tutorial for an example), then an additional extension of "." plus an integer indicating the order in which the model was constructed from the alignment is added to the name. For example, if you run **cmbuild --ctarget 3** with the alignment file **RNaseP.sto**, **cmbuild** will cluster the alignment by sequence identity into 3 clusters and build 3 CMs, one from each cluster. These CMs will be named: RNaseP-1.1, RNaseP-1.2 and RNaseP-1.3.

If the alignment file only has 1 alignment in it, you can override the automatic naming conventions and provide your own name with the **-n <s>** option, where **<s>** is any string.

Saving the model

The model is now saved to a file, according to the filename specified on the command line. By default, a new file is created, and the model is saved in a portable ASCII text format.

If the cmfile already exists, the program exits with an error. The **-F** option causes the new model to overwrite an existing cmfile. The **-A** option causes the new model to be appended to an existing cmfile (creating a growing CM database, perhaps).

5 How `cmsearch` scores alignments and determines significance

The **`cmsearch`** program gives you a ranked list of hits in a sequence database. Which ones are likely to be true homologues and which ones are likely to be nonhomologous to your query CM?

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

Executive summary

- The best criterion of statistical significance is the E-value. The E-value is calculated from the bit score. It tells you how many false positives you would have expected to see at or above this bit score. Therefore a low E-value is best; an E-value of 0.1, for instance, means that there's only a 10% chance that you would've seen a hit this good in a target database of random sequences of the same size as the one you're searching. *Typically, I trust the results of searches at about E=0.1 and below, and I examine the hits manually down to E=10 or so.* However, be alert; INFERNAL E-values are not perfect.

- INFERNAL bit scores are a stricter criterion: they reflect whether the sequence is a better match to the profile model (positive score) or to the null model of nonhomologous sequences (negative score). A bit score above $\log_2$ of the size of the target database is likely to be a true homologue. For a 10 Mb genome, this rule-of-thumb number is on the order of 24 bits (remember a 10 Mb genome is really 20 Mb when you search both strands). Whereas the E-value measures how statistically significant the bit score is, the bit score itself is telling you how well the sequence matches your model. Because these things should be strongly correlated, usually, true homologues will have both a good bit score and a good E-value. However, sometimes (and these are the interesting cases), you will find remote homologues which do not match the model well (and so do not have good bit scores – possibly even negative), but which nonetheless have significant E-values, indicating that the bit score, though "bad", is still better than you would've expected by chance, so it is suggestive of homology. However, as above, be alert; it is possible to get artifactually high bit scores simply because of highly biased residue composition.

- For Rfam CMs, you can also examine three other numbers that represent bit score thresholds: a TC (trusted cutoff) score, a GA (gathering) score, and a NC (noise cutoff) score. The meaning of these numbers is described below.

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

In more detail: INFERNAL bit scores

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

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

$P(\text{seq}|\text{CM})$ is the probability of the target sequence according to your CM. $P(\text{seq}|\text{null})$ is the probability of the target sequence given a "null hypothesis" model of the statistics of random sequence. In INFERNAL, this null model is a simple one-state CM that says that random sequences are i.i.d. sequences with a specific residue composition, which by default is equiprobable across the four RNA nucleotides ($P(A) = P(C) = P(G) = P(U) = 0.25$). This "null model distribution" is part of the CM save file, and it can be altered when you run **cmbuild**.

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

You can specify the E-value or bit score cutoff to the **cmsearch** program. By default, for calibrated models (see below) the E-value cutoff is set as 1.0, and for non-calibrated models (for which E-values are not available) the cutoff is set as a bit score of 0.0 bits. (This is discussed in more detail in section 6). Alternatively, for a specific sensible use cutoffs, read about how the Rfam TC/NC/GA cutoffs are set and used (below).

In more detail: INFERNAL E-values

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

Unlike the raw score, the E-value is dependent on the size of the database you search. If you detect a hit of length 100 residues with an E-value of 0.1 in a search of a sequence database of size 100 Mb, then you happen to re-score the sequence all by itself, you will get an E-value 1 million times better. The E-value is quite literally the expected number of false positives at this raw score; the larger the database you search, the greater the number of expected false positives.

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

To calculate E-values, models must be calibrated with the **cmcalibrate** program. Unfortunately, **cmcalibrate** is painfully slow. We're working on making it faster, but for now, it is highly recommended that you spend the compute time to calibrate your models before searching with them. Not only will calibrated models report E-values, but they are often much faster at searching. This is because, in addition to calibrating E-value statistics, **cmcalibrate** also determines appropriate HMM filter cutoffs to use for each model, as explained in section 6.

cmcalibrate writes several parameters into your CM file on lines with labels starting with "E-". Among these parameters are the μ (location) and λ (scale) parameters of an exponential tail that best fits a histogram of scores calculated on randomly generated sequences. You only need to do this calibration once for a given CM. All the Rfam CMs come pre-calibrated.

WARNING: Using negative bit score thresholds for local searches

If the bit score threshold you use with local searches lower than -5, you may begin to observe some strange behavior. At a -10 bit threshold, **cmsearch** will start to report hits of a single residue in the target, that

are clearly not homologous sequences. This is because the local alignment of a single residue to an average sized CM scores about -10 bits. In general, for local searches the lowest recommended bit score cutoff is about -5 bits. If you commonly use E-value cutoffs for local searches, be sure to check that the bit score threshold that corresponds to your E-value cutoff is at least -5 bits. In practice this should only happen if you use a high E-value cutoff for a small target database, for example an E-value cutoff of 1000 for a 1000 nucleotide database. This issue is specific to local searches, and does not apply to glocal searches (enabled with the **-g** option).

fitting exponential tails to INFERNAL score histograms

The **cmcalibrate** program fits exponential tails to scores of the model against random sequences. When **cmsearch** is run and hits are found, the exponential tail parameters are used to calculate E-value for the hits. **cmsearch** implements different search algorithms and allows searching with both local and glocally configured models. Importantly, the configuration and algorithm used affect the scores reported, so **cmcalibrate** must fit separate exponential tails for each. In total, **cmcalibrate** fits 8 different exponential tail distributions. These are listed by the program as it proceeds through each of the 8 stages. Because **cmcalibrate** is so slow, it has a **--forecast** option which will list each stage and predict it's required running time. Let's examine the output with this option for the **my.cm** model from the tutorial:

```
> cmcalibrate --forecast 1 my.cm
```

```
# Forecasting time for 1 processor(s) to calibrate CM 1: trna.5-1
#
# stage     mod  cfg  alg  expL (Mb)   filN predicted time
# --------  ---  ---  ---  --------- ------ --------------
  exp tail  hmm  glc  vit      15.00      -       00:02:29
  exp tail  hmm  glc  fwd      15.00      -       00:04:59
  exp tail   cm  glc  cyk       1.50      -       00:06:29
  exp tail   cm  glc  ins       1.50      -       00:23:34
  filter      -  glc    -          -  10000       00:02:15
  exp tail  hmm  loc  vit      15.00      -       00:02:45
  exp tail  hmm  loc  fwd      15.00      -       00:05:59
  exp tail   cm  loc  cyk       1.50      -       00:05:45
  exp tail   cm  loc  ins       1.50      -       00:23:32
  filter      -  loc    -          -  10000       00:07:40
# --------  ---  ---  ---  ---------  -----  --------------
# all         -    -    -          -      -       01:25:33
```

Let's go through what each column of the output is telling us:

stage what type of stage this row pertains to, either "exp tail" for exponential tail fitting, or "filter" for filter threshold calculation. For now we're just focusing on the exponential tail stages.

mod the model we're fitting an exponential tail for, either the CM or a CP9 HMM. We use the HMM exponential tails during HMM filtering for faster searches.

cfg the configuration of the model for this stage, either "glc" for glocal or "loc" for local. **cmcalibrate** has to calibrate exponential tails *separately* for glocal and local modes.

alg the search algorithm for this stage. There are two algorithms that need to be separately calibrated for each the CM and HMM. These are explained in more detail in section 6.

expL the length of random sequence we'll search for this stage. For CM stages, this length is 1,500,000 residues (1.5 Mb) by default. For HMM stages, this length is is the minimum of 15 Mb and a length x, where x is the length that will cause this HMM stage to require 10% as much compute time as a CM calibration stage. The reasoning behind this is that longer sequence lengths tend to result in more accurate E-values.

filN the number of random sequences that will be searched for the HMM filter threshold calculation. This isn't relevant now; see section 6 for more on HMM filter threshold calculation.

predicted time the predicted run time of this stage.

As you can see, **cmcalibrate** will fit separate exponential tails for each combination of model and algorithm (4 choices) and configuration (2 choices) that's $4 * 2 = 8$ exponential tails. We explained the difference between local versus glocal configuration in the tutorial, now we'll discuss different search algorithms.

In more detail: different search algorithms in INFERNAL

cmsearch implements four several different search algorithms, Inside and CYK for profile SCFG search with CMs, and Forward and Viterbi for profile HMM search with CP9 HMMs. These algorithms differ in the meaning of the score that they calculate. Above we oversimplified an INFERNAL bit score as:

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

This is actually only correct for the Inside algorithm. The other three algorithms listed above are calculating something slightly different in the *numerator* (only) of the above equation, as follows:

Inside	$P(\text{seq}|\text{CM})$
CYK	$P(\text{seq}, \pi|\text{CM})$
Forward	$P(\text{seq}|\text{HMM})$
Viterbi	$P(\text{seq}, \pi|\text{HMM})$

Here, π represents the optimal (most probable) parse, or alignment, of the sequence to the model. So, the CYK and Viterbi algorithms report the log-odds score for the optimal alignment of the sequence given the model (either CM or HMM respectively), while the Inside and Forward algorithms report the log odds score for the sequence *summed over all possible alignments* of the sequence to the model.

From a probabilistic inference standpoint, the Inside and Forward scores are giving us what we want to know, the log-odds score that the sequence we're looking at is a homolog of the family we're modelling. The specific alignment π to the model is a nuisance variable that is appropriately integrated out. Not only are

Inside and Forward better in theory, but empirically, benchmarks show that they're more sensitive than CYK and Viterbi. So, why use CYK and Viterbi at all? Actually, we rarely use Viterbi really, it's only implemented for testing purposes (it's okay; calibrating Viterbi E-values takes a very small fraction of the running time of **cmcalibrate**, you can see for yourself in the previous example). CYK *is* used, but mainly as a filter in **cmsearch**. CYK makes a good filter for two reasons. First, it can be more efficiently implemented than the Inside algorithm. INFERNAL's current implementation of CYK is about three times faster than Inside (you can see this in the predicted run times in the **--forecast** example). Secondly, high scoring CYK hits, the ones where interested in, tend to have a single well-defined alignment and consequently approximate Inside scores well. Combined, these two features mean we can safely and effectively filter with CYK, and have the speed of CYK and sensitivity of Inside. You can read more about how we use CYK as a filter in **cmsearch** in section 6.

▷ ***Why are you talking about HMM algorithms. I thought CMs were more appropriate for structural RNA sequence analysis than HMMs, isn't that the whole reason you've developed* INFERNAL*?*** *We've implemented HMMs in* INFERNAL *to help accelerate CM algorithms. As described in section 6, HMMs are used as filters to speedup database searches with* ***cmsearch****. They're also used to develop constraints for CM alignment with* ***cmalign****.*

▷ ***Why is cmcalibrate so slow?*** *The necessity of doing 8 separate exponential tail fitting stages (see above), and the fact that CM search algorithms scale more than LN^2 with sequence length L and model consensus length N make the program excruciatingly slow. Notice that some of the 8 stages are much faster than others. The two Inside stages usually take a large majority of the total run time. You might argue this means we shouldn't use Inside, but it's the most sensitive algorithm we have; and sensitivity trumps speed in our design goals.*

Accuracy of E-values

The E-values reported by **cmsearch** are not extremely accurate. This is because CM scores don't fit the exponential tail distribution extremely well. Empirically, local alignment scores returned from local searches fit exponential tails fairly well, and glocal alignment scores fit less well. We will continue to work on making E-values more accurate in future versions of INFERNAL.

One important factor in the accuracy of E-values is the GC content of the target database. This is because many CMs are biased towards particular GC content (usually low GC), in many cases giving A-rich sequence high scores simply due to their high composition of As. **cmsearch** partially alleviates the problem of biased composition using a post-hoc correction called null3 (see "Biased composition filtering: the null3 model below"). However, the problem still persists and is particularly evident when the target database being searched has a very low (about 25%) or very high (about 65%) GC content. **cmcalibrate** calibrates the E-value statistics for a genome composition of about 46% GC (based on an average measure from a sampling of genomes from the three domains of life). When a target database with GC content lower than 46% is searched, the effect is that there are usually more high scoring hits than the calibrated CM expects, and lower E-values for random hits can result (that is you might observe 5-25 hits with an E-value less than 1 for example). Conversely, when a target database with GC content above 46% is searched, the effect is the opposite, and higher E-values often result (you might observe only 1-5 hits with an E-value less than 25). These effects are the average case, but different models will give different results. We consider the E-values in INFERNAL a work in progress, so feel free to report examples of particularly bad (or good) performance of E-values to us.

In more detail: Rfam TC/NC/GA cutoffs

When a Rfam model is built, the Rfam curation team keeps track of scores of every hit in a large nonredundant database. They record three types of score cutoffs on Rfam CM files:

GA (gathering cutoff) : the score used as cutoff in constructing Rfam. All database hits that score at or above the GA bit score will be included in the Rfam "full alignment".

TC (trusted cutoff) : the scores of the lowest-scoring hit(s) that were included as true member(s) of the Rfam family. Hits above the TC score are "within" the Rfam family and almost certainly members.

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

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

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

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

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

> ▷ ***Why isn't sequence X included in a Rfam full alignment? It has a significant score!*** *For the reasons above, the sequences in Rfam full alignments are harvested using curated GA thresholds, rather than using score or E-value thresholds. Please don't go writing a paper that claims CMs don't detect some similarity until you've done the experiment with CMs (and E-values) instead of just looking at curated Rfam classifications.*

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

Predicting running times for searches with Rfam cutoffs

The next section explains how **`cmcalibrate`** determines HMM filter thresholds to use to accelerate **`cmsearch`**. These filter thresholds are dependent on the final threshold used in **`cmsearch`**, in general, the stricter the final threshold the stricter the filter threshold and the greater the acceleration from the filter. Because the

Rfam GA/NC/TC cutoffs are relatively strict it is often possible to achieve large speedups of up to 100-fold or more when they're used as the final threshold. Section 6 explains this in more detail, and shows an example of predicting the running time of filtered searches with Rfam cutoffs using the **cmstat** program.

Biased composition filtering: the null3 model

I've lied. INFERNAL bit scores are actually calculated as log odds scores relative to *two* null hypotheses. The first is the null model built into the CM when it was built with **cmbuild**. The second, called *null3*, is an *ad hoc* model calculated on the fly for each alignment, from the characteristics of that alignment. (Why null3? To differentiate it from the null2 model used by the HMMER software package). The purpose of null3 is to compensate for false positive hits caused by simple biased composition regions in the target sequence.

Common biased composition filters like XNU, DUST, and SEG are qualitative filters – if a region is detected as biased composition, it is masked (the residues are converted to X's). The INFERNAL composition filter is a quantitative filter, that tests whether the sequence is a better match to the CM, the null (random composition) model, or a null3 model of biased nucleotide composition.

This is a Good Thing, but on the other hand, the null3 model is not very sophisticated. It is a single-state HMM just like the main null model, which means it only captures residue composition, like DUST; no attempt is made in INFERNAL to filter short-period repetitive sequences like the XNU algorithm does.

The null3 model was motivated by the observation that many high-scoring false positive hits in **cmsearch** are to regions of the target database with with highly biased residue composition, in particular regions with high percentages of A and U residues. The first null model used by INFERNAL is by default 25% A, C, G, and U (this model can be changed with the **--null** option to **cmbuild**). If a model has a bias for a particular residue, for example A, and a target regions is composed of an overrepresentation of that residue then it will receive a high score simply because it is A-rich.

A different null3 model is calculated for every alignment. The 4 emission probabilities of the null3 model are calculated as simply their occurence within the region of the hit. For example, if the hit is 50 residues long and contains 20 As, 5 Cs, 5 Gs and 20 Us, then the null3 model probabilitiles will be calculated as $(0.4, 0.1, 0.1, 0.4)$.

But now we've got *two* null hypotheses. We said we report a bit score that's a log-odds ratio of our model likelihood and *one* null hypothesis likelihood. How do we calculate a score if we have more than one null hypothesis? INFERNAL does a bit of algebraic sleight of hand here, to arrive at an additive correction to the original score that it calls the "null3 score correction".

derivation of the null3 score correction

We arrived at the parameters of the null3 model in a very *ad hoc* way. However, after that, the way INFERNAL arrives at the final bit score once the null3 parameters have been determined is clean (e.g. derivable) Bayesian probability theory. It is analagous to the way HMMER uses the *null2* score correction.

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

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

an equation which forces us to state explicit probabilistic models not just for the hypothesis we want to test, but also for the alternative hypotheses we want to test against. Up until now, we've considered two hypotheses for an observed sequence D: either it came from our CM (call that model M), or it came from

our null hypothesis for random, unrelated sequences (call that model N). If these are the only two models we consider, the Bayesian posterior for the model M is:

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

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

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

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

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

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

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

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

for *any* number of alternative hypotheses under consideration.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

s_2 The (simple, uncorrected) log odds score for the null3 hypothesis, calculated by rescoring the residues of the alignment under the null3 model.

π_2 The log odds of the priors, $\log P(N_2)/P(N_1)$. INFERNAL arbitrarily assumes that the null3 model is $\frac{1}{32}$ as likely as the main null model, so this factor is -5 bits.

The code that calculates the null3 correction is in **`cm_parsetree.c:ScoreCorrectionNull3()`**.

The null3 correction is usually close to zero, for random sequences, but becomes a significant quantitative penalty on biased composition sequences. It gets added to the original alignment score to form INFERNAL's final bit score.

The null3 score correction is introduced in version 1.0 and was not present in any of the 0.x versions of INFERNAL. This can lead to large differences in the scores reported by 1.0 and previous versions.

The following table shows the penalty for a 100 nucleotide hit with varying compositions of A, C, G, and U residues. This table is included to give you an idea of how severe the null3 correction is, and can be useful for comparing bit scores from INFERNAL 1.0 to previous versions (which did not use the null3 correction). These are just a sample of the possible composition of hits you might see. Again, these scores are for 100 nucleotide hits, to determine the correction for a hit of length x simply multiply the corresponding correction below by $x/100$. For example, a 35% A, 15% C, 15% G, 35% U hit of length 100 nt would receive a 6.88 bit penalty from null3 (row 4). A 200 nt hit of the same composition would receive a penalty of 13.76 bits. A 50 nt hit of the same composition would receive a 3.44 bit penalty.

GC%	A%	C%	G%	U%	NULL3 correction (bits)
0.0	50.0	0.0	0.0	50.0	95.00
10.0	45.0	5.0	5.0	45.0	48.10
20.0	40.0	10.0	10.0	40.0	22.81
30.0	35.0	15.0	15.0	35.0	6.88
35.0	32.5	17.5	17.5	32.5	2.01
40.0	30.0	20.0	20.0	30.0	0.30
45.0	27.5	22.5	22.5	27.5	0.07
50.0	25.0	25.0	25.0	25.0	0.04
55.0	22.5	27.5	27.5	22.5	0.07
60.0	20.0	30.0	30.0	20.0	0.30
65.0	17.5	32.5	32.5	17.5	2.01
70.0	15.0	35.0	35.0	15.0	6.88
80.0	10.0	40.0	40.0	10.0	22.81
90.0	5.0	45.0	45.0	5.0	48.10
100.0	0.0	50.0	50.0	0.0	95.00

By default, the null3 score correction is used by **`cmcalibrate, cmsearch`** and **`cmalign`**. It can be turned off in any of these programs by using the **`--no-null3`** option (which can only be seen in the list of options if **`--devhelp`** is invoked)). However, be careful, the E-values for models that are calibrated with **`--no-null3`** are only valid when **`--no-null3`** is also used with **`cmsearch`**. Likewise, if **`--no-null3`** is *not* used during calibration, it should not be used during search.

6 How `cmsearch` uses filters to accelerate search

A major limitation to the practical use of CMs is the slow running time of homology search implementations. **`cmsearch`** has two acceleration strategies for alleviating this. In this section we'll briefly describe these two strategies and how they're employed within **`cmsearch`**.

The two acceleration strategies are both filtering approaches. The main idea of filtering is to use a fast algorithm to do a first-pass scan of the database with a wide net, allowing everything that could possibly be a good hit to survive and eliminating everything that is very unlikely to be a good hit. After the filter is run, the expensive CM algorithms are run only on the surviving fraction of the database.

The first filtering approach uses an HMM as the fast first-pass algorithm. HMM filtering for CM homology search was introduced by Weinberg and Ruzzo (Weinberg and Ruzzo, 2004a; Weinberg and Ruzzo, 2004b; Weinberg and Ruzzo, 2006). We use a specific type of HMM architecture, essentially a reimplementation of Weinberg and Ruzzo's maximum-likelihood HMMs (Weinberg and Ruzzo, 2006), that we call CM plan 9 HMMs or CP9 HMMs (to distinguish them from the plan 7, or p7 HMMs of the HMMER software package http://hmmer.janelia.org). HMMs are unable to model the interactions between base-paired columns that a CM can model, which makes them less sensitive and specific for RNA sequence analysis, but in exchange they are more efficient to compute with, so they're faster than CMs and useful for filtering.

The second filtering approach is a banded dynamic programming technique called query-dependent banding (QDB). QDB precalculates regions of the CM dynamic programming matrix that have negligible probability and can be skipped to save time (Nawrocki and Eddy, 2007). This calculation is dependent only on the query CM itself, it's independent of the database being search, so it only has to be done once per model.

`cmsearch` uses both of these filtering strategies in combination. HMM filtering is faster but less specific than QDB filtering, so it is used first. Any hit that scores above the HMM filter threshold survives the HMM filter and is then searched with the QDB filter. Any hit that surives both filters is reevaluated again using the Inside algorithm, which is the slowest but most specific algorithm we have, which determines the final scores of the hits in the database.

The previous section discussed how **`cmcalibrate`** calibrates models, a step which we highly recommend before searching. **`cmsearch`** can be run with non-calibrated models but the filters are employed differently than for calibrated models. Next, we'll discuss how filters are used for searches with calibrated models, and then we'll discuss how they're used for searches with non-calibrated models.

Filtered searches with calibrated models

determining appropriate HMM filter score thresholds with `cmcalibrate`

A goal of our filtering approach is to sacrifice only a small amount of sensitivity for the win in speed. In other words, we want faster searches, but only if they won't miss potential homologs that a slow, non-filtered search would have found. Weinberg's rigorous HMM filters addressed this very nicely, by guaranteeing that all hits above a certain CM threshold will survive the filter (Weinberg and Ruzzo, 2004a). Our approach takes a slightly different tact, and relaxes the 100% guarantee to a 99.5% probability using a empirical sampling technique. That is, we estimate our HMM filter will cause us to miss 0.5% of the hits that a non-filtered search would have found. This sampling technique is performed by **`cmcalibrate`** and takes advantage of the CM as a generative probabilistic model. Sequences are generated, or emitted, from the model and searched with both an expensive CM algorithm (either CYK or Inside) and the faster HMM Forward search algorithm. The HMM scores are ranked and the HMM filter threshold is set as x, the 99.5%

worst score. If we assume the sample of sequences we drew from the HMM is representative of real RNA homologs of the family we're modelling, then using an HMM filter and a cutoff of x will allow 99.5% of the real homologs to survive. Of course, this is a strong and potentially dangerous assumption to make, but it essentially underlies our entire modelling approach - we're just assuming our CM is a good model of the family we're trying to model! In other words, if this assumption is wrong, we'll get poor performance for a variety of reasons, and an inappropriate filter threshold would be a relatively weak concern. In the future, as CMs get better due to improved parameterization, etc., our assumption will become more true, and this filtering strategy will improve as well.

When you calibrate a model with **cmcalibrate** the HMM filter thresholds are calculated using the sampling technique and printed to the CM file. (HMM filter threshold determination is one of two purposes of **cmcalibrate**, the other is fitting exponential tails for E-value calculation as described in section 5).

how cmsearch sets filter thresholds

When **cmsearch** sets its thresholds for each round of the search, it first sets the final round threshold. This is because the determination of the filter thresholds depends heavily on the final threshold. By default, the final round threshold used for calibrated model searches is $E = 1$ (see "manually setting filter and final thresholds" for information on how to change this). This E-value is then converted to a bit score based on the database size by using the exponential tail fit parameters stored in the CM file by **cmcalibrate**. **cmsearch** then sets the HMM filter threshold and then the QDB filter threshold based on the final round bit score threshold. In general, the more strict the final round threshold the more strict the filter thresholds will be.

cmcalibrate automatically determines the appropriate HMM filter score threshold as described above by generating sequences from the model and scoring them. During search, the appropriate threshold is automatically chosen dependent on the final bit score search threshold and set as the HMM filter score threshold for that search. In some cases, it is determined that HMM filtering is a bad idea because the expected survival fraction is close to 1.0, which mean the filter would save little if any time. In these cases, the HMM filter is turned off. This usually happens when searching small databases, using low final bit score thresholds (or high final E value thresholds), or when searching with a model that has very little primary sequence conservation.

cmsearch sets the QDB filter threshold based on the final round threshold in an *ad hoc* way that seems to work well empirically. Our goal with the QDB filter is the same as with the HMM filter, we want the filter to accelerate searches as much as possible while minimizing sensitivity loss. Our *ad hoc* approach sets the QDB filter threshold so that the following two conditions are met:

1. Our QDB filter will always allow hits with E-values of 100 times our final E-value threshold to survive.
2. The final round of search will require at least 3% of the total predicted number of dynamic programming (DP) calculations for the entire search (filters plus final round).

The reasoning here is that the vast majority of hits that will score below our final round threshold should satisfy condition 1, so our sensitivity loss from using filter will be small. Empirically we observe this to be true on internal benchmarks (data not shown). However, condition 2 states that we're also always willing to perform at least 3% of our overall number of DP calculations in the final round. The number of DP calculations in the final round can be predicted based on the size of the model and on the QDB E-value cutoff. So given the E-value x that satisfies condition 1, we can predict the fraction $f(x)$ of the total number of DP calculations that will occur in the final round. If $f(x)$ is *less* than 0.03 we can raise our QDB E-value

cutoff to the value y that satisfies $f(y) = 0.03$, and still meet both of our conditions. In raising the E-value cutoff to y, we're making the filter less strict, which can only reduce our potential sensitivity loss, but only slowing down the search a small amount because we've increased the number of calculations by at most 3%. The value 3% was chosen as a good tradeoff between speed and sensitivity in our benchmarks.

One final point: if the predicted survival fraction for the QDB filter E-value thresold as calculated above is greater than the predicted survival filter for the HMM filter then **cmsearch** will turn off the QDB filter mode and use only the HMM filter and the final round for searching.

▷ ***I ran cmsearch with a calibrated model, but HMM filters were not used for accleration. How come?*** *This is because* **cmcalibrate** *has determined that it's not "safe" to use an HMM filter for the search you're running. This means that with the final threshold you're using, if you used an HMM filter, to miss no more than 1% of the hits below your final threshold, the filter would have to let nearly the entire database survive. In this case* **cmsearch** *judges it's not worth it to use the HMM filter and skips that step of the search. If you really want to use an HMM filter you can either (A) make your final threshold stricter (***cmstat** *can help you determine what effect this will have, see below) or (B) manually set the HMM filter threshold as explained in the "manually setting filter and final thresholds" section.*

▷ ***I used cmsearch twice with the same model and the same E-value cutoff on two different databases, but the filter thresholds were set differently. Why?*** *The reason is that the databases were different size. The filter thresholds are set based on the bit score cutoff of the final round of the search you're performing. The bit score is dependent on both the E-value and the database size. For example, if you search two databases of size* 1 *Mb and* 100 *Mb respectively and use an E-value cutoff of* 0.1 *for both searches, the bit score cutoff for the 1 Mb database may be* 20 *bits, but it is* 27 *bits for the* 100 *Mb database. The filters can be more strict for the* 100 *Mb search because they only have to allow all hits that might score above* 27 *bits to survive instead of all hits that might score above* 20 *bits.*

filtered versus non-filtered search

Let's see an example of the filters in action. Then we'll compare the results to a search without using the filters. We'll use a calibrated CM from the tutorial section, the **purine.2.c.cm** file, a calibrated Purine riboswitch model, and we'll use it to search the *Colwellia psycherythraea* genome. In contrast to the tutorial, we'll use a more strict E-value cutoff of 0.001.

```
> cmsearch -E 0.001 purine2.c.cm C.psychrerythraea.genome.fa
```

In the tutorial we were only concerned with the hits reported by the program, but in this section we're interested in the rest of the output. Let's look at the header and pre-search info section:

```
# command:    cmsearch -E 0.001 purine.2.c.cm C.psychrerythraea.genome.fa
# date:       Tue Jun 10 22:58:15 2008
# num seqs:   1
# dbsize(Mb): 10.746360
#
# Pre-search info for CM 1: purine.2-1
#
#                                  cutoffs            predictions
#                              -------------------  --------------------
# rnd  mod  alg  cfg   beta     E value   bit sc     surv     run time
# ---  ---  ---  ---  -----  ----------  -------  -------  -----------
    1  hmm  fwd  loc      -    4041.305     6.61   0.0541  00:02:24.21
    2   cm  cyk  loc  1e-07      93.161     8.91   0.0012  00:02:29.60
    3   cm  ins  loc  1e-15     1.0e-03    29.33  8.9e-09  00:00:18.23
# ---  ---  ---  ---  -----  ----------  -------  -------  -----------
  all    -    -    -      -           -        -        -  00:05:12.05
```

First comes the command used and date of execution. Then the database size is printed in number of nucleotides. Remember, by default, **cmsearch** searches both strands of the database, so even though the sequence in **C.psychrerythraea.genome.fa** is only about 5 Mb long, the database size according to **cmsearch** is twice that, about 10 Mb. Following that is the number of sequences in the database. Then comes the tabular output section summarizing the filtering search strategy that **cmsearch** will use in this search. Here are descriptions of the different columns:

rnd the round of searching each row pertains to. In this search there's three rounds, two filter rounds plus the final round. The hits reported by **cmsearch** are those that survive both filters and the final round of searching.

mod the model each round of searching will use. For the first round we'll use a CP9 HMM to filter, for the last two rounds the CM is used.

alg the search algorithm used by each round. The HMM filter round always uses the Forward ("fwd") HMM algorithm. The QDB filter round always uses the CYK CM algorithm. The final round uses the Inside ("ins") CM algorithm by default, but the CYK algorithm can be used instead with the **--cyk** option.

cfg the configuration of the model during each round. In this case they are all locally configured. Glocal configuration is enabled with the **-g** option (see the tutorial).

beta the tail loss probability for the QDB calculation. This is the amount of probability mass allowed outside each band on the DP matrix. For round 2, the QDB filter round, this value is 1e-07, so 99.9999% of the probability mass is within each band. For the final round of searching, the bands are less strict, with beta values of 1e-15. For more information on this see (Nawrocki and Eddy, 2007).

cutoffs the cutoffs used for each round of searching, in **E value** and in **bit sc**. Notice how the E-values go down and the bit scores go up as you progress through the rounds, this means the cutoffs are getting stricter in each successive round.

predictions the predicted survival fraction (**surv**) and running time (**run time**) for each round. These are based on the E-value cutoffs, and are just predictions. A survival fraction of 0.0541 means that our E-values predict that 94.59% of the database will be removed by the HMM filter round. We observe these predictions are often inaccurate by up to five-fold, but they are the best predictions that we can make currently.

Look at the pre-search info for this search: the final round (round 3) E-value cutoff is set as 0.001 (we manually set it to this) which translates to bit score of 29.33 in a database of this size. (If we had not used the **-E** options, the E-value cutoff would have been the default value of 1 used when searching with a calibrated model). The HMM filter cutoff is a bit score of 6.61 bits which has an E-value of 4041. That seems high, but the **surv** column tells us that if 4041 hits survive the HMM filter, we'll filter out about 96% of the database. The QDB CYK filter threshold is 8.81 bits which has an E-value of about 93. This threshold has been set with our *ad hoc* method as a presumably safe bet to allow any hit that would exceed our final round threshold of 29.33 bits with the Inside algorithm to survive. These thresholds have all been set using the default methods in **cmsearch** for calibrated models, but if you want, you can change any or all of them using command-line options as described below in "Manually setting filter and final thresholds".

After a couple of minutes the search finishes, a single hit is output to the screen (see the tutorial for more on this), and the "Post-search info" section is printed:

```
# Post-search info for CM 1: purine.2-1
#
#                               number of hits       surv fraction
#                             ------------------  -----------------
# rnd  mod  alg  cfg   beta    expected   actual  expected   actual
# ---  ---  ---  ---  -----  ----------  -------  --------  -------
    1  hmm  fwd  loc      -    4041.305     2572    0.0541   0.0376
    2   cm  cyk  loc  1e-07      93.161       22    0.0012   0.0003
    3   cm  ins  loc  1e-15     1.0e-03        1   8.9e-09  6.2e-06
#
# expected time    actual time
# -------------  -------------
    00:05:12.05    00:04:02.00
```

Much of this information we already saw in the pre-search info, except the columns labelled **`actual`**. Our predictions were not very accurate, but they are within an order of magnitude for the most part. One striking disparity is the number of actual hits to survive round 2, the QDB filter round. We predicted that about 93 hits would survive, but only 22 did. One explanation for this is that the calculation of the expected number ignores the fact that the HMM filter will be run first, and remove much of the database. In reality, the QDB filter was only run on 3.76% of the full database, but the 93 estimate is based on searching the full database. (You might think that a better prediction would be to multiply our expected QDB hits of 93 by 0.0376, our expected survival fraction from round 1, but this is wrong - the fraction that survives the HMM filter is not just a random 3.76% of the database, it's the 3.76% that scores highest with our HMM, and there's no good theory we can come up with for predicting how many hits we'll get in a biased fraction of the database like that).

▷ ***Why are the `cmsearch` "run time" predictions are so inaccurate?*** *Short answer: our E-values are not perfect. In fact, they're far from perfect. We're working on making E-values in* INFERNAL *more accurate and easier to calculate. For now, this is the best we can do. Another reason is that for rounds 2 and 3 of* ***`cmsearch`*** *the search is being performed on the biased fraction of the database that has survived all the previous rounds, which makes it harder to predict the actual number of hits that will survive with accuracy (see above).*

So, did our filters do a good job? Well, it's hard to say from what we've seen. Using an E-value cutoff of 0.001 they allowed a single hit through that looks like a real Purine riboswitch (see the secondary structure figure of the tutorial). That's good, but who knows, we may have missed 100 others! And it's impossible to say how much time they've saved us. Well in this case, the experiment that answers both of these questions is easy enough to perform, we can run **`cmsearch`** with the filters turned off to get the non-filtered running time and to see if anything else is found. The command to do that is:

```
> cmsearch -E 0.001 --fil-no-hmm --fil-no-qdb purine.2.c.cm C.psychrerythraea.genome.fa
```

To save you the time of actually doing this we've saved the output of this search in the file **`purine.2.nofilter.cmsearch`**, but feel free to run it on your own. Take a look at the file. Here is the

post-search information:

```
# Post-search info for CM 1: purine2-1
#
#                               number of hits       surv fraction
#                            ------------------  -----------------
# rnd  mod  alg  cfg   beta    expected   actual  expected   actual
# ---  ---  ---  ---  -----  ----------  -------  --------  -------
    1   cm  ins  loc  1e-15       0.100        1   8.9e-07  6.2e-06
#
# expected time    actual time
# -------------  -------------
    03:13:25.22    03:55:36.00
```

The non-filtered search only finds this one hit below our E-value cutoff also. Looks like our filters did a good job in this case! Notice the time it took to run, about four hours, compared to about five minutes for our filtered search, so our filters give us an acceleration of about 50 fold.

▷ ***I just did a search and it took about as long as a non-filtered search. How come I didn't see a significant speedup?*** *The speedup does vary a lot depending on the model you're using and the size of the database you're searching.* ***cmcalibrate*** *estimates a HMM filter score cutoff that will accelerate the search as much as possible with without losing appreciable sensitivity. Sometimes this cutoff is very strict, leading to very effective filters, and sometimes it's not. As you might expect, in general we find that we can use HMM filters effectively for models with regions of high primary sequence similarity, and not so effectively for models lacking such regions. You can determine how how well an HMM can filter for your model* after *you've calibrated it using the* ***cmstat*** *program as described below.*

using `cmstat` to get information on HMM filters

The **`cmstat`** program can be used to predict how effective an HMM filter will be for searches with a calibrated CM file. The **`--lfi, --gfi, --lfc`** and **`--gfc`** options will display stats on HMM filtering for the four different final round search strategies: local Inside, glocal Inside, local CYK and glocal CYK modes respectively (by default **`cmsearch`** uses local Inside as the final round search strategy). The **`--seqfile <f>`** option can be used to specify the target database you want predictions for. Here's an example:

```
> cmstat --lfi --seqfile C.psychrerythraea.genome.fa purine.2.c.cm
```

```
#  idx  name         clen       F     nseq  db (Mb)  always?
# ----  ----------  -----  ------  -------  -------  -------
     1  purine.2-1    103  0.9950    10000     10.7       no
#
#
#       CM E-value cutoff / HMM Forward E-value filter cutoff pairs:
#
#       idx         cm E  cm bit       hmm E  hmmbit    surv     xhmm  speedup
#       ----  ----------  ------  ----------  ------  ------  -------  -------
           1       0.046    23.1   36629.398     3.4  0.4905     36.4      2.0
           2       0.040    23.3   29737.326     3.7  0.3982     29.7      2.4
           3       0.028    23.9   26541.287     3.8  0.3554     26.6      2.7
           4       0.023    24.3   23640.775     4.0  0.3165     23.8      3.0
           5       0.021    24.4   20096.482     4.2  0.2691     20.4      3.5
           6       0.017    24.7   17599.957     4.4  0.2357     18.0      4.0
           7       0.014    25.1   15361.491     4.6  0.2057     15.8      4.6
           8       0.011    25.4   13553.667     4.8  0.1815     14.1      5.1
           9    9.55e-03    25.7   12162.658     5.0  0.1629     12.7      5.7
          10    7.45e-03    26.1   10565.534     5.2  0.1415     11.2      6.4
          11    5.26e-03    26.6    8933.020     5.4  0.1196      9.6      7.5
          12    3.58e-03    27.3    7775.779     5.6  0.1041      8.5      8.5
          13    2.25e-03    28.0    6791.427     5.8  0.0909      7.6      9.5
```

```
14    1.48e-03    28.7    5931.661     6.0  0.0794      6.7     10.7
15    1.39e-03    28.8    5330.116     6.2  0.0714      6.1     11.7
16    1.21e-03    29.0    4725.174     6.4  0.0633      5.6     13.0
17    1.15e-03    29.1    4041.305     6.6  0.0541      4.9     14.7
18    8.80e-04    29.5    3398.422     6.9  0.0455      4.3     16.8
19    7.15e-04    29.9    2962.177     7.1  0.0397      3.9     18.7
20    5.19e-04    30.4    2608.278     7.3  0.0349      3.5     20.5
21    3.78e-04    30.9    2239.857     7.5  0.0300      3.2     22.8
22    3.73e-04    30.9    1985.648     7.7  0.0266      2.9     24.7
23    3.41e-04    31.1    1765.061     7.8  0.0236      2.7     26.7
24    3.06e-04    31.3    1523.977     8.1  0.0204      2.5     29.2
25    2.69e-04    31.5    1352.845     8.2  0.0181      2.3     31.3
26    2.48e-04    31.6    1202.553     8.4  0.0161      2.2     33.4
27    1.94e-04    32.0    1037.597     8.6  0.0139      2.0     36.0
28    1.30e-04    32.7     894.061     8.8  0.0120      1.9     38.7
29    8.97e-05    33.3     796.893     9.0  0.0107      1.8     40.8
30    6.56e-05    33.8     704.541     9.2  0.0094      1.7     42.9
31    4.90e-05    34.2     633.229     9.4  0.0085      1.6     44.8
32    1.33e-08    47.6      63.323    12.8  0.0008      1.1     68.0
```

The first non-# prefixed line includes information about our calibrated CM. There are seven columns, here's what each one means:

idx simply the index of the model this row pertains to in the CM file. In this case we have only 1 CM.

name name of the model.

F this is the fraction of hits **cmcalibrate** required the HMM to be able to recognize during filter threshold calculation. By default this number is 0.995 as I mentioned above, but can be changed with the **-F** option to **cmcalibrate** (see the manual page). This means that we expect the HMM filter to miss 0.5% of the hits that a non-filtered search would find.

nseq this is the number of sequences generated in **cmcalibrate** and searched with the CM and HMM during filter threshold calculation. In this case, 10,000 sequences were generated and searched, and the HMM must have been able to recognize 99.5% of the sequences above any given CM score threshold.

db (Mb) the database size in megabases (Mb) the statistics in the next section (see below) pertain to.

always? **cmcalibrate** determines if it is always possible to use an HMM filter for the model it is calibrating, no matter what the database size and final round CM E value cutoff is. If it is, this column reads "yes", if not it reads "no".

After the CM information is a section with a heading labelled "CM E-value cutoff / HMM Forward E-value filter cutoff pairs". This section explains how effective the HMM filter is predicted to be if particular CM E-value cutoffs are used for the search in our sequence file **C.psychrerythraea.genome.fa**. Here's a description of each column:

idx simply the row number.

cm E final round CM E-value cutoff this row of stats pertains to.

cm bit the bit score the E-value corresponds to in a database of this size (10.7 Mb in this case).

hmm bit the HMM filter bit score threshold that would be used for this search.

surv the *predicted* survival fraction of the database that would survive the HMM filter.

xhmm the *predicted* ratio of millions of dynamic programming (DP) calculations required for an HMM filtered search using the cutoff given in the "hmmbit" column to the millions of DP calcs required by a search with only the HMM. For row 1, this means our filtered search (that's the HMM filter plus the CM Inside scan of the predicted 49.05% surviving fraction of the database) should take roughly 36 times the amount of time an HMM only search would take.

speedup the *predicted* speedup of the HMM filtered search versus a non-filtered scan with just the CM Inside algorithm.

There are 32 rows of this data, each row shows the characteristics that would be used if a particaular CM E-value cutoff were used when searching **C.psychrerythraea.genome.fa**. When searching with a CM E-value cutoff that falls between that of two rows, the HMM filter from the bordering row with the lower cutoff is used. For example, we just ran a search with a 0.001 E-value cutoff and our HMM filter was predicted to have a survival fraction of 0.0541. This corresonds to row 17 of the data above.

Using **cmstat** in this way allows the user to see how effective an HMM filter is predicted to be over the choice of different E-value thresholds. If running time is a primary concern, the CM E-value for which the HMM filter that gives an appropriately high speedup can be chosen.

using cmstat to predict running times for filtered searches with Rfam cutoffs

Section 5 describes how the Rfam curators set GA, NC and TC bit score cutoffs for each Rfam model. Because these cutoffs are relatively strict it is often possible to achieve large speedups of up to 100-fold or more when using an Rfam cutoff as the final threshold. The **cmstat** can print these predicted speedups with the **--ga, --nc** and **--tc** options. Here's an example, using the **rfam10.c.cm** CM file from the tutorial directory that includes 10 randomly chosen Rfam 8.1 models, that have been calibrated with default parameters to cmcalibrate:

```
> cmstat --ga --lfi rfam10.c.cm
```

```
# local Inside filter threshold stats for Rfam GA gathering cutoff from CM file
#
#  idx  name              clen      cm E  cm bit  hmmbit    surv     xhmm  speedup
# ----  --------------  -----  --------  ------  ------  ------  -------  -------
     1  Retroviral_psi    118  9.47e-06    27.0     4.9  0.0320      2.9     20.3
     2  snoR43             73  2.32e-07    40.0    11.9  0.0010      1.1     60.5
     3  snoMe28S-U3344     82  7.04e-11    50.0    11.8  0.0010      1.1     63.5
     4  SNORD36            79  4.99e-04    28.0    12.0  0.0009      1.1     65.0
     5  snoMe18S-Um1356    86  1.02e-09    46.0    11.8  0.0010      1.1     62.0
     6  SNORND104          70  1.94e-07    35.0    10.4  0.0011      1.1     56.4
     7  HgcC              130  9.13e-05    24.0    10.1  0.0034      1.8    129.4
     8  SL1               103  1.69e-04    30.0     7.4  0.0162      2.0     30.5
     9  sroH              161  7.63e-13    50.0     7.8  0.0091      1.6     39.8
    10  snoR9             128  3.91e-11    50.0     9.1  0.0007      1.1     80.9
# ----  --------------  -----  --------  ------  ------  ------  -------  -------
     -  *Average*         103  7.69e-05    38.0     9.7  0.0067      1.5     60.8
     -  *Total*             -         -       -       -       -      1.5     57.0
```

The last two lines that report the average and total statistics are new, they appear with the **--lfi, --lfc, --gfi, --gfc** if the CM file you're running **cmstat** on includes more than 1 model. Notice that we're predict about a 57-fold speedup when searching with all 10 of these models, which means our CM search would be only about 1.5X slower than an HMM search.

Filtered searches with non-calibrated models

It is possible to run **cmsearch** with models that are not calibrated. Results from searches with non-calibrated models will not include E-values and will not be automatically accelerated with appropriate HMM filter thresholds. The tradeoff is that calibration is expensive, so skipping it can save you time.

When using a non-calibrated model **cmsearch** will by default use an HMM filter with a cutoff threshold of 3.0 bits, and a QDB filter with a cutoff threshold of 0.0 bits, although these thresholds can be changed using command-line options. Let's walk through a quick example of default search with a non-calibrated version of the tRNA model from the tutorial. We'll assume here that you've gone through tutorial section of the guide and the examples earlier in this section (if not, this part may not make sense). First, build a model from the **tRNA.5.sto** Stockholm file (from the **/tutorial/** subdirectory of INFERNAL). We'll use the **-F** option which allows us to overwrite the file **my.cm** if it exists.

```
> cmbuild -F my.cm trna.5.sto
```

Now, let's redo the search from the tutorial of the 300 Kb database, but this time our model is not calibrated. First, let's try to use **--forecast** to predict the run time:

```
> cmsearch --forecast 1 my.cm tosearch.300Kb.db
```

You'll get an error message saying that you can't use **--forecast** for a non-calibrated model. The reason is because **cmsearch** has no E-value statistics and thus no good way of estimating how many hits will survive the HMM filter with a threshold of 3.0 bits.

Let's run the actual search:

```
> cmsearch my.cm tosearch.300Kb.db
```

First, the "Pre-search" info is printed to the screen, which has less information than it would if the model were calibrated (see earlier examples in this section):

```
# Pre-search info for CM 1: trna.5-1
#
# rnd  mod  alg  cfg   beta  bit sc cut
# ---  ---  ---  ---  -----  ----------
    1  hmm  fwd  loc      -        3.00
    2   cm  cyk  loc  1e-07        0.00
    3   cm  ins  loc  1e-15        0.00
```

Then about 270 hits are printed to the screen. There's so many because our cutoff was set as final cutoff was set as 0.0 bits. Notice that the top hit is still the real tRNA from 101 to 173, and it has the same bit score as in the search with the calibrated model, which it must, but there's no E-value reported. In this case, it's pretty clear that the sequence is a good hit to the model, both because of the high bit score and because of the primary sequence and structural similarity apparent in the **cmsearch** alignment.

After all the hits, the "Post-search" information is printed showing you how many hits survived each round:

```
# Post-search info for CM 1: trna.5-1
#
# rnd  mod  alg  cfg   beta  bit sc cut  num hits  surv fract
# ---  ---  ---  ---  -----  ----------  --------  ----------
    1  hmm  fwd  loc      -        3.00       914      0.1638
    2   cm  cyk  loc  1e-07        0.00       212      0.0496
    3   cm  ins  loc  1e-15        0.00       269      0.0159
#
#    run time
# -----------
     00:00:42
```

▷ ***My search results say that more hits were found in the final round than in a previous filter round, how is this possible?*** *The reason this can happen is because the filter rounds detect hits and then extend the boundaries of the hit to include a short stretch of neighboring residues because we don't want to rely on the filter's definition of the hit end points. Some hits in the filter rounds can overlap following the extension, in which case the hits are merged into one single hit. In the final round it's possible to find more than one hit in the regions that were overlapping and merged, so you can get more than one final hit in a region counted as just one hit in a filter round. This is what happens in the above example. Notice that even though there's more hits in round 3 than round 2, the survival fraction is less in round 3 than round 2. Again, this is because hit boundaries are extended in the filter round to allow subsequent rounds to refine the endpoints, whereas hit boundaries are not extended in the final round.*

You may be thinking that some of these thresholds are inappropriate. For example, you may not want to look at all 269 hits that have a bit score above 0.0 bits, because most of them are clearly not tRNAs. You're right, and these default thresholds for non-calibrated models are usually inappropriate. Without E-values it's difficult to automatically predict appropriate thresholds so **cmsearch** doesn't even try. **cmsearch** does have several command-line options that allow the user to manipulate the thresholds to their liking. This is described next.

Manually setting filter and final thresholds

We described above how **cmsearch** will automatically pick filter and final thresholds differently depending on if the query model is calibrated or not. For calibrated models the HMM filter threshold that will theoretically maximize speed while maintaining 99.5% sensitivity is chosen (sometimes this means turning HMM filtering off if the predicted speedup is insignificant). The QDB threshold is set in a more *ad hoc* way, that works well empirically. For non-calibrated models, the thresholds are automatically set in a very simplistic way with a 3.0 bit threshold for the HMM filter and 0.0 bit threshold for the QDB filter and final round.

Regardless of whether you're searching with a calibrated or non-calibrated model, **cmsearch** offers you several options manually overriding these automatic choices of thresholds. Here's a list:

Final threshold related options:

option	sets final round threshold to	requirements
-E <x>	E-value of **<x>**	model must be calibrated with **cmcalibrate**
-T <x>	bit score of **<x>**	none
--ga	Rfam GA bit score	#=GF GA annotation in model training alignment to **cmbuild**
--tc	Rfam TC bit score	#=GF TC annotation in model training alignment to **cmbuild**
--nc	Rfam NC bit score	#=GF NC annotation in model training alignment to **cmbuild**

HMM filter threshold related options:

option	effect	requirements
--fil-no-hmm <x>	turn HMM filter off	none
--fil-E-hmm <x>	sets threshold as E-value **<x>**	model must be calibrated
--fil-T-hmm <x>	sets threshold as bit score **<x>**	none
--fil-Smax-hmm <x>	sets maximum allowable E-value threshold as that which gives predicted survival fraction of **<x>**	model must be calibrated

QDB filter threshold related options:

option	effect	requirements
--fil-no-qdb <x>	turn QDB filter off	none
--fil-E-qdb <x>	sets threshold as E-value **<x>**	model must be calibrated
--fil-T-qdb <x>	sets threshold as bit score **<x>**	none

You can use either 0 or 1 of the options from each table in a search. The Rfam **--ga, --tc** and **--nc** options are explained more in section 5.

We'll go through two examples of using these options with our tRNA model. First, with a calibrated or non-calibrated model to set the HMM filter cutoff to 5.0 bits, the QDB filter cutoff to 10.0 bits and the final cutoff to 15.0 bits:

```
> cmsearch --fil-T-hmm 5 --fil-T-qdb 10 -T 15 my.cm tosearch.300Kb.db
```

If your model is calibrated, you can force the HMM filter to be set to achieve a specified (predicted) survival fraction with **--fil-Smax-hmm <x>**. With this option for values of **<x> << 1** you can expect roughly a **1 / <x>** speedup relative to a non-filtered search. To try it:

```
> cmsearch --fil-Smax-hmm 0.001 my.c.cm tosearch.300Kb.db
```

```
# Pre-search info for CM 1: trna.5-1
#
#                                  cutoffs            predictions
#                           -------------------  --------------------
# rnd  mod  alg  cfg   beta     E value   bit sc     surv     run time
# ---  ---  ---  ---  -----  ----------  -------  -------  -----------
    1  hmm  fwd  loc      -       5.472    11.46   0.0010  00:00:05.62
    2   cm  ins  loc  1e-15       1.000    11.92   0.0001  00:00:00.55
# ---  ---  ---  ---  -----  ----------  -------  -------  -----------
  all    -    -    -      -           -        -        -  00:00:06.18
```

Notice that the HMM filter round 1 cutoff was set as the E-value that corresponds to predicted survival fraction of 0.001.

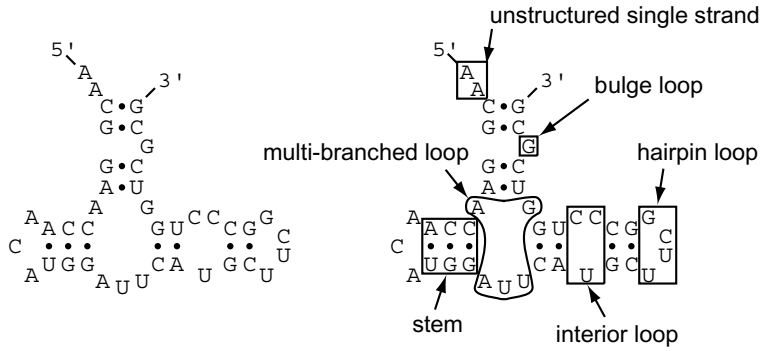
7 File and output formats

RNA secondary structures: WUSS notation

INFERNAL annotates RNA secondary structures using a linear string representation called "WUSS notation" (Washington University Secondary Structure notation).

The symbology is extended from the common bracket notation for RNA secondary structures, where open- and close-bracket symbols (or parentheses) are used to annotate base pairing partners: for example, `((((...))))` indicates a four-base stem with a three-base loop. Bracket notation is difficult for humans to interpret, for anything much larger than a simple stem-loop. WUSS notation makes it somewhat easier to interpret the annotation for larger structures.

The following figure shows an example with the key elements of WUSS notation. At the top left is an example RNA structure. At the top right is the same structure, with different RNA structural elements marked. Below both structure pictures : the WUSS notation string for the structure.




```
::((((,<<<___>>>,,,<<-<<____>>-->>,))-))
AACGGAACCAACAUGGAUUCAUGCUUCGGCCCUGGUCGCG
```

Full (output) WUSS notation

In detail, symbols used by WUSS notation in *output* structure annotation strings are as follows:

Base pairs Base pairs are annotated by nested matching pairs of symbols `<>`, `()`, `[]`, or `{}`. The different symbols indicate the "depth" of the helix in the RNA structure as follows: `<>` are used for simple terminal stems; `()` are used for "internal" helices enclosing a multifurcation of all terminal stems; `[]` are used for internal helices enclosing a multifurcation that includes at least one annotated `()` stem already; and `{}` are used for all internal helices enclosing deeper multifurcations.

Hairpin loops Hairpin loop residues are indicated by underscores, `_`. Simple stem loops stand out as, e.g. `<<<<_____>>>>`.

Bulge, interior loops Bulge and interior loop residues are indicated by dashes, `-`.

Multifurcation loops Multifurcation loop residues are indicated by commas, `,`. The mnemonic is "stem 1, stem2", e.g. `<<<___>>>,,<<<___>>>`.

External residues Unstructured single stranded residues completely outside the structure (unenclosed by any base pairs) are annotated by colons, `:`.

Insertions Insertions relative to a known structure are indicated by periods, `.`. Regions where local structural alignment was invoked, leaving regions of both target and query sequence unaligned, are indicated by tildes, `~`. These symbols only appear in alignments of a known (query) structure annotation to a target sequence of unknown structure.

Pseudoknots WUSS notation allows pseudoknots to be annotated as pairs of upper case/lower case letters: for example, `<<<<_AAAA____>>>>aaaa` annotates a simple pseudoknot; additional pseudoknotted stems could be annotated by `Bb`, `Cc`, etc. INFERNAL cannot handle pseudoknots, however; pseudoknot notation never appears in INFERNAL output; it is accepted in input files, but ignored.

An example of WUSS notation for a complicated structure (*E. coli* RNase P) is shown in Figure 5. An example of WUSS notation for a local INFERNAL alignment of *B. subtilis* RNase P to *E. coli* RNase P, illustrating the use of local alignment annotation symbols, is in Figure 6.

Shorthand (input) WUSS notation

While WUSS notation makes it easier to visually interpret INFERNAL *output* structural annotation, it would be painful to be required to *input* all structures in full WUSS notation. Therefore when INFERNAL reads input secondary structure annotation, it uses simpler rules:

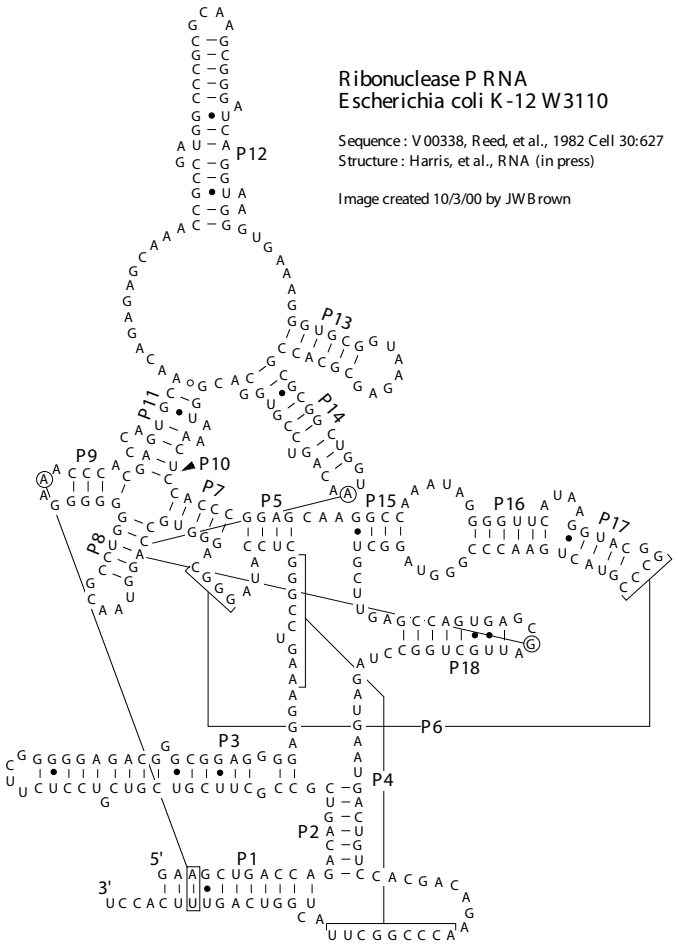
Base pairs Any matching nested pair of `()`, `()`, `[]`, `{}` symbols indicates a base pair; the exact choice of symbol has no meaning, so long as the left and right partners match up.

Single stranded residues All other symbols `_-,:.~` indicate single stranded residues. The choice of symbol has no special meaning. Annotated pseudoknots (nested matched pairs of upper/lower case alphabetic characters) are also interpreted as single stranded residue in INFERNAL input.

Thus, for instance, `<<<<....>>>>` and `((((____))))` and `<(<(._._)>)>` all indicate a four base stem with a four base loop (the last example is legal but weird).

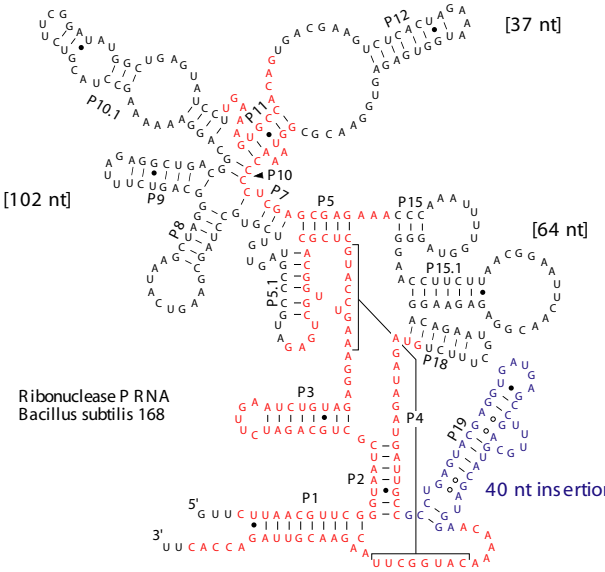
Remember that the key property of canonical (nonpseudoknotted) RNA secondary structure is that the pairs are *nested*. `((<<....))>>` is not a legal annotation string: the pair symbols don't match up properly. INFERNAL will reject such an annotation and report an input format error, suspecting a problem with your annotation. If you want to annotate pseudoknots, WUSS notation allows alphabetic symbols Aa, Bb, etc. see above; but remember that INFERNAL ignores pseudoknotted stems and treats them as single stranded residues.

Because many other RNA secondary structure analysis programs use a simple bracket notation for annotating structure, INFERNAL's ability to input this format makes it easier to use data generated by other RNA software packages. Conversely, converting INFERNAL output WUSS notation to simple bracket notation is a matter of a simple Perl or sed script, substituting the symbols appropriately.




```
   {{{{{{{{{{{{{{{{{{,<<<<<<<<<<<<<-<<<<<_____>>>>>>>>>->>>>>>>>
 1 GAAGCUGACCAGACAGUCGCCGCUUCGUCGUCGUCCUCUUCGGGGGAGACGGGCGGAGGG 60

   >,,,,,,,,,,,,,[[[[--------[[[[[<<<<<_____>>>>>>>>>____>>>->(
61 GAGGAAAGUCCGGGCUCCAUAGGGCAGGGUGCCAGGUAACGCCUGGGGGGGGAAACCCACG 120

   (---(((((,,,,,,,,,,,,<<<<<--<<<<<<<<_____>>>>>->>>>>>-->>,,,,
121 ACCAGUGCAACAGAGAGCAAACCGCCGAUGGCCCGCGCAAGCGGGAUCAGGUAAGGGUGA 180

   ,,,<<<<<<_______>>>>>>>>>>>>>>>>____>>>->>>>>->,,)))--))))]]]
181 AAGGGUGCGGUAAGAGCGCACCGCGCGGCUGGUAACAGUCCGUGGCACGGUAAACUCCAC 240

   ]]]]]],,,<<<<------<<<<<<----<<<<<_____>>>>>>>>>>>------>>>>,
241 CCGGAGCAAGGCCAAAUAGGGGUUCAUAAGGUACGGCCCGUACUGAACCCGGGUAGGCUG 300

   ,,,,,<<<<<<<<____>>>>>>>>,,,,,,,,,,}}}}}}}----------------
301 CUUGAGCCAGUGAGCGAUUGCUGGCCUAGAUGAAUGACUGUCCACGACAGAACCCGGCUU 360

   -}-}}}}}}}}}}::::
361 AUCGGUCAGUUUCACCU 377
```

Figure 5: **Example of WUSS notation.** Top: Secondary structure of *E. coli* RNase P, from Jim Brown's RNase P database (Brown, 1999). Bottom: WUSS notation for the same structure, annotating the *E. coli* RNase P sequence. The P4 and P6 pseudoknots are not annotated in this example.




```
hit 0   :      4    399    52.56 bits
          {{{{{{{{{{{{{{{{{{,<<<<<<<<<<<<<-<<<<<____>>>>>>>>>->>>>>>>>
           1 ggAGuggGgcaGgCaguCGCugcuucggccuuGuucaguuaacugaaaaggAccgaagga 60
             +: :::G::C:GG:A:UCGCU+C::::            U+            ::::G+A
           4 CUUAACGUUCGGGUAAUCGCUGCAGAUC----------UUG----------AAUCUGUA 42

             >,,,,,,,,,,,,,[[[.[--------[[[[[~~~~~~~((---(((((,,,,~~~~~~)
          61 GAGGAAAGUCCGGGCUC.CACAGGGCAgGGUG*[ 29]*GGAAAGUGCCACAG*[96]*G 229
             GAGGAAAGUCC  GCUC C  A GG   :G G       :GAAAGUGCCACAG      G
          43 GAGGAAAGUCCAUGCUCgC--ACGGUGCUGAG*[102]*UGAAAGUGCCACAG*[37]*G 226

             ))--))))]]]]]].]]],,,~~~~~~,,,,,,,,,,}}}}}}}--..............
         230 GUAAACCCCACCcG.GAGCAA*[77]*CuAGAUGAAUGacuGcCCA.............. 344
             GUAAACC:C C: G GAG AA       UAGAU++AUGA:U:CC
         227 GUAAACCCCUCGAGcGAGAAA*[64]*GUAGAUAGAUGAUUGCC--gccugaguacgagg 342

             ..........................----------------}-}}}}}}}}}}:::::
         345 ........................CGACAGAACCCGGCUUAuagcCccaCUccucuu 377
                                     ACA AAC  GGCUUA:AG::C::: :+ C
         343 ugaugagccguuugcaguacgaugga--ACAAAACAUGGCUUACAGAACGUUAGACCAC 399
```

Figure 6: **Local alignment annotation example.** Top: Secondary structure of *B. subtilis* RNase P, from Jim Brown's RNase P database (Brown, 1999). Residues in red are those that INFERNAL aligns to a CM of *E. coli* type RNase P's. The local structural alignment is in four pieces; three regions of the structure (102, 37, and 64 nt long) are skipped over. One additional stem is treated as a 40 nt insertion. Bottom: the INFERNAL output, showing the *E. coli* query structure aligned to the *B. subtilis* sequence.

Multiple alignments: Stockholm format

The Pfam consortium developed an annotated alignment format called "Stockholm format", and this format has been adopted as the standard alignment format in HMMER and INFERNAL, and by the Rfam consortium. The reasons for inventing a new alignment format were two-fold. First, there really is no standard accepted format for multiple sequence alignment files, so we don't feel guilty about inventing a new one. Second, the formats of popular multiple alignment software (e.g. CLUSTAL, GCG MSF, PHYLIP) do not support rich documentation and markup of the alignment. Stockholm format was developed to support extensible markup of multiple sequence alignments, and we use this capability extensively in both RNA work (with structural markup) and the Pfam database (with extensive use of both annotation and markup).

A minimal Stockholm file

```
# STOCKHOLM 1.0

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

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

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

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

In the alignment, each line contains a name, followed by the aligned sequence. A dash or period denotes a gap. If the alignment is too long to fit on one line, the alignment may be split into multiple blocks, with blocks separated by blank lines. The number of sequences, their order, and their names must be the same in every block. Within a given block, each (sub)sequence (and any associated `#=GR` and `#=GC` markup, see below) is of equal length, called the *block length*. Block lengths may differ from block to block; the block length must be at least one residue, and there is no maximum.

The sequence names must be unique. (They are used to associate markup tags with the sequences.)

Other blank lines are ignored. You can add comments to the file on lines starting with a #.

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

Syntax of Stockholm markup

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

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

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

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

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

Semantics of Stockholm markup

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

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

HMMER and INFERNAL use Stockholm markup to pick up a variety of information from the multiple alignment files. The Pfam and Rfam consortiums therefore agree on additional syntax for certain tag types, so software can parse some markups for useful (or necessary) information. This additional syntax is imposed by Pfam, HMMER, INFERNAL, 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. "RNaseP. Mandatory, if the file is an alignment database used as input for **cmbuild**, because each CM must get a unique name. One word. Unique in file.

AC <s> Accession. **<s>** is a unique accession number for the alignment; e.g. "PF00001". Used by the Rfam database, for instance. Often a alphabetical prefix indicating the database (e.g. "RF") 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. "Ribonuclease P RNA". 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.

Recognized #=GS annotations

WT <f> Sequence 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. **cmbuild** uses this annotation to determine which columns are consensus versus insertion; insertion columns are annotated by a gap symbol, and consensus columns by any non-gap symbol.

SS_cons Secondary structure consensus. When this line is generated by INFERNAL, it is generated in full WUSS notation. When it is read by **cmbuild**, it is interpreted more loosely, in shorthand (input) WUSS notation: pairs of symbols <>, `()`, `[]`, or `[]` mark consensus base pairs, and symbols `:_-,.~` mark single stranded columns.

Recognized #=GR annotations

SS Secondary structure for this sequence. See **#=GC SS_cons** above.

Sequence files: FASTA format

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

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

For better or worse, FASTA is not a documented standard. Minor (and major) variants are in widespread use in the bioinformatics community, all of which are called "FASTA format". My software attempts to cater to all of them, and is tolerant of common deviations in FASTA format. Certainly anything that is accepted by the database formatting programs in NCBI BLAST or WU-BLAST (e.g. setdb, pressdb, xdformat) will also be accepted by my software. Blank lines in a FASTA file are ignored, and so are spaces or other gap symbols (dashes, underscores, periods) in a sequence. Other non-amino or non-nucleic acid symbols in the sequence are also silently ignored, mostly because some people seem to think that "*" or "." should be added to protein sequences to (redundantly) indicate the end of the sequence. The parser will also accept unlimited line lengths, which allows it to accomodate the enormous description lines in the NCBI NR databases.

(On the other hand, any FASTA files *generated* by my software adhere closely to community standards, and should be usable by other software packages (BLAST, FASTA, etc.) that are more picky about parsing their input files. That means you can run a sloppy FASTA file thru the **`sreformat`** utility program to clean it up.)

Partly because of this tolerance, the software may have a difficult time dealing with files that are *not* in FASTA format, especially if you're relying on file format autodetection (the "Babelfish"). Some (now mercifully uncommon) file formats are so similar to FASTA format that they be erroneously called FASTA by the Babelfish and then quietly and lethally misparsed. An example is the old NBRF file format. If you're afraid of this, you can use the **`--informat fasta`** option to bypass the Babelfish and improve robustness. However, it is still possible to construct files perversely similar to FASTA that will still confuse the parser. (The gist of these caveats applies to all formats, not just FASTA.)

CM file format

The default CM file format is a simple, extensible tag-value format. The format being used right now is tentative and likely to change. Therefore, it is not currently documented here. If you absolutely need to interpret it, see the file `cm_io.c` in the source code.

Null model file format

The Infernal source distribution includes an example prior file, **`rna.null`**. This null model is identical to the hardcoded default prior used by Infernal, all four RNA nucleotides are equiprobable in the null, background model.

A null model file must contain exactly four non-comment lines. A comment line begins with a "# ", that is a # followed by a single space. Each of the four non-comment lines must contain a single floating point number, the four of which sum to 1.0. The first non-comment line is interpreted as the background probability of an "A" residue, the second, third, and fourth non-comment lines are interpreted as the background probabilities of a "C", "G" and "U" respectively.

Dirichlet prior files

A prior file is parsed into a number of whitespace-delimited, non-comment fields. These fields are then interpreted in order. The order and number of the fields is important. This is not a robust, tag-value save file format.

All whitespace is ignored, including newlines. The number of fields per line is unimportant.

Comments begin with a # character. The remainder of any line following a # is ignored.

The Infernal source distribution includes an example prior file, **`default.pri`**. This prior is identical to the hardcoded default prior used by Infernal. The following text may only make sense if you're looking at that example while you read.

The order of the fields in the prior file is as follows:

Strategy. The first field is the keyword **`Dirichlet`**. Currently Dirichlet priors (mixture or not) are the only prior strategy used by Infernal.

Transition prior section. The next field is the number **`74`**, the number of different types of transition distributions. (See Figure 7 for an explanation of where the number 74 comes from.) Then, for each of these 74 distributions:

`<from-uniqstate> <to-node>`: Two fields give the transition type: from a unique state identifier, to a node identifier. Example: **`MATP_MP MATP`**.

`<n>`: One field gives the number of transition probabilities for this transition type; that is, the number of Dirichlet parameter vector $\alpha_1^q..\alpha_n^q$ for each mixture component q.

`<nq>`: One field gives the number of mixture Dirichlet components for this transition type's prior. Then, for each of these **`nq`** Dirichlet components:

`p(q)`: One field gives the mixture coefficient $p(q)$, the prior probability of this component q. For a single-component "mixture", this is always 1.0.

$\alpha_{\mathbf{1}}^{\mathbf{q}}..\alpha_{\mathbf{n}}^{\mathbf{q}}$: The next n fields give the Dirichlet parameter vector for this mixture component q.

Base pair emission prior section. This next section is the prior for MATP_MP emissions. One field gives **`<K>`**, the "alphabet size" – the number of base pair emission probabilities – which is always 16 (4x4), for RNA. The next field gives **`<nq>`**, the number of mixture components. Then, for each of these **`nq`** Dirichlet components:

`p(q)`: One field gives the mixture coefficient $p(q)$, the prior probability of this component q. For a single-component "mixture", this is always 1.0.

$\alpha_{\mathbf{AA}}^{\mathbf{q}}..\alpha_{\mathbf{UU}}^{\mathbf{q}}$: The next 16 fields give the Dirichlet parameter vector for this mixture component, in alphabetical order (AA, AC, AG, AU, CA ... GU, UA, UC, UG, UU).

Consensus singlet base emission prior section. This next section is the prior for MATL_ML and MATR_MR emissions. One field gives **`<K>`**, the "alphabet size" – the number of singlet emission probabilities – which is always 4, for RNA. The next field gives **`<nq>`**, the number of mixture components. Then, for each of these **`nq`** Dirichlet components:

`p(q)`: One field gives the mixture coefficient $p(q)$, the prior probability of this component q. For a single-component "mixture", this is always 1.0.

$\alpha_{\mathbf{A}}^{\mathbf{q}}..\alpha_{\mathbf{U}}^{\mathbf{q}}$: The next 4 fields give the Dirichlet parameter vector for this mixture component, in alphabetical order (A, C, G, U).

Nonconsensus singlet base emission prior section. This next section is the prior for insertions (MATP_IL, MATP_IR, MATL_IL, MATR_IR, ROOT_IL, ROOT_IR, BEGR_IL) as well as nonconsensus singlets (MATP_ML, MATP_MR). One field gives **`<K>`**, the "alphabet size" – the number of singlet emission probabilities – which is always 4, for RNA. The next field gives **`<nq>`**, the number of mixture components. Then, for each of these **`nq`** Dirichlet components:

p(q): One field gives the mixture coefficient $p(q)$, the prior probability of this component q. For a single-component "mixture", this is always 1.0.

$\alpha_{\mathbf{A}}^{\mathbf{q}}..\alpha_{\mathbf{U}}^{\mathbf{q}}$: The next 4 fields give the Dirichlet parameter vector for this mixture component, in alphabetical order (A, C, G, U).

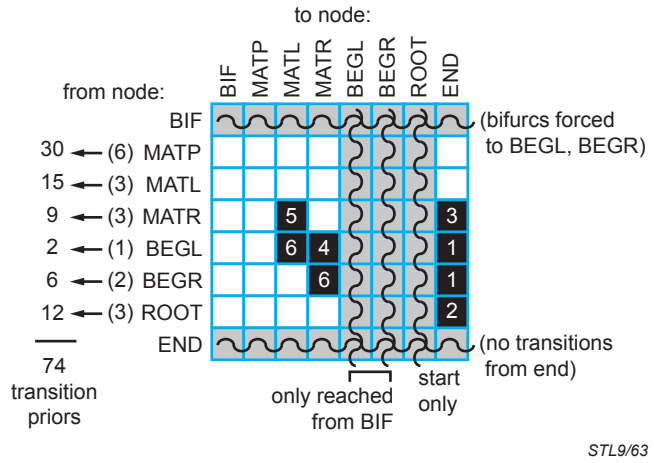


Figure 7: **Where does the magic number of 74 transition distribution types come from?** The transition distributions are indexed in a 2D array, from a unique statetype (20 possible) to a downstream node (8 possible), so the total conceivable number of different distributions is $20 \times 8 = 160$. The grid represents these possibilities by showing the 8×8 array of all node types to all node types; each starting node contains 1 or more unique states (number in parentheses to the left). Two rows are impossible (gray): bifurcations automatically transit to determined BEGL, BEGR states with probability 1, and end nodes have no transitions. Three columns are impossible (gray): BEGL and BEGR can only be reached by probability 1 transitions from a bifurcation, and the ROOT node is special and can only start a model. Eight individual cells of the grid are unused (black) because of the way **`cmbuild`** (almost) unambiguously constructs a guide tree from a consensus structure. These cases are numbered as follows. (1) BEGL and BEGR never transit to END; this would imply an empty substructure. A bifurcation is only used if both sides of the split contain at least one consensus pair (MATP). (2) ROOT never transits to END; this would imply an alignment with zero consensus columns. Infernal models assume ≥ 1 consensus columns. (3) MATR never transits to END. Infernal always uses MATL for unpaired columns whenever possible. MATR is only used for internal loops, multifurcation loops, and 3' bulges, so MATR must always be followed by a BIF, MATP, or another MATR. (4) BEGL never transits to MATR. The single stranded region between two bifurcated stems is unambiguously assigned to MATL nodes on the right side of the split, not to MATR nodes on the left. (5) MATR never transits to MATL. The only place where this could arise (given that we already specified that MATL is used whenever possible) is in an interior loop; there, by unambiguous convention, MATL nodes precede MATR nodes. (6) BEGL nodes never transit to MATL, and BEGR nodes never transit to MATR. By convention, at any bifurcated subsequence i, j, i and j are paired but not to each other. That is, the smallest possible subsequence is bifurcated, so that any single stranded stretches to the left and right are assigned to MATL and MATR nodes above the bifurcation, instead of MATL nodes below the BEGL and MATR nodes below the BEGR. Thus, the total number 74 comes from multiplying, for each row, the number of unique states in each starting node by the number of possible downstream nodes (white), and summing these up, as shown to the left of the grid.

8 Manual pages

`cmalign` - use a CM to make a structured RNA multiple alignment

Synopsis

Align sequences to a CM: **cmalign** *[options] cmfile seqfile*

Merge two alignments: **cmalign --merge** *[options] cmfile msafile1 msafile2*

Description

cmalign aligns the RNA sequences in *seqfile* to the covariance model (CM) in *cmfile*, and outputs a multiple sequence alignment. Alternatively, with the **--merge** option, **cmalign** merges the two alignments *msafile1* and *msafile2* created by previous runs of **cmalign** with *cmfile* into a single alignment.

The sequence file *seqfile* must be in FASTA, EMBL, or Genbank format.

CM files are profiles of RNA consensus secondary structure. A CM file is produced by the **cmbuild** program, from a given RNA sequence alignment of known consensus structure.

The alignment that **cmalign** makes is written in Stockholm format. It can be redirected to a file using the **-o** option.

cmalign uses an HMM banding technique to accelerate alignment by default as described below for the **--hbanded** option. HMM banding can be turned off with the **--nonbanded** option.

By default, **cmalign** computes the alignment with maximum expected accuracy that is consistent with constraints (bands) derived from an HMM, using a banded version of the Durbin/Holmes optimal accuracy algorithm. This behavior can be changed, as described below and in the User's Guide, with the **--cyk, --sample,** or **--viterbi** options.

It is possible to include the fixed training alignment used to build the CM within the output alignment of **cmalign.** This is done using the **--withali** option, as described below and in the User's Guide.

Output

cmalign first outputs tabular information on the scores of each sequence being aligned, then the alignment itself is printed. The alignment can be redirected to an output file <*f*> with the **-o** <*f*> option. The tabular output section includes one line per sequences and seven fields per line: "seq idx": the index of the sequence in the input file, "seq name": the sequence name, "len": the length of the sequence, "total": the total bit score of the sequence, "struct": an approximation of the contribution of the secondary structure to the bit score, "avg prob": the average posterior probability (confidence estimate) of each aligned residue, and "elapsed": the wall time spent aligning the sequence. The fields can change if different options are selected. For example if the **--cyk** option is enabled, the "avg prob" field disappears because posterior probabilities are not calculated by the CYK algorithm.

Options

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

-o <*f*> Save the alignment in Stockholm format to a file <*f*>. The default is to write it to standard output.

-l Turn on the local alignment algorithm, which allows the alignment to span two or more subsequences if necessary (e.g. if the structures of the query model and target sequence are only partially shared), allowing certain large insertions and deletions in the structure to be penalized differently than normal indels. The default is to globally align the query model to the target sequences.

-p Annotate the alignment with posterior probabilities calculated using the Inside and Outside algorithms. The **-p** option causes additional annotation to appear in the output alignment, but does not modify the alignment itself (that is, the relative positions of the residues are unchanged). Two characters for each residue are used to annotate the posterior probability that the corresponding residue aligns at the corresponding position in the Stockholm alignment. These characters have the Stockholm markup tags "#=GR <seq name> POSTX." and "#=GR <seq name> POST.X", and can only have the values: "0-9", "*" or ".". They indicate the tens and ones place for the posterior probability: an "8" for "POSTX." and a "3" for "POST.X" indicates that the posterior probability is between 0.83 and 0.84. A "*" for both "POSTX." and "POST.X" indicates that the confidence estimate is "very nearly" 1.0 (it's hard to be exact here due to numerical precision issues) A "." in both "POSTX." and "POST.X" indicates that that column aligns to a gap. When used in combination with **--nonbanded,** the calculation of the posterior probabilities considers all possible alignments of the target sequence to the CM. Without **--nonbanded** (in HMM banded mode), the calculation considers only possible alignments within the HMM bands.

-q Quiet; suppress the verbose banner, and only print the resulting alignment to stdout. This allows piping the alignment to the input of other programs, for example.

--informat <*s*> Assert that the input *seqfile* is in format <*s*>. Do not run Babelfish format autodection. This increases the reliability of the program somewhat, because the Babelfish can make mistakes; particularly recommended for unattended, high-throughput runs of Infernal. Acceptable formats are: FASTA, EMBL, UNIPROT, GENBANK, and DDBJ. <*s*> is case-insensitive.

--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.

--mpi Run as an MPI parallel program. This option will only be available if Infernal has been configured and built with the "--enable-mpi" flag (see User's Guide for details).

Expert Options

--optacc Align sequences using the Durbin/Holmes optimal accuracy algorithm. This is default behavior, so this option is probably useless. The optimal accuracy alignment will be constrained by HMM bands for acceleration unless the **--nonbanded** option is enabled. The optimal accuracy algorithm determines the alignment that maximizes the posterior probabilities of the aligned residues within it. The posterior probabilites are determined using (possibly HMM banded) variants of the Inside and Outside algorithms.

--cyk Do not use the Durbin/Holmes optimal accuracy alignment to align the sequences, instead use the CYK algorithm which determines the optimally scoring alignment of the sequence to the model.

--sample Sample an alignment from the posterior distribution of alignments. The posterior distribution is determined using an HMM banded (unless **--nonbanded)** variant of the Inside algorithm.

-s <*n*> Set the random number generator seed to <*n*>, where <*n*> is a positive integer. This option can only be used in combination with **--sample.** The default is to use time() to generate a different seed for each run, which means that two different runs of **cmalign --sample** on the same alignment will give slightly different results. You can use this option to generate reproducible results.

--viterbi Do not use the CM to align the sequences, instead use the HMM Viterbi algorithm to align with a CM Plan 9 HMM. The HMM is automatically constructed to be maximally similar to the CM. This HMM alignment is faster than CM alignment, but can be less accurate because the structure of the RNA family is ignored.

--sub Turn on the sub model construction and alignment procedure. For each sequence, an HMM is first used to predict the model start and end consensus columns, and a new sub CM is constructed that only models consensus columns from start to end. The sequence is then aligned to this sub CM. This option is useful for aligning sequences that are known to truncated, non-full length sequences. This "sub CM" procedure is not the same as the "sub CMs" described by Weinberg and Ruzzo.

--small Use the divide and conquer CYK alignment algorithm described in SR Eddy, BMC Bioinformatics 3:18, 2002. The **--nonbanded** option must be used in combination with this options. Also, it is recommended whenever **--nonbanded** is used that **--small** is also used because standard CM alignment without HMM banding requires a lot of memory, especially for large RNAs. **--small** allows CM alignment within practical memory limits, reducing the memory required for alignment LSU rRNA, the largest known RNAs, from 150 Gb to less than 300 Mb. This option can only be used in combination with **--nonbanded** and **--cyk.**

--hbanded This option is turned on by default. Accelerate alignment by pruning away regions of the CM DP matrix that are deemed negligible by an HMM. First, each sequence is scored with a CM plan 9 HMM derived from the CM using the Forward and Backward HMM algorithms and calculate posterior probabilities that each residue aligns to each state of the HMM. These posterior probabilities are used to derive constraints (bands) on the CM DP matrix. Finally, the target sequence is aligned to the CM using the banded DP matrix, during which cells outside the bands are ignored. Usually most of the full DP matrix lies outside the bands (often more than 95%), making this technique faster because fewer DP calculations are required, and more memory efficient because only cells within the bands need be allocated. Importantly, HMM banding sacrifices the guarantee of determining the optimally accurarte or optimal alignment, which will be missed if it lies outside the bands. The tau paramater (analagous to the beta parameter for QDB calculation in **cmsearch**) is the amount of probability mass considered negligible during HMM band calculation; lower values of tau yield greater speedups but also a greater chance of missing the optimal alignment. The default tau is 1E-7, determined empirically as a good tradeoff between sensitivity and speed, though this value can be changed with the **--tau ”** <**x**>**”** option. The level of acceleration increases with both the length and primary sequence conservation level of the family. For example, with the default tau of 1E-7, tRNA models (low primary sequence conservation with length of about 75 residues) show about 10X acceleration, and SSU bacterial rRNA models (high primary sequence conservation with length of about 1500 residues) show about 700X. HMM banding can be turned off with the **--nonbanded** option.

--nonbanded Turns off HMM banding. The returned alignment is guaranteed to be the globally optimally accurate one (by default) or the globally optimally scoring one (if **--cyk** is enabled). The **--small** option is recommended in combination with this option, because standard alignment without HMM banding requires a lot of memory (see **--small**).

--tau <*x*> Set the tail loss probability used during HMM band calculation to <*x*>. This is the amount of probability mass within the HMM posterior probabilities that is considered negligible. The default value is 1E-7. In general, higher values will result in greater acceleration, but increase the chance of missing the optimal alignment due to the HMM bands.

--mxsize <*x*> Set the maximum allowable DP matrix size to <*x*> megabytes. By default this size is 2,048 Mb. This should be large enough for the vast majority of alignments, however if it is not **cmalign** will exit prematurely and report an error message that the matrix exceeded it’s maximum allowable size. In this case, the **--mxsize** can be used to raise the limit. This is most likely to occur when the **--nonbanded** option is used without the **--small** option, but can still occur when **--nonbanded** is not used.

--rna Output the alignments as RNA sequence alignments. This is true by default.

--dna Output the alignments as DNA sequence alignments.

--matchonly Only include match columns in the output alignment, do not include any insertions relative to the consensus model.

--resonly Only include match columns in the output alignment that have at least 1 residue (non-gap character) in them. By default all match columns are printed to the alignment, even those that are 100% gaps. **--resonly** replicates the default behavior of previous versions of **cmalign.**

--fins Change the behavior of how insert emissions are placed in the alignment. By default, all contiguous blocks of inserts are split in half, and half the residues are flushed left against the nearest consensus column to the left, and half are flushed right against the nearest consensus column on the right. With **--fins** inserts are not split in half, instead all inserted residues from IL states are flushed left, and all inserted residues from IR states are flushed right. **--fins** replicates the default behavior of previous versions of **cmalign.**

--onepost Modifies behavior of the **-p** option. Use only one character instead of two to annotate the posterior probability of each aligned residue. Specifically, only the "#=GR <seq name> POSTX." tag is printed to the alignment. An "8" for "POSTX." indicates a posterior probability between 0.8 and 0.9 for the corresponding residue.

--merge With **--merge** the usage of cmalign changes to **cmalign --merge** *[options] cmfile msafile1 msafile2*. Merge the two alignments in *msafile1* and *msafile2* created by previous runs of **cmalign** with *cmfile* together into a single alignment and exit. *msafile1* and *msafile2* must only have one alignment per file. This option allows the user to split up large sequence files into many smaller files, align them independently to *cmfile* on different computers to get many small alignments, and then merge them into a single large alignment.

--withali <*f*> Reads an alignment from file <*f*> and aligns it as a single object to the CM; e.g. the alignment in <*f*> is held fixed. This allows you to align sequences to a model with **cmalign** and view them in the context of an existing trusted multiple alignment. The alignment in the file <*f*> must be exactly the alignment that the CM was built from, or a subset of it with the following special property: the definition of consensus columns and consensus secondary structure must be identical between <*f*> and the alignment the CM was built from. One easy way to achieve this is to use the **--rf** option to **cmbuild** (see man page for **cmbuild**) and to maintain the "#=GC RF" annotation in the alignment when removing sequences to create the subset alignment <*f*>. To specify that the **--rf** option to **cmbuild** was used, enable the **--rf** option to **cmalign** (see **--rf** below).

--withpknots Must be used in combination with **--withali** <*f*>**.** Propogate structural information for any pseudoknots that exist in <*f*> to the output alignment.

--rf Must be used in combination with **--withali** <*f*>**.** Specify that the alignment in <*f*> has the same "#=GC RF" annotation as the alignment file the CM was built from using **cmbuild** and further that the **--rf** option was supplied to **cmbuild** when the CM was constructed.

--gapthresh <*x*> Must be used in combination with **--withali** <*f*>**.** Specify that the **--gapthresh** <*x*> option was supplied to **cmbuild** when the CM was constructed from the alignment file <*f*>.

--tfile <*f*> Dump tabular sequence tracebacks for each individual sequence to a file <*f*>. Primarily useful for debugging.

`cmbuild` - construct a CM from an RNA multiple sequence alignment

Synopsis

cmbuild *[options] cmfile alifile*

Description

cmbuild read an RNA multiple sequence alignment from *alifile*, constructs a covariance model (CM), and saves the CM to *cmfile.*

The alignment file must be in Stockholm format, and must contain consensus secondary structure annotation. **cmbuild** uses the consensus structure to determine the architecture of the CM.

The alignment file may be a database containing more than one alignment. If so, the resulting *cmfile* will be a database of CMs, one per alignment.

The expert options **--ctarget, --cmindiff,** and **--call** result in multiple CMs being built from each alignment in *alifile* as described below.

Output

The default output from **cmbuild** is tabular, with a single line printed for each model . Each line has the following fields: **aln:** the index of the alignment used to build the CM, **cm idx:** the index of the CM in the *cmfile;* **name:** the name of the CM, **nseq:** the number of sequences in the alignment used to build the CM, **eff_nseq:** the effective number of sequences used to build the model (see the User Guide); **alen:** the length of the alignment used to build the CM; **clen:** the number of columns from the alignment defined as consensus columns; **rel entropy, CM:** the total relative entropy of the model divided by the number of consensus columns; **rel entropy, HMM:** the total relative entropy of the model *ignoring* secondary structure divided by the number of consensus columns.

Options

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

-n <*s*> Name the covariance model <*s*>. (Does not work if *alifile* contains more than one alignment). The default is to use the name of the alignment (given by the #=GF ID tag, in Stockholm format), or if that is not present, to use the name of the alignment file minus any file type extension plus a ”-” and a positive integer indicating the position of that alignment in the file (that is, the first alignment in a file ”myrnas.sto” would give a CM named ”myrnas-1”, the second alignment would give a CM named ”myrnas-2”).

-A Append the CM to *cmfile,* if *cmfile* already exists.

-F Allow *cmfile* to be overwritten. Normally, if *cmfile* already exists, **cmbuild** exits with an error unless the *-A* or *-F* option is set.

-v Run in verbose output mode instead of using the default single line tabular format. This output format is similar to that used by older versions of Infernal.

--iins Allow informative insert emissions for the CM. By default, all CM insert emission scores are set to 0.0 bits. The motivation for zero bit scores is to avoid high-scoring hits to low complexity sequence favored by high insert state emission scores.

--Wbeta<*x*> Set the beta tail loss probability for query-dependent banding (QDB) to <*x*> The QDB algorithm is used to determine the maximium length of a hit to the model. For more information on QDB see (Nawrocki and Eddy, PLoS Computational Biology 3(3): e56). The beta paramater is the amount of probability mass considered negligible during band calculation, lower values of beta will result in shorter maximum hit lengths, which will yield faster searches. The default beta is 1E-7: determined empirically as a good tradeoff between sensitivity, specificity and speed.

--devhelp Print help, as with **-h,** but also include undocumented developer options. These options are not listed below. They 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

--rsearch <*f*> Parameterize emission scores a la RSEARCH, using the RIBOSUM matrix in file <*f*>. (Actually, the emission scores will not be identical to RIBOSUM scores due of differences in the modelling strategy between Infernal and RSEARCH, but they will be as similar as possible.) RIBOSUM matrix files are included with Infernal in the ”matrices/” subdirectory of the top-level Infernal directory. RIBOSUM matrices are substitution score matrices trained specifically for structural RNAs with separate single stranded residue and base pair substitution scores. For more information see the RSEARCH publication (Klein and Eddy, BMC Bioinformatics 4:44, 2003). Actually, the emission scores will not exactly With **--rsearch** enabled, all alignments in *alifile* must contain exactly one sequence or the **--call** option must also be enabled.

--binary Save the model in a compact binary format. The default is a more readable ASCII text format.

--rf Use reference coordinate annotation (#=GC RF line, in Stockholm) to determine which columns are consensus, and which are inserts. Any non-gap character indicates a consensus column. (For example, mark consensus columns with ”x”, and insert columns with ”.”.) The default is to determine this automatically; if the frequency of gap characters in a column is greater than a threshold, gapthresh (default 0.5), the column is called an insertion.

--gapthresh <*x*> Set the gap threshold (used for determining which columns are insertions versus consensus; see **--rf** above) to <*x*>. The default is 0.5.

--ignorant Strip all base pair secondary structure information from all input alignments in *alifile* before building the CM(s). All resulting CM(s) will have zero MATP (base pair) nodes, with zero bifurcations.

--wgsc Use the Gerstein/Sonnhammer/Chothia (GSC) weighting algorithm. This is the default unless the number of sequences in the alignment exceeds a cutoff (see **--pbswitch),** in which case the default becomes the faster Henikoff position-based weighting scheme.

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

--wpb Use the Henikoff position-based weighting scheme. This weighting scheme is automatically used (overriding **--wgsc** and **--wblosum)** if the number of sequences in the alignment exceeds a cutoff (see **--pbswitch).**

--wnone Turn sequence weighting off; e.g. explicitly set all sequence weights to 1.0.

--wgiven Use sequence weights as given in annotation in the input alignment file. If no weights were given, assume they are all 1.0. The default is to determine new sequence weights by the Gerstein/Sonnhammer/Chothia algorithm, ignoring any annotated weights.

--pbswitch <*n*> Set the cutoff for automatically switching the weighting method to the Henikoff position-based weighting scheme to <*n*>. If the number of sequences in the alignment exceeds <*n*> Henikoff weighting is used. By default <*n*> is 5000.

--wid <*x*> Controls the behavior of the --*wblosum* weighting option by setting the percent identity for clustering the alignment to <*x*>.

--eent Use the entropy weighting strategy to determine the effective sequence number that gives a target mean match state relative entropy. This option is the default, and can be turned off with **--enone.** The default target mean match state relative entropy is 0.59 bits but can be changed with **--ere.** The default of 0.59 bits is automatically changed if the total relative entropy of the model (summed match state relative entropy) is less than a cutoff, which is is 6.0 bits by default, but can be changed with the expert, undocumented **--eX** option. If you really want to play with that option, consult the source code.

--enone Turn off the entropy weighting strategy. The effective sequence number is just the number of sequences in the alignment.

--ere <*x*> Set the target mean match state relative entropy as <*x*>. By default the target relative entropy per match position is 0.59 bits.

--null <*f*> Read a null model from <*f*>. The null model defines the probability of each RNA nucleotide in background sequence, the default is to use 0.25 for each nucleotide. The format of null files is documented in the User's Guide.

--prior <*f*> Read a Dirichlet prior from <*f*>, replacing the default mixture Dirichlet. The format of prior files is documented in the User's Guide.

--ctarget <*n*> Cluster each alignment in *alifile* by percent identity. Find a cutoff percent id threshold that gives exactly <*n*> clusters and build a separate CM from each cluster. If <*n*> is greater than the number of sequences in the alignment the program will not complain, and each sequence in the alignment will be its own cluster. Each CM will have a positive integer appended to its name indicating the order in which it was built. For example, if **cmbuild --ctarget 3** is called with *alifile* "myrnas.sto", and "myrnas.sto" has exactly one Stockholm alignment in it with no #=GF ID tag annotation, three CMs will be built, the first will be named "myrnas-1.1", the second, "myrnas-1.2", and the third "myrnas-1.3". (As explained above for the **-n** option, the first number "1" after "myrnas" indicates the CM was built from the first alignment in "myrnas.sto".)

--cmaxid <*x*> Cluster each sequence alignment in *alifile* by percent identity. Define clusters at the cutoff fractional id similarity of <*x*> and build a separate CM from each cluster. No two sequences will be be more than <*x*> fractionally identical (<*x*> * 100 percent identical) if those two sequences are in different clusters. The CMs are named as described above for **--ctarget.**

--call Build a separate CM from each sequence in each alignment in *alifile.* Naming of CMs takes place as described above for **--ctarget.** Using this option in combination with **--rsearch** causes a separate CM to be built and parameterized using a RIBOSUM matrix for each sequence in *alifile.*

--corig After building multiple CMs using **--ctarget, --cmindiff** or **--call** as described above, build a final CM using the complete original alignment from *alifile.* The CMs are named as described above for **--ctarget** with the exception of the final CM built from the original alignment which is named in the default manner, without an appended integer.

--cdump<*f*> Dump the multiple alignments of each cluster to <*f*> in Stockholm format. This option only works in combination with **--ctarget, --cmindiff** or **--call.**

--refine <*f*> Attempt to refine the alignment before building the CM using expectation-maximization (EM). A CM is first built from the initial alignment as usual. Then, the sequences in the alignment are realigned optimally (with the HMM banded CYK algorithm, optimal means optimal given the bands) to the CM, and a new CM is built from the resulting alignment. The sequences are then realigned to the new CM, and a new CM is built from that alignment. This is continued until convergence, specifically when the alignments for two successive iterations are not significantly different (the summed bit scores of all the sequences in the alignment changes less than 1% between two successive iterations). The final alignment (the alignment used to build the CM that gets written to *cmfile)* is written to <*f*>.

--gibbs Modifies the behavior of **--refine** so Gibbs sampling is used instead of EM. The difference is that during the alignment stage the alignment is not necessarily optimal, instead an alignment (parsetree) for each sequences is sampled from the posterior distribution of alignments as determined by the Inside algorithm. Due to this sampling step **--gibbs** is non-deterministic, so different runs with the same alignment may yield different results. This is not true when **--refine** is used without the **--gibbs** option, in which case the final alignment and CM will always be the same. When **--gibbs** is enabled, the **-s "** <**n**>**"** option can be used to seed the random number generator predictably, making the results reproducible. The goal of the **--gibbs** option is to help expert RNA alignment curators refine structural alignments by allowing them to observe alternative high scoring alignments.

-s <*n*> Set the random seed to <*n*>, where <*n*> is a positive integer. This option can only be used in combination with **--gibbs.** The default is to use time() to generate a different seed for each run, which means that two different runs of **cmbuild --refine** <*f*> --gibbs on the same alignment will give slightly different results. You can use this option to generate reproducible results.

-l With **--refine,** turn on the local alignment algorithm, which allows the alignment to span two or more subsequences if necessary (e.g. if the structures of the query model and target sequence are only partially shared), allowing certain large insertions and deletions in the structure to be penalized differently than normal indels. The default is to globally align the query model to the target sequences.

-a With **--refine,** print the scores of each individual sequence alignment.

--cyk With **--refine,** align with the CYK algorithm. By default the optimal accuracy algorithm is used. There is more information on this in the **cmalign** manual page.

--sub With **--refine,** turn on the sub model construction and alignment procedure. For each sequence to be realigned an HMM is first used to predict the model start and end consensus columns, and a new sub CM is constructed that only models consensus columns from start to end. The sequence is then aligned to this sub CM. This option is useful for building CMs for alignments with sequences that are known to truncated, non-full length sequences. This option is experimental and not rigorously tested, use at your own risk. This "sub CM" procedure is not the same as the "sub CMs" described by Weinberg and Ruzzo.

--nonbanded With **--refine,** do not use HMM bands to accelerate alignment. Use the full CYK algorithm which is guaranteed to give the optimal alignment. This will slow down the run significantly, especially for large models.

--tau <*x*> With **--refine,** set the tail loss probability used during HMM band calculation to <*f*>. This is the amount of probability mass within the HMM posterior probabilities that is considered negligible. The default value is 1E-7. In general, higher values will result in greater acceleration, but increase the chance of missing the optimal alignment due to the HMM bands.

--fins With **--refine,** change the behavior of how insert emissions are placed in the alignment. By default, all contiguous blocks of inserts are split in half, and half the residues are flushed left against the nearest consensus column to the left, and half are flushed right against the nearest consensus column on the right. With **--fins** inserts are not split in half, instead all inserted residues from IL states are flushed left, instead all inserted residues from IR states are flushed right. This was the default behavior of previous versions of Infernal.

--mxsize <*x*> With **--refine,** set the maximum allowable matrix size for alignment to <*x*> megabytes. By default this size is 2 Gb. This should be large enough for the vast majority of alignments, however it is possible that when run with **--refine, cmbuild** will exit prematurely, reporting an error message that the matrix exceeded it's maximum allowable size. In this case, the **--mxsize** can be used to raise the limit.

--rdump<*x*> With **--refine,** output the intermediate alignments at each iteration of the refinement procedure (as described above for **--refine**) to file <*f*>.

`cmcalibrate` - fit exponential tails for E-values and determine HMM

Synopsis

cmcalibrate *[options] cmfile*

Description

cmcalibrate calibrates E-value statistics and HMM filter thresholds for the covariance models (CMs) in *cmfile.* The E-values and HMM filter threshold statistics are added to the *cmfile* and are used by **cmsearch** for increased sensitivity and acceleration in RNA homology search.

CM files are profiles of RNA consensus secondary structure. A CM file is produced by the **cmbuild** program, from a given RNA sequence alignment of known consensus structure. **cmcalibrate** is very slow. It takes several hours to calibrate a single average sized CM. **cmcalibrate** can be run in parallel with MPI. To do this, use the **--mpi** option and run cmsearch inside a MPI wrapper program such as **mpirun.** For example: **mpirun C cmcalibrate** --*mpi [other* options] *cmfile* The **--forecast** <*n*> option can be used to estimate how long the program will take to run on <*n*> processors. Unless you plan on running **cmcalibrate** in MPI mode, <*n*> should be set as 1.

cmcalibrate performs two main tasks. The first is to calibrate E-value statistics. This is done by generating random sequences and searching them with the CM and collecting hits. The histogram of the bit scores of the hits is fit to an exponential tail, and the parameters of the fitted tail are saved to the CM file. The exponential tail is used to predict the expected number of hits (E-values) at a given bit score in **cmsearch.** The random sequences are generated by an HMM that was trained on real genomic sequences with various GC contents. The goal is to have the GC distributions in the random sequences to be similar to actual genomic sequences. The second task is to determine appropriate HMM filter thresholds for the CM over the possible range of final CM bit score thresholds. This is done by sampling 10,000 sequences from the CM itself and searching them with the CM and HMM. The appropriate HMM bit score threshold for a given CM threshold is set as the HMM threshold that will recognize 99.5% of the hits that score above the CM threshold. This HMM threshold is calculated over the range of reasonable CM thresholds. Both tasks must be performed for each configuration and algorithm that **cmsearch** might use. These include HMM Viterbi, HMM Forward, CM CYK and CM Inside algorithms for E-value calibration, and CM CYK and CM Inside algorithms for HMM filter thresholds. Additionally, for each algorithm, each task must be performed twice, once for a locally configured model and once for a globally configured model. The E-values and HMM filter thresholds determined by **cmcalibrate** are only used by the **cmsearch** program. If you are not going to use **cmsearch,** do not waste time calibrating your models. The majority of the options to **cmcalibrate** fall into one of two categories, depending on which of the two main tasks they're associated with. Options that affect the exponential tail E-value fitting are prefixed with **--exp.** Options that affect the HMM filter threshold determination are prefixed with **--fil.** The calibration of E-value statistics takes the majority of the running time of **cmcalibrate.** This is because CM search algorithms are slow, and the random sequences that must be searched have to be long enough to include enough random hits that can be binned into a histogram to which an exponential tail can be reliably fit. By default the random sequence length for CM searches is 1.5 megabases (Mb), for all search modes, but 1.5 can be changed to <*x*> with **--exp-cmL-glc** <*x*> or **--exp-cmL-loc** <*x*> options for glocal and local CM search calibrations respectively. Because **cmsearch** uses HMM search algorithms to filter, **cmcalibrate** must also fit exponential tails for HMM search algorithms.

HMMs are much faster than CMs so it is possible to search much longer random sequence than 1.5 MB and not significantly increase the running time of **cmcalibrate.** The length of sequence searched with the HMM is controlled by the **--exp-fract** <*x*>**, --exp-hmmLn-glc** <*x*>**, --exp-hmmLn-loc** <*x*>**,** and the **--exp-hmmLx** <*x*> options. By default, the sequence length for HMM calibration is set as the length that will require 0.10 times the number of dynamic programming calculations as a CM E-value calibration step. (The value 0.10 can be changed to <*x*> with the **--exp-fract** <*x*> option). If this sequence length is less than a minimum value, which by default is 15.0 MB, then the minimum value is used. The minimum value can be changed to <*x*> with **--exp-hmmLn-glc** <*x*> and **--exp-hmmLn-loc** <*x*> for glocal and local HMM search calibrations separately. Similarily if this value is more than a maximum value, which by default is 1000.0 MB, then the maximum value is used. The maximum value can be changed to <*x*> with the **--exp-hmmLx** <*x*> option.

Options

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

-s <*n*> Set the random number generator seed to <*n*>, where <*n*> is a positive integer. The default is to use time() to generate a different seed for each run, which means that two different runs of **cmcalibrate** on the same CM will give slightly different E-value and HMM filter threshold parameters. You can use this option to generate reproducible results.

--forecast <*n*> Predict the running time of the calibration for *cmfile* and provided options and exit, DO NOT perform the calibration. The predictions should be used as rough estimates. The value <*n*> is the number of processors the calibration will be run on, so <*n*> equal to 1 is appropriate unless you will run **cmcalibrate** in parallel with MPI.

--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.

--mpi Run as an MPI parallel program. This option will only be available if Infernal has been configured and built with the "--enable-mpi" flag (see User's Guide for details).

Expert Options

--exp-cmL-glc <*x*> Set the length of random sequence to search for the CM **glocal** exponential tail fits to <*x*> megabases (Mb). By default, <*x*> is 1.5 Mb. Searching more sequences will make the exponential tail fits more precise, but will take longer: using <*x*> of 3.0 instead of the default of 1.5 will cause the running time of **cmcalibrate** to increase by roughly 50%.

--exp-cmL-loc <*x*> Set the length of random sequence to search for the CM **local** exponential tail fits to <*x*> megabases (Mb). By default, <*x*> is 1.5 Mb. Searching more sequences will make the exponential tail fits more precise, but will take longer: using <*x*> of 3.0 instead of the default of 1.5 will cause the running time of **cmcalibrate** to increase by roughly 50%.

--exp-hmmLn-glc <*x*> Set the minimum random sequence length to search for the HMM **glocal** exponential tail fits to <*x*> megabases (Mb). By default, <*x*> is 15.0. For more information, see the explanation regarding sequence lengths for E-value calibration above before the Options section.

--exp-hmmLn-loc <*x*> Set the minimum random sequence length to search for the HMM **local** exponential tail fits to <*x*> megabases (Mb). By default, <*x*> is 15.0. For more information, see the explanation regarding sequence lengths for E-value calibration above before the Options section.

--exp-hmmLx <*x*> Set the maximum random sequence length to search when determining HMM E-values to <*x*> megabases (Mb). By default, <*x*> is 1000.0. For more information, see the explanation regarding sequence lengths for E-value calibration above before the Options section.

--exp-fract <*x*> Set the HMM/CM fraction of dynamic programming calculations to <*x*>. By default, <*x*> is 0.10. For more information, see the explanation regarding sequence lengths for E-value calibration above before the Options section.

--exp-tailn-cglc <*x*> During E-value calibration of **glocal CM** search modes fit the exponential tail to the high scores in the histogram tail that includes <*x*> hits per Mb searched. By default this <*x*> is 25. The value 25 was chosen because it works well empirically for glocal CM modes relative to other values.

--exp-tailn-cloc <*x*> During E-value calibration of **local CM** search modes fit the exponential tail to the high scores in the histogram tail that includes <*x*> hits per Mb searched. By default this <*x*> is 75. The value 75 was chosen because it works well empirically for local CM modes relative to other values.

--exp-tailn-hglc <*x*> During E-value calibration of **glocal HMM** search modes fit the exponential tail to the high scores in the histogram tail that includes <*x*> hits per Mb searched. By default this <*x*> is 250. The value 250 was chosen because it works well empirically for glocal HMM modes relative to other values.

--exp-tailn-hloc <*x*> During E-value calibration of **local HMM** search modes fit the exponential tail to the high scores in the histogram tail that includes <*x*> hits per Mb searched. By default this <*x*> is 750. The value 750 was chosen because it works well empirically for glocal HMM modes relative to other values.

--exp-tailp <*x*> Ignore the **--exp-tailn** prefixed options and fit the <*x*> fraction right tail of the histogram to exponential tails, for all search modes.

--exp-tailxn <*n*> With **--exp-tailp** enforce that the maximum number of hits in the tail that is fit is <*n*>.

--exp-beta <*x*> During E-value calibration, by default query-dependent banding (QDB) is used to accelerate the CM search algorithms with a beta tail loss probability of 1E-15. This beta value can be changed to <*x*> using the **--exp-beta** <*x*> option. The beta parameter is the amount of probability mass excluded during band calculation, higher values of beta give greater speedups but sacrifice more accuracy than lower values. A recommended value is 1E-7 (0.00001). QDB is explained in more detail in the manual page for **cmsearch** and in (Nawrocki and Eddy, PLoS Computational Biology 3(3): e56).

--exp-no-qdb Turn of QDB during E-value calibration. This will slow down calibration, and is not recommended unless you plan on using **--no-qdb** in **cmsearch.**

--exp-hfile <*f*> Save the histograms fit for the E-value calibration to file <*f*>. The format of this file is two tab delimited columns. The first column is the x-axis values of bit scores of each bin. The second column is the y-axis values of number of hits per bin. Each series is delimited by a line with a single character "&". The file will contain one series for each exponential tail fit, i.e. one series of empirical data for each line of output from **cmcalibrate** that begins with "exp tail".

--exp-sfile <*f*> Save a survival plot for the E-value calibration to file <*f*>. The format of this file is two tab delimited columns. The first column is the x-axis values of bit scores of each bin. The second column is the y-axis values of fraction of hits that meet or exceed the score for each bin. Each series is delimited by a line with a single character "&". The file will contain three series' of data for each exponential tail fit, i.e. three series for each line of output from **cmcalibrate** that begins with "exp tail". The first series is the empirical survival plot from the histogram of hits to the random sequence. The second series is the exponential tail fit to the empirical distribution. The third series is the exponential tail fit if lambda were fixed and set as the natural log of 2 (0.691314718).

--exp-qqfile <*f*> Save a quantile-quantile plot for the E-value calibration to file <*f*>. The format of this file is two tab delimited columns. The first column is the x-axis values, and the second column is the y-axis values. The distance of the points from the identity line (y=x) is a measure of how good the exponential tail fit is, the closer the points are to the identity line, the better the fit is. Each series is delimited by a line with a single character "&". The file will contain one series of empirical data for each exponential tail fit, i.e. one series for each line of output from **cmcalibrate** that begins with "exp tail".

--exp-ffile <*f*> Save statistics on the exponential tail statistics to file <*f*>. The file will contain the lambda and mu values for exponential tails fit to tails of different sizes. For example, by default **cmcalibrate** fits exponential tails to the rightmost 0.01 (1) of the score histogram and stores the parameters of that exponential tail to the CM file. (The value of 0.01 can be changed to <*x*> with the **--exp-tailp** <*x*> option). When **--exp-ffile** <*f*> is used the file <*f*> will include the exponential tail parameters for fits to various fractions of the histogram tail, instead of just to 0.01.

--fil-N <*n*> Set the number of sequences sampled and searched for the HMM filter threshold calibration to <*n*>. By default, <*n*> is 10,000.

--fil-F <*x*> Set the fraction of sample sequences the HMM filter must be able to recognize, and allow to survive, to <*x*>, where <*x*> is a positive real number less than or equal to 1.0. By default, <*x*> is 0.995.

--fil-xhmm <*x*> Set the target number of dynamic programming calculations for a HMM filtered CM QDB search with beta = 1E-7 to <*x*> times the number of calculations required to do an HMM search. By default, <*x*> is 2.0.

--fil-tau <*x*> Set the tail loss probability during HMM band calculation for HMM filter threshold calibration to <*x*>. This is the amount of probability mass within the HMM posterior probabilities that is considered negligible. The default value is 1E-7. In general, higher values will result in greater acceleration, but increase the chance of missing the optimal alignment due to the HMM bands.

--fil-gemit During HMM filter calibration, always sample sequences from a globally configured CM, even when calibrating local modes. By default, sequences are sampled from a globally configured CM when calibrating the global search modes, and sampled from a locally configured CM when calibrating the local search modes.

--fil-dfile <*f*> Save statistics on filter threshold calibration, including HMM and CM scores for all sampled sequences, to file <*f*>.

--mxsize <*x*> Set the maximum allowable DP matrix size to <*x*> megabytes. By default this size is 2,048 Mb. This should be large enough for the vast majority of calibrations, however if it is not **cmcalibrate** will exit prematurely and report an error message that the matrix exceeded it's maximum allowable size. In this case, the **--mxsize** can be used to raise the limit.

`cmemit` - generate sequences from a covariance model

Synopsis

cmemit *[options] cmfile seqfile*

Description

cmemit reads the covariance model(s) (CMs) in *cmfile* and generates a number of sequences from the CM(s); or if the **-c** option is selected, generates a single majority-rule consensus. This can be useful for various application in which one needs a simulation of sequences consistent with a sequence family consensus. By default, **cmemit** generates 10 sequences and outputs them in FASTA (unaligned) format to *seqfile.*

General Options

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

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

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

-u Write the generated sequences in unaligned format (FASTA). This is the default, so this option is probably useless.

-a Write the generated sequences in an aligned format (STOCKHOLM) with consensus structure annotation rather than FASTA.

-c Predict a single majority-rule consensus sequence instead of sampling sequences from the CM's probability distribution. Highly conserved residues (base paired residues that score higher than 3.0 bits, or single stranded residues that score higher than 1.0 bits) are shown in upper case; others are shown in lower case.

-l Configure the CMs into local mode before emitting sequences. See the User's Guide for more information on locally configured CMs.

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

--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

--rna Specify that the emitted sequences be output as RNA sequences. This is true by default.

--dna Specify that the emitted sequences be output as DNA sequences. By default, the output alphabet is RNA.

--tfile <*f*> Dump tabular sequence parsetrees (tracebacks) for each emitted sequence to file <*f*>. Primarily useful for debugging.

--exp <*x*> Exponentiate the emission and transition probabilities of the CM by <*x*> and then renormalize those distributions before emitting sequences. This option changes the CM probability distribution of parsetrees relative to default. With <*x*> less than 1.0 the emitted sequences will tend to have lower bit scores upon alignment to the CM with **cmalign.** With <x> greater than 1.0, the emitted sequences will tend to have higher bit scores upon alignment to the CM. This bit score difference will increase as <x> moves further away from 1.0 in either direction. If <x> equals 1.0, this option has no effect relative to default. This option is useful for generating sequences that are either difficult ($\langle x \rangle < 1.0$) or easy ($\langle x \rangle > 1.0$) for the CM to distinguish as homologous from background, random sequence.

--begin <*n*> Truncate the resulting alignment by removing all residues before consensus column <*n*>, where <n> is a positive integer no greater than the consensus length of the CM. Must be used in combination with **--end** and either **-a** or **--shmm** (a developer option).

--end <*n*> Truncate the resulting alignment by removing all residues after consensus column <*n*>, where <n> is a positive integer no greater than the consensus length of the CM. Must be used in combination with **--begin** and either **-a** or **--shmm** (a developer option).

`cmscore` - align and score one or more sequences to a CM

Synopsis

cmscore *[options] cmfile seqfile*

Description

cmscore uses the covariance model (CM) in *cmfile* to align and score the sequences in *seqfile*, and output summary statistics on timings and scores. **cmscore** is a testbed for new CM alignment algorithms, and it is also used by the testsuite. It is not intended to be particularly useful in the real world. Documentation is provided for completeness, and to aid our own memories.

CM files are profiles of RNA consensus secondary structure. A CM file is produced by the **cmbuild** program, from a given RNA sequence alignment of known consensus structure.

cmscore aligns the sequence(s) in *seqfile* using two alignment algorithms and compares the scores and timings of each algorithm. By default the two algorithms compared are the divide and conquer (D&C) CYK algorithm (SR Eddy, BMC Bioinformatics 3:18, 2002), and the HMM banded standard CYK algorithm. When the **--nonbanded** option is enabled D&C CYK is compared with the standard CYK alignment algorithm. In this case, because both algorithms should find the optimal alignment the parsetree scores should be nearly identical (within 0.01 bits), if this is not the case for any sequence **cmscore** exits and prints an error message. When the **--viterbi** option is enabled D&C CYK is compared with Viterbi alignment to a CM Plan 9 HMM constructed to be maximally similar to the CM. While non-banded CYK variants are guaranteed to find the optimal alignment and score of each sequence, HMM banded CYK sacrifices this guarantee for acceleration. The level of acceleration can be controlled by the tau parameter, which is set with the **--tau** <*x*> option. This is described in more detail in the man page for **cmalign,** but in short, <*x*> is a rough estimate at the probability that the optimal alignment will be missed. The greater <*x*> is, the greater the acceleration, but the greater the chance of missing the optimal alignment. By default tau is set as 1E-7. **cmscore** is useful for testing for values of tau that give the best trade-off between acceleration versus accuracy. To make this testing easier, multiple tau values can be tested within a single cmscore call. The **--taus** <*x*> and **--taue** <*x*> option combination allow the user to specify a beginning tau value and an ending tau value. For example, **--taus** *3* and and **--taue** *5* would first align the sequences in *seqfile* with non-banded D&C CYK, and then perform 3 additional HMM banded alignments, first with tau=1E-3, next with tau=1E-4 and finally with tau=1E-5. Currently, only values of 1E-<x> can be used. Summary statistics on timings and how often the optimal alignment is missed for each value or tau are printed to stdout.

When comparing the non-banded standard CYK and D&C CYK algorithms with the **--nonbanded** option, the two parse trees should usually be identical for any sequence, because the optimal alignment score is guaranteed. However, there can be cases of ties, where two or more different parse trees have identical scores. In such cases, it is possible for the two parse trees to differ. The parse tree selected as ”optimal” from amongst the ties is arbitrary, dependent on order of evaluation in the DP traceback, and the order of evaluation for D&C vs. standard CYK is different. Thus, in its testsuite role, **cmscore** checks that the scores are within 0.01 bits of each other, but does not check that the parse trees are absolutely identical.

The alignment algorithms can be run in ”search” mode within **cmscore** by using the **--search** option. When **--search** is enabled, **--inside** specifies that the Inside algorithm be used instead of CYK and **--forward**

specifies that the HMM Forward algorithm be used instead of CYK.

The sequences are treated as single stranded RNAs; that is, only the given strand of each sequence is aligned and scored, and no reverse complementing is done.

Options

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

-n <*n*> Set the number of sequences to generate and align to <*n*>. This option is incompatible with the **--infile** option.

-l Turn on the local alignment algorithm, which allows the alignment to span two or more subsequences if necessary (e.g. if the structures of the query model and target sequence are only partially shared), allowing certain large insertions and deletions in the structure to be penalized differently than normal indels. The default is to globally align the query model to the target sequences.

-s <*n*> Set the random seed to <*n*>, where <n> is a positive integer. The default is to use time() to generate a different seed for each run, which means that two different runs of **cmscore** on the same CM will give different results. You can use this option to generate reproducible results. The random number generator is used to generate sequences to score, so **-s** is incompatible with the **--infile** option which supplies the sequences to score in an input file.

-a Print individual timings and score comparisons for each sequence in *seqfile.* By default only summary statistics are printed.

--sub Turn on the sub model construction and alignment procedure. For each sequence, an HMM is first used to predict the model start and end consensus columns, and a new sub CM is constructed that only models consensus columns from start to end. The sequence is then aligned to this sub CM. This option is useful for aligning sequences that are known to truncated, non-full length sequences. This ”sub CM” procedure is not the same as the ”sub CMs” described by Weinberg and Ruzzo. When used in combination with **--tfile** the parsetree printed is not the sub CM parsetree, but rather the sub CM parstree mapped onto the topology of the original CM. This mapped parsetree will likely have a different score (sometimes much worse) than the sub CM parsetree, both of those scores are printed to the parsetree file for each sequence.

--mxsize <*x*> Set the maximum allowable DP matrix size to <*x*> megabytes. By default this size is 2048 Mb. This should be large enough for the most alignments, however if it is not **cmscore** will exit prematurely and report an error message that the matrix exceeded it’s maximum allowable size. In this case, the **--mxsize** can be used to raise the limit, or if **--nonbanded** is enabled, the **--scoreonly** option will solve the memory issue. This memory error is most likely to occur when the **--nonbanded** option

is used without the **--scoreonly** option, but can still occur when **--nonbanded** is not used.

--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.

--mpi Run as an MPI parallel program. This option will only be available if Infernal has been configured and built with the **--enable-mpi** flag (see User's Guide for details).

Expert Options

--emit Generate sequences to score by sampling from the CM. This option is on by default. The number of sequences generated is 10 by default but can be changed with the **-n** option. The sequences generated from the CM can be saved to an output file in FASTA format with the **--outfile** option.

--random Generate sequences to score by sampling from the CMs null distribution. This option turns the **--emit** option off. By default the CM distribution will be 0.25 for each of the four RNA nucleotides, but it may be different if the **--null** option was used when **cmbuild** created the *cmfile.* By default, the length of the sequences generated is sampled from the length distribution of the CM. The average length of the random sequences will be the consensus length of the RNA family modelled by the CM (or very close to it). Alternatively, the **--Lmin** <*n*> and **--Lmax** <*n*> options can be used to specify a length distribution. The number of sequences generated is 10 by default but can be changed with the **-n** option. The random sequences generated can be saved to an output file in FASTA format with the **--outfile** option.

--infile <*f*> Sequences to score are read from the file <*f*>. All the sequences from <*f*> are read and scored, the **-n** and **-s** options are incompatible with **--infile.**

--outfile <*f*> Save generated sequences that are scored to the file <*f*> in FASTA format. This option is incompatible with the **--infile** option.

--Lmin <*n1*> Must be used in combination with **--random** and **--Lmax** <*n2*>. The lengths of the random sequences generated and scored will be uniform between the range of <*n1*>..<*n2*>.

--Lmax <*n2*> Must be used in combination with **--random** and **--Lmin** <*n1*>**.** The lengths of the random sequences generated and scored will be uniform between the range of <*n1*>..<*n2*>.

--pad Must be used in combination with **--emit** and **--search.** Add <*n*> cm->W (max hit length) minus L (sequence <x> length) residues to the 5' and 3' end of each emitted sequence <x>.

--hbanded Specify that the second stage alignment algorithm be HMM banded CYK. This option is on by default. For more information on this option, see the description of the **--hbanded** option in the man page for **cmalign.**

--tau <*x*> For stage 2 alignment, set the tail loss probability used during HMM band calculation to <*x*>. This is the amount of probability mass within the HMM posterior probabilities that is considered negligible. The default value is 1E-7. In general, higher values will result in greater acceleration, but increase the chance of missing the optimal alignment due to the HMM bands.

--aln2bands With **--search,** when calculating HMM bands, use an HMM alignment algorithm instead of an HMM search algorithm. In general, using this option will result in greater acceleration, but will increase the chance of missing the optimal alignment.

--hsafe For stage 2 HMM banded alignment, realign any sequences with a negative alignment score using non-banded CYK to guarantee finding the optimal alignment.

--nonbanded Specify that the second stage alignment algorithm be standard, non-banded, non-D&C CYK. When **--nonbanded** is enabled, the program fails with a non-zero exit code and prints an error message if the parsetree score for any sequence from stage 1 D&C alignment and stage 2 alignment differs by more than 0.01 bits. In theory, this should never happen as both algorithms are guaranteed to determine the optimal parsetree. For larger RNAs (more than 300 residues) if memory is limiting, **--nonbanded** should be used in combination with **--scoreonly.**

--scoreonly With **--nonbanded** during the second stage standard non-banded CYK alignment, use the ”score only” variant of the algorithm to save memory, and don’t recover a parse tree.

--viterbi Specify that the second stage alignment algorithm be Viterbi to a CM Plan 9 HMM.

--search Run all algorithms in scanning mode, not alignment mode. This means the highest scoring subsequence within each sequence is returned as the score, not necessarily the score of an alignment of the full sequence.

--inside With **--search** Compare the non-banded scanning Inside algorithm to the HMM banded scanning Inside algorith, instead of using CYK versions.

--forward With **--search** Compare the scanning Forward scoring algorithm against CYK.

--taus <*n*> Specify the first alignment algorithm as non-banded D&C CYK, and multiple stages of HMM banded CYK alignment. The first HMM banded alignment will use tau=1E-<x>, which will be the highest value of tau used. Must be used in combination with **--taue.**

--taue <*n*> Specify the first alignment algorithm as non-banded D&C CYK, and multiple stages of HMM banded CYK alignment. The final HMM banded alignment will use tau=1E-<x>, which will be the lowest value of tau used. Must be used in combination with **--taus.**

--tfile <*f*> Print the parsetrees for each alignment of each sequence to file <*f*>.

`cmsearch` - search a sequence database for RNAs homologous to a CM

Synopsis

cmsearch *[options] cmfile seqfile*

Description

cmsearch uses the covariance model (CM) in *cmfile* to search for homologous RNAs in *seqfile,* and outputs high-scoring alignments.

Currently, the sequence file must be in FASTA format.

CM files are profiles of RNA consensus secondary structure. A CM file is produced by the **cmbuild** program, from a given RNA sequence alignment of known consensus structure. CM files can be calibrated prior to running **cmsearch** with the **cmcalibrate** program. Searches with calibrated CM files will include E-values and will use appropriate filter thresholds for acceleration. It is strongly recommended to calibrate your CM files before using **cmsearch.** CM calibration is described in more detail below and in chapters 5 and 6 of the User Guide.

cmsearch output consists of alignments of all hits in the database sorted by decreasing score per sequence and per strand. That is, all hits for the same sequence and the same (Watson or Crick) strand are sorted, but hits across sequences or strands are not sorted.

The threshold for reporting scores is different depending on whether the CM file has been calibrated or not. If the CM file has been calibrated, the default reporting threshold is an E-value of 1.0. This is the threshold at which 1 hit is expected by chance. It is possible to manually set the threshold to bit score <*x*> using the **-T** <*x*> option as described below, or to E-value <*x*> using the **-E** <*x*> option. The **-E** option will only work if the CM file has been calibrated.

RNA homology search with CMs is slow. To speed it up, **cmsearch** by default uses two rounds of filters with faster algorithms to prune the database prior to searching with the slow CM algorithm. The first round of filtering is faster but less strict than the second round. First, the full database is searched with the first round filter, then any hits that survive the first round are searched with the second round filter. Finally any hits that survive the first and second round of filtering are searched with the final round search strategy. During the filter rounds, hits are padded with a short stretch of residues on either side prior to searching with the subsequent round. The exact number of residues is dependent on the size of the model being searched with. The first round of filtering is performed with an HMM. If the CM file is calibrated, the threshold for the HMM filter will be automatically chosen as an appropriate one as determined in **cmcalibrate.** If the model is not calibrated, the default HMM filter threshold is 3.0 bits. The HMM filter threshold can be manually set to bit score <*x*> using the **--fil-T-hmm** <*x*> option as described below, or to E-value <*x*> using the **--fil-E-hmm** <*x*> option. The **--fil-E-hmm** option will only work if the CM file has been calibrated. The HMM filter can be turned off with the **--fil-no-hmm** option. The second round of filtering is performed with the CM (not an HMM) using query-dependent banding (QDB) for acceleration. Briefly, QDB precalculates regions of the dynamic programming matrix that have negligible probability based on the query CM's transition probabilities. During search, these regions of the matrix are ignored to make searches faster. For more information on QDB see (Nawrocki and Eddy, PLoS Computational Biology 3(3):

e56). The beta paramater is the amount of probability mass considered negligible during band calculation, lower values of beta yield greater speedups but also a greater chance of missing the optimal alignment. The default beta is 1E-7: determined empirically as a good tradeoff between sensitivity and speed, though this value can be changed with the **--fil-beta** <*x*> option. If the CM file has been calibrated, the QDB filter threshold will be automatic set to an appropriate value using an ad-hoc procedure (see the User Guide). If the CM file has not been calibrated, the default QDB filter threshold is 0.0 bits. The QDB filter threshold can be manually set to bit score <*x*> using the **--fil-T-qdb** <*x*> option as described below, or to E-value <*x*> using the **--fil-E-qdb** <*x*> option. The **--fil-E-qdb** option will only work if the CM file has been calibrated. The QDB filter can be turned off with the **--fil-no-qdb** option. Another way to accelerate **cmsearch** is to run it in parallel with MPI on multiple computers. To do this, use the **--mpi** option and run **cmsearch** inside a MPI wrapper program such as **mpirun.** For example: **mpirun C cmsearch** --*mpi [other* options] *cmfile seqfile* The **--forecast** <*n*> option will estimate how long a search will take for your *cmfile* and *seqfile* on <*n*> processors. Unless you plan on running **cmsearch** in MPI mode, <*n*> should be set as 1.

Another technique for accelerated CM homology search with HMM filters is the construction and use of a ”rigorous filter” HMM which was developed by Zasha Weinberg and Larry Ruzzo. All hits above a certain CM bit score threshold are guaranteed to survive the HMM filtering step. Their implementation of rigorous filters has been included in previous versions of Infernal, but not in the current version. For more information see the User’s Guide.

Output

By default, **cmsearch** outputs the alignments of search hits that score above the final search round threshold. The format of this output is described in the ”Tutorial” section of the User guide. This format has purposefully not been changed from the 0.x versions of Infernal so as not to break existing parsers. However, it can be augmented with a new line of output that marks non-compensatory basepairs with an ’x’ by using the **-x** option. This option was added to facilitate quick analysis of the secondary structure of hits by eye. Additionally, the **-p** option can be used to annotate the posterior probability of each aligned residue in the hit alignments as described below. The **--tabfile** <*f*> outputs a tabular representation of the hits found by **cmsearch** to the file <*f*>. Each non-# prefixed line of this file corresponds to a hit, and each such line has 8 fields: ”target name” the name of the target sequence the hit was found in, ”target coord - start”: the start position of the hit in the target sequence, ”target coord - stop”: the end position of hit in the target sequence, ”query coord - start”: the start position of the hit in the query model, ”query coord - stop”: the end position of hit in the query sequence, ”bit sc”: the bit score of the hit, ”E-value”: the E-value of the hit (if available, ”-” if not), and ”GC” the percentage of G and C residues in the hit within the target sequence. **cmsearch** tab files can be used as input to the Easel miniapp **esl-sfetch** (included in the easel/miniapp/ subdirectory of infernal) with the **-C -f --tabfile** options to extract all the hits from the target database file to a new FASTA file. This file can then be aligned to a CM with **cmalign.**

Options

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

-o <*f*> Save the high-scoring alignments of hits to a file <*f*>. The default is to write them to standard output.

-g <*f*> Turn on the 'glocal' alignment algorithm, local with respect to the target database, and global with respect to the model. By default, the local alignment algorithm is used which is local with respect to both the target sequence and the model. In local mode, the alignment to span two or more subsequences if necessary (e.g. if the structures of the query model and target sequence are only partially shared), allowing certain large insertions and deletions in the structure to be penalized differently than normal indels. Local mode performs better on empirical benchmarks and is significantly more sensitive for remote homology detection. Empirically, glocal searches return many fewer hits than local searches, so glocal may be desired for some applications.

-p Append posterior probabilities to alignments of hits. For more information on posterior probabilities see the description of the **-p** option in the manual page for **cmalign.**

-x Annotate negative scoring basepairs and basepairs that include a gap in the left or right half of the pair (but not both) with x's in the alignments of hits. The x's appear above the structural annotation in the alignment output. Basepairs without x's above them are compensatory with respect to the model. Compensatory mutations are good evidence for structural homology.

-Z <*x*> Calculate E-values as if the target database size was <*x*> megabases (Mb). Ignore the actual size of the database. This option is only valid if the CM file has been calibrated. Warning: the predictions for timings and survival fractions will be calculated as if the database was of size <*x*> Mb, which means they will be inaccurate.

--toponly Only search the top (Watson) strand of the sequences in *seqfile.* By default, both strands are searched.

--bottomonly Only search the bottom (Crick) strand of the sequences in *seqfile.* By default, both strands are searched.

--forecast <*n*> Predict the running time of the search with provided files and options and exit, **DO NOT** perform the search. This option is only available with calibrated CM files. The predictions should be used as rough estimates and can be fairly inaccurate, especially for highly biased target databases (for example 80% AT genomes). The value for <*n*> is the number of processors the search will be run on, so <*n*> equal to 1 is appropriate unless you will run **cmsearch** in parallel with MPI.

--informat <*s*> Assert that the input *seqfile* is in format <*s*>. Do not run Babelfish format autodection. This increases the reliability of the program somewhat, because the Babelfish can make mistakes; particularly recommended for unattended, high-throughput runs of Infernal. <*s*> is case-insensitive. Acceptable formats are: FASTA, EMBL, UNIPROT, GENBANK, and DDBJ. <*s*> is case-insensitive.

--mxsize <*x*> Set the maximum allowable DP matrix size to <*x*> megabytes. By default this size is 2,048 Mb. This should be large enough for the vast majority of alignments,

however if it is not **cmsearch** will exit prematurely and report an error message that the matrix exceeded it's maximum allowable size. In this case, the **--mxsize** can be used to raise the limit.

--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.

--mpi Run as an MPI parallel program. This option will only be available if Infernal has been configured and built with the "--enable-mpi" flag (see User's Guide for details).

Expert Options

--inside Use the Inside algorithm for the final round of searching. This is true by default.

--cyk Use the CYK algorithm for the final round of searching.

--forward Search only with an HMM. This is much faster but less sensitive than a CM search. Use the Forward algorithm for the HMM search.

--viterbi Search only with an HMM. This is much faster but less sensitive than a CM search. Use the Viterbi algorithm for the HMM search.

-E <*x*> Set the E-value cutoff for the per-sequence/strand ranked hit list to <*x*>, where <*x*> is a positive real number. Hits with E-values better than (less than) or equal to this threshold will be shown. This option is only available if the CM file has been calibrated. This threshold is relevant only to the final round of searching performed after all filters have been used, not to the filter rounds themselves.

-T <*x*> Set the bit score cutoff for the per-sequence ranked hit list to <*x*>, where <*x*> is a positive real number. Hits with bit scores better than (greater than) this threshold will be shown. This threshold is relevant only to the final round of searching performed after all filters have been used, not to the filter rounds themselves.

--nc Set the bit score cutoff as the NC cutoff value used by Rfam curators as the noise cutoff score. This is the highest scoring hit found by this model during Rfam curation that the Rfam curators defined as a noise (false positive) sequence. The NC cutoff is defined as <*x*> bits in the original Stockholm alignment the model was built from with a line: *#=GF NC* <*x*> positioned before the sequence alignment. If such a line existed in the alignment provided to **cmbuild** then the **--nc** option will be available in **cmsearch.** If no such line existed when **cmbuild** was run, then using the **--nc** option to **cmsearch** will cause the program to print an error message and exit.

--ga Set the bit score cutoff as the GA cutoff value used by Rfam curators as the gathering threshold. The GA cutoff is defined in a stockholm file used to build the model in the same way as the NC cutoff (see above), but with a line: *#=GF GA <x>*

--tc Set the bit score cutoff as the TC cutoff value used by Rfam curators as the trusted cutoff. The TC cutoff is defined in the stockholm file used to build the model in the same way as the NC cutoff (see above), but with a line: *#=GF TC <x>*

--no-qdb Do not use query-dependent banding (QDB) for the final round of search. By default, QDB is used in the final round of search with beta = 1E-15, after all filtering is finished.

--beta <*x*> For query-dependent banding (QDB) during the final round of search, set the beta parameter to <*x*> where <*x*> is any positive real number less than 1.0. Beta is the probability mass considered negligible during band calculation. The default beta for the final round of search is 1E-15.

--hbanded Use HMM bands to accelerate the final round of search. Constraints for the CM search are derived from posterior probabilities from an HMM. This is an experimental option and it is not recommended for use unless you know exactly what you're doing.

--tau <*x*> Set the tail loss probability during HMM band calculation to <*x*>. This is the amount of probability mass within the HMM posterior probabilities that is considered negligible. The default value is 1E-7. In general, higher values will result in greater acceleration, but increase the chance of missing the optimal alignment due to the HMM bands. This option only makes sense in combination with **--hbanded**

--fil-no-hmm Turn the HMM filter off.

--fil-no-qdb Turn the QDB filter off.

--fil-beta For the QDB filter, set the beta parameter to <*x*> where <*x*> is any positive real number less than 1.0. Beta is the probability mass considered negligible during band calculation. The default beta for the QDB filter round of search is 1E-7.

--fil-T-qdb <*x*> Set the bit score cutoff for the QDB filter round to <*x*>, where <*x*> is a positive real number. Hits with bit scores better than (greater than) this threshold will survive the QDB filter and be passed to the final round.

--fil-T-hmm <*x*> Set the bit score cutoff for the HMM filter round to <*x*>, where <*x*> is a positive real number. Hits with bit scores better than (greater than) this threshold will survive the HMM filter and be passed to the next round, either a QDB filter round, or if the QDB filter is disabled, to the final round of search.

--fil-E-qdb <*x*> Set the E-value cutoff for the QDB filter round. <*x*>, where <*x*> is a positive real number. Hits with E-values better than (less than) or equal to this threshold will survive and be passed to the final round. This option is only available if the CM file has been calibrated.

--fil-E-hmm <*x*> Set the E-value cutoff for the HMM filter round. <*x*>, where <*x*> is a positive real number. Hits with E-values better than (less than) or equal to this threshold will survive and be passed to the next round, either a QDB filter round, or if the QDB filter is disable, to the final round of search. This option is only available if the CM file has been calibrated.

--fil-Smax-hmm <*x*> Set the maximum predicted survival fraction for an HMM filter as <*x*>, where <*x*> is a positive real number less than 1.0. The E-value cutoff for the HMM filter will be set as the value <*y*>, such that if <*y*> hits survived the filter it is predicted that exactly <*x*> fraction of the residues in the database would survive.

--hmm-W <*n*> Set the HMM window size W (maximum size of a hit) to <*n*>. This option only works in combination with **--forward** or **--viterbi.** By default, W is calculated automatically, but this automatic calculation is time consuming for large models.

--hmm-cW <*x*> Set the HMM window size W (maximum size of a hit) as <*x*> times the consensus length of the CM. The consensus length (clen) of the CM can be determined using the **cmstat** program. This option only works in combination with **--forward** or **--viterbi.** By default, W is calculated automatically, but this automatic calculation is time consuming for large models. To find potential full length hits to the model <**x**> should be greater than 1.0, but values above 2.0 are probably wasteful.

--noalign Do not calculate and print alignments of each hit, only print locations and scores.

--aln-hbanded Use HMM bands to accelerate alignment during the hit alignment stage.

--aln-optacc Calculate alignments of hits from final round of search using the optimal accuracy algorithm which computes the alignment that maximizes the summed posterior probability of all aligned residues given the model, which can be different from the highest scoring one.

--tabfile <*f*> Create a new output file <*f*> and print tabular results to it. The format of the tabular results is listed in the **OUTPUT** section. The tabular results can be more easily parsed by scripts than the default **cmsearch** output. The **esl-sfetch** miniapp included in the easel/miniapps/ subdirectory of infernal has a **--tabfile** option that allows it to read **cmsearch** tab files and fetch the hits reported within them from the target database into a new sequence file.

--gcfile <*f*> Create a new output file <*f*> and print statistics of the GC content of the sequences in *seqfile* to it. The sequences are partitioned into 100 nt non-overlapping windows, and the GC percentage of each window is calculated. A normalized histogram of those GC percentages is then printed to <*f*> This file can be generated even if **cmsearch** is run with **--forecast** and no search is performed.

--rna Output the hit alignments as RNA sequences alignments. This is true by default.

--dna Output the hit alignments as DNA sequence alignments.

`cmstat` - display summary statistics for a CM

Synopsis

cmstat *[options] cmfile*

Description

cmstat calculates and displays various types of statistics describing the covariance models (CMs) in *cmfile.*

CM files are profiles of RNA consensus secondary structure. A CM file is produced by the **cmbuild** program, from a given RNA sequence alignment of known consensus structure. CM files can be calibrated with the **cmcalibrate** program. Searches with calibrated CM files will include E-values and will use appropriate filter thresholds for faster speed. It is strongly recommended to calibrate your CM files before using **cmsearch.** CM calibration is described in more detail below and in chapters 5 and 6 of the User Guide. **cmstat** is useful for determining statistics on calibrated or non-calibrated CM files. By default, **cmstat** prints general statistics of the model and the alignment it was built from. If the model(s) in *cmfile* have been calibrated with **cmcalibrate** the **--le** and **--ge** options can be used to print statistics on the the exponential tails used for calculating E-values for the various possible search modes for locally (**--le**) and globally configured (**--ge**) models in **cmsearch.** If *cmfile* is calibrated, HMM filter threshold statistics can be printed for local inside CM search with **--lfi,** for glocal inside CM search with **--gfi,** for local CYK CM search with **--lfc,** and for glocal CYK CM search with **--gfc.** The **--search** option causes **cmstat** performing a timing experiment for homology search. Statistics will be printed on how many kilobases can be scanned per second for the different possible algorithms in **cmsearch.**

Options

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

-g Turn on the 'glocal' alignment algorithm, local with respect to the target database, and global with respect to the model. By default, the model is configured for local alignment which is local with respect to both the target sequence and the model.

-m print general statistics on the models in *cmfile* and the alignment it was built from.

-Z <*x*> Calculate E-values as if the target database size was <*x*> megabases (Mb). Ignore the actual size of the database. This option is only valid if the CM file has been calibrated.

--all print all available statistics

--le print local E-value statistics. This option only works if *cmfile* has been calibrated with **cmcalibrate.**

--ge print glocal E-value statistics. This option only works if *cmfile* has been calibrated with **cmcalibrate.**

--beta <*x*> With the **--search** option set the beta parameter for the query-dependent banding algorithm stages to <*x*> Beta is the probability mass considered negligible during band calculation. The default is 1E-7.

--qdbfile <*f*> Save the query-dependent bands (QDBs) for each state to file <*f*>

Expert Options

--lfi Print the HMM filter thresholds for the range of relevant CM bit score cutoffs for searches with locally configured models using the Inside algorithm.

--gfi Print the HMM filter thresholds for the range of relevant CM bit score cutoffs for searches with globally configured models using the Inside algorithm.

--lfc Print the HMM filter thresholds for the range of relevant CM bit score cutoffs for searches with locally configured models using the CYK algorithm.

--gfc Print the HMM filter thresholds for the range of relevant CM bit score cutoffs for searches with globally configured models using the CYK algorithm.

-E <*x*> Print filter threshold statistics for an HMM filter if a final CM E-value cutoff of <*x*> were to be used for a run of **cmsearch** on 1 MB of sequence. (Remember **cmsearch** considers a 500,000 nucleotide sequence file as 1 MB of sequence because by default both strands of the sequence are searched). The size 1 MB of sequence can be changed to the size of a given database in file <*f*> using the **--seqfile** <*f*> option.

-T <*x*> Print filter threshold statistics for an HMM filter if a final CM bit score cutoff of <*x*> were to be used for a run of **cmsearch.**

--nc Print filter threshold statistics for an HMM filter if a CM bit score cutoff equal to the Rfam NC cutoff were to be used for a run of **cmsearch.** The NC cutoff is defined as <*x*> bits in the original Stockholm alignment the model was built from with a line: *#=GF NC <x>* positioned before the sequence alignment. If such a line existed in the alignment provided to **cmbuild** then the **--nc** option will be available in **cmstat** If no such line existed when **cmbuild** was run, then using the **--nc** option to **cmstat** will cause the program to print an error message and exit.

--ga Print filter threshold statistics for an HMM filter if a CM bit score cutoff of Rfam GA cutoff value were to be used for a run of **cmsearch.** The GA cutoff is defined in a stockholm file used to build the model in the same way as the NC cutoff (see above), but with a line: *#=GF GA <x>*

--tc Print filter threshold statistics for an HMM filter if a CM bit score cutoff equal to the Rfam TC cutoff value were to be used for a run of **cmsearch.** The TC cutoff is defined in a stockholm file used to build the model in the same way as the NC cutoff (see above), but with a line: *#=GF TC <x>*

--seqfile <*x*> With the **-E** option, use the database size of the database in <*x*> instead of the default database size of 1 MB.

--toponly In combination with **--seqfile** <*x*> option, only consider the top strand of the database in <*x*> instead of both strands. **--search** perform an experiment to determine how fast the CM(s) can search with different search algorithms.

--cmL <*n*> With the **--search** option set the length of sequence to search with CM algorithms as <*n*> residues. By default, <*n*> is 1000.

--hmmL <*n*> With the **--search** option set the length of sequence to search with HMM algorithms as <*n*> residues. By default, <*n*> is 100,000.

--efile <*f*> Save a plot of **cmsearch** HMM filter E value cutoffs versus CM E value cutoffs in xmgrace format to file <*f*>. This option must be used in combination with **--lfi, --gfi, --lfc** or **--gfc.**

--bfile <*f*> Save a plot of **cmsearch** HMM bit score cutoffs versus CM bit score cutoffs in xmgrace format to file <*f*>. This option must be used in combination with **--lfi, --gfi, --lfc** or **--gfc.**

--sfile <*f*> Save a plot of **cmsearch** predicted survival fraction from the HMM filter versus CM E value cutoff in xmgrace format to file <*f*>. This option must be used in combination with **--lfi, --gfi, --lfc** or **--gfc.**

--xfile <*f*> Save a plot of 'xhmm' versus CM E value cutoff in xmgrace format to file <*f*> 'xhmm' is the ratio of the number of dynamic programming calculations

predicted to be required for the HMM filter and the CM search of the filter survivors versus the number of dynamic programming calculations for the filter alone. So, an 'xhmm' value of 2.0 means the filter stage of a search requires the same number of calculations as the CM search of the filter survivors does. This option must be used in combination with **--lfi, --gfi, --lfc** or **--gfc.**

--afile <*f*> Save a plot of the predicted acceleration for an HMM filtered search versus CM E value cutoff in xmgrace format to file <*f*>. This option must be used in combination with **--lfi, --gfi, --lfc** or **--gfc.**

--bits With **--efile, --sfile, --xfile,** and **--afile** use CM bit score cutoffs instead of CM E value cutoffs for the x-axis values of the plot.

Index of frequently asked questions

References

Brown, J. W. (1999). The ribonuclease P database. *Nucl. Acids Res.*, 27:314.

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

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

Eddy, S. R. (2002). A memory-efficient dynamic programming algorithm for optimal alignment of a sequence to an RNA secondary structure. *BMC Bioinformatics*, 3:18.

Eddy, S. R. and Durbin, R. (1994). RNA sequence analysis using covariance models. *Nucl. Acids Res.*, 22:2079–2088.

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

Giegerich, R. (2000). Explaining and controlling ambiguity in dynamic programming. In Giancarlo, R. and Sankoff, D., editors, *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, number 1848, pages 46–59, Montréal, Canada. Springer-Verlag, Berlin.

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

Karplus, K., Barrett, C., and Hughey, R. (1998). Hidden Markov models for detecting remote protein homologies. *Bioinformatics*, 14:846–856.

Klein, R. J. and Eddy, S. R. (2003). RSEARCH: finding homologs of single structured RNA sequences. *BMC Bioinformatics*, 4:44.

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

Nawrocki, E. P. and Eddy, S. R. (2007). Query-dependent banding (QDB) for faster RNA similarity searches. *PLoS Comput. Biol.*, 3:e56.

Weinberg, Z. and Ruzzo, W. L. (2004a). Exploiting conserved structure for faster annotation of non-coding RNAs without loss of accuracy. *Bioinformatics*, 20 Suppl. 1:I334–I341.

Weinberg, Z. and Ruzzo, W. L. (2004b). Faster genome annotation of non-coding RNA families without loss of accuracy. *RECOMB '04*, pages 243–251.

Weinberg, Z. and Ruzzo, W. L. (2006). Sequence-based heuristics for faster annotation of non-coding RNA families. *Bioinformatics*, 22:35–39.