

SAM

Sequence Alignment and Modeling
Software System

Richard Hughey Kevin Karplus Anders Krogh

sam-info@cse.ucsc.edu
Baskin Center for Computer Engineering and Science
University of California
Santa Cruz, CA 95064
Technical Report UCSC-CRL-99-11
Updated for SAM Version 3.4
July 31, 2003

Contributors:
Mark Diekhans, Leslie Grate
Christian Barrett, Michael Brown, Jonathan Casper, Melissa Cline,
Terry Figel, Rachel Karchin, Kimmen Sjölander, Christopher Tarnas
supersedes UCSC-CRL-95-07 and UCSC-CRL-96-22

Abstract

The Sequence Alignment and Modeling system (SAM) is a collection of flexible software tools for creating, refining, and using linear hidden Markov models for biological sequence analysis. The model states can be viewed as representing the sequence of columns in a multiple sequence alignment, with provisions for arbitrary position-dependent insertions and deletions in each sequence. The models are trained on a family of protein or nucleic acid sequences using an expectation-maximization algorithm and a variety of algorithmic heuristics. A trained model can then be used to both generate multiple alignments and search databases for new members of the family.

SAM includes programs and scripts for the SAM-T2K method of remote homology detection. SAM-T2K is an iterative HMM search method for creating an HMM from a single protein sequence or seed alignment using iterative search of a protein database. The method is currently the most sensitive purely-sequence-based remote homology detection algorithm. SAM-T2K is based on successful methods created for the CASP2 and CASP3 protein structure prediction experiments.

The algorithms and methods used by SAM have been described in several pioneering papers from the University of California, Santa Cruz. These papers, as well as the SAM software suite, several servers, and links to related sites are available on the World-Wide Web to

<http://www.cse.ucsc.edu/research/compbio/sam.html>

Contents

1	Introduction	7
1.1	Acknowledgments	9
2	New to this version	9
2.1	Version 3.5	9
2.2	Version 3.4	10
2.3	Version 3.3.2	11
2.4	Version 3.3.1	12
2.5	Version 3.2.1	12
2.6	Version 3.2	13
2.7	Version 3.1	14
2.8	Version 3.0	14
3	Quick overview	15
3.1	Building a model	16
3.2	Aligning sequences	18
3.3	Examining models	19
3.4	Scoring sequences	20
3.5	Parameter selection	23
4	SAM-T2K	24
4.1	SAM-T2K for superfamily modeling	25
4.2	Improved verification of homology	27
4.3	Family-level multiple alignments	27
4.4	Modeling non-contiguous domains	28
4.5	Building an HMM from a structural alignment	31

4.6	Improving existing multiple alignments	33
4.7	Creating a multiple alignment from unaligned sequences	33
4.8	Parameters for <code>target2k</code> perl script	34
4.9	The model building scripts	39
4.10	Support scripts	40
4.11	Installing SAM-T2K	43
4.12	SAM-T2K references	44
5	File types	45
6	Parameter specification	47
7	Sequence formats	49
7.1	Alphabets	49
7.1.1	User-defined alphabets	52
7.1.2	User-defined Numeric Alphabets	53
7.1.3	Alphabet Background Files	55
7.2	Sequences	56
7.3	Models as sequences	58
7.4	Training sets, test sets, and databases	58
8	Regularizers and models	59
8.1	Regularizers	59
8.1.1	Regularizer alternatives	60
8.1.2	Transition regularizers with structural information	62
8.2	Initial model	64
8.3	Initial alignment	65
8.4	Model format	65

8.4.1	Model length	68
8.4.2	Special nodes	69
8.4.3	Node labels	70
8.4.4	Codon models	70
8.4.5	Binary models	71
8.5	Free-insertion modules	71
8.6	FIM, insert and match tables	73
9	The buildmodel estimation process	74
9.1	Noise and annealing	74
9.2	Frequency-based Surgery	76
9.2.1	Sequence-based Surgery	77
9.3	Training statistics	77
9.4	Weighted training	80
9.4.1	External weighting	80
9.4.1.1	Multi-subfamily weighting	81
9.4.2	Annealing with Weights	82
9.4.3	Internal weighting of alignments	83
9.4.4	Internal weighting during training	83
9.5	Viterbi training	85
9.6	Global and semi-local training	86
9.7	Building models with constraints	87
9.7.1	Model Node Labels	87
9.7.2	Constraints Definition File	87
9.7.3	Using Constraints	88
9.8	Reducing buildmodel runtime	89

10 Related programs	90
10.1 align2model and prettyalign	90
10.1.1 prettyalign	91
10.1.2 Aligning fragments and SW	92
10.2 hmmscore	100
10.2.1 NLL-NULL scoring	102
10.2.2 Reducing hmmscore runtime	106
10.2.3 Selecting sequences, scores, and alignments	106
10.2.4 Scoring Fragments	107
10.2.5 Selecting multiple domain alignments	111
10.2.6 Multi-track HMM scoring	115
10.2.7 Distributed scoring	116
10.2.8 Single Sequence Smith & Waterman Scoring and Alignment	117
10.2.9 Scoring using the Kestrel parallel processor	118
10.2.10 Calibration and Model libraries	119
10.3 addfims	122
10.4 fragfinder	123
10.5 grabdp	124
10.6 get_fisher_scores	126
10.7 modelfromalign	127
10.8 pathprobs	127
10.9 predict_track	129
10.10 Model manipulation	130
10.10.1 checkmodel	130
10.10.2 drawmodel	130

10.10.3	<code>hmmconvert</code>	131
10.10.4	<code>makelogo</code>	131
10.10.5	<code>modifymodel</code>	133
10.10.6	<code>sam2psi</code> and <code>psi2sam</code>	135
10.10.7	Conversion between SAM and HMMer	135
10.10.7.1	Internal HMM Structure	135
10.10.7.2	Extra Information in HMMer	136
10.10.7.3	Extra Information in SAM	136
10.11	Plotting Programs	137
10.11.1	<code>makehist</code>	137
10.11.2	<code>makeroc</code>	138
10.11.3	<code>makeroc2</code>	139
10.11.4	<code>makeroc3</code>	140
10.12	Sequence manipulation	141
10.12.1	<code>listalphabets</code>	141
10.12.2	<code>checkseq</code>	141
10.12.3	<code>genseq</code>	143
10.12.4	<code>permuteseq</code>	143
10.12.5	<code>randseq</code>	144
10.12.6	<code>splitseq</code>	144
10.12.7	<code>sampleseqs</code>	144
10.12.8	<code>sortseq</code>	145
10.12.9	<code>uniqueseq</code>	146
10.12.10	<code>mk_kestrel_db</code>	147
11	System installation	147

11.1 Environment variables	147
11.2 Runtime statistics	147
11.3 Manipulating large collections of data	148
11.4 Future Features	148
11.5 Prior versions	149
11.5.1 Version 2.2.1	149
11.5.2 Version 2.2	149
11.5.3 Version 2.1.2	149
11.5.4 Version 2.1.1	150
11.5.5 Version 2.1	150
11.5.6 Version 2.0	151
11.5.7 Version 1.4	151
11.5.8 Version 1.3	152
11.5.9 Version 1.2	153
11.5.10 Version 1.1	154
11.5.11 Version 1.0	154
12 Parameter descriptions	154

1 Introduction

The Sequence Alignment and Modeling system (SAM) is a collection of software tools for creating, refining, and using a type of statistical model called a linear hidden Markov model for biological sequence analysis. Linear hidden Markov models only model primary structure (sequence) information; long-range interactions, such as base pairing in RNA, require more complex models such as stochastic context-free grammars, as described by Sakakibara *et. al* (NAR 22(23):5112–5120), also available from the UCSC computational biology WWW site.

The algorithms and methods have been described in several papers, some of which are available on our WWW site,

<http://www.cse.ucsc.edu/research/compbio/sam.html>.

A tutorial on the use of SAM and the iterative SAM-T98 method (the direct predecessor of SAM-T99) is also available at our WWW site,

<http://www.cse.ucsc.edu/research/compbio/ismb99.tutorial.html>.

SAM-T2K is used at the official SCOP Superfamily server at <http://stash.mrc-lmb.cam.ac.uk/SUPERFAMILY>

The primary papers from UCSC (copies of these papers and several others are available from the SAM WWW site) include:

- R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method, *CABIOS*, 12(2):95–107, 1996.
- K. Karplus, C. Barrett, and R. Hughey, Hidden Markov Models for Detecting Remote Protein Homologies, *Bioinformatics*, 14(10):846–856, 1998.
- A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology*, 235:1501–1531, February 1994.
- K. Karplus, R. Karchin, C. Barrett, S. Tu, M. Cline, M. Diekhans, L. Grate, J. Casper, and R. Hughey, “What is the value added by human intervention in protein structure prediction?,” *Proteins: Structure, Function, and Genetics*, 2001.
- R. Wheeler and R. Hughey, Optimizing Reduced Space Sequence Analysis, *Bioinformatics*, 16(12): 1082–1090, 2000.
- K. Karplus, C. Barrett, M. Cline, M. Diekhans, L. Grate, R. Hughey, “Predicting Protein Structure using Only Sequence Information,” *Proteins: Structure, Function, and Genetics*, Supplement 3:121–125, 1999.
- J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia, Sequence Comparisons Using Multiple Sequences Detect Twice as many Remote Homologues as Pairwise Methods, *Journal of Molecular Biology*, 284(4):1201–1210, 1998.

- R. Karchin and R. Hughey. Weighting hidden Markov models for maximum discrimination *Bioinformatics*, 14(9):772–782, 1998.
- C. Tarnas and R. Hughey. Reduced space hidden Markov model training. *Bioinformatics*, 14(5):401–406, 1998.
- K. Karplus, Kimmen Sjölander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm, and C. Sander, “Predicting protein structure using hidden Markov models,” *Proteins: Structure, Function, and Genetics*, Supplement 1, 1997.
- K. Sjolander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I.S. Mian, and D. Haussler. Dirichlet Mixtures: A Method for Improving Detection of Weak but Significant Protein Sequence Homology. *CABIOS* 12(4):327–345, 1996.
- C. Barrett and R. Hughey and K. Karplus. Scoring Hidden Markov Models. *CABIOS* 13(2):191–199, 1997.
- J. A. Grice, R. Hughey, and D. Speck. Reduced space sequence alignment. *CABIOS* 13(1):45–53, 1997.
- D. Haussler, A. Krogh, I. S. Mian, and K. Sjölander. Protein modeling using hidden Markov models: Analysis of globins. In *Proceedings of the Hawaii International Conference on System Sciences*, volume 1, pages 792–802, Los Alamitos, CA, 1993. IEEE Computer Society Press.
- A. Krogh, I. S. Mian, and D. Haussler. A hidden Markov model that finds genes in *E. coli* DNA. *Nucleic Acids Research*, 1994.
- R. Hughey and A. Krogh. SAM: Sequence alignment and modeling software system. Technical Report UCSC-CRL-96-22, University of California, Santa Cruz, CA, September 1996.
- K. Karplus. Regularizers for Estimating Distributions of Amino Acids from Small Samples. Technical Report UCSC-CRL-95-11, University of California, Santa Cruz, CA, 30 March 1995.

We would appreciate references to the first article in all work that cites or uses the SAM system, the second for all work that cites or uses the SAM-T2K method, and the third article in work that cites or uses HMM methods similar to SAM.

Because the software is an active research tool, there are a vast selection of options, many of which have, through experimental study, been set to reasonable defaults.

The SAM software and documentation copyright is held by the Regents of the University of California. A signed license is required to obtain a copy of SAM, downloadable from the SAM WWW site, with no fee for educational research use. If you have suggestions for enhancements, new ways of using SAM, or other comments, please contact us.

SAM incorporates the readseq package by D. G. Gilbert, who allows it to be freely copied and used. The `hmmedit` and `sae` programs use ACEDb by Richard Durbin and Jean Thierry-Mieg. The source code for `hmmedit` and `sae` is available from <ftp://ftp.cse.ucsc.edu/pub/protein/hmmeditsaesrc.tar.Z>.

SAM includes the BLAST matrix library for use with SAM’s Smith and Waterman implementation. This work of the U. S. Government is available at <http://www.ncbi.nlm.nih.gov/BLAST>

To be informed of future releases, please send your e-mail address to sam-info@cse.ucsc.edu for addition to our mailing list. Please also use this address for any questions or comments you may have.

You will also find Sean Eddy's system, HMMER (<http://genome.wustl.edu/eddy/hmm.html>), to be of interest.

Martin Madera and Julian Gough have written a perl converter between SAM and HMMer 2.0 formats. (The SAM programs only work with HMMer 1.7.)

<http://www.mrc-lmb.cam.ac.uk/genomes/julian/convert/convert.html>

1.1 Acknowledgments

We thank I. Saira Mian and Finn Drablos for their important early evaluations of the system. Finally, we thank the entire UCSC Computational Biology Group (now forming the core of a Center for Biomolecular Engineering), led by David Haussler, who got this whole thing started. This work was supported in part by NSF grants CDA-9115268, IRI-9123692, BIR 94-08579, MIP-9423985, DBI-9808007, and EIA-9905322; DOE grants 94-12-048216 and DE-FG03-99ER62849; ONR grant N00014-91-J-1162; NIH grant GM17129; a grant from the Danish Natural Science Research Council; and a gift from Digital Equipment Corporation.

2 New to this version

Information on previous versions follows the program descriptions. See Section 11.5 on page 149.

2.1 Version 3.5

July, 2005.

- Addition of guide sequence labeling (`dbgguide`) to probabilistic sequences read from SAM HMM files. When HMM's are used as database sequences. Probabilistic sequences are presently being tested; users may wish to wait until version 3.6 to make use of this feature. See Section 7.3 on page 58.
- The `randomize` variable, which indicates the number of synthetic sequences to generate from the model as a means of introducing random noise has been reduced from 50 to 5. This parameter was originally set at 50 when very large training sets are used with SAM. Because of SAM's improved performance on small sets, we have reduced the number.
- Numeric alphabets to enable more than 25 characters. See Section 7.1.2 on page 53.
- The `simple_threshold` variable was previously used in the internal iterations of `multdomain`, sometimes reducing the number of hits when reverse null models were used. This has been corrected. See Section 10.2.5 on page 111.

- Codon models. An experimental HMM architecture that enables the mixing of codon and nucleotide states. Not fully tested. See Section 8.4.4 on page 70 and Section 7.1 on page 49.
- The default value of `simple_threshold` has been increased from 0 to 10000. The variable controls when the reverse null model is calculated. At the setting of 0, only sequences with 0 or better simple null model scores are scored with the reverse null model, reducing by half the amount of time required for a database search. However, since the reverse null model is far more sensitive, this could result in missed sequences. We have therefor changed the default to be the most rigorous search, and leave it to the user to decide whether or not to reduce `simple_threshold` to 0. See Section 10.2.2 on page 106.
- The manual includes a new section on reducing `hmmscore` execution time. See Section 10.2.2 on page 106.
- A new type of surgery, *sequence surgery* has been added to `buildmodel`. When a model is being built to match or canonical sequence, a single-sequence a2m alignment of that sequence can be provided to `buildmodel` to guide surgery. See Section 9.2.1 on page 77.
- The default value of `SW` has been changes to 2, meaning that by default local scoring and alignment is performed. See Section 10.1.2 on page 92 and Section 10.2.4 on page 107.
- A new variable `SW_train`, with a default of 2, controls global or local training in `buildmodel`. To prevent model creap, with frequency-based surgery (the default), the initial and final node of a model are never deleted in the surgery procedure when when `SW_train` is set to local (2) or domain (3).
- Background distributions for internal or user-defined alphabets can now be specified from a file with the `alphabackfile` option. See Section 7.1.3 on page 55.
- A sign error in score adjustment of simple null models was corrected, and the default adjustment by log of the sequence length was scaled from 1.0 to 1.5. See Section 10.2.1 on page 102.
- The `uniqueseq` program previously removed duplicate sequences from alignments before performing alignment-based (`percentid`) thinning. The default now is to only perform thinning based on the alignment. If the prior behaviour is desired, set `aligncheckonly` to 0. See Section 10.12.9 on page 146.
- The `verbose` option default has been changed to 0, reducing the number of messages `hmmscore` and `uniqueseq` produce by default. For example, the progress dots `hmmscore` prints are no longer the default.
- Support for a variant NBRF format has been added, in which sequence identifiers are specified with '>' and sequence lines are prefaced with digits specifying the index of the first character.

2.2 Version 3.4

July 2003.

- The default internal protein prior library has been changed from `reccode4.20comp` to our current favorite `recode3.20comp`. See Section 8.1.1 on page 60.

- Posterior-decoded alignment has been made more efficient. Also, the `maxposdecodemem` variable sets the maximum amount of memory to use for the required very large dynamic programming matrix. Sequences that are too long for alignment within this matrix are aligned to the HMM using the Viterbi algorithm.
- Addition of `not_id` for selecting sequences not to be used. `id` and `not_id` are used whenever a database is loaded with `db`. See Section 7.4 on page 58.
- Corrections to PAUP alignment file reading. Performing multitrack length check after removing non-sequence characters.
- New programs `sam2psi` and `psi2sam` for converting to and from PSI-BLAST checkpoint files, losing information about insertion states and transition probabilities. See Section 10.10.6 on page 135.
- The SAM-Target2K model building script `target2k` is included in the distribution, as well as the previous `target99` script. The documentation has largely been updated to discuss `target2k`, though some of the examples are based on `target99` or use obsolete model building scripts. `target99` is no longer supported. See Section 4 on page 24.
- The number of Sam-T99/T2K model building scripts has been reduced. Based on our experiences, we have eliminated all the “fw” scripts, “varh50”, and “fh” scripts. Choices are now reduced to 3 basic model building scripts, respectively for tuning models to find remote potential homologs, close potential homologs, and very close potential homologs. See Section 4.9 on page 39.
- A `buildmodel` error that caused transitions into node 1 of a model to be incorrectly tabulated has been fixed. This error was introduced in July 2002, and also affected the surgery procedure, causing too many new nodes to be added to a model, destroying the effectiveness of surgery.

2.3 Version 3.3.2

December 2002.

- Probabilistic sequences read from SAM HMM file format. See Section 7.3 on page 58.
- Scaled reverse null model scoring.
- Several new local structure alphabets are available. The `listalphabets` program provides detailed information on all internal alphabets and regularizers. See Section 7.1 on page 49 and Section 10.12.1 on page 141.
- Posterior-decoded alignment has been made more efficient. Also, the `maxposdecodemem` variable sets the maximum amount of memory to use for the required very large dynamic programming matrix. Sequences that are too long for alignment within this matrix are aligned to the HMM using the Viterbi algorithm.
- The `fragfinder` program will find short, gap-free sequence fragments that strongly match a model. See Section 10.4 on page 123.
- The `pathprobs` numerical output has changed, and can now trim alignments based on path probabilities. See Cline, Karplus, & Hughey, *Bioinformatics* 18(2):306–314, 2002. See Section 10.8 on page 127.

2.4 Version 3.3.1

December 2001.

- Model libraries. Model libraries may now be specified and scored with `hmmscore`. Each member of a model library is a set of SAM settings, a model, and a model name. RDB files now always provide model names. Model libraries can be calibrated (highly recommended) as an option to `hmmscore`. See Section 10.2.10 on page 119.
- The `genseq` program will generate random sequences based on a regularizer or Dirichlet mixture regularizer. See Section 10.12.3 on page 143.
- The `makelogo` program is a new viewing tool for SAM models. See Section 10.10.4 on page 131.
- The new `get_fisher_scores` outputs Fisher score vectors for input to an external discriminative learning program. See Section 10.6 on page 126.
- Posterior decoded alignment has been found to have a slight edge over Viterbi alignment. However, the current implementation of this algorithm requires a very large dynamic programming matrix that may be beyond the memory limits of the platform. New to this version, sequences that are too long for the posterior decoded alignment calculation will be aligned to the model using the Viterbi algorithm. See Section 10.1 on page 90.
- Complex null models are no longer supported. An error in `hmmscore` that ignored user's null models has been fixed. See Section 10.2 on page 100.
- The ability of `hmmscore` to read and sort score files, undocumented for the last several versions, has been removed.
- The postscript header file used by `drawmodel` is now produced directly by the program. The `SAM_PS` environment variable, which previously indicated the location of the postscript header, has been eliminated. See Section 10.10.2 on page 130.
- For proteins, the `recode4.20comp` prior has been made an internal default so that `buildmodel` will use it even without proper setting of the `PRIOR_PATH` environment variable. See Section 8.1.1 on page 60.

2.5 Version 3.2.1

October 2000

- Null sequences are now properly scored and aligned by `hmmscore` and other programs. The `buildmodel` program automatically removes null sequences from the training set. See Section 7.4 on page 58.
- In the `hmmscore` program, setting `many_files` to 2 causes the score file to be printed to standard output rather than to the normal `.dist` file, setting to 4 causes `.mstat` files to be printed to standard out, and to 6, both are sent to standard out with undefined interleaving. This variable is treated as a binary bit-vector, so setting to, for example, 3, will result in `buildmodel` generating many files and `hmmscore` sending `.dist` output to standard output. See Section 10.2.3 on page 106.

- Martin Madera and Julian Gough have written a perl converter between SAM and HMMer 2.0 formats that can be downloaded from the SAM WWW page or, for the most up-to-date copy, from <http://www.mrc-lmb.cam.ac.uk/genomes/julian/convert/convert.html>.
- Problems with Irix 64 distribution corrected.

2.6 Version 3.2

July 2000

- Secondary structure alphabets based on DSSP labels. See Section 7.1 on page 49.
- Multi-track HMMs. The `hmmscore` and `align2model` programs can now make use of multi-track HMMs and sequences. For example, a set of protein sequences with associated secondary structure sequences can be scored or aligned to a protein model with character emission probabilities calculated according to the protein model and a second secondary structure model (track). See Section 10.2.6 on page 115.
- The `predict_track` program can be used to make consensus predictions from a primary sequence alignment about a secondary structure track. The program is experimental and not yet fully optimized. See Section 10.9 on page 129.
- The `pathprobs` program can be used to generate the posterior probabilities of each character in an alignment given an alignment and a model, in RDB format and (with reduced information) a2m format. Changes to accommodate this format in `prettyalign` mean that a2m files with compressed insertions (as generated by `prettyalign`) will not be read as alignments. In general, `prettyalign` output should not be used as an input format. See Section 10.8 on page 127.
- The SAM-T99 `target99` script now supports NCBI Blast 2 as well as WU-Blast. SAM-T99 has been developed using WU-Blast; results with Blast2 will differ slightly. See Section 4 on page 24.
- An error that caused the `regularizer_file` to be read for a user's null model rather than the `nullmodel_file` has been fixed.
- Previous to this release, `buildmodel` local training incorrectly always performed global alignment to a model with FIMs on both ends. This has been corrected. To approximately duplicate previous work with local training settings, use `sw 3`. See Section 9.6 on page 86.
- Simple null model selection has been improved. If FIM tables are automatically set (i.e., `FIM_method_score` is positive), the null model emission probabilities correspond to that table, whether or not any FIMs are present in the model. If it is negative, the null model is taken from the specified source, and any automatically-added FIMs are taken from that source as well. If it is zero, the null model corresponds to the first FIM present in the model, or to geometric average of the match states. The null model transition probability is set according to the value of `fimtrans`. The null model probabilities are affected by `fimstrength` whether or not there are FIMs in the model. This eliminates the previous inconsistencies that arose from using the first insert node as a null model whether or not it was a FIM. SAM is now quite explicit when insert/FIM tables are changed and when FIMs are added.

- The `viterbi_threshold` can be used with `hmmscore` to prefilter sequences with a Viterbi NLL-NULL calculation before performing the more expensive (and more sensitive) all-paths (EM style) calculation. See Section 10.2.1 on page 102.
- Previously, when E-values were calculated but a sequence did not have a reverse-null-model score due to `simple_theshold`, the E-value was calculated from the simple null model, leading to incomparable E-values. Now, when the reverse score is not calculated, the E-value is reported as the maximum possible E-value (database size). See Section 10.2.1 on page 102.

2.7 Version 3.1

- SAM now includes a dedicated FASTA reader that is far quicker than the more flexible `readseq` package. Sequence I/O, measured by running `checkseq`, has been sped by a factor of 10. Sequence memory use can be reduced by unsetting the `keepannotations` parameter.
- Models can be created to enable `hmmscore` to perform Smith & Waterman alignment and scoring. SAM will calculate E-values using the reverse-sequence null model. See Section 10.2.8 on page 117.
- Default value of `segment_size` has been increased from 100 to 1000 sequences so that ~ 0.5 –1MB of protein sequence data is in memory at any one time. For particularly long sequences, you may wish to reduce.
- Addition of `adpstyle`, the dynamic programming style used for alignments and multiple domain alignments. Posterior-decoded alignment based solely on character emission posteriors is now available. Scoring according to either posterior alignment option can also be performed. See Section 10.1 on page 90.
- Reoptimization of inner loop checkpoint placement. The default setting of `maxmem` has been increased to use up to 20MB of memory for dynamic programming. $\sim 5 - 20\%$ speedup for `buildmodel` and sequence alignment. Posterior-decoded alignment does not yet use reduced space.
- Addition of the `randseq` program, which can be used to randomly select sequences from a database, and `splitseq` which can split a database according to sequence length (particularly useful to filter sequences that can cause posterior-decoded alignment to run out of memory). See Section 10.12.5 on page 144 and Section 10.12.6 on page 144.

2.8 Version 3.0

October 1999

- The SAM-T99 iterative method for remote homology detection. This is the vastly preferred method for building an HMM from a single protein sequence and for weighting sequences when an alignment is available. See Section 4 on page 24 and Section 9.4.3 on page 83.
- Inclusion of the `view_pdoc` program for viewing the posterior decoded alignment. This may assist in checking alternate paths in an alignment. See Section 10.5 on page 124.

- The `uniqueseq` program can now be used on alignments to eliminate sequences that match other sequences in the alignment. See Section 10.12.9 on page 146.
- The `hmmscore` program will calculate E-values for reverse-sequence null model scoring (scores better than $1e-300$ are reported as $1e-300$). Internal Z-scoring has been eliminated. Score data can be output in the RDB format. See Section 10.2 on page 100.
- Also in `hmmscore`, the reverse null model is now the default null model calculation. This doubles runtime over the simple null model, but is more accurate. See Section 10.2 on page 100.
- The default null model (as well as all FIMs) now includes by default a self-loop transition probability equal to the geometric average of the match to match transitions in the HMM (`fimtrans` is 1.0). The way in which insert to insert arcs are set for negative values of `fimtrans` has changed. See Section 8.5 on page 71.
- Regardless of input format, selected sequence output is always in FASTA format. Sequence annotation lines are now preserved in sequence output files (`sel`, `a2m`, and `mult` files), and are truncated to the first 50 characters in `dist` and `mstat` files.
- In `buildmodel`, constrained training is now supported. Specific residues can be constrained to specific model nodes during training. This serves as a method of incorporation prior knowledge about the training sequence, such as structurally similar regions. See Section 9.7 on page 87.
- The `buildmodel seed` parameter has been renamed `randseed` to avoid confusion with the SAM-T99 seed alignment parameter.

3 Quick overview

The Sequence Alignment and Modeling (SAM) suite of programs includes several tools for modeling, aligning, and discriminating related DNA, RNA, and protein sequences. Given a set of related sequences, the system can automatically train and use a linear HMM representing the family. The SAM-T2K method, available in SAM 3.0 and higher, is a more automated method for building and using HMMs. Readers may wish to read the T2K example before the current section. See Section 4 on page 24.

SAM uses a linear hidden Markov model (Figure 1) to represent biological sequences. The model is a linear sequence of *nodes*, each of which includes *match* (square), *insert* (diamond), and *delete* (circle) states. Each match state has a distribution over the appropriate alphabet indicating which characters are most likely. The chain of match states forms a model of the family, or of columns of a multiple alignment. Some sequences may not have characters in specific positions — delete states enable them to skip through a node without ‘using up’ any characters. Other sequences may have extra characters, which are modeled with the insert states. Insertions are thus used when a small number of sequences have positions not found in most other sequences, while delete states are used when a small number of sequences do not have a character in a position found in most other sequences. As with the match states, all transition probabilities (the chance of having a delete or moving to an insertion state) are local, enabling, for example, the system to strongly penalize sequences that delete conserved regions.

The primary programs include:

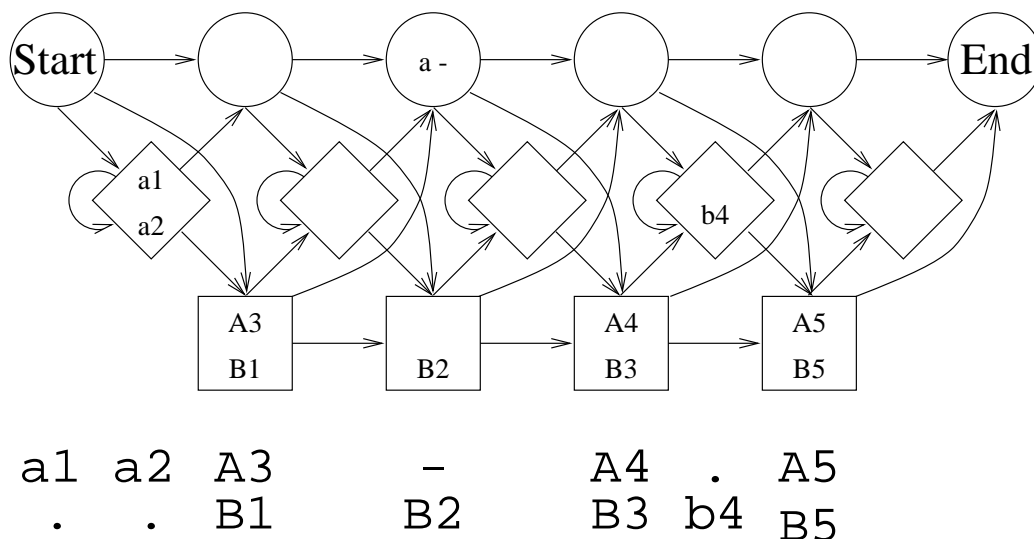


Figure 1: A linear hidden Markov model and example alignment.

buildmodel Create a new model from a family of sequences, or refine an existing model.

align2model Create a multiple alignment of sequences to an existing model. The **prettyalign** program will make **align2model** output more readable.

hmmscore Calculate the negative log-likelihood (NLL) scores for a file of sequences given a model, as well log-odds scores and E-values in the case of reverse null models. This program is used for discrimination experiments. Sequences that score better than (or worse than) a threshold can be saved, as can their alignments or multiple domain alignments.

modelfromalign Use an existing multiple alignment to create an initial model. Such a model is usually then refined using **buildmodel**.

target2k A script that uses SAM to iteratively create a model from a single protein sequence and its close homologues.

A basic flowchart for using SAM is shown in Figure 2.

As a simple example, consider the task of modeling the 10 tRNAs included in the file **trna10.seq** of the distribution. For this experiment, default program settings will be used: the many adjustable parameters are described Sections 6 and 12.

3.1 Building a model

To start, we need to create a model from the sequence file using **buildmodel**. This program always requires a name for the run: if the name is **test**, the system will create the model output file **test.mod**, which will include parameter settings, iteration statistics, and CPU usage, as well as the initial and final model.

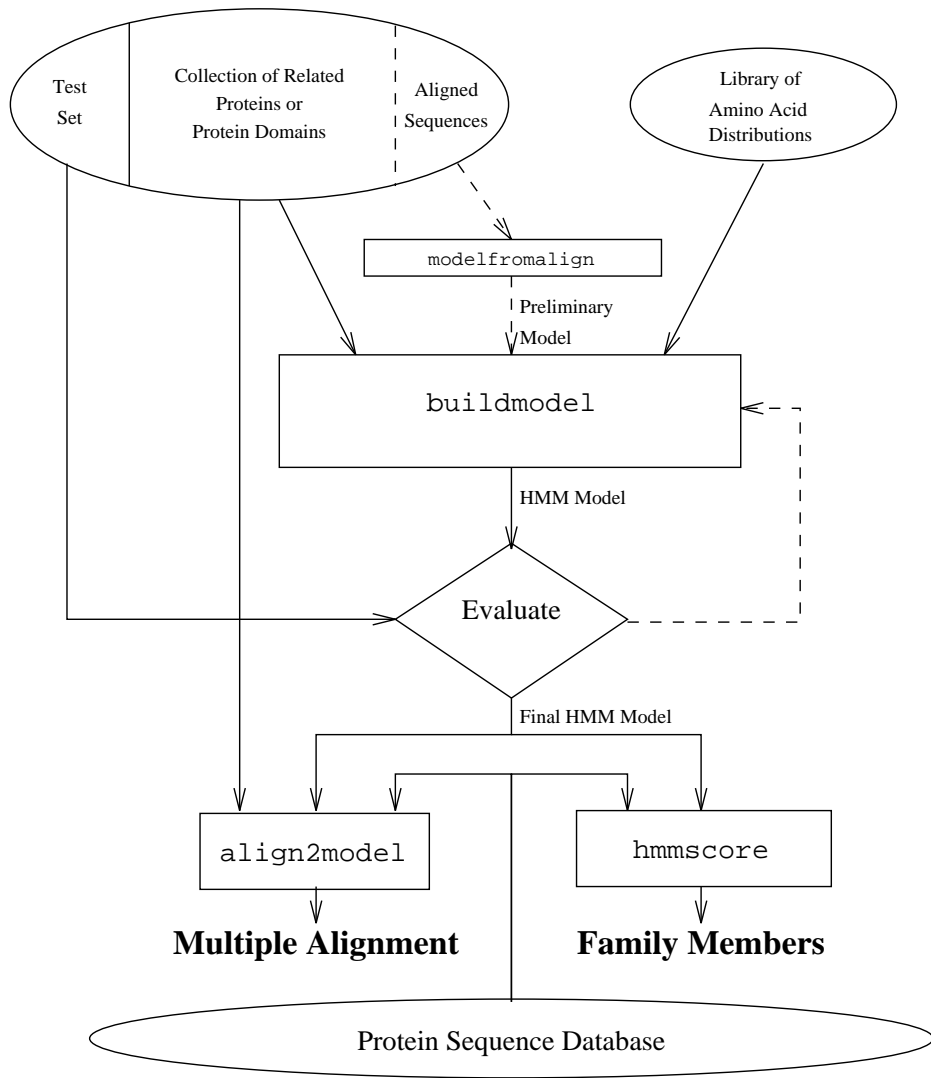


Figure 2: Overview of SAM.

Parameters to `buildmodel` are specified with hyphens. For this experiment, first we try the command:

```
buildmodel test -train trna10.seq -randseed 0
```

This specifies the run name and where the sequences for training can be found. Here, to hopefully make this example reproducible, a seed for the random number generator has also been specified.

`Buildmodel` then prints out a line on standard output such as:

```
-39.71  -36.21  -37.54   1.02  36 2 78
```

This is a brief summary of the statistics provided in the output model file, and is discussed in more detail in Section 11.2. If no random seed had been specified, `buildmodel` would use the process number, and the statistics line would be different for multiple runs.

The run has generated a file called `test.mod`. This file contains various statistics, described later, as well as the final model. Statistics are printed to the file after each re-estimation so that the progress of a run can be readily checked.

3.2 Aligning sequences

To generate a multiple alignment, a command such as the following is used:

```
align2model trna10 -i test.mod -db trna10.seq  
prettyalign trna10.a2m -l90 > trna10.pretty
```

This aligns each sequence to the model, places the alignment in the file `trna10.a2m`, and then cleans up the output and places it in the file `trna10.pretty`, with 90 characters per line. The alignment will look something like:

```

; SAM: prettyalign v3.5 (July 15, 2005) compiled 07/15/05_11:25:33
; (c) 1992-2001 Regents of the University of California, Santa Cruz
;
;           Sequence Alignment and Modeling Software System
;           http://www.cse.ucsc.edu/research/compbio/sam.html
;
; ----- Citations (SAM, SAM-T2K, HMMs) -----
; R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
;   Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
; K. Karplus, et al., What is the value added by human intervention in protein
;   structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
; A. Krogh et al., Hidden Markov models in computational biology:
;   Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
; -----
;
;           10           20           30           40           50           60
;           |           |           |           |           |           |
TRNA1  gg.....GGAUGUAGCUCAG-.UGG...U.AGAGCGCAUGCUUCG.CAUGUAU.G.A.GGcCCCgGGUUCGAUCCCCGGC
TRNA2  gc.....GGCCGUCGUCUAGU.CUGgauU.AGGAGCGCCUGGCCUCC.CAAGCCA.GcA.AU.CCCGGGUUCGAAUCCCGGC
TRNA3  ggcccugugg-----CUAGC.UGG..UcAAAGCGCCUGUCUAG.UAAACAG.G.AgAU.CCUGGGUUCGAAUCCCGGC
TRNA4  gggcga.....----AUAGUGUCAG.CGG...G.AGCACACCAGACUUG.CAAUCUG.G.U.AG.GGAGGGUUCGAGUCCUCU
TRNA5  gccggg.....----AUAGCUCAGU.UGG...U.AGAGCAGAGGACUGA.AAAUCCUcG.U.GU.CACCAGUUCAAAUCUGGUU
TRNA6  gg.....GGCCUAGCUCAGC.UGG...G.AGAGCGCCUGCUUUG.CACGCAG.G.A.GG.UCAGCGGUCGA-CCCUCUA
TRNA7  gg.....GCACAUGGCGCAGU.UGG...U.ACGCGCUUCCCUUG.CAAGGAA.G.AgGU.CAUCGGUUCGAUUCGGUU
TRNA8  gg.....GCCGUGGCCUAGU.CUGga.U.ACGGCACCGGCCUUC.UAAGCCG.GgG.AU.CGGGGGUUCAAAUCCCUCC
TRNA9  cg.....GCACGUAGCGCAGCcUGG..U.ACGGCACCGUCCUGGgGUUGCGG.G.G.GU.CGGAGGUUCAAAUCCUCUC
TRNA10 uccgu.....---CGUAGUCUAGG.UGGu..U.AGGAUACUCGGCUUU.CACCCGA.G.A.GA.CCCGGGUUCAAGUCCCGGC
;
;           70
;           |
TRNA1  AUCUCCA-----.....
TRNA2  GGCCGCA-----.....
TRNA3  GGGCCUCCA-----.....
TRNA4  UUGUCCACCA----.....
TRNA5  -----ccuggca
TRNA6  GGCUCCACCA----.....
TRNA7  GCGUCCA-----.....
TRNA8  GGGUCC-----g.....
TRNA9  GUGCCGACCA----.....
TRNA10 GACGGAACCA----.....

```

Here, hyphens indicate deletes while lower-case letters, and the corresponding periods (‘.’) indicate inserts. The column numbers refer to match states in the model, not to column numbers, so that insertions are disregarded in calculating these index points. It is important to remember that insertions are not aligned among them selves: the fact that two insertion characters are in the same printed column only means that they were generated by the same insertion state, not that they should be aligned.

3.3 Examining models

Model structure can be quite interesting. SAM includes two programs for examining models: `makeLogo` and `drawmodel`

The first, `makeLogo`, shows the negative entropy in bits of each match state and the relative frequen-

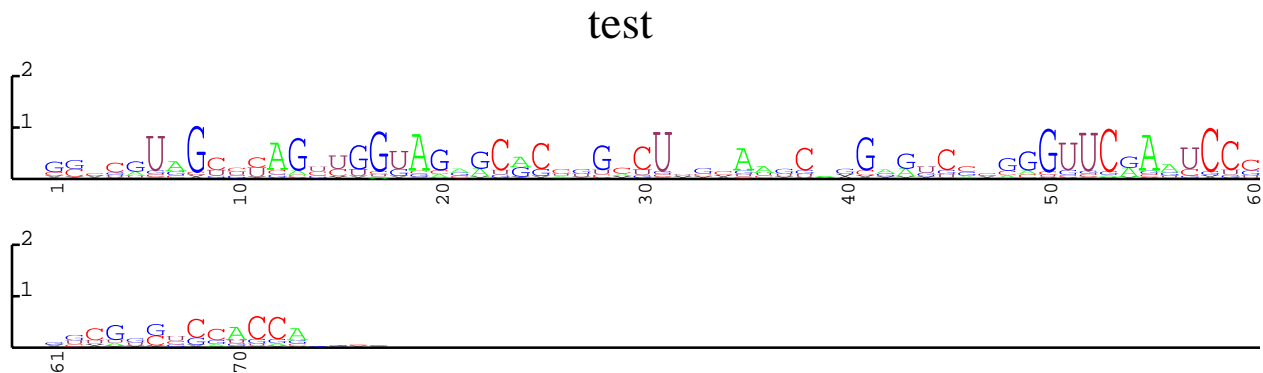


Figure 3: Output of `makelogo`. Bar height corresponds to information in bits, relative character heights correspond to relative frequencies.

cies of the different letters in a sequence logo format, invented by Tom Schneider and Mike Stephens (NAR 18:6097–6100, 1990) <http://www-lmmb.ncifcrf.gov/toms/index.html>. The command:

```
makelogo test -modelfile test.mod
```

creates the `test.eps` file of Figure 3. The program has many options. See Section 10.10.4 on page 131.

The second, `drawmodel`, program generates postscript drawings of models that include match-state histograms and transition line styles that correspond to their frequency of use. These drawings are most useful when derived from frequency counts, values that can be optionally included in the output file:

```
buildmodel test -train trna10.seq -randseed 0 -print_frequencies 1
```

The command `drawmodel test.mod test.ps` will run the `drawmodel` program, which scans the file finding a model and frequency count data. By selecting the frequency count data in ‘overall’ mode, the postscript drawing in Figure 4 is generated.¹ The histograms in the match states correspond to the columns of the multiple alignment above, the numbers in the diamond insert states correspond to the average number of insertions for each sequence that uses that state, and the node numbers are given in the circular delete states. Transitions that are not used by a significant number of the sequences are not drawn. See Section 10.10.2 on page 130.

3.4 Scoring sequences

The scoring program, `hmm_score`, generates a file of the negative log-likelihood minus NULL model (NLL-NULL, or log-odds) scores for each sequence given the model. Let’s see how the 10 tRNA

¹Unfortunately, `drawmodel` does not generate postscript bounding boxes. For insertion in this L^AT_EX file using the `psfig` macros, `ghostview` was used to determine the bounding box.

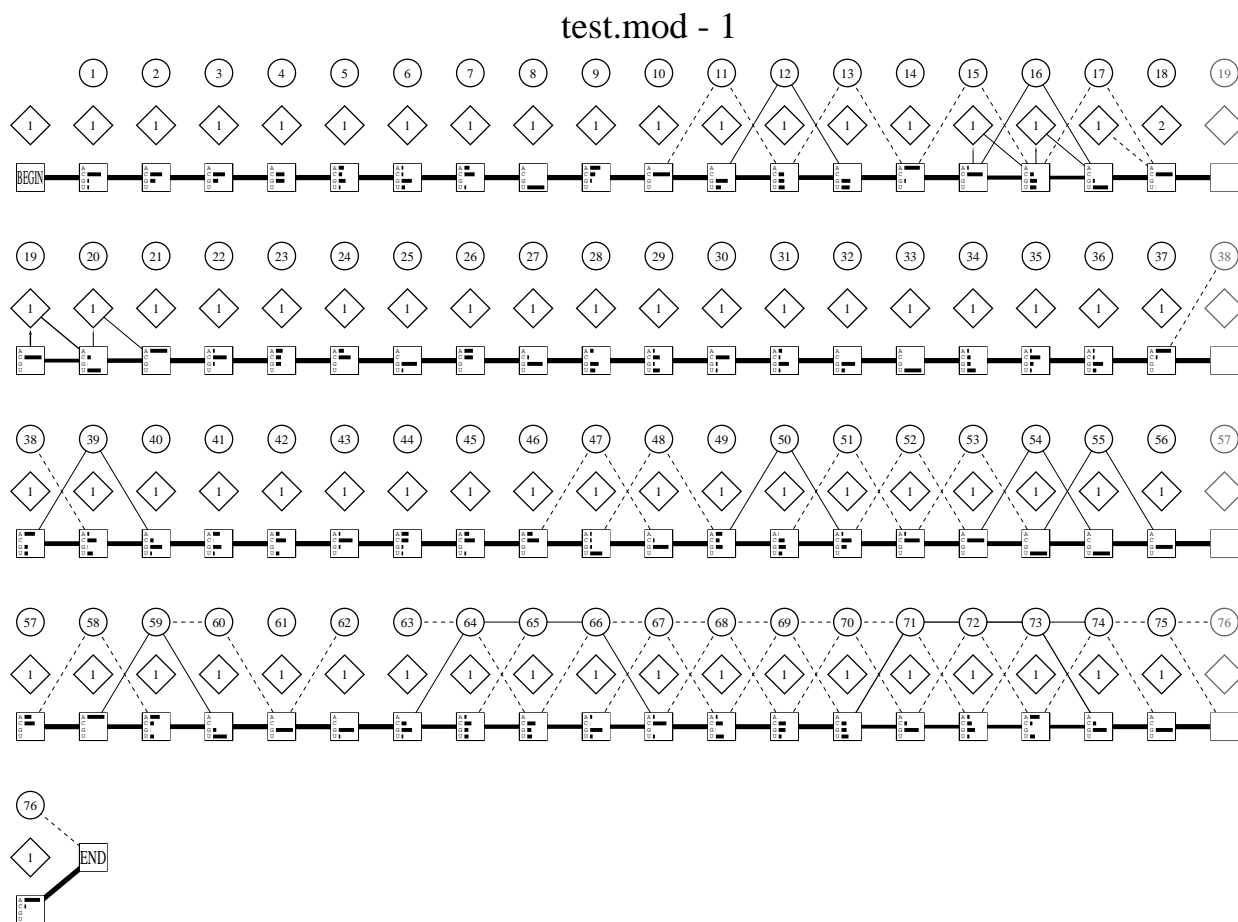


Figure 4: Output of `drawmodel`. Match state histograms are in AGCU order.

sequences fit the model (`test.mod`):

```
hmmscore test -i test.mod -db trna10.seq -sw 2
```

The arguments are the name of the run, the model file, the database sequence file, and a specifier to use fully-local scoring similar to that of the Smith & Waterman algorithm (this is suggested for all scoring runs unless semi-local, domain, or global scoring is specifically required).

`hmmscore` produces the `test.dist` file of Figure 5.

The score file contains six columns. The first is the sequence identifier, followed by sequence length, the 'NLL-NULL' score using a simple null model, the reverse sequence 'NLL-NULL' and an E-value based on the size of the database and the reverse sequence score. The simple null model is just the product of the character probabilities in each sequence multiplied together. Thus, the NLL-NULL score show how much improvement modeling with an HMM gained versus modeling with just a single insertion state from the HMM. The reverse sequence null model is more expensive to calculate, but

```

% SAM: hmmscore v3.5 (July 15, 2005) compiled 07/15/05_11:25:31
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%       Sequence Alignment and Modeling Software System
%       http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% test  Host: peep    Fri Jul 15 13:45:00 2005
% rph   Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Inserted Files:  test.mod
% Database Files:  /projects/kestrel/rph/sam32/demos/trna10.seq
%
% Subsequence-submodel (local) (SW = 2)
% Simple scores adjusted by +1.5*ln(seq len) (adjust_score = 2)
% Track 0 FIMs added (geometric mean of match probabilities (6))
% Single Track Model:  test.mod
% Score DP Method: forward all-paths (dpstyle = 0)
% Align DP Method: viterbi (adpstyle = 1)
% 10 sequences, 747 residues, 78 nodes, 0.01 seconds
%
% Sequence scores selected:  All (select_score=8)
%
% Simple: NLL=NULL using FIM probabilities
% Reverse: NLL=NULL for the reverse sequence NULL model
%   Calculated when Simple < simple_threshold (10000.00)
% E-value on N (=10) sequences:
%   N / (1 + exp(-(lambda(=1.0000) * Reverse)**tau(=1.0000)))
%   Calculated when Simple < simple_threshold (10000.00)
%   Rescale E-values or use -dbsize for multiple scoring runs.
%   WARNING:  E-VALUES ARE NOT CALIBRATED!
% Scores sorted by E-value, best first
%
% Sequence ID   Length   Simple   Reverse   E-value   X count
TRNA7          73      -36.19   -28.50   4.18e-12
TRNA2          76      -35.48   -28.13   6.08e-12
TRNA4          75      -35.66   -27.90   7.65e-12
TRNA9          77      -35.54   -27.82   8.29e-12
TRNA10         76      -35.21   -27.69   9.39e-12
TRNA1          72      -34.85   -27.32   1.37e-11
TRNA8          75      -36.32   -27.26   1.44e-11
TRNA5          73      -35.10   -26.36   3.58e-11
TRNA3          76      -34.43   -26.26   3.93e-11
TRNA6          74      -34.59   -23.81   4.56e-10

```

Figure 5: Distance output from hmmscore

a much better indicator of sequence similarity to the model. The reverse sequence NLL-NULL score is the difference between the raw negative log-likelihood (NLL) sequence score from that of the raw NLL score of the reversed sequence. Thus, there for sequence null model matches exactly the length and composition of the sequence, but does not match the ordering of the residues.

Other options of `hmmscore` enable the selective output of sequences according to their NLL scores, NLL per base scores, or E-value. The `hmmscore` program enters an interactive mode when called with no command line arguments. See Section 10.2 on page 100.

3.5 Parameter selection

As you will see, the SAM system, being an active research tool, has a vast number of parameters. This section briefly mentions a few of the parameters and ways we have found to make SAM work reasonably well.

At UCSC, our primary use of SAM is for modelling proteins. In this case, the SAM-T2K method (described next) is the method to use for creating a multiple alignment of database sequences, which can then be turned into an HMM using the, for example, `w0.5` script.

We have had less experience with DNA and RNA uses of SAM, though we have had many reports of excellent results in these fields. The SAM DNA alignment page uses the following parameters to create an HMM from a file of aligned sequences:

```
buildmodel run -train train.a2m -modellength 0 -alphabet DNA -alignfile train.a2m
-aweight_method 2 -aweight_bits 0.3 -aweight_exponent 0.5
```

For the final alignment the command is:

```
hmmscore run -i run.mod -dpstyle 0 -adpstyle 5 -sw 2 -select_align 8 -db
train.a2m -db database.seq -alphabet DNA
```

A few of the guiding principles in these two commands are:

- If you are building a model for one or more protein sequences, use SAM-T2K.
- Use global alignment (`sw 0`, the default) for building models, especially if the sequences are trimmed to the region you wish to model, and nearly always use fully local (`sw 2`, must be set) scoring.
- For scoring and building models, the forward algorithm, or forward-backward, works best (`dpstyle 0`, the default).
- For alignment, posterior-decoded alignment is more costly but performs better (`adpstyle 5`).
- For sequence weighting, alignment-based weighting works well. If you do not have an alignment, use `buildmodel` or `target2k` to create an alignment and then weight that as above.
- Always use the default reverse-sequence null model scoring.
- If you would like to vary parameters beyond these guidelines for your specific domain, be sure to perform extensive tests within your domain.

4 SAM-T2K

The SAM-T99 method is an iterative HMM method. It is a refinement of the methods used at UCSC for the 1996 CASP2 protein structure prediction contest:

K. Karplus, K. Sjölander, C. Barrett, M. Cline, D. Haussler, R. Hughey, L. Holm, and C. Sander, “Predicting Protein Structure Using Hidden Markov Models,” in *Proteins: Structure, Function, and Genetics*, Supplement 1:134–139, 1997.

as well as the 2000 CASP4 and 1998 CASP3 contests:

K. Karplus, R. Karchin, C. Barrett, S. Tu, M. Cline, M. Diekhans, L. Grate, J. Casper, and R. Hughey, “What is the value added by human intervention in protein structure prediction?,” invited for *Proteins: Structure, Function, and Genetics*, 2001.

K. Karplus, C. Barrett, M. Cline, M. Diekhans, L. Grate, R. Hughey, “Predicting Protein Structure using Only Sequence Information” *Proteins: Structure, Function, and Genetics*, Supplement 3:121–125, 1999.

The CASP3 SAM-T98 method is described in detail in the following article, which should be cited along with the SAM paper whenever the SAM-T2K is used:

K. Karplus, C. Barrett, and R. Hughey, “Hidden Markov Models for Detecting Remote Protein Homologies,” *Bioinformatics*, 14(10):846–856, 1998.

SAM-T98 is shown to be better for remote homology detection than BLAST, FASTA, ISS, and PSI-BLAST in

J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia, “Sequence Comparisons Using Multiple Sequences Detect Twice as many Remote Homologues as Pairwise Methods,” *Journal of Molecular Biology*, 284(4):1201–1210, 1998.

The performance of SAM-T2K, SAM-T98, Double-BLAST (finding a set of close BLAST hits to a sequence and using them for a second BLAST search), and Smith and Waterman with a reverse-sequence null model are shown in Figure 6.

This section illustrates the use of the SAM 3.0 programs and scripts to perform a remote homology search.

The method has several options for handling somewhat different tasks. The next sections will describe a few such tasks:

- Building a multiple alignment and HMM for a superfamily, starting with a single sequence. See Section 4.1 on page 25.

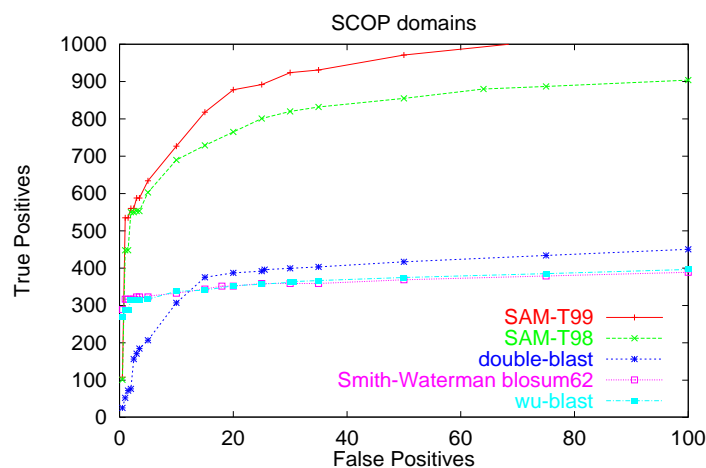


Figure 6: Performance of SAM-T2K and other methods on SCOP super-family discrimination.

- Building a family-level multiple alignment (rather than a superfamily multiple alignment). See Section 4.3 on page 27.
- Modeling a non-contiguous domain. See Section 4.4 on page 28.
- Building a model from a structural alignment. See Section 4.5 on page 31.
- Improving an existing multiple alignment (perhaps from some other tool, such as ClustalW [13] or MaxHom [11]). See Section 4.6 on page 33.
- Creating a multiple alignment from a set of unaligned sequences. See Section 4.7 on page 33.

4.1 SAM-T2K for superfamily modeling

The SAM-T2K method has been encapsulated in a single perl script: `target2k`. This script accepts a single protein sequence as input (the *target* sequence), does a search of a non-redundant protein database, and returns a multiple alignment of sequences similar to the target. The default parameters have been adjusted to give good performance at recognizing sequences related at the superfamily level of the SCOP database [7].

For example, let's start with a single hemoglobin (say PDB sequence 1babA), and try to find as many other globins as we can. The initial protein sequence should be in FASTA format, and all uppercase, since it is used by `modelfromalign` to build an HMM. The 1babA sequence is in the demos subdirectory as `1babA.seq`.

The command

```
target2k -seed 1babA.seq -out 1babA-t2k
```

will search the non-redundant protein database (see Section 4.11) and return a multiple alignment of protein sequences similar to 1babA. The seed sequence is provided first, followed by the similar

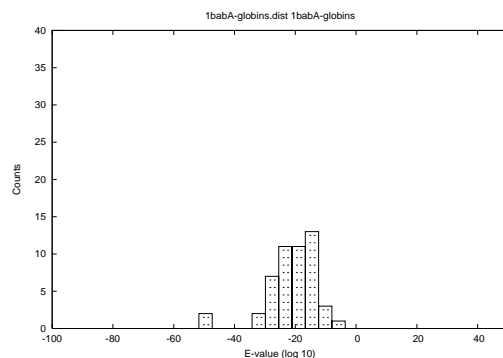


Figure 7: Globin score histogram for SAM-T2K model trained on 1babA.

sequences sorted in order of how well they fit the HMM implied by the multiple alignment. The best-scoring sequence is quite frequently not the sequence that was used as the seed.

Since `target2k` produces voluminous output to the standard error output (so you can see the progress of the run), you may want to redirect the standard-error output to a file (the syntax for this depends on which UNIX shell you run). Note that the multiple alignment, with over 1250 sequences, contains not only hemoglobins, but also myoglobins and leghemoglobins.

You may want to create an HMM from the multiple alignment, in order to search another database (for example, the PDB database), or to score sequences that were selected by some other method. Although you can use `modelfromalign` directly to do the HMM construction, we have provided some simple scripts that set the parameters appropriately. The `build-weighted-model` script is a simple wrapper for `uniqueseq` and `modelfromalign`, and scripts like `w0.5` and `w0.5` set the parameters of `build-weighted-model`.

To build a model for searching for globins, try the following command:

```
w0.5 1babA-t2k.a2m 1babA-t2k-w0.5.mod
```

This model can then be used for scoring a set of sequences:

```
hmmscore 1babA-globins -i 1babA-t2k-w0.5.mod -db globins50.seq -sw 2
```

producing `1babA-globins.dist`. Note that all the globins in this small database score extremely well (Figure 7) with this model (E-values $1.6e-11$ or smaller, $NLL\text{-reverse} \leq -28.79$)—compare with the results presented in Section 8.1.1 on page 60 for models built from a few globin sequences.

In a 1995 test set of 12,219 sequences based on the PIR database [10], the `1babA-t2k-w0.5` model scored all 505 globins with E-values ≤ 0.0024 (and $\leq 9.2e-12$, if you exclude the 45-residue fragment GGNW3B) and all non-globins with E-values ≥ 25 providing a very clean separation of globins from non-globins. Note that this HMM does much better than prior reports of HMMs constructed automatically for globin recognition, which did not achieve perfect separation even when given 400 or 10 known globins [3, 6]. Warning: globins are a fairly easy protein domain to recognize, and this clean a separation cannot be expected for more difficult domains.

On the SCOP pdb90d domain database (version 1.37 with 2466 sequences), the model scored all globins with E-values $\leq 4.0\text{e-}07$, and non-globins with E-values ≥ 0.82 . The phycocyanins, which are in the same superfamily as globins, but a different family, had E-values ranging from 630 to 2000, and were emphatically not recognized by the HMM. Thus the model generated from 1babA should be regarded as a family model, and not as a superfamily model.

One other important warning: the E-value reported is an estimate of how many sequences would score that well with the final model in a random database. A very low E-value means that the sequence or a very similar one was included in the training set from which the model was built—it does not necessarily mean that the sequence is very similar to the seed sequence. With any iterated search method (SAM-T98, SAM-T2K, PSIBLAST [1], ISS [9], ...) including a false positive on any iteration results in much too strong a score for that sequence and similar sequences on subsequent iterations.

Since the default thresholds for SAM-T2K include sequences with E-values as large as 0.01 (measured in the non-redundant database) for the last iteration, E-values less than

$$0.01 \frac{\text{size}(\text{searched database})}{\text{size}(\text{nonredundant protein database})}$$

are roughly equivalent. For the PIR test set, this threshold is about $0.01 \frac{12,216}{415,133} = 0.0004$ and for the pdb90d, the threshold is about $0.01 \frac{2466}{415,133} = 0.00006$. Since all the globins score much better than that, about all we can say is that they fit the model—there are no sequences being found that are very different from the ones used in creating the HMM.

On a test of 935 SCOP domains (the PDB40D Jong Families [8]), the E-values can be seen to be somewhat conservative for values larger than 0.1, but the false-positive rate remains above 0.01 even for very small E-values (see Figure 8). This “fat-tail” phenomenon occurs not only for SAM-T2K, but for almost all homology-detection methods—there are some statistically significant sequence similarities that do not cause proteins to fold the same way. The strongest “false positive” in this test set is actually a mislabeling of one of the domains, fixed in more recent releases of the SCOP domains.

4.2 Improved verification of homology

If we have a conjectured homology between two sequences, perhaps as a result of a search using a SAM-T2K model, we can improve our confidence in the result by combining the results from two different `.dist` files. For each sequence, we build a multiple alignment using `target2k`, then create an HMM from the alignment and use it to score the other sequence. The two “Reverse” scores can be averaged and converted to an E-value (See Section 10.2 on page 100.). Figures 9 and 10 show the improvement one gets from averaging scores in this way.

4.3 Family-level multiple alignments

The default parameters for the `target2k` script have been set for fairly good performance on recognizing superfamilies. If you want to separate one family from another (as Pfam does [12]), this level of generalization is usually too extreme.

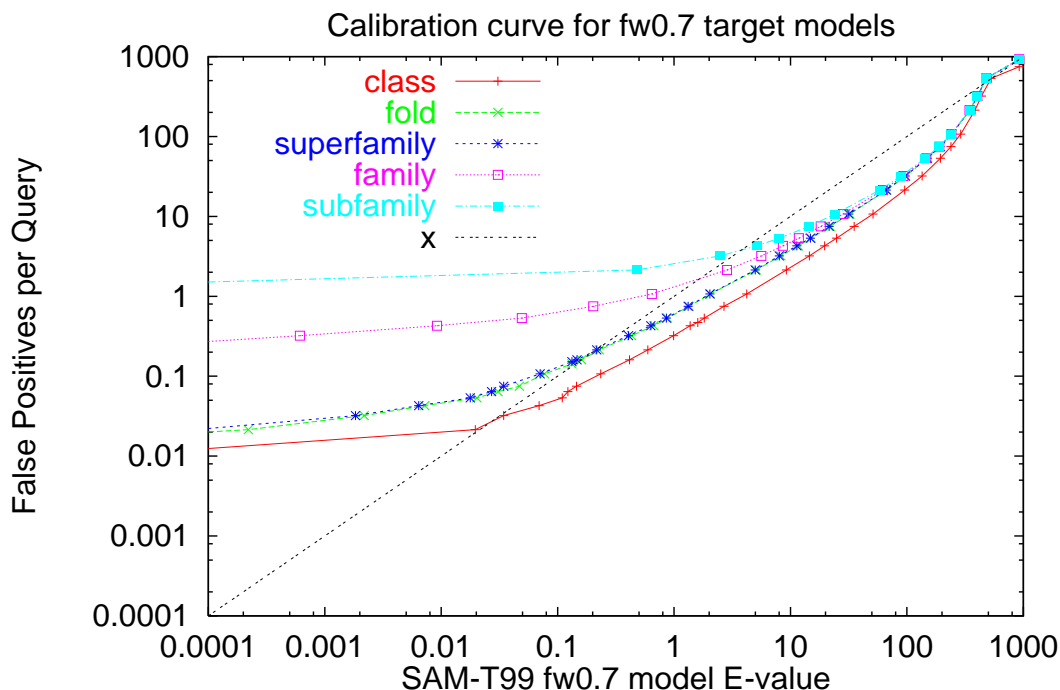


Figure 8: This graph compares the theoretically expected number of false positives per query (the E-value), with the observed number of false positives per query in an all-vs-all test of 935 SCOP domains. The different curves reflect the different levels of similarity one can get from the SCOP hierarchy. The multiple alignments were built with the default values of the `target99` script, and the HMMs were built using `fw0.7`, and scored with `hmmscore -sw 2 -fimtrans 1`.

By specifying the option `-family`, you can request a different set of parameters intended for building family-level multiple alignments from a seed sequence. These parameters have not been tested yet, and are offered only as a starting point for further tweaking.

The Pfam database creates family-level HMMs from hand-curated seed alignments [12]. The SAM-T2K method can also be started with a seed alignment, rather than a single sequence—simply provide the alignment as an a2m file with `target2k`'s `-seed` option.

The `-seed` option is also useful for superfamily recognition—especially if a structural alignment is available to use as a seed.

4.4 Modeling non-contiguous domains

Some protein domains are formed from non-contiguous pieces of the backbone. For example, the SCOP database [7] has a GroES-like fold for alcohol dehydrogenase (d2ohxa1) consisting of residues 1–174 and 325–374, with a Rossmann-fold domain in the middle.

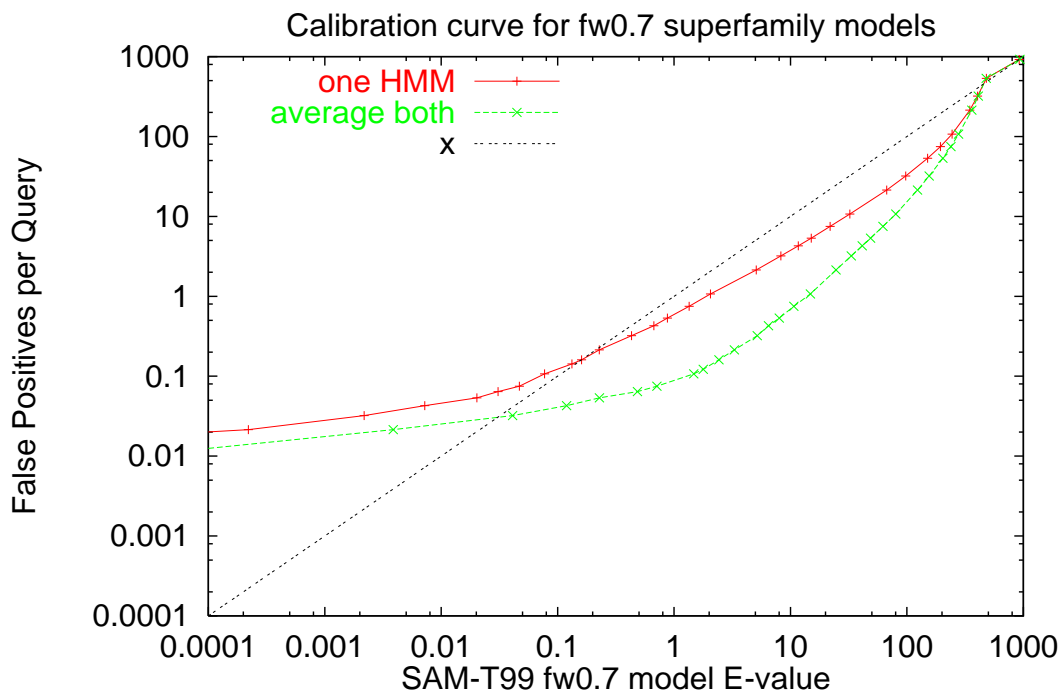


Figure 9: This graph compares the theoretically expected number of false positives per query (the E-value), with the observed number of false positives per query in an all-vs-all test of 935 SCOP domains. False positives are sequences in different folds (in the SCOP hierarchy), true positives are sequences in the same superfamily. Note that the averaging technique makes the E-value estimates considerably more conservative. The multiple alignments were built with the default values of the `target2k` script, and the HMMs were built using `w0.5`, and scored with `hmmscore -sw 2 -fimtrans 1`.

To model a domain like this, the HMM should have a FIM (See Section 8.5 on page 71.) for the long insert. The `target2k` script can handle this—simply replace the amino acids of the inserted domain by a single `O`, which will create a FIM (See Section 10.7 on page 127.).

The demo file `d2ohxa1.seq` has the appropriate seed alignment for the multi-helical domain,

```
>d2ohxa1 2.22.1.2.1 (1-174,325-374) Alcohol dehydrogenase [horse (Equus caballus)]
STAGKVIKCKAAVLWEEKKPFSEIEVEVAPPKAHEVRIKMOVATGICRSDDHVVSGLVTP
LPVIAGHEAAGIVESIGEVTTVRPGDKVIPLFTPQCGKCRVCKHPEGNFCLKNDLSMPR
GTMQDGTSRFTCRGKPIHHFLGTSTFSQYTVVDEISVAKIDAASPLEKVCCLIGC
O
KDSVPKLVADFMKFKALDPLITHVLPFEKINEGFDLLRSGESIRITLTF
```

and the command

```
target2k -seed d2ohxa1.seq -out d2ohxa1-t2k
```

will build the multiple alignment.

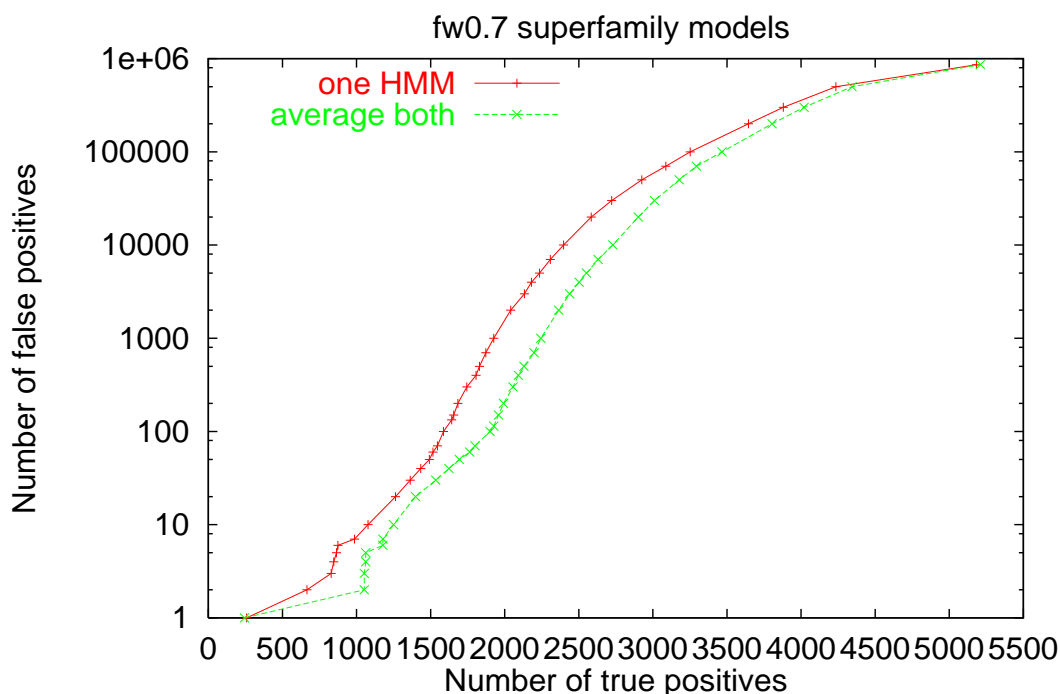


Figure 10: This graph the number of false positives (different folds in the SCOP hierarchy) versus the number of true positives (same superfamily) in an all-vs-all test of 935 SCOP domains. The multiple alignments were built with the default values of the `target99` script, and the HMMs were built using `fw0.7`, and scored with `hmmscore -sw 2 -fimtrans 1`. The averaging method finds considerably more superfamily relationships at every level of false positives accepted.

The FIM will be created in the HMMs that are used for aligning the sequences found by the SAM-T2K method. Furthermore, since the first sequence of the resulting multiple alignment will have an O, the FIM will be created in any HMMs built from the output alignment.

The HMM `d2ohxa1-t2k-w0.5.mod` scores the alcohol dehydrogenase family members (in `pdb90d` version 1.37) with E-values $\leq 6.0e-31$. The best-scoring sequence outside the superfamily is the central domain of the alcohol dehydrogenase with E-value $2.0e-08$, which creeps into the model due to misalignments despite the FIM. The best-scoring completely unrelated domain is `d2tmda3` (Trimethylamine dehydrogenase, middle, ADP-binding domain) with E-value 0.027

The chaperonin-10 sequences in the GroES-like superfamily but not the alcohol dehydrogenase family have E-values 16 and 270, so this HMM is best viewed as a family-level model, not a superfamily model. The E-values are reasonable estimates of the number of false positives (46 and 255) in the `pdb90d` database for this search.

The central domain of `2ohxA` is an NAD(P)-binding Rossmann-fold domain, a large superfamily of alpha/beta/alpha units with a parallel beta-sheet of 6 strands (order 321456). This superfamily is fairly diverse, and difficult to capture in a single HMM. Let's start with `d2ohxa2.seq`:

```
>d2ohxa2 3.19.1.1.1 (175-324) Alcohol dehydrogenase [horse (Equus caballus)]
GFSTGYGSAVKVAKVTQGSTCAVFLGGVGLSVIMGCKAAGAARIIGVDINKDKFAKAKE
VGATECVNPQDYKKPIQEVLTEMSNGGVDFSFVEVIGRLDTMVTALSCCQEAYGVSIVIGV
PPDSQNLSPMPLLLSGRTWKGAIFGGFKS
```

and build the HMM as before

```
target2k -seed d2ohxa2.seq -out d2ohxa2-t2k
w0.5 d2ohxa2-t2k.a2m d2ohxa2-t2k-w0.5.mod
```

Scoring the SCOP pdb90d (version 1.37) database gives us the first false positive at E-value 0.022 (Dihydrolipoamide dehydrogenase) and at the minimum-error point (E-value 1.1), there are 35 true positives, 4 false positives, and 19 true negatives. All seven members of the same family as d2ohxa2 score extremely well (E-values $\leq 9.0e-25$), but there are 20 other members of the superfamily before the first false positive, so this can genuinely be viewed as a superfamily model, and not just a family model.

4.5 Building an HMM from a structural alignment

We can try to make a better model for the Rossmann-fold superfamily containing d2ohxa2, by starting from a structural alignment of two of its members. The model built using just d2ohxa2 had particular difficulty recognizing the lactate dehydrogenases and the malate dehydrogenases, so perhaps we could improve the model by including one of them in the alignment. In the FSSP [5] alignment for 2ohxA, the highest Z-score of these is for 2cmd.

From the FSSP alignment, we can extract the aligned section that seems to match the domain we are interested in:

```
>2ohxA-domain Alcohol dehydrogenase (holo form) complex with nadh and
STCAVFLGGVGLSVIMG
CKAAGAARIIGVDINKDKFAKAKEVGATECVNPQDYKKPIQEVLTEMSNGGVDFSFVEVIGRLDTMVTALS
CCQEAYGVSIVIGVPPDSQNLSPMPLLLSGRTWKGAIFGGFKSKDS
>2cmd-domain Malate dehydrogenase
MKVAVLGAAGGIGQALAL
LLKTQLpsg-SELSLYDIAPVTPGVAVDLshiptavk--IKGFSGE-----DATPAL----EGADVVLIS
AGvrrkpgmdrsdlfnvNAGIVKNLVQQVaktCPKACIGIITNPvnttvaiiaa-----EVLKkaGVYDk
n---KLFVGT-TLDIiRSNT
```

The a2m file can be a bit unreadable, but we can use prettyalign to make sure that the alignment is the one we intend:

```

                10      20      30      40
                |      |      |      |
2ohxA-domain  STCAVFG.LGGVGLSVIMGCKAAG...AARIIGVDINKDKFAKAKEV.....GA
2cmd-domain   MKVAVLGAAGGIGQALALLLKTQLpsg-SELSLYDIAPVTPGVAVDLshiptavk--
                50      60      70      80
                |      |      |      |
2ohxA-domain  TECVNPQDYKKPIQEVLTEMSNGGVDFSFEVIG.....RLDTMVTAL
2cmd-domain   IKGFSGE-----DATPAL----EGADVVLISAGvrrkpgmdrsdlfnvNAGIVKNLV
                90      100     110     120     13
                |      |      |      |
2ohxA-domain  SCC..QEAYGVSIVGVVP.....PDSQNLsmnp..MLLL..SGRTWKGAIFGG
2cmd-domain   QQVakTCPKACIGIITNPvnttvaiaa-----EVLKkaGVYDkn---KLFGVT-TL
                0
                |
2ohxA-domain  FK.SKDS
2cmd-domain   DIiRSNT

```

If we build the HMM as before

```

target2k -seed 2ohxA-2cmd.a2m -out 2ohxA-2cmd-t2k
w0.5 2ohxA-2cmd-t2k.a2m 2ohxA-2cmd-t2k-w0.5.mod

```

and score the SCOP pdb90d (version 1.37) database, we get the first false positive at E-value at $3.2e-04$ and at the minimum-error point (E-value $9e-04$), there are 31 true positives, 1 false positive, and 23 false negatives.

Although this model scores the superfamily well (40 of the 54 members have E-values less than 1.0), it also scores 19 incorrect domains as well—14 of them dihydrodoliipoamide dehydrogenase and related folds. If we selected a threshold for the `d2ohxA2-t2k-w0.5.mod` HMM that accepted 40 of the superfamily members, we would have had 54 false positives, so adding a structurally-aligned homolog has clearly let us create a better model of the superfamily.

If we add one more poorly-found sequence to the structural alignment (say 2pgd), we can improve the model further. Starting from the alignment

```

                10      20      30      40      50
                |      |      |      |      |
2pgd  DIALIG.LAVMQNLILNMNDHG...F.VVCAFNRVTSKVDDFLANEAKGT...KVLGAH..S
2cmd  KVAVLGAAGGIGQALALLLKTQLpsgS.ELSLYDIA-PVTPGVAVD-LSHIptavKIKGFSgeD
2ohxA TCAVFG.LGGVGLSVIMGCKAAG...AARIIGVDINKDKFAKAKEV----ga..-TECVN..P
                60      70      80      90      100
                |      |      |      |      |
2pgd  ....LEEMVSKLKKPR.RIILLVK....AGQAVDNFIEKL...VPLLDIGDIIIDGGNSEY
2cmd  ....ATPALEGA--D.VVLISAGv15nvNAGIVKNLVQQV...AKTCP-KACIGIITNPVN
2ohxA qd7piQEVLTEMSNGGVdFSFEVIG.....----RLDTMVTAlsc---QEAYGVSIVGVPPD
                110     120
                |      |
2pgd  RD....TMRRCRDLK..DKGI..LFVGSgVS
2cmd  TT....VAIAAEVLKkaGVYDknKLFgVTTL
2ohxA SQnlsmn-----PMLL..LSGR..TWKGAIFG

```

and building a SAM-T2K alignment as before, the `2pgd-2cmd-2ohxA-t2k-w0.5.mod` HMM finds 24 true positives before the first false positive, and 19 false positives at 40 true positives. There are still some bad true negatives—the worst is `d1scua2` (Succinyl-CoA synthetase, alpha-chain, N-terminal (CoA-binding) domain) at E-value 570, with 346 incorrect sequences scoring better.

If we have a structural alignment of dissimilar homologs, it may not seem necessary to run the `target2k` method. If we build a model from the structural alignment with the command

```
w0.5 2pgd-2cmd-2ohxA.a2m 2pgd-2cmd-2ohxA-w0.5.mod
```

we can get 23 true positives before any false positives, 26 true positives with only 4 false positives, and 40 true positives with 129 false positives.

For very low false-positive rates, our best results on this domain were with the SAM-T2K method applied to a single sequence, with 27 true positives before a false positive. If we want to find the remote homologs, accepting a few false positives, we get better results by using a structural alignment as the seed (40 true positives with only 19 false positives). The SAM-T2K method does get better results on this example than using just a structural alignment without searching for more homologs.

4.6 Improving existing multiple alignments

You can clean up an existing multiple alignment by using it as a seed and specifying the `-tuneup` option. This option turns off the search for similar sequences and turns off the `-force_seed 1` option that normally copies the seed alignment without modification.

The seed alignment will be used to create an HMM, then the sequences in the alignment will be used as the set of potential homologs to search and align. The output alignment may contain only a subset of the original sequences, if some of the sequences score too poorly to meet the thresholds. If you want to include all the sequences, set the `-thresholds` variable to have a very large final threshold.

The `-tuneup` option can also be used to add unaligned sequences to an existing multiple alignment with the `-homologs` option. For example,

```
target2k -seed 1babA.seq -homologs globins50.seq -tuneup
      -db_size 400000 -out 1babA-50
```

adds 50 globin sequences to the single 1babA globin alignment, creating a multiple alignment of 51 sequences. The `-db_size` option says what size database should be assumed for computing E-values. If it is omitted, the size of the non-redundant database that would normally be searched is used.

4.7 Creating a multiple alignment from unaligned sequences

It isn't really necessary to specify a seed for the SAM-T2K method—if the `-close` or `-homologs` option is used, then an initial multiple alignment can be created by `buildmodel` from the specified unaligned sequences. If both `-close` and `-homologs` are given, then only the close set are used for the initial alignment, but the full set of homologs are used for subsequent iterations.

For example,

```
target2k -homologs globins50.seq -tuneup -db_size 400000 -out globins50-tuned
```

will align the 50 globin sequences without using a seed sequence, leaving the alignment in `globins50-tuned.a2m`.

4.8 Parameters for `target2k` perl script

The parameters of the `target2k` perl script are not described in the parameters section (See Section 6 on page 47.), since they are not parameters to the SAM programs and are not parsed by the same routines. One important difference is that the `target2k` parameters are interpreted in the order they are seen on the command line—if two options conflict, the one that is set later overrides the earlier one.

The `target2k` parameters can be roughly grouped into four classes:

required parameters The `-out` parameter is always required, to specify where the results are to be put, and at least one of `-seed`, `-close`, and `-homologs` (usually `-seed`) must be specified to tell the script what to build a model for. The `-seed`, `-close`, and `-homologs` files may be gzipped.

modes The `-superfamily`, `-family`, and `-tuneup` parameters set many internal variables for three of the most common tasks expected for the script. The `superfamily` settings are the default values if they are not overridden by explicit parameter settings.

major parameters There are many parameters that can be used to override the settings given by the major modes. The most commonly used parameters are probably `-iter`, `-thresholds`, `-db_size`, `-homologs`, `-close`, and `-db`.

minor parameters Many of the other parameters are provided mainly for developers who want to tweak various internal parameters to get better performance.

Here are the individual parameters to `target2k` in alphabetical order:

`-all` Requests that the intermediate multiple alignments created on each iteration be kept as `root_1.a2m`, `root_2.a2m`, This option is very useful for determining when contaminating sequences were introduced into a multiple alignment, so that thresholds can be appropriately modified to exclude them.

`-aweight_bits` 0.5 How many bits/column the HMM should save relative to the background distribution after regularizing a multiple alignment with the Dirichlet mixture specified by `-mixture`.

We have gotten the best results with about 0.8 bits/column for the mixture `recode4.20comp`, and around 0.5 bits/column for `recode3.20comp`. You may want to increase the bits/column for short models or for doing family-level models.

`-aweight_exponent` 0.5 This exponent is interpreted in two different ways, depending on the setting of `-aweight_method`. (See Section 9.4.2 on page 82.) For `-aweight_method 1` (the EntropyWeight method), the exponent should be large (around 10 works well).

For `-aweight_method 2` (the Henikoffs' method[4]), the exponent only affects the initial values, and a value around 0.5 works well.

-aweight_method 2 The **-aweight_method** is used with **-aweight_exponent** and **-aweight_bits** to specify how sequence weighting is done: 1=EntropyWeight, 2=HenikoffWeight, 3=FlatWeight. See Section 9.4.3 on page 83..

-blast_max_report 20000 Maximum number sequences reported by wu-blastp or blast2, passed to the wu-blast-prefilter or ncbi-blast-prefilter script. You can set this parameter smaller to speed up the script, at the cost of losing a fair amount of generality on some models (such as immunoglobulins and zinc fingers), where there are many very similar sequences.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

-close filename File containing close homologs of the seed sequence (may be compressed with gzip). These close homologs are used to train the model on the first training iteration. If this parameter is not specified, but **-homologs** is specified, then the file from the **-homologs** option is used for the close homologs. If neither **-close** nor **-homologs** are specified, then the close homologs and potential homologs are searched for in the non-redundant protein database using blastp, but with a much tighter threshold for the close set than the potential-homolog set.

It is a good idea to set **-db_size** explicitly when no search is being done.

-constraints 1 Should constraints be used? Zero means no constraints; one means use constraints generated from the seed alignment. See Section 9.7 on page 87. Even if the **-force_seed** option is set, and the seed alignment will be copied directly to the multiple alignment on each iteration, the constraints are useful to keep **buildmodel** from modifying the seed alignment while it works.

Not only does **-constraints 1** keep the seed alignment intact, it also propagates **.cst** files through the entire process, so that they can be used with the final multiple alignment.

-cut_match 0.2 This parameter is passed to **buildmodel** to control how surgery is done. Since **-nsurgery** is normally set to 0 to turn surgery off, this parameter is usually irrelevant.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

-cut_insert 0.25 This parameter is passed to **buildmodel** to control how surgery is done. Since **-nsurgery** is normally set to 0 to turn surgery off, this parameter is usually irrelevant.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

-db nr Specifies the filename for the non-redundant database to search and to use for determining **-db_size** for E-value calculations. The database *must* be in FASTA format and have a wu-blastp index. Because it is searched by wu-blastp, it cannot be compressed with gzip.

The script provided has “nr” as a legal value for **-db**, which expands to the filename of the non-redundant protein database specified when the script was installed. It is straightforward to add more such shorthand names for databases (in the **process_command_line** subroutine).

If **-db** is not specified, then the filename that “nr” expands to is used.

-db_size <int> Specifies number of sequences to assume are being searched for E-value calculations. If not specified, then size of database specified in **-db** is assumed.

Note: when using **-tuneup**, **-no_search**, **-homologs**, or **-close**, no search of the database is done, but the count of the default database (or database specified with **-db**) is still done to get the appropriate size for E-value calculations unless **-db_size** is explicitly set.

- family** Sets options appropriately for modeling a family. These options have **not** been carefully tuned, and are just a guess at initial parameters. Individual parameters can be overridden by specifying them later in the command.
- fimstrength** 1.0 Probability multiplier for letters in FIMs. This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.
- fimtrans** -1.0 FIM and standard insert-to-insert transitions. Absolute value is the cost (negative-log probability) of FIM insert-to-insert transition divided by average match-to-match cost. When negative, non-FIM insert-to-insert transitions are set to $p - (1 - f)p^2$, where p is the regularized and normalized frequency counts for the transition and f is the FIM insert-to-insert transition. This parameter controls how much the model prefers to absorb letters into FIMs rather than accepting poor matches at the ends of the matched region. Values around 1.0 and -1.0 generally work best.
- tt -final_adpstyle** 5 Alignment style to use for making final alignment from trained HMM. 1 indicates Viterbi, and 5 indicates the much slower but more accurate posterior-decoded style. Alignment style is controlled by `init_adpstyle`, `search_adpstyle`, `realign_adpstyle`, and `final_adpstyle`.
- final_trans_reg** `fssp-trained.regularizer` Transition regularizer used for the last iteration. See Section 8.1 on page 59.
This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.
- force_seed** 1 Set to 1 (default) to force all of the seed alignment into the training set and the multiple alignment at each iteration.
The `-tuneup` option implies `-force_seed` 0, with the expectation that the seed alignment is to be modified. It is possible to override `-tuneup`'s setting, which might be desirable when adding a new homologs to a seed alignment that isn't to be changed.
- frac_insert** 0.3 This parameter is passed to `buildmodel` to control how surgery is done. Since `-nsurgery` is normally set to 0 to turn surgery off, this parameter is usually irrelevant.
This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.
- full_seq_align** 0 Set to 1 to use full-sequence alignment, not multiple-domain alignment for final alignment. If you want to preserve the original sequence names, this option is essential, since the normal action of multiple-domain alignment modifies the names to include information about which domain was found.
- homologs** *filename* specifies a file containing potential homologs of the seed (may be compressed with `gzip`). When `-homologs` is set, no search of the database is done, as it is assumed that all potential homologs have been identified. If this parameter is not specified, but `-close` is specified, then the file from the `-close` option is used for the potential homologs as well as for the close homologs. If neither `-close` nor `-homologs` are specified, then the close homologs and potential homologs are searched for in the non-redundant protein database using `blastp`, but with a much tighter threshold for the close set than the potential-homolog set.
If the option is not used, the database specified in `-db` (or the default non-redundant database) is searched with `blastp` to find potential homologs.
It is a good idea to set `-db_size` explicitly when no search is being done.

include_all 0 In each call to `select_seq`, include all sequences from the `homologs` set, no matter how bad the scores. Used for the `-tuneup` option.

-init_trans_reg `gap1.5.regularizer` Transition regularizer used for the first iteration. See Section 8.1 on page 59.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

tt -init_adpstyle 1 Initial alignment style. 1 indicates Viterbi, and 5 indicates the much slower but more accurate posterior-decoded style. Alignment style is controlled by `init_adpstyle`, `search_adpstyle`, `realign_adpstyle`, and `final_adpstyle`.

-iter 4 The number of iterations of `hmmscore` and `buildmodel` to perform. Normally, `-iter` is set to match the number of E-value thresholds set—if `-iter` is smaller, the extra thresholds are ignored, and if `-iter` is larger the final threshold is repeated for subsequent iterations.

-jump_in 0.2 Probability associated with jump-in on local alignment.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

-jump_out 1.0 Probability associated with jump-out on local alignment.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

-keep_temporary Don't delete temporary directories. The temporary directories can get fairly large, so they are deleted when the `target2k` script finished normally. When a part of the script fails badly, the temporaries are usually kept to help track down the problem.

When the script runs to completion, but produces a poor result, finding the problem can be difficult, so the `-keep_temporary` option is available to aid debugging. This option is not only useful to script developers, but also to users who want to look “under-the-hood” to see how the script is producing the results it gets.

-mid_trans_reg `stiff-gap5.regularizer` Transition regularizer for all iterations except the first (`-init_trans_reg`) and last (`-final_trans_reg`) iteration. See Section 8.1 on page 59.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

-mixture `recode3.20comp` Dirichlet mixture to use for regularizing amino acid distributions. We have gotten the best results on aligning remote homologs by using `recode3.20comp`. See Section 8.1.1 on page 60.

-ncbiblast Use NCBI BLAST2 for initial prefilter via the `ncbi-blast-prefilter` script. The `-wublast` option specifies use of WU-BLAST, when available. See Section 4.11 on page 43.

-no_search Use only the sequences from `-seed`, `-homologs`, or `-close` as potential homologs—don't look for more.

The `-no_search` option is implied by `-homologs -close`, and `-tuneup`. There is currently no option for turning the search back on.

It is a good idea to set `-db_size` explicitly when no search is being done.

-no_sort Turn off sorting of sequences by score.

-nsurgery 0 Specifies the number of surgery steps to allow on each iteration (default 0). Surgery is always allowed when creating an initial alignment (**-seed** not specified).
 Setting **-nsurgery** positive is not compatible with **-force_seed 1**—try **-force_seed 0** and **-constraints 1** instead.

-out *root* creates *root.a2m* for output (also *root.cst* if constraints are turned on). This is the only absolutely essential parameter.

prefilter_thresholds 0.01, 1.0, 10.0, 400.0 E-value thresholds used with each iteration with the blast prefilter.

tt -realign_adpstyle 1 Alignment style to use for making alignment from trained HMM on intermediate iterations. 1 indicates Viterbi, and 5 indicates the much slower but more accurate posterior-decoded style. Alignment style is controlled by **init_adpstyle**, **search_adpstyle**, **realign_adpstyle**, and **final_adpstyle**.

-reverse_diff 4 How much tighter is the simple threshold than the reversed-sequence threshold, in nats The selection of sequences to include in the multiple alignment at each iteration requires passing four tests:

1. NLL—null score for all-paths local scoring better than some threshold.
2. NLL—reversed E-value for all-paths local scoring better than some threshold.
3. NLL—null score for Viterbi local scoring of found domain better than some threshold.
4. NLL—reversed E-value for Viterbi local scoring of found domain better than some threshold.

The simple tests (steps 1 and 3) require a threshold in nats. This threshold is computed from the E-value threshold used for steps 2 and 4, then tightened by **-reverse_diff**.

Making **-reverse_diff** very small (say -10) essentially turns off steps 1 and 3, slowing down the script but possibly increasing the number of things found (both true and false positives).
 Making **-reverse_diff** very large requires very good matches to the model before more detailed scoring is done, speeding the script up, but possibly losing some true positives.

This is a minor parameter, intended only to allow developers to tweak parameters without modifying the installed script.

tt -search_adpstyle 1 Alignment style to use for selecting training set for retraining HMM. 1 indicates Viterbi, and 5 indicates the much slower but more accurate posterior-decoded style. Alignment style is controlled by **init_adpstyle**, **search_adpstyle**, **realign_adpstyle**, and **final_adpstyle**.

-seed *file.a2m* a guide sequence (or multiple alignment) in a2m format (may be compressed with gzip). See Section 10.1 on page 90. The seed is usually expected, but if none is provided and either **-homologs** or **-close** is specified, then an initial alignment will be built from the homologs (the close ones being preferred if both are specified).

-superfamily Sets many options appropriately for modeling a superfamily given a single sequence. These are the default options. Individual parameters can be overridden by specifying them later in the command.

-thresholds 0.00001, 0.0002, 0.001, 0.005 specifies the E-value thresholds to use for each iteration (E-value is expected number of false positives). The values should be given as a comma-separated list with no spaces. If the number of thresholds is more than the **-iter**

parameter, then the extra ones are ignored. If there are fewer thresholds than iterations, the last threshold is repeated as needed. The E-values are computed for a database the size of the non-redundant database searched (unless `-db_size` is specified).

`tmp_dir` Directory in which to place the temporary directory used to hold intermediate files. The default is set by the configuration program `sam-t2k.conf`.

`-tuneup` Sets many options, under the assumption that what is wanted is an improvement of an exiting alignment, or the alignment of an already selected set of homologs. Some of the important parameters it sets are `-no_search`, `-force_seed 0`, `-full_seq_align`, and the thresholds. Eliminates search, assumes that all individual parameters can be overridden by specifying them later in the command, except that there is no way to turn searching back on.

`-wublast` Use WU-BLAST for initial prefilter via the `wu-blast-prefilter` script. The `-ncbiblast` option specifies use of NCBI BLAST2, when available. See Section 4.11 on page 43.

4.9 The model building scripts

The model building scripts primarily differ on the alignment weighting methods that is used. The primary parameter is the number of bits to save in each column of the alignment in comparison to the background. Saving more bits will result in a more specialized model, while saving fewer bits will result in a more general model. The various scripts default to certain transition regularizers that also affect the generality of the model. Readers may wish to read the perl source code to find out more about these routine, for example to change the parameter settings passed to `build-weighted-model` to only drop of sequences with 100% identity. See Section 9.4.3 on page 83 and Section 8.1 on page 59.

Our favored all-around model building script is `w0.5`, which builds a SAM HMM with a target entropy of 0.5 bits per column. The `w1.0` script produces an HMM that is more general than `w0.5`, and thus will have more positives at a given scoring threshold. Figure 12 shows the 2ohxA model built using `w0.5`, while Figure 13 shows the `w1.0` model. Note the shorter columns in the second case, indicating less negative entropy in the distribution. Searchers with this more general model will find more false positives, and may also find more true positives.

The model building scripts include:

build-weighted-model A general script for building models using most of the features of SAM.

The default Dirichlet mixture prior `recode2.20comp`. The default transition regularizers is `fssp-trained.regularizer`, uniform alignment waiting, 0.5 bits per column is the target. This script is called by the various scripts discussed next, each of which will build a model from a multiple alignment (possibly with constraints).

w0.5 Build a model with target entropy weighting of 0.5 bits per column using `recode3.20comp` and `fssp-trained.regularizer`. The alignment is trimmed to sequences with less than 80% identity. This script is our standard for building HMMs to find remote potential homologs.

w0.7 Build a model with target entropy weighting of 0.7 bits per column using `recode3.20comp` and `fssp-trained.regularizer`. The alignment is trimmed to sequences with less than 85% identity. This script will create a model suitable for finding close homologs.

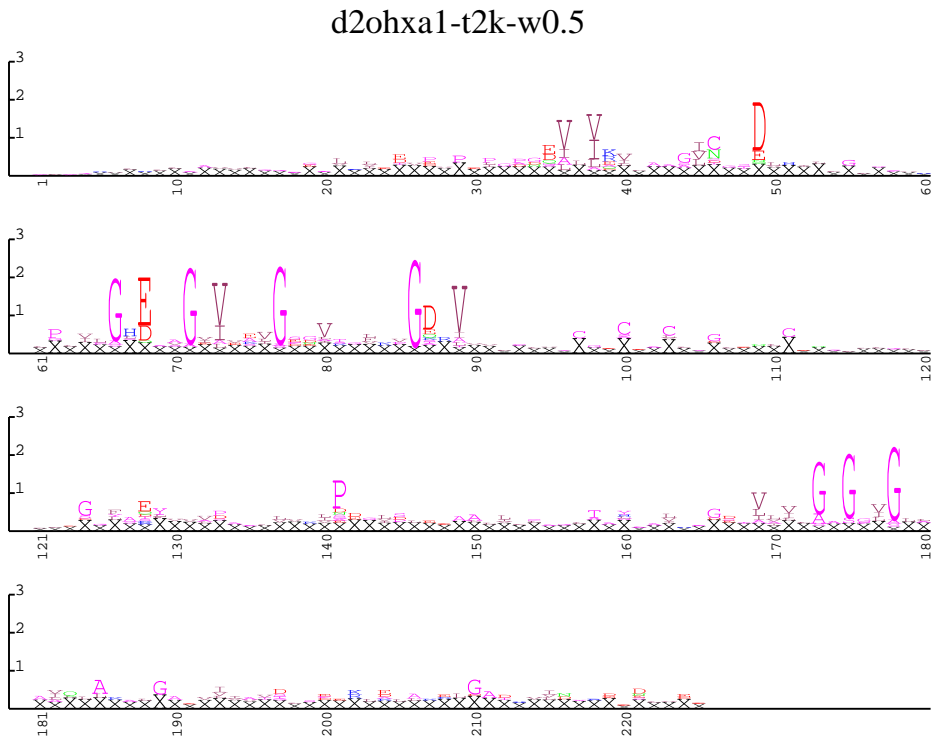


Figure 11: Sequence logo of the 2ohxA model built using the w0.5 model building script.

w1.0 Build a model with target entropy weighting of 1.0 bits per column using `recode3.20comp` and `fssp-trained.regularizer`. The alignment is trimmed to sequences with less than 90% identity. This script makes a strongly specialized model; it is primarily appropriate to use with `makelogo` as an aid to visualizing models.

4.10 Support scripts

Two support scripts are also included with SAM-T2K.

The `select-by-key-residues` and `select-by-gapless-regions` scripts can be used to modify a2m alignments. They both take an a2m file as input, and produce an a2m file as output, with the output being a subset of the lines of the input. That is, they both delete lines in the alignment that do not meet certain criteria.

The `select-by-key-residues` script takes a description of the key residues that must be present, and removes all sequences from the alignment that do not include those residues. For example,

```
select-by-key-residues -residues a.res < b.a2m > c.a2m
```

d2ohxa1-t2k-w0.7

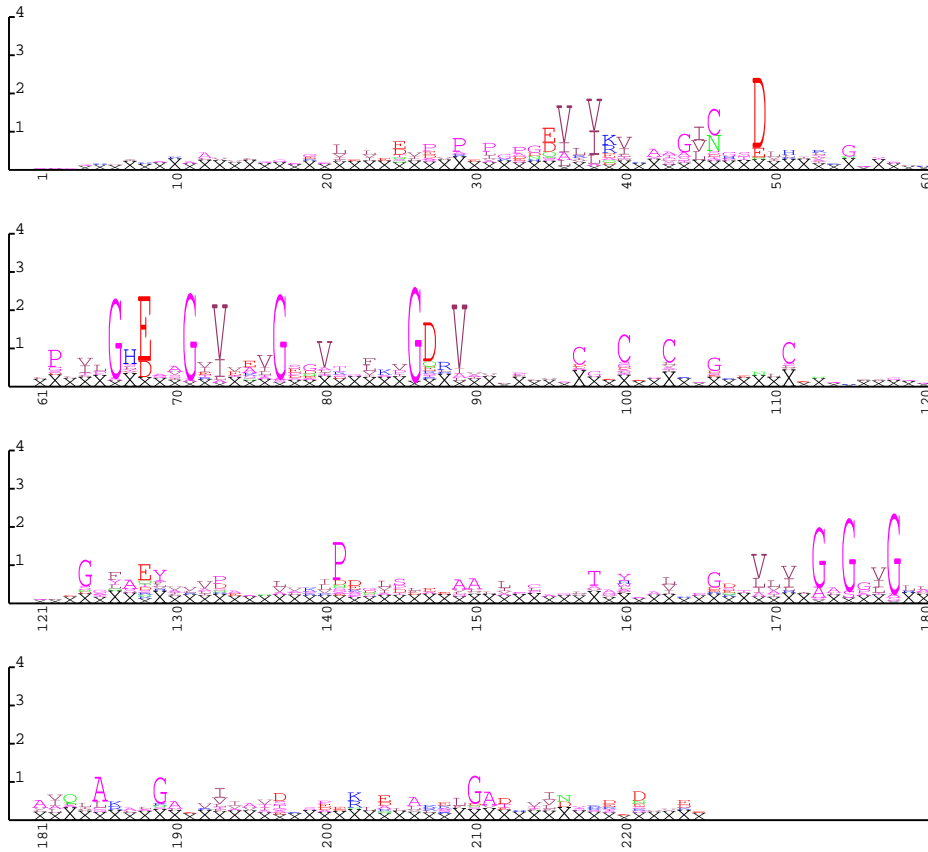


Figure 12: Sequence logo of the 2ohxA model build using the w0.7 model building script. The specificity of the model is increased, as indicated by the higher bits per column.

will create the file `c.a2m` based on the alignment `b.a2m` and the specification on `a.res`. The residue file is a white-space separated list of residues that must be present. For example,

```
N150 AS180 P-225
# Comments can be prefixed with #
```

would mean that alignment column 150 (with 1 as the first column) must have Asparagine, 180 must have Alanine or Serine, and 225 must have Proline or a gap. The script has two optional parameters, used as `-mismatch m.a2m`, `-err m.err`, or both, to create an `a2m` file containing all rejected sequences and an error file that explains why each sequence was rejected.

The `select-by-gapless-regions` script takes a description of the regions that must have no indels, and removes all sequences from the alignment that have insertions or deletions in those positions. For example,

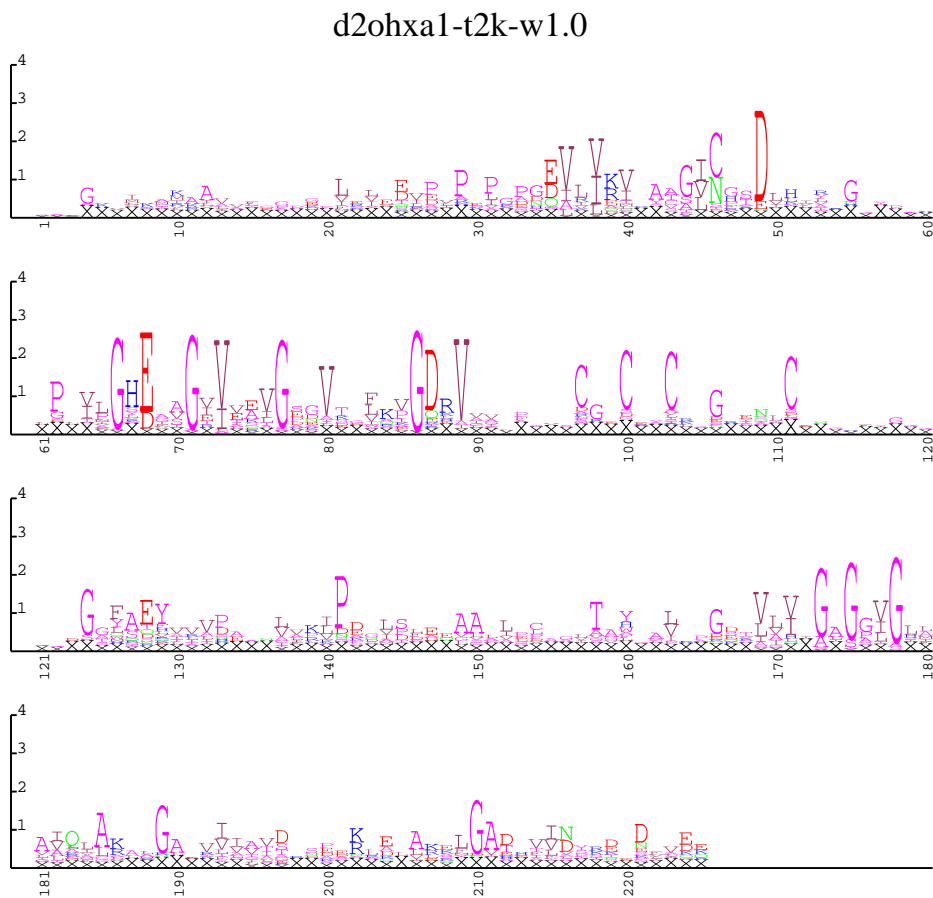


Figure 13: Sequence logo of the 2ohxA model build using the w1.0 model building script. The specificity of the model is greatly increased. The large ‘O’ indicates that one of the sequences in the alignment had been cut (such as in FSSP), resulting in an internal FIM position. (See Section 10.7 on page 127.) This sequence is trimmed by the w0.5 script as having greater than 80% identity to another sequence.

```
select-by-gapless-regions -nogaps a.nogap < b.a2m > c.a2m
```

will create the file `c.a2m` based on the alignment `b.a2m` and the specification on `a.nogap`. The `nogap` file is a white-space separated list of pairs of residues between which there should be no insertions. For example,

```
40 60
74 95 # Comment
```

would mean that alignment columns 40...60 and 74...95 must have no deletions, and that there cannot be insertions after columns 40...59 and 74...94.

4.11 Installing SAM-T2K

To install SAM-T2K, you must have version 5 of PERL installed, as `target2k` is a PERL5 script. The `sam-t2k-configure` script can take care of this when run in the directory in what the scripts are installed.

The other changes needed are in the in the `sam-t2k.conf` perl module.

- You need to have `blastp` from the Washington University Blast 2.0 installed (<http://blast.wustl.edu/>) and/or `blastall` from the NCBI version of Blast 2.0 (<http://www.ncbi.nlm.nih.gov/BLAST/>). For WU-BLAST, the full path of the program (usually called `blastp` or `wu-blastp`) should be named in `$blast_prog`. For NCBI BLAST2, the full path of the program (`blastall`) should be named in `$blast2_prog`. Also for NCBI BLAST, the appropriate `.ncbirc` text (the matrix data directory path) should be provided in `$blast2_ncbirc_text`.

If you would like to make NCBI Blast the default prefilter, change the line of the `target2k` that says

```
$blast2 = 0; # 1 for DEFAULT NCBI BLAST 2
```

to

```
$blast2 = 1; # 0 for DEFAULT WU BLAST 2
```

If both programs are available, the `-blast2` and `-wublast` options can be used to specify which BLAST prefilter to use. The `target2k` script has not been extensively evaluated with NCBI's version of `blastp`.

- For the standard use of the SAM-T2K method, you need a non-redundant protein database in `fasta` format, which has been indexed for use with the Blast prefilter program. The variable `$$:NR` should be set to the full filename for the non-redundant protein database. We use the NR database from NCBI available from <ftp://ftp.ncbi.nih.gov/blast/db/nr.Z>, which we then index using `setdb` for WU-BLAST and `formatdb` for BLAST2. **The T2K script requires both the flat and the indexed database.** Therefore, just downloading the indexed database will not work.
- The variable `$$:reg_lib_dir` should be set to the directory in which regularizers and Dirichlet mixtures are kept. The `lib` directory of the SAM installation package should contain all the necessary regularizers, and others can be downloaded from the web site

<http://www.cse.ucsc.edu/research/compbio/dirichlets/index.html>.

- The script needs to create a temporary directory (which it discards when it is done). This temporary directory can be created under `/tmp`, under the current directory (`.`), or elsewhere. The superdirectory for the temporary directory should be specified in `::$tmp_root_dir`.
- The directory for the SAM executables should be specified in `$sam_bin_dir`.
- The search path for executables should be completed with whatever standard directories need to be searched for commands like `cat`, `cp`, `echo`, `gunzip`, `rm`, and so forth. The Unix `whereis` and `which` commands may help you locate these programs. Need of the `gunzip` program can be avoided if the initial alignment is not a gnu-compressed file.
- The `make_directories` subroutine (at the end of `SamT2K.pm` changes the permission and group ownership of the temporary directories. This was done to make it easier for us to clean up temporary directories that were left by processes that failed. You may want to change the name of the group that is used (from “protein”), or you may want to comment out the lines that do the “chmod” and “chgrp”.

The scripts `build-weighted-model`, `w1.0`, `w0.7`, and `w0.5` use the `Setpaths.pm` perl module, and so should not need any changes except for the location of perl.

4.12 SAM-T2K references

- [1] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, September 1997.
- [2] Stephen F. Altschul. Amino acid substitution matrices from an information theoretic perspective. *Journal of Molecular Biology*, 219:555–565, 1991.
- [3] D. Haussler, A. Krogh, I. S. Mian, and K. Sjölander. Protein modeling using hidden Markov models: Analysis of globins. In *Proceedings of the Hawaii International Conference on System Sciences*, volume 1, pages 792–802, Los Alamitos, CA, 1993. IEEE Computer Society Press.
- [4] Steven Henikoff and Jorja G. Henikoff. Position-based sequence weights. *Journal of Molecular Biology*, 243(4):574–578, November 1994.
- [5] Liisa Holm and Chris Sander. Mapping the protein universe. *Science*, 273(5275):595–603, Aug 2 1996.
- [6] R. Karchin and R. Hughey. Weighting hidden Markov models for maximum discrimination. *Bioinformatics*, 14(9):772–782, 1998.
- [7] A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995. Information on SCOP is available at <http://scop.mrc-lmb.cam.ac.uk/scop>.
- [8] J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia. Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods. *Journal of Molecular Biology*, 284(4):1201–1210, 1998.
- [9] J. Park, S. Teichmann, T. Hubbard, and C. Chothia. Intermediate sequences increase the detection of homology between sequences. *Journal of Molecular Biology*, 273:349–354, 1997.

- [10] William Pearson. Comparison of methods for searching protein sequence databases. *Protein Science*, 4:1145–1160, 1995.
- [11] C. Sander and R. Schneider. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins: Structure, Function, and Genetics*, 9(1):56–68, 1991.
- [12] E.L.L. Sonnhammer, S.R. Eddy, and R. Durbin. Pfam: A comprehensive database of protein families based on seed alignments. *Proteins: Structure, Function, and Genetics*, 28:405–420, 1997.
- [13] Julie D. Thompson, Desmond G. Higgins, and Toby J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties, and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

5 File types

SAM produces a variety of files. This section summarizes both the extensions SAM uses and the extensions we conventionally use when naming output that goes to standard output.

The following file extensions are automatically used:

- .a2m** A SAM alignment file, as created by `align2model` or `hmmscore`. A FASTA-compatible format. See Section 10.1 on page 90.
- .apr** A SAM alignment probabilities file, as created by `pathprobs`. An RDB-compatible format. See Section 10.8 on page 127.
- .ckp** A PSI-BLAST checkpoint profile file created by `sam2psi`, which can be used with PSI-BLAST in conjunction with the matching `.cks` checkpoint sequence file. See Section 10.10.6 on page 135.
- .cks** A PSI-BLAST checkpoint sequence file created by `sam2psi`, which can be used with PSI-BLAST in conjunction with the matching `.ckp` checkpoint profile file. See Section 10.10.6 on page 135.
- .cst** A SAM constraints file containing the definition of sequence position constraints. See Section 9.7 on page 87.
- .data** A `gnuplot` data file generated by `makehist`. See Section 10.11 on page 137.
- .dist** A scoring file listing sequence identifiers, lengths, and scores, produced by `hmmscore`. See Section 10.2 on page 100.
- .dist-rdb** A scoring file in RDB format including sequence identifiers, lengths, scores, and information about scoring method produced by `hmmscore`. See Section 10.2 on page 100.
- .frag** An a2m alignment file generated by `fragfinder`. See Section 10.4 on page 123.
- .freq** A frequency model generated by `buildmodel` when `many_files` is set or by `grabdp`. This is a model that reports the frequency of each letter in each state when the training set is evaluated according to the model. See Section 9.3 on page 77 and Section 10.5 on page 124.

- .fstat** A scoring file produced by the `fragfinder` program. See Section 10.4 on page 123.
- .hmmer** An HMMER-format model produced by `sam2hmmer`. See Section 10.10.7 on page 135.
- .kids** A Kestrel sequence id file. The file is an ordered list of identifiers for sequences in a `.kseq` or `.krseq` file. See Section 10.12.10 on page 147.
- .kseqs** A Kestrel sequence database, containing an encoding of the sequences optimized for Kestrel `hmmscore`. See Section 10.12.10 on page 147.
- .kseqs** A Kestrel reverse null model scoring sequence database. This contains each sequence in the original database, plus a reverse of the sequence. See Section 10.12.10 on page 147.
- .match-rdb** An RDB file created by `grabdp`, listing the posterior match probabilities of each amino acid type at each node, plus the prior probabilities of each amino acid type. Used for diagnostic purposes. See Section 10.5 on page 124.
- .mod** A model file created by a `buildmodel`, `addfims`, or `modelfromalign`. When `many_files` is not set (the default), `buildmodel` places the run statistics, parameter settings, and the frequency model into this file as well. See Section 9.3 on page 77.
 - .f.mod** A subfamily model for family number 'f'. See Section 9.4 on page 80.
 - .a.mrrr.mod** If `print_all_models` is set, model number *m* during re-estimate cycle 'rrr' will be placed in the file.
 - .s.rrr.mod** If `print_surg_models` is set, the winning model is printed after each surgery iteration, where 'rrr' is the re-estimation index.
- .mult** One of two multiple domain alignment output files created by `hmmscore`. This will contain all alignments to a motif that were found. See Section 10.2.5 on page 111.
- .mlib** A model library file created by `hmmscore` model calibration. Each entry includes a single- or multi-track model specification and various scoring parameters. See Section 10.2.10 on page 119.
- .mstat** The other multiple domain output file. This file contains the sequence identifier and scores for the data in the corresponding `.mult` file. See Section 10.2.5 on page 111.
- .pa2m** A SAM alignment file that alternates sequences and posterior probability annotations in sequence format, as created by `pathprobs`. See Section 10.8 on page 127.
- .pdoc** Created by `grabdp`, this file contains the posterior decoding of the dynamic programming for match states and transitions into match states. The files tend to be rather large. See Section 10.5 on page 124.
- .plt** A `gnuplot` command file created by `makehist`. See Section 10.11 on page 137.
- .ptrack** Created by `predict_track`, this file contains secondary structure (or other second alphabet) predictions for a given sequence based on a primary sequence model, and databases of primary and secondary structure for a group of sequences. See Section 10.9 on page 129.
- .samrc** A file of default parameters either in a home directory or the current directory. See Section 6 on page 47.
- .sel** Sequences that passed the selection criteria used in `hmmscore`. See Section 10.2 on page 100.
- .seq** A file of sequences, as for example created by `sampleseqs`.

- .stat** The statistics of a `buildmodel` run when `many_files` is set, including re-estimation scores and initial parameters. See Section 9.3 on page 77.
- .ta2m** An alignment trimmed by the `pathprobs` program. See Section 10.8 on page 127.
- .weightoutput** A list of sequence weights from the internal weighting option of `buildmodel`. Present when `print_all_weights` and internal weighting are both enabled. See Section 9.4.4 on page 83.

The following file extensions are conventionally used in this manual.

- .colors** A color file for the `makelogo` program, usually located in the same directory as the Dirichlet priors. See Section 10.10.4 on page 131.
- .log** Standard error output from running `target2k` can be redirected to this file. See Section 4 on page 24.
- .ncomp** A Dirichlet mixture prior library file, where `n` is the number of components to the mixture, as in `null.1comp` or `mall-opt.9comp`. Information on specific mixtures can be found at <http://www.cse.ucsc.edu/research/compbio/dirichlets/index.html>. See Section 8.1 on page 59.
- .plib** A Dirichlet mixture prior library file (old naming convention). The same extension is used both for match state regularizers and HSSP-based transition regularizers. See Section 8.1 on page 59.
- .pretty** The output (stdout) from `prettyalign`. See Section 10.1 on page 90.
- .regularizer** A transition regularizer, generally used in conjunction with a Dirichlet mixture regularizer for the match states. See Section 8.1 on page 59.
- .weights** A file of sequence weights. See Section 9.4 on page 80.

In most cases, SAM can read compressed (`.gz` or `.Z`) files, specified either as their complete name or as their root name without the compression suffix. If the root name is given and both a compressed and an uncompressed file exist, the uncompressed file is read. If both a `.gz` and a `.Z` file is present, the `.gz` file is used. Reading compressed files requires that the GNU decompression program `gunzip` and the Unix decompression program `uncompress` be in the user's path.

6 Parameter specification

Parameter values are drawn from four sources: command line arguments, inserted parameter files, default parameter files, and the program itself. Initial models and regularizers cannot be specified on the command line. Several programs require certain parameters to be set; if they are not (or if parameters are specified incorrectly), a usage message is output to standard error.

Each parameter, including the initial model and regularizer, has a reasonable setting hardwired in the SAM code. These are the default values listed in Section 12. The default regularizer is actually two defaults, one for RNA or DNA, and the other for proteins.

These hardwired values can be overridden by a user-specific default file or command line specification. This file can be one of three alternatives. First, if the environment variable SAMRC is set, new default values are read from that file. Second, if the SAMRC variable was not set and a `.samrc` file exists in the current directory, that file is used as the default. Third, if SAMRC was not set and `.samrc` was not found in the current directory, `$HOME/.samrc` is checked.

Parameter files can cause other parameter files to be read using the `insert` directive. When this directive is used in a default such as `.samrc`, the inserted file is assumed to have defaults as well. Non-default files are specified on the command line as, for example,

```
buildmodel test -alphabet RNA -insert trna.init
```

In this case, the alphabet is set to RNA, and the file `trna.init` will override default parameters hardwired in the program or specified in one of the `.samrc` files. If the file contained, for example, the line `alphabet DNA`, the alphabet would be switched to DNA with an appropriate warning message. Values are set and insert files are read according to their position on the command line or within a file.

Command line arguments are evaluated in the order they are presented to the program. If one of the command line arguments specifies an inserted parameter file, that file is processed before the next command line argument. If one file inserts another, the inserted file is processed before completing the original file. Thus, to override values specified in an inserted file, insert the file *first* on the command line, and then specify the parameters to reset—the last specified values win.

It is often important to conditionally specify initialization information. In addition to the `insert`, three conditional insertion directives are also available: `insert_file_dna`, `insert_file_rna`, and `insert_file_protein`. These cause a file to be inserted if the alphabet matches the directive. If the alphabet is not yet set when one of these is encountered, and warning message is generated.

Two parameter names have abbreviated forms: `i` can be used in place of `insert`, and `a` can be used in place of `alphabet`. The following will set the alphabet to RNA and read in the file named `parameters`.

```
buildmodel test -a RNA -i parameters
```

The model output (such as `test.mod`, in the command line above) includes statistics about the run and a listing of all parameters that have been changed from their default values. Inserted file names are listed, but commented out, because their effect has been recorded in the list of all changed parameter values. Random number seeds created based on the pid are also commented out so that new seeds will be created if the program is rerun on the file.

Models are usually specified using the insert file (`-i`) command line argument. In this case, the model type (i.e., model, regularizer, frequency counts, or null model, discussed in Section 8.4) is read from the file. Alternatively, a `model_file`, `regularizer_file`, or `nullmodel_file` can be specified, in which case the very first model structure in that file (which could be a regularizer or frequency count model, for example) is read in. These file names will override any models present in the inserted files, even if the inserted file occurs after the `model_file` parameter on the command line. This option is particularly useful for discrimination training with positive and negative examples, in which case a model generated by the negative examples can be used as the null model. See Section 10.2.1 on page 102.

There are three special parameter names, `db`, `id`, and `not_id`, that form lists of strings. That is, when multiple database or sequence identifiers are found on the command line or in a parameter file, they are added to the current list of databases or sequence identifiers, rather than replacing the previously-specified value.

7 Sequence formats

The SAM system understands several alphabets and many sequence formats.

7.1 Alphabets

The SAM system currently supports two nucleotide alphabets ('DNA' and 'RNA'), one amino acid alphabet ('protein'), and several secondary structure alphabets (including 'DSSP', 'EHL', 'EHL2', 'EHTL', 'TCO', 'ALPHA', 'PB', 'STR', and 'ANG'), as well as user-defined alphabets of up to 25 letters. The predefined alphabets can be specified by setting the `alphabet` variable. If no alphabet is chosen, the first sequence in a specified file will be examined using `readseq` (discussed below) to determine if a nucleotide or protein alphabet should be used. If this method does not work, the protein alphabet is the default. The SAM software includes several warning messages if it appears that an incorrect alphabet has been chosen.

The alphabets use standard characters. DNA sequences are composed of the characters "AGCTRYN" and RNA of "AGCURYN," where 'R' is a purine ('G' or 'A'), 'Y' is a pyrimidine ('C,' or 'T' or 'U,' as appropriate), and 'N' is a wildcard character that could be any of the four normal characters. SAM's sequence I/O routines can convert between DNA and RNA alphabets if the alphabet is specified incorrectly.

The protein alphabet is "ACDEFGHIKLMNPQRSTVWYBZX." In addition to the twenty amino acids, 'X' is the general wildcard character, 'B' matches 'N' or 'D', and 'Z' matches 'Q' or 'E.' Protein alignments (specified with `alignfile` to `buildmodel` or `modelfromalign`) converted to models can additionally use the letter 'O' to indicate insertion of a free-insertion module. Except for these two instances, the 'O' character is converted to the all-matching 'X' wildcard.

The DSSP alphabet is "EHTSGBICX", including 'E' (beta strand), 'H' (alpha helix), 'T' (turn), 'S' (bend), 'G' (3-10 helix), 'B' (short beta bridge), 'I' (pi helix), and 'C' (random coil). The character 'L' (loop) is an alias for 'C'. The character 'C' is used to indicate coils rather than the space character used by DSSP. The following secondary structure alphabets are subsets of the DSSP alphabets with various groups merged.

- The 3-character EHL alphabet (with characters "EHCX") aliases 'G' to 'H'; and 'L', 'T', 'S', 'B', and 'I' to 'C'.
- The 3-character EHL2 alphabet (with characters "EHCX") aliases 'G' and 'I' to 'H'; 'B' to 'E'; and 'L', 'T', and 'S' to 'C'.
- The 4-character EHTL alphabet (with characters "EHTCX") aliases 'G' to 'H'; and 'L', 'S', 'B', and 'I' to 'C'.

- The 4-character EHTL2 alphabet (with characters “EHTCX”) aliases ‘G’ and ‘I’ to ‘H’; and ‘L’, ‘S’, and ‘B’ to ‘C’.
- The 5-character EBHTL alphabet (with characters “EBHTCX”) aliases ‘G’ and ‘I’ to ‘H’; and ‘L’ and ‘S’ to ‘C’.
- The 6-character EBGHTL alphabet (with characters “EBGHTCX”) aliases ‘I’ to ‘H’; and ‘L’ and ‘S’ to ‘C’.
- The 7-character STRIDE alphabet (with characters “EBGHITCX”) aliases ‘L’ and ‘S’ to ‘C’.

The TCO alphabet is “EFGH”. ‘TCO’ is the cosine of the dihedral angle defined by the carbonyl double bonds of residue i and residue $i-1$ (near +1 for helices and near -1 for sheet). The cosine values are assigned to four classes ‘E’ (extended), ‘F’ (transition-1), ‘G’ (transition-2) and ‘H’ (helical), determined by the *k-means* algorithm.

The ALPHA alphabet is “GHISTAEBCDF”. ‘ALPHA’ is the virtual dihedral angle between the C_α atoms of residues $j-1$, j , $j+1$ and $j+2$. The angles are assigned to eleven classes: $G \in [8^\circ, 31]$, $H \in (31^\circ, 58^\circ)$, $I \in (58^\circ, 85^\circ)$, $S \in (85^\circ, 140^\circ)$, $T \in (140^\circ, 165^\circ)$, $A \in (165^\circ, -170^\circ)$, $E \in (-170^\circ, -136^\circ)$, $B \in (-136^\circ, -103^\circ)$, $C \in (-103^\circ, -68^\circ)$, $D \in (-68^\circ, -17^\circ)$, $F \in (-17^\circ, 8^\circ)$, according to peaks in the distributions of ALPHA angles for the twenty possible side-chains. Assignment was done manually, by Kevin Karplus.

The PB (‘Protein Blocks’) alphabet is “ABCDEFGHIJKLMNPO”. PB was designed by de Brevern et. al. They used overlapping residue fragments of length five (chosen empirically), extracted from a non-redundant set of structures, and encoded them as sequence “windows” of (PHI, PSI) pairs called *dihedral vectors*. An unsupervised *Kohonen self-organizing map* network was trained on the dihedral vectors with an *RMSDA* (root mean square deviations on angular values) distance metric, to produce a set of 16 clusters, each with a representative Protein Block or *PB*. The clusters are ‘A’ (N-cap beta), ‘B’ (N-cap beta), ‘C’ (N-cap beta), ‘D’ (beta), ‘E’ (C-cap beta), ‘F’ (C-cap beta), ‘G’ (mainly coil), ‘H’ (mainly coil), ‘I’ (mainly coil), ‘J’ (mainly coil), ‘K’ (N-cap alpha), ‘L’ (N-cap alpha), ‘M’ (alpha), ‘N’ (C-cap alpha), ‘O’ (C-cap alpha) and ‘P’ (C-cap alpha to N-cap beta).

The STR (strand) alphabet is “BGHSTLPAMZQE” an enhanced version of DSSP, conceived by Yael Mandel-Gutfreund. It subdivides DSSP letter E (beta strand) into 6 letters, according to properties of a residue’s relationship to its strand partners. ‘P’ is surrounded by two parallel partners, ‘A’ by two anti-parallel partners, ‘M’ by one anti-parallel and one parallel partner. Edge strands may be ‘Q’ (parallel partner) or ‘Z’ (anti-parallel partner). A strand with no partners is assigned the letter ‘E’.

The ANG (angle) alphabet is based on a design by Bystoff et. al. Bystoff partitioned the (PHI, PSI) plane into eleven regions, using the *k-means* algorithm on all trans (PHI, PSI) pairs in PDB with $k=10$. Cluster boundaries were calculated with a *Voronoi* method. All cis (PHI, PSI) pairs were assigned to an eleventh cluster, which ANG does not use. In the ANG assignments, a (PHI,PSI) pair is associated with a cluster representative from which it has the minimum Euclidean distance. Representatives are as follows: $H = [-61.91^\circ, -45.20^\circ]$, $G = [-109.78^\circ, 20.88^\circ]$, $P = [-70.58^\circ, 147.22^\circ]$, $E = [-132.89^\circ, 142.43^\circ]$, $D = [-135.03^\circ, 77.26^\circ]$, $N = [-85.03^\circ, 72.26^\circ]$, $Y = [-165.00^\circ, 175.00^\circ]$, $L = [55.88^\circ, 38.62^\circ]$, $T = [85.82^\circ, -0.03^\circ]$, $S = [80.00^\circ, -170.00^\circ]$.

The BYS (Bystoff) alphabet is the same as the ANG alphabet with the addition of the letter ‘C’ for cis peptides.

The RELSA2 alphabet (Rost and Sander, 1994) is a two-letter alphabet based on a residue’s relative solvent-accessible surface area. Residues with relative solvent accessibility less than 16% are assigned to the buried state and those above 16% are assigned to the exposed state.

The RELSA3 alphabet (Rost and Sander, 1994) is a three-letter alphabet based on a residue’s relative solvent-accessible surface area. Residues with relative solvent accessibility less than 9% are assigned to the buried state, those between 9% and 36% to the intermediate state, and those above 36% to the exposed state.

The RELSA10 alphabet (Rost and Sander, 1994) is a ten-letter alphabet based on a residue’s relative solvent-accessible surface area. A residue is assigned to one of the states according to the formula $int(\sqrt{100 * RelAcc})$.

The ABSSA7 alphabet is a seven-letter alphabet (all seven states are equiprobable) based on a residue’s absolute (unnormalized) solvent-accessible surface area.

The RELSA7 alphabet is an equiprobable, seven-letter alphabet based on a residue’s relative solvent-accessible surface area. This alphabet was selected to provide a direct comparison of relative and absolute solvent-accessibility measures, by measuring its performance against that of ABS-SA-7.

Neighborhood count alphabets with seven equiprobable classes:

The BURIAL_GEN_8_7, BURIAL_GEN_9_7, and BURIAL_GEN_10_7 alphabets are based on neighborhood counts within a spherical cut-off of a residue of interest. These alphabets use maximally informative spots and count all atoms within the spherical-cutoff region. The radii of the spheres are 8, 9, and 10Å, respectively.

The CA_BURIAL_12_7 and CA_BURIAL_14_7 alphabets use the C_α atom as the sphere center and 12 and 14Å sphere radii. Residues are represented by their C_α atoms, and only C_α atoms inside the sphere are counted. The C_α of the residue of interest is not included in the count.

The CB_BURIAL_12_7, CB_BURIAL_14_7, and CB_BURIAL_16_7 alphabets use the C_β atom as the sphere center (C_α for glycine), with 12, 14, and 16Å sphere radii. Residues are represented by their C_β atoms and only C_β atoms (except C_α for glycines) inside the sphere are counted. The C_β of the residue of interest is not included in the count.

- R. Karchin et. al. “Hidden Markov models that use predicted local structure for fold recognition: alphabets of backbone geometry,” *PSFG*, 51(4):504–514, 2003.
- C. Bystroff et.al. “HMMSTR: a Hidden Markov Model for Local Sequence-Structure Correlations in Proteins” *JMB*, 301(1):173–190, 2000.
- A.G. de Brevern et. al. “Bayesian probabilistic approach for predicting backbone structures in terms of protein blocks” *PSFG*, 41:271–287, 2000.
- B. Rost and C. Sander “Conservation and prediction of solvent accessibility in protein families,” *PSFG*, 20(3):216–226, 1994.

The secondary and local structure alphabets are not automatically detected; they must be specified with a `alphabet` command line option.

In all alphabets, unknown characters are converted to wildcards and a warning message is printed.

When a model is created, a wildcard character's probability is the sum of the probabilities of the component letters. Thus, the 'X' character will have unity probability, giving it no preference to one state over another. During the training process, wildcard character frequency counts are proportioned among the appropriate true characters according to the relative probabilities of those characters.

All alphabets have internal regularizers (single-component background distributions), and the protein alphabets includes several in Dirichlet mixture prior libraries in the SAM library directory. The `listalphabets` program should be relied on (rather than this documentation) for determining what alphabets are available, what character aliases are used, and what the default character and transition regularizers are. Alphabet background distributions can be changed with an `alphabackfile`. See Section 8.1 on page 59, Section 8.1.1 on page 60, Section 10.12.1 on page 141, and Section 7.1.3 on page 55.

7.1.1 User-defined alphabets

SAM also supports user-defined alphabets of 2 to 25 user-selected letters ('A'-'Z') and one (required) wildcard letter. The restriction to alphabetic characters is a result of the need for both uppercase and lowercase letters in the sequence alignment format. As the system always requires an all-matching wildcard, only 25 letters are allowed.

User-defined alphabets are specified with the `alphabet_def` variable. As with the standard alphabets, the definition will be included in all resulting models, so future specification of the alphabet on the command line is not required.

For example, performing the commands

```
buildmodel texttest -train text.seq -alphabetdef "text QWERTYUIOPASDFGHJKLZCVBNMX"
align2model texttest -i texttest.mod -db text.seq
```

results in the alignment file:

```
>sentence1
thequ..ICKBROWNFOXJUMPEDOVERTHES.LOWLAZ.YDOG----..
.....
>sentence1
thequ..ICKBROWNFOXJUMPEDOVERTHES.LOWLAZ.YDOG----..
.....
>sentence2
thequickERGREENFOXHOPPEDOVERTHES.LOWLUCkYPIG----..
.....
>sentence3
.....-----THES.LOWLAZ.YPIGWADDle
dintothequickpurplefox
>sentence4
thef...ASTBROWNFOXHOPPEDINTOTHEQuICKLAZ.YDOG----..
.....
```

Note that the above example does not model the letter 'X' because it is a wildcard: the 'X' character

was not trained and does not have a preference for any state over any other state.

A minimum of three characters, 2 normal and one wildcard, is required to define an alphabet. Default flat regularizers are created automatically, but users may wish to create their own alphabet-specific regularizers with `regularizer_file`.

As with alphabets, models are tagged with the `alphabet_def` line, for example

```
MODEL -- Final model for run text
alphabet_def text QWERTYUIOPASDFGHJKLZCVBNMx
GENERIC
1.886984 0.254944 0.376488
.....
```

See Section 8.4 on page 65.

Alphabet definition statement set include a standard alphabet name and a hyphen for the character list referred to the standard alphabet and use its regularizers. For example, both of the following refer to the predefined protein alphabet:

```
listalphabets foo -alphabetdef "protein -"
listalphabets foo -a protein
```

7.1.2 User-defined Numeric Alphabets

Data sets alphabets larger than 25 characters can also be modeled using SAM and user-defined alphabets.

In the numeric alphabet, characters are numbers and sequences use whitespace to separate the individual characters. For example, the `trna10n.seq` file is a translation of `trna10.seq` into a numeric alphabet. One sequence is:

```
>TRNAn1
1 1 1 1 0 3 1 3 0 1 2 3 2 0 1 3 1 1 3 0 1 0 1 2 1 2 0 3 1 2 3 3 2 1 2
0 3 1 3 0 3 1 0 1 1 2 2 2 2 1 1 1 3 3 2 1 0 3 2 2 2 2 1 1 2 0 3 2 3 2
2 0
```

All the normal commands can be used, as in:

```
listalphabets testn -alphabetdef "RNANumbers 5"
buildmodel testn -train trna10num.seq -randseed 0 -alphabetdef "RNANumbers 5"
hmmscore testn -modelfile testn.mod -db trna10n.seq -alphabetdef "RNANumbers 5"
align2model testn -modelfile testn.mod -db trna10n.seq -id TRNAn1
-id TRNAn2 -id TRNAn3 -alphabetdef "RNANumbers 5"
```

In this alphabet, the five legal characters are the number is zero through three as regular characters, in the number four as the required any-character wildcard. When such an alphabet is defined, SAM creates a flat regularizer as with other defined alphabets. Especially for large alphabets, the user is strongly recommended to provide SAM with a different regularizer based on the background probabilities of the alphabet. (For numeric alphabets, the Order line is not used.) See Section 7.1.3 on page 55..

The alphabet information from the above commands is:

```
##### Track 0 alphabet #####
#InternalID:      35
Name:            RNANumbers
Comment:        <none>
Length:          4
WildCards:       1
FLength:         4
FWildCards:      1
Letters:         <numeric alphabet>
Convert:         <none>
MatchRegularizer:
  0 0.250000  1 0.250000  2 0.250000  3 0.250000
# Total match regularizer weight is 1.000000
InsertRegularizer:
  0 0.250000  1 0.250000  2 0.250000  3 0.250000
# Total insert regularizer weight is 1.000000
TransitionRegularizer:
  IntoDeleteFrom: D 1.886984 M 0.254944 I 0.376488
  IntoMatchFrom:  D 1.819972 M 15.521340 I 3.764209
  IntoInsertFrom: D 0.225758 M 0.265967 I 4.006562
##### Track 1 alphabet #####
##### Track 2 alphabet #####
##### Track 3 alphabet #####
##### Track 4 alphabet #####
##### Track 5 alphabet #####
```

Alignment (.a2m) files use the symbols '*', '-', and '.' in a column proceeding each numeric character to indicate whether or not that character was matched, is a blank (delete) character, or is an insert character. Presently, SAM and the prettyalign program are not able to read these alignments.

```

>TRNAn1
- *1*1*1
*1*0*3*1*3*0*1*2*3*2
*0*1- - *3*1*1 *3*0*1
*0*1*2*1*2*0*3*1*2*3
*3*2*1*2*0*3*1*3*0*3
*1*0*1*1*2*2*2*1*1
*1*3*3*2*1*0*3*2*2*2
*2*1*1*2*0*3*2*3*2*2
*0
>TRNAn2
- *1*2*1
*1*2*2*1*3*2*1*3*2*3
*0*1*3*2*3*1*1.0.3*3
*0*1*1*0*2*1*2*3*1*1
*2*2*3*2*2*2*0*0*1*2
*2*0*1*2*0*0*3*2*2*2
*1*1*1*3*3*2*1*0*0*3
*2*2*2*1*1*2*1*1*2*2
*1*2*0
>TRNAn3
.1.1.2.2.2.3.1.3.1.1
- - - - - *2*3
*0*1*2*3*1*1*3 *2*0
*0*0*1*2*1*2*2*3*1*3
*2*3*0*1*3*0*0*0*2*0
*1*1*0*1*0*3*2*2*3*1
*1*1*3*3*2*1*0*0*3*2
*2*2*0*1*2*1*1*1*1*2
*2*3*2*2.0

```

The distribution of SAM limits the use of these alphabets to 255 characters and a wildcard for efficiency concerns with respect to the standard alphabets. A version capable of handling larger alphabets can be obtained by sending electronic mail to sam-info@soe.ucsc.edu.

7.1.3 Alphabet Background Files

Background frequencies for internal alphabets may be modified, and background frequencies for user-defined alphabets may be set by specifying one or more background files with the `alphbackfile` keyword.

If one or more `alphabets` have been set, then only the set alphabets are modified, otherwise all internal alphabets are modified (useful when using `listalphabets` to verify the changes). When reading an `alphbackfile`, SAM proceeds as follows:

- Lines are ignored until a line begins with ‘Alphabet’ (with any capitalization)
- Spaces, tabs, and equal signs are skipped, and the next token (delimited by space, tab, equal sign, or newline, referred to as whitespace henceforth) is assumed to be an alphabet name.
- If this name is an active alphabet, then SAM expects the next line to begin with ‘Order’ followed by whitespace, followed by a number of single letters separated by whitespace. These

define the order in which probabilities are presented on the next line.

- SAM then expects the next line to start with ‘Probs’ followed by whitespace and a matching number of floating-point numbers.

The `alphbackfile` has a number of error messages related to missing characters, mismatched alphabets, and the like. The `listalphabets` program is a good way to view the results of the `alphbackfile` reading.

The file format reads files such as the following, but only pays attention to the Alphabet, Order, and Probs lines.

```
# Alphabet background file
ClassName = BackgroundProbs
Alphabet = dssp_eh12
Order =      E      H      L
Probs = 0.2277 0.3668 0.4055
EndClassName = BackgroundProbs
ClassName = BackgroundProbs
Alphabet = STRIDE
Order =      B      C      E      G      H      I      T
Probs = 0.0114 0.1810 0.2221 0.0384 0.3420 0.0001 0.2049
EndClassName = BackgroundProbs
```

For the Codon alphabet, the order statement only includes the four nucleotides. For example, an order statement that listed “T G C A” would be followed by a line of 64 values all on a single line in the order of TTT, TTG, TTC, TTA, TGT, . . . AAAA.

For numeric alphabets, the order statement, if included, is ignored. Numeric alphabets must always be presented in numeric order.

7.2 Sequences

SAM has three ways of reading sequences. SAM’s FASTA (and a2m-format alignment) format reader is by far the quickest. SAM also includes an HSSP alignment file reader and, for many other formats, D. G. Gilbert’s `readseq` package. Because SAM’s FASTA reader is tuned both to SAM and to a single format, it can be up to 10 times faster than `readseq`. Users are advised to convert large databases into FASTA format using `readseq` or some other program.

When working with formats other than FASTA (including FASTA variants), it is always a good idea to use the `checkseq` program to ensure that SAM is reading your sequence format correctly. The `checkseq` program will let you know the number of sequence in a file, the format type, and the total length. See Section 10.12.2 on page 141.

SAM’s modified version of the `readseq` package by D. G. Gilbert of the Indiana University. The code is based on the February 1, 1993 release, and is included as a subdirectory of the SAM source directory. We are grateful that Gilbert has provided this useful package that may be used by anyone.

The `readseq` package can read most common formats: examples of all these formats are included in the `readseq` directory. The formats include

- IG/Stanford, used by Intelligenetics and others
- GenBank/GB, genbank flatfile format
- NBRF format (SAM modifications include accepting sequences without semicolon in ID and without a comment line)
- EMBL, EMBL flatfile format
- GCG, single sequence format of GCG software
- DNASTrider, for common Mac program
- Fitch format, limited use
- Pearson/Fasta, a common format used by Fasta programs and others.
- Zuker format, limited use. Input only.
- Olsen, format printed by Olsen VMS sequence editor. Input only.
- Phylip3.2, sequential format for Phylip programs
- Plain/Raw, sequence data only (no name, document, numbering)
- MSF multi sequence format used by GCG software
- PAUP's multiple sequence (NEXUS) format
- PIR/CODATA format used by PIR

We usually use FASTA format, which looks like this:

```

; Comments are ignored.
>IDENTIFIER
LMLDQQTINI IKATVPVLKE HGVTITTTTFY KNLFAKHPEV
RPLFDMGRQE SLEQPKALAM T
>SEQ2      Annotations after identifier are preserved
AKHPEVRPLFDMGRQESLEQPKALAMT

```

For information on other formats, please look through the test files and the `Formats` file in the `readseq` directory.

Sequence output will be in FASTA format regardless of the input file format.

Alignment output by `align2model` and `hmm_score` is in a FASTA-compatible format in which uppercase letters indicate match states and lowercase letters indicate insertion states and hyphens indicate deletion states (model positions for which the given sequence has no corresponding character). The `prettyalign` program can be used to line up the match columns of an a2m-format alignment file.

Additionally, `align2model` can include periods so that its sequence outputs can be visually aligned without the use of `prettyalign`. If, for example, the longest sequence in a collection is 2000 characters long, all sequences will be filled (using periods) to that longest sequence's alignment length, which will be more than 2000 if any deletion states are used. Thus, allowing the periods to be printed can greatly expand the size of the alignment file. If periods are not desired, the

parameter `a2mdots` can be set to 0. The `prettyalign` program will work whether or not the `a2m` format alignment has periods.

SAM can also read HSSP files.

7.3 Models as sequences

As a relatively untested feature, probabilistic sequences may be read into SAM. A probabilistic sequence is one that, instead of being a sequence of letters from an alphabet, has in sequence a probability distribution across all letters in the alphabet. The input format is a SAM model. When a SAM model is used as a sequence, the probabilities from the match state are used to define each position in the sequence. When comparing a probabilistic sequence to an HMM, the dot product of the HMM state and the probabilistic character is used to decide the probability of the character being generated by the HMM state.

A probabilistic sequence can be specified by provided a model file as a database, for example, `-db test.mod`.

When alignments are performed between a probabilistic sequence and an HMM, the `a2m` file uses the most likely character of each state in the probabilistic sequence as a letter. If different letters are desired, the `dbgguide` parameter may be used multiple times to specify files that contain sequences that exactly match the model length of each sequence specified in the list of `db` databases.

This feature is in the midst of development, and is subject to change.

7.4 Training sets, test sets, and databases

The `buildmodel` program uses two sets of sequences: the training and the test set. Training is performed exclusively on the training set, and at the end of the model creation, all sequences in the test set are checked against the model, and the average NLL distance is reported for both the training and the test set.

Training and test sets can be specified in up to two files each: `train`, `train2`, `test`, and `test2`. At most `Nseq` sequences will be read from any one file, so that at most `4Nseq` sequences will be read in if four files are specified. The `buildmodel` program ignores zero-length sequences in the training set file(s) after printing a warning.

The system can also randomly partition sequences into the training and the test set. If `Ntrain` is set, the system will randomly pick `Ntrain` sequences from all files specified (training and testing) using the random seed `trainseed`, and reserve the rest for the test set. By default, the seed is set to the process ID number, which is printed on the output file so that the partition can be reproduced. Sequence partitioning and model training use different random seeds, though both default to the process ID.

Several other programs, such as `hmmscore` and `align2model`, take an arbitrary number of sequence database files specified as `db`. Unlike most variables, repeating the `db` declaration adds a new file to the list, rather than replacing the previous database file. Zero-length sequences are processed the

same way as all other sequences. The related variables, `id` and `not_id`, restrict sequences to those with specific identifiers, or exclude those with specific identifiers. If both `id` and `not_id` are used on the same command line, sequences that are selected by `id` but are not selected by `not_id` will be used.

8 Regularizers and models

The SAM system handles a type of hidden Markov model that was developed specifically for biological sequences. It consists of a chain of ‘nodes’, each of which consists of a ‘match’ state, an ‘insert’ state, and a ‘delete’ state (Figure 1 on page 16). The only way the structure can be varied is in the length, *i.e.*, how many nodes the model has. There are three transitions out of each state, which can be taken with some probability. One of these transitions leads to the insert state in the same node, whereas the others lead to the match and delete states in the next node. Two states are special: the begin state (numbered 0) and the end state (numbered $L + 1$ for a model of length L). The model is completely specified when all the probabilities are given for all transitions and all the letters in the match and insert states.

8.1 Regularizers

The word *regularizer* is often used in (Bayesian) estimation for a method to keep estimates from overfitting the data, and in Bayesian statistics it is tightly connected with the so-called *prior* distribution. We use a Bayesian method of model estimation, and we have chosen to let the regularizer play several important roles in the program. The regularizer should reflect your prior expectations about how a model will look like for the family you are about to model. For instance, one may not think a model that only uses inserts and deletes is a good one, and that expectation can be built into the regularizer.

The regularizer has three functions:

Regularizer: During model estimation the regularizer should make sure that the model does not diverge too much from your expectations. It is done by adding ‘fake’ observations to the real ones. The model is re-estimated by ‘counting’ how many times each probability parameter is used by the data, and then normalizing these counts. Before the normalization the ‘fake’ counts in the regularizer are added.

Initial model: By normalizing the regularizer, a valid model is obtained. If no initial model is specified to the program it will use this normalized regularizer as a starting point, but usually some noise is added first (see below).

Noise: The normalized regularizer also determines the noise added both initially and during learning (if annealing is used). See below.

Therefore, to run the program, you always have to specify a regularizer. Some good default ones are shipped with the program, so you need not worry about it in the beginning. With some experience however, it can be used as a powerful tool for guiding the learning process.

A	0.08713	C	0.03347	D	0.04687	E	0.04953	F	0.03977
G	0.08861	H	0.03362	I	0.03689	K	0.08048	L	0.08536
M	0.01475	N	0.04043	P	0.05068	Q	0.03826	R	0.04090
S	0.06958	T	0.05854	V	0.06472	W	0.01049	Y	0.02992

Table 1: Default amino acid match-state frequencies for protein regularizer.

	DSSP	EHL	EHL2	EHTL
E	0.104	0.104	0.110	0.104
H	0.159	0.178	0.1781	0.178
T	0.058			0.058
S	0.047			
G	0.019			
B	0.006			
I	0.0001			
C	0.103	0.2141	0.208	0.1561

Table 2: Default frequencies for secondary structure regularizers.

The default RNA and DNA regularizer assumes a uniform distribution over the four letters, while the default protein and secondary structure regularizers use the frequencies specified in Tables 1 and 2 (the other reduced-character secondary structure alphabets similarly sum the aliased characters). Often it is a good idea to use the actual letter frequencies in the training data instead of the default distribution. This can be achieved by setting `Insert_method_train` to 1.

Without editing all the numbers in the regularizer, one can change the strength of it by changing some parameters called ‘confidences’. All regularizer numbers corresponding to transitions leaving the delete states are multiplied by the parameter `del_jump_conf` before being used. Similarly for the parameters `ins_jump_conf` and `match_jump_conf`. The numbers corresponding to the letter probabilities in match states and delete states are multiplied by `matchconf` and `insconf` respectively.

8.1.1 Regularizer alternatives

For training protein sequences, we always recommend the use of a Dirichlet mixture prior, which is enabled by setting `prior_library` to the name of a prior library. The prior library (discussed in the Sjölander et. al. paper mentioned in Section 1) encapsulates information about what distributions are expected to be found in match states. That is, columns in a multiple alignment are not all drawn from the same background distribution: some columns are highly conserved, others are primarily hydrophobic, and so on. The SAM distribution includes both the mixture from the CABIOS paper as well as several other prior libraries created by Kevin Karplus (karplus@cse.ucsc.edu), and others can be downloaded from the web site

<http://www.cse.ucsc.edu/research/compbio/dirichlets/index.html>.

Here are some of the included Dirichlet mixtures:

recode4.20comp This was previously Karplus's favorite mixture for HMMs intended for finding distantly related proteins, superseding `recode1.20comp`, `recode2.20comp`, and (for about a year) `recode3.20comp`. This prior was used in SAM-T99.

The `recode4.20comp` mixture was re-optimized from `fournier-fssp.20comp` on the `fssp-3-5-98-select-0.8-3.cols` data set, to minimize the errors in estimating distributions from samples of 1, 2, or 3 amino acids. It differs from earlier mixtures in the *recode* series in predicting a broader distribution for any given set of counts. It does a better job of matching distributions structural alignments than earlier alignments in the series.

The `recode3.20comp` mixture may still be more appropriate for modeling close homologs, as it does slightly better on the SAM-T98 alignments. (Note: this may be an artifact, as the SAM-T98 alignments were built using either the `recode3.20comp` mixture or the similar `recode2.20comp` mixture.)

recode3.20comp Karplus' current favorite, and the default SAM protein prior starting with version 3.4.

Optimized for the SAM-T98 alignments built for all the leaves of the FSSP tree (version from 3-5-98). The sequences were weighted to obtain an average information content of 1.0 bits/column (relative to using background frequencies). The optimization was to minimize the encoding cost of the observed distributions given small samples from them.

This regularizer was trained on a dataset that included columns with few counts, so it probably overestimates the probability of residues being conserved.

uprior9.plib The 9-component library discussed in the aforementioned paper. Optimized for unweighted blocks data.

null.1comp A one-component regularizer with tiny alpha, to get effectively no regularization.

rsdb-comp2.32comp.gen A 32-component prior, the default used by `genseq` and `hmmscore` random sequence calibration for proteins. Do not use this prior for building HMMs; it is only meant for generating random protein sequences. See Section 10.12.3 on page 143 and Section 10.2.10 on page 119.

t99-ebghtl-comp.6comp.gen A 6-component prior, the default used by `genseq` and `hmmscore` random sequence calibration for the EBGHTL alphabet. Do not use this prior for building HMMs; it is only meant for generating random secondary structure sequences. See Section 10.12.3 on page 143 and Section 10.2.10 on page 119.

t99-2d-comp.9comp.gen A 9-component prior, the default used by `genseq` and `hmmscore` random sequence calibration for the EHL2 alphabet. Do not use this prior for building HMMs; it is only meant for generating random secondary structure sequences. See Section 10.12.3 on page 143 and Section 10.2.10 on page 119.

The `protein_prior` command, in conjunction with another file that specifies the prior library, can be used to ensure that a prior library is used whenever protein analysis is performed.

The distributions of nucleic acids do not lend themselves to effective use of Dirichlet mixture priors. We have not yet created Dirichlet mixtures for the secondary structure alphabets.

When a prior library is used, it overrides the match-state character emission values of the regularizer. Similarly, the insert-state character emission values of the regularizer are by default overridden to be the geometric average of the match state probabilities. Thus, as a result of the historical development

of this code, for protein sequence analysis, only the transition probabilities of the regularizer are actually used in training. Again, the distribution contains several different transition regularizers optimized for different purposes, all created by Kevin Karplus. With Version 2.0, the default protein transition regularizer has been changed to `trained.regularizer`, good general regularizer. The old values are in the `sam1.3.regularizer` file of the `lib` directory.

trained.regularizer Regularizer optimized for unweighted transition counts on some set of re-estimated HSSP alignments

cheap_gap.regularizer Makes gap opening and closing very cheap allowing exploration of many different alignments, but giving too high a cost to long matches

long_match.regularizer Assigns somewhat reasonable gap costs for unweighted data, useful for sweeping away "chatter" into big matches and big gaps, by making gap opening expensive but gap extension fairly cheap.

We intend to further evaluate nucleotide regularizers in the future.

Prior libraries and regularizers can be specified by their path name. If the `$PRIOR_LIBRARY` environment variable is set to a path name including a trailing `'/'` (or if it was not set but the proper directory was specified at compile time), SAM will check that directory for prior libraries and regularizers.

Mixtures and regularizers make the biggest difference for small training sets. The file `globins50.seq` contains 50 globins. To test this, generate and score two models from four sequences:

```
buildmodel train4 -train globins50.seq -randseed 0 -trainseed 0
-ntrain 4 -priorlibrary 0
buildmodel train4reg -train globins50.seq -priorlibrary recode1.20comp
-regularizerfile weak-gap.regularizer -randseed 0 -trainseed 0 -ntrain 4
hmmscore train4 -i train4.mod -db globins50.seq -sw 2
hmmscore train4reg -i train4reg.mod -db globins50.seq -sw 2
```

Note here that the four training sequences are also in the test set.

The results of these two runs, as well as two similar ones with 10 training sequences, are shown in Figure 14, in the form of score histograms. See Section 10.11.1 on page 137. Note how in both cases, the use of the optimized regularizer improves scores of the globin sequences, and also that in both cases, the jump from 4 sequences to 10 sequences greatly improves model scores.

8.1.2 Transition regularizers with structural information

Just as the Dirichlet mixtures were used to incorporate prior information about amino acid distributions in the match states of an HMM, one can now use analogous information concerning the transition probabilities in various structural environments. To derive this information, we built HMMs from about 1050 HSSP database files and aligned the sequences that made up the file back to the HMM. Using sequence weighting and noting the structural environment, we generated weighted counts for transitions in every structural environment. Structural environment was defined in terms of secondary structure and accessibility. From these weighted counts we derive pseudocounts and incorporate them when building an HMM. The net effect of this is to impose general structural

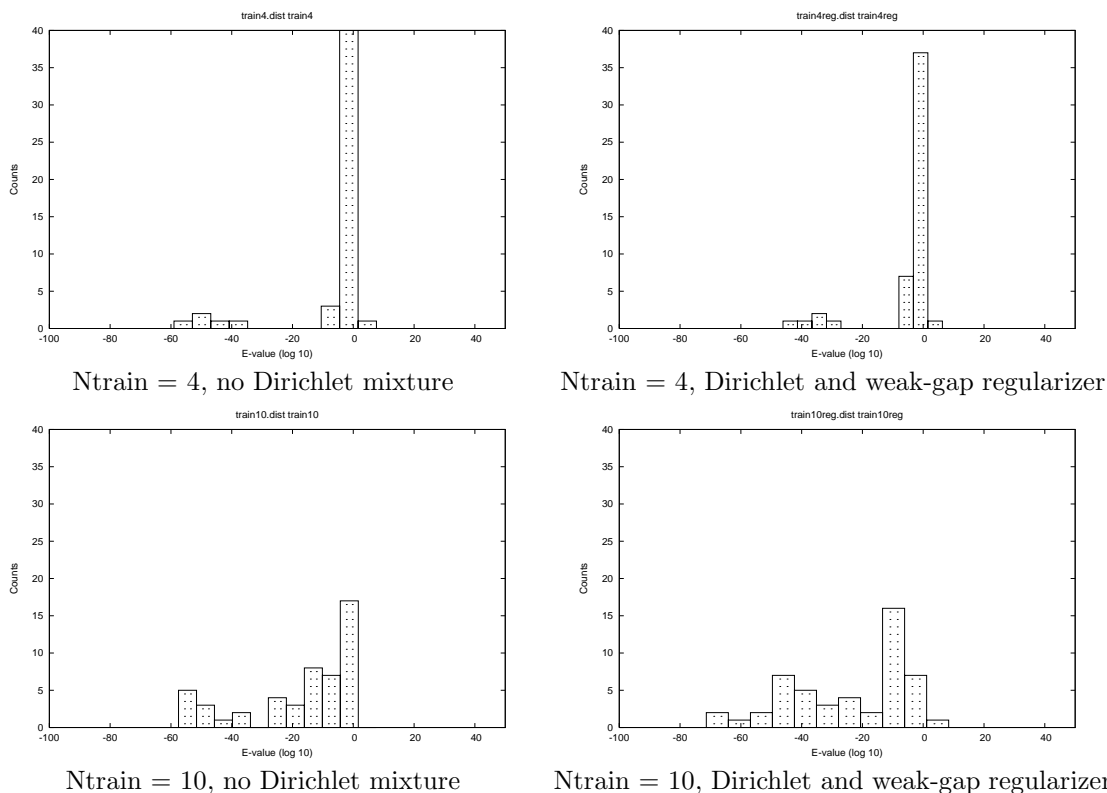


Figure 14: Regularizer performance (also see Figure 7 on page 26).

information, such as the low probability of an insert into the middle of a helix, into the HMM estimation process.

There are three relevant parameters. The first is the specification of the structural transition prior library, which one specifies with `trans_priors`. The library incorporated into the current SAM suite is `TransFromRev15.plib`. In order to use this library, one must specify a template file with `template`. This is a three-column file: amino acid sequence, secondary structure, and accessibility (as defined by HSSP). During model estimation, the sequence in the template file is aligned to the HMM. The alignment of the template sequence to the HMM dictates the assignment of the values in the second and third columns of the template file to each model node. These values in the last two columns specify a structural environment, whose pseudocounts are used to re-estimate the node's transition parameters. One may change the influence of pseudocounts with a real-valued multiplier using the parameter `transweight`.

The program `make_template` is included with the SAM distribution for generating template files from HSSP files.

The following is a command line example involving the use of the transition prior library `TransFromRev15.plib`, the template file for the PDB structure `2prd`, and the weight multiplier.

```
buildmodel 2prd -train 2prd.training.seqs -priorlibrary recode1.20comp
-transpriors TransFromRev15.plib -template 2prd.tplate -transweight 2.5
```


above. That is done by specifying the model explicitly in the initialization file, by using the heading ‘MODEL’ instead of ‘REGULARIZER’ that starts the regularizer specifications. See section 8.4, below. Most other programs in the SAM package also take already-formed models as input.

If desired, the first model in some other file (which might have a keyword other than ‘MODEL’) can be read using the `model_file` directive. See Section 6 on page 47.

8.3 Initial alignment

One of the best ways to train a hidden Markov model is to use an existing rough alignment to get the process started. There are two equivalent ways to do this. First, a model could be generated using the `modelfromalign` program (Section 10.7). Second, an alignment file can be specified on the `buildmodel` command line using the `alignfile` directive. In this case, any initial models are ignored in favor of this starting alignment.

The format of the alignment file is determined automatically as follows. First, if the key letters ‘HSSP’ begin the files first line, it is read in as an HSSP file. Second, the `align2model` (a2m) format is checked. In this case, lowercase letters are treated as insertions, periods are ignored, and uppercase letters and hyphens refer to match columns. If all sequences do not have the same number of match columns under these assumptions, the sequences are checked for a general alignment format, in which all upper and lower case letters count as match columns, and all periods and hyphens count as deletions. If this fails as well, SAM will continue on using this last format, but will print error messages about the sequences with non-matching lengths.

The `alignment_weights` parameter can be used to specify a file of weights, or `aweight_method` can be used to internally calculate sequence weights based on the initial alignment. If both are set, the external file is used. See Section 9.4 on page 80.

As a subcase of an initial alignment, `buildmodel` can be instructed to create models from randomly chosen single sequences in the training set. This is done by setting the `sequence_models` to a value greater than 0. For each of the initial models required by `buildmodel`, a random sequence will be chosen and a model created based on that sequence regularized with a weight equal to the value of `sequence_models`. As long as fewer models are created than sequences in the training set, a different sequence will be chosen for each model. Noise will be reduced according to `retrain_noise_scale`.

8.4 Model format

Regularizers and models are specified by one set of numbers for each node in the structure. One can also specify a generic node for nodes not specified explicitly (internal nodes or the special Start and End states). The simplest model (for DNA) looks like this:

```

MODEL
alphabet DNA
Generic
dd md id
dm mm im
di mi ii
mA mG mC mT
iA iG iC iT
ENDMODEL

```

where `dd`, `md`, and `id` are numbers specifying probabilities of transitions INTO the delete state from delete, match and insert respectively. Similarly, `dm`, `mm`, and `im` are probabilities for the transitions INTO a match state and `di`, `mi`, and `ii` into insert. (The states come in the order: delete (`d`), match (`m`), and insert (`i`)). In Figure 1 on page 16, $T(m_3|d_2)$, for example, corresponds to the `dm` entry of node 3. The next four numbers (`mA`, `mG`, `mC`, `mT`) are the letter probabilities in the match state, *i.e.*, probabilities for producing the letters A, G, C, and T. Similarly the last four are the letter probabilities for the insert state. Here DNA was assumed; there would be 20 probabilities for proteins in each of the last two groups, in the alphabetical order of the single-letter amino acid abbreviations given in Section 7.1. Wildcards do not have corresponding entries in either the match or insert tables: their probabilities are calculated by SAM.

A model of length 4, in which all nodes are different, looks like this:

```

MODEL
alphabet DNA
0 0 0 0 0 0 0 0 mi ii 0 0 0 0 iA iG iC iT
1 0 md id 0 mm im di mi ii mA mG mC mT iA iG iC iT
2 dd md id dm mm im di mi ii mA mG mC mT iA iG iC iT
3 dd md id dm mm im di mi ii mA mG mC mT iA iG iC iT
4 dd md id dm mm im di mi ii mA mG mC mT iA iG iC iT
5 0 0 0 dm mm im 0 0 0 mA mG mC mT 0 0 0 0
ENDMODEL

```

The first number in each line is the model position (the node number). Position 0 is the begin state, and position `length+1` (5 in the example) is the end state.

In the two first positions (0 and 1) and the last (5) some probabilities are zero. These will always be set to zero by the program, whether or not a number different from zero is specified. Referring to Figure 1, the begin and end states look like match states, but really only match beginning-of-sequence and end-of-sequence, rather than real characters. In the case of position 0, initial insertions are allowed (the `mi` and `ii` transitions), as are transitions to the next position's match or delete states. Since position 0 has no delete state, the `dd`, `dm` transitions for position 1 are zero (the `di` transition is between d_1 and i_1 in the figure).

At position 5, all sequences are required to match the implicit end-of-sequence. Because the end position has no insert or delete states, all transitions into node 5's insert or delete state are zero.

The use of regularizers is discussed in section 8.1. A regularizer specification looks exactly the same as a model specification, except that it starts with 'REGULARIZER' instead of 'MODEL'. Frequency count output (`print_frequencies`) is similarly formatted, but with the starting word 'FREQUENCIES'. A trained model can be turned into a user-specified NULL model (See Section 10.2 on page 100.) by replacing 'MODEL' with 'NULLMODEL'. Any text on the same line as the initial word is ignored — `buildmodel` places a brief comment after the word 'MODEL'.

When specifying regularizers and models, it is sometimes convenient to specify the first and last node differently than the remainder. Since the length of the model can vary, the final node cannot be specified as being, for example, node 100. Instead, one can use *negative* numbers to specify nodes relative to the end, rather than the beginning. For example,

```
REGULARIZER
Generic .....
Begin .....
1 .....
3 .....
-2 .....
-1 .....
End .....
ENDMODEL
```

'Begin' (or anything beginning with 'B') is synonymous with node number 0, and 'End' with the end node. If this regularizer is used with a model of length 100, node number 0, 1, 3, 99(-2), 100(-1), and 101 (End) will be specified individually, and for all the rest of the nodes the Generic specification would be used.

The `buildmodel` program adds two informational nodes to models it produces. The first, called 'LETTCOUNT', has the distribution of characters in the set of training sequences. The letter counting procedure adds a small offset to avoid zero counts. Wildcard counts are proportioned among the appropriate letters according to the distribution of non-wildcard letters. The second, 'FREQAVE', has the average frequency of each letter in the match states. If the match states are only modeling a portion of the training sequences, these averages may be different from the 'LETTCOUNT' values. These nodes can be used as null models during the scoring procedure, and during future `buildmodel` runs. See Section 10.2 on page 100.

It is often easiest to specify regularizers by changing an existing regularizer. For example, the default protein regularizer can be printed out to the model file by setting `dump_parameters` to 1.

```
buildmodel params -a protein -train trna10.seq -dump_parameters 1 -reestimates 0
```

This command writes all parameter values to the file `params.mod` using the protein alphabet (several alphabet warning messages will be printed because the sequences are not protein sequences). The last argument is required to ensure that `buildmodel` constructs a regularizer.

The `params.mod` file contains among other lines, the following regularizer specification (several digits have been truncated):

```
REGULARIZER: Initial setting
alphabet protein
GENERIC 1.89 0.25 0.38 1.82 15.52 3.76 0.23 0.27 4.01
0.16 0.04 0.11 0.12 0.07 0.12 0.07 0.11 0.13 0.14
0.06 0.11 0.07 0.10 0.11 0.17 0.15 0.14 0.03 0.07
0.16 0.04 0.11 0.12 0.07 0.12 0.07 0.11 0.13 0.14
0.06 0.11 0.07 0.10 0.11 0.17 0.15 0.14 0.03 0.07
ENDMODEL
```

The numbers are in order, the transition probabilities, the 20 match state values, and the 20 insert state probabilities. The match and insert state values correspond to those in Table 1. Versions of SAM before Version 1.1 had a uniform distribution in the insert states, rather than a background

distribution. To change to a uniform distribution for insert states, but maintaining the default transition regularization, the following could be placed in a parameter file (or the `insert_method_train` variable could be used, as discussed in Section 8.1):

```
REGULARIZER: Background in match, 1/20 in insert
alphabet protein
GENERIC
1.886984 0.254944 0.376488
1.819972 15.521340 3.764209
0.225758 0.265967 4.006562
0.162339 0.037220 0.107508 0.123557 0.074544
0.122092 0.072662 0.112151 0.128548 0.138534
0.063912 0.113368 0.074824 0.103722 0.110612
0.170739 0.154307 0.143584 0.028017 0.069302
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05 0.05
ENDMODEL
```

Alphabet (or `alphabet_def`) specification within a model or regularizer is optional, though whenever `buildmodel` prints one of these structures, the name of the alphabet will be included.

8.4.1 Model length

The length of the model(s) can be determined by the program in several ways (listed in order of importance):

- If there is an initial alignment (`alignfile`), the length of the alignment will be used.
- If there is an initial model of fixed length (no `GENERIC`) or `modellength` is 0, the initial model's length will be used.
- If there is an initial regularizer of fixed length (no `GENERIC`), the regularizer's length will be used.
- If the value of `modellength` is greater than 0, all models will be of that length.
- If instead `maxmodlen` is set to greater than 0, model lengths will be chosen randomly between `maxmodlen` and `minmodlen`.
- If `maxmodlength` is left at its default 0 and `modellength` is set to 0, and no initial model is specified, all model lengths will be set to the average length of the training sequences.
- If `maxmodlength` is left at its default 0 and `modellength` is left at its default value of `-1`, then model lengths will be randomly chosen between 90% and 110% of either the initial model length (if present) or the average training sequence length (if no initial model is present).

When model lengths are randomly selected, it is done with the same random number generator that creates model noise (distinct from the random number generator used to divide sequences into the training and the test set).

The surgery heuristic may lengthen or shorten a model. See Section 9.2 on page 76.

8.4.2 Special nodes

There are several special node types that can be used to hand-tune a model. These are indicated with type declarations within the model description, such as

```
TYPE 29 NO_SURGERY
TYPE 12 KEEP
TYPE 1 FIM
TYPE -1 F
```

Here, the two parameters are the node number (a negative node number indexes from the end of the model, as above) and a type. Type declarations may appear anywhere within the model specification, and in any order. If more than one specification for a node appears, the last one is used. Only the first character of the type matters, and it must be one of 'N', 'K', 'T', 'A', or 'F'.

To tie a type declaration to a specific node (rather than a generic node), its number must match that of the node declaration. That is, if a model consists of node declarations for nodes 1...5, a 'TYPE 6 F' statement will either generate an error if there is no generic node declaration, or create a FIM node number 6 using the generic node specification.

Model output from `buildmodel` is fully specified: the model will include begin and end nodes and a sequence of positively-numbered nodes. If you wish to change change node types of a node, you must specify the exact same positive node number as that node. Specifying, for example `TYPE -1 FIM`, will result in an error because there is no generic node, and node -1 has not been specified. To achieve this effect, you will need to add either a generic or a new node description (for node -1) in addition to the type statement. Alternatively (and preferably), you can use the `addfims` program or in some cases the `auto_fim` variable to add a FIM at the start and end of the model.

Type 'N' nodes are no-surgery nodes. During the surgery heuristic, these nodes will neither be deleted (if they are used by too few sequences) nor expanded (if their insert states are used by too many sequences). The parameters of the node will be trained as normal. No-surgery nodes are usually not used explicitly: they are a building block upon which keep and FIM nodes are based.

Type 'K' nodes are keep nodes. The match and insert probabilities of these states will not be trained, however their transition probabilities will. Keep nodes are also immune to surgery.

Type 'T' nodes are transition keep notes. The *outgoing* transitions associated with that node are not trained but the match and insert tables are. Note that the outgoing transitions for a node include the incoming transitions to that node's insert state as well as the incoming transitions to the next node's delete and match states. Transition keep nodes are immune to surgery.

Type 'A' nodes are all keep nodes. This has the attributes of both 'K' and 'T', though of course one could not specify both 'K' and 'T' because nodes can only have one type. The outgoing transitions and the character distributions are not trained. All keep nodes are immune to surgery. In general, if you do not wish to train a node (for example, it is a conserved region from an existing alignment that is known to be correct), the nodes of that region should be of type 'A'. The 'K' and 'T' nodes are for more specialized use, such as learning transition probabilities for an existing profile.

Keep nodes can be particularly useful when training from an existing model or alignment (using the `modelfromalign` program, Section 10.7). If a region is identified as being particularly important to preserve during training, its nodes can be identified as keep nodes (determining which node

numbers correspond to the region can be done using `drawmodel` or `align2model`). For example, if node numbers 10–12 are identified as being ‘correct’ (and thus should not be trained), the lines

```
TYPE 10 K
TYPE 11 K
TYPE 12 K
```

should be added between the `MODEL` and matching `ENDMODEL` statements. After training, the kept nodes may no longer be numbered 10–12 because of model surgery, however the nodes will still be part of the model, and will be identified as kept nodes in the model output. The program `modifymodel` can be used to change node types.

Type ‘F’ nodes are free-insertion modules (FIMs). See Section 8.5 on page 71.

If the program is being run without an initial model, node types are taken from the regularizer. If there is an initial model, types in the regularizer are ignored preference to any types present in the model.

8.4.3 Node labels

Node labels used by constraints are specified using `NODELABELS` directives. The form of this directive is

```
NODELABELS firstnode- lastnode: firstlabel
```

Where *firstnode* and *lastnode* specify a contiguous range of nodes to label and *firstlabel* is the first label to use. For example,

```
NODELABELS 10-22: 79
```

See Section 9.7 on page 87.

8.4.4 Codon models

Codon models are standard SAM models based on the `codon` alphabet. Codon models are a mixture of nucleotide and codon states, but nodes always include 64 match numbers and 64 insert numbers. For nucleotide nodes, only the first 4 numbers are read by SAM, so if the first 4 numbers are all, for example, 0.25, a flat distribution among the amino acids will be used. Each codon is represented by 3 model nodes in the HMM with types C, D, and E (in order). These 3 nodes are used to represent a single codon node. The Node C match state contains the 64 probabilities (or regularizer values) of the different codons in in the order AAA, AAC,...TTG, TTT. The Node C insert state is not used. The node D match state automatically becomes a free match state (unity probability for any character emission), and the node D insert state is not used. The node E match state is similarly free, and the node E insert state emits nucleotides in the normal way.

This model architecture allows the nucleotides in codons to have a joint probability in the C nodes. It does not, however, enable the insertion of complete codons; insertions are always of nucleotides. This architecture was chosen as one that could be somewhat readily implemented in SAM and

enabled the mixed treatment of nucleotides and codons (a numeric alphabet could be used for pure codon models that do not include wildcards).

8.4.5 Binary models

Files containing models can be written in either text (human readable) or binary format. You can recognize a binary model by the keyword `BINARY` which appears on the line directly after the model declaration. The advantages of using binary formatted models are decreased file size/disk space usage and faster model reading and writing. The disadvantage is that you can't read or modify your model files.

By selecting binary format, you can reduce the size of an 81-node model file from 16,240 bytes to 8726 bytes. A file containing 249 nodes shrinks from 116,218 bytes in text format to 51,647 bytes. We achieve this striking savings by taking advantage of the fact that nodes of a given type frequently have the same letter probabilities for the insert state.

Before a binary model is written to file, the program scans all the nodes of each type. If it finds the nodes all have the same letter probabilities for insert state, the data is stored in a table. The probabilities are considered 'identical' if their difference is less than .000002, therefore they are rounded to this magnitude in the binary model.

If you run `buildmodel` and use binary output you are likely to shave a few seconds from the program's run time.

When generating model files, use the command-line option `-binary_output 1` for binary format. The default for this option is currently set to 0 or off.

The program `hmmconvert` is available to switch models from one format to another. It will read your model, determine its format and then write the model to a new file in the opposite format. See Section 10.10.3 on page 131.

8.5 Free-insertion modules

It is often desirable to be able to model motifs that occur in long sequences. This can be done with a free-insertion module. The idea is to have a model of the motif flanked by insert states with particular character distributions. The cost of aligning a sequence to such a model does not strongly depend on the position of the motif in the sequence, and thus it will pick up the piece of the sequence that fits the motif model the best.

These insert states are added to the model by the so-called free-insertion modules (FIMs), in which only the delete state and the insert states are used. All the transitions from the previous node go into the delete state, which is ensured by setting the other probabilities to zero. From the delete state there is a transition to the insert state with the probability set to one. In the insert state, character probabilities are set according to `FIM_method_train` (for `buildmodel`) or `FIM_method_score` (for `hmmcore`). Use of these parameters allows you to embody information particular to the problem domain and can significantly affect performance. Discussion on their use appears in Section 10.2. From the delete and insert states there are transitions to the next node which also have unity

probability (delete to match, insert to match, delete to delete, and insert to delete). Note that the probabilities do not sum up to one properly in such a module. Since all sequences must pass through a FIM's delete node (excepting the case of when the Begin node is a FIM), a delete (or dash) will be present in any alignment to the model. (This delete is automatically removed whenever `auto_fim` is set, whether or not the FIMs were already there.)

The FIM is also used as the null model during scoring. (See Section 10.2 on page 100.)

These special nodes can be used to learn, align, or discriminate motifs that occur once per sequence. Typically, FIMs are used at the beginning and the end of a model to allow an arbitrary number of insertions at either end. For example, if a model for a motif has been learned from truncated sequences, adding FIMs to the resulting model will enable detection of that motif within longer sequences. Before trimming sequences by hand, one should try learning with FIMs, as in

```
REGULARIZER
GENERIC .....
TYPE 0 FIM
TYPE -1 FIM
ENDMODEL
```

The begin node (number 0) can be used as a FIM as shown above. However, the end node has no insert state, so the FIM is put just before the end, which is specified as node `-1`. (See Section 9.6 for some important comments on adding FIMs to models without the `addfims` program, as in the regularizer example above.)

Given a sequence, an HMM will only identify one occurrence of the domain or motif on which it was trained even if there are multiple copies. Multiple occurrences can be found with `multdomain`.

To train a model to find several (different) motifs, one can add FIMs at different positions in the model. For instance to model sequences with two motifs of lengths 5 and 10 one should specify the model as

```
REGULARIZER
GENERIC .....
TYPE 0 FIM
TYPE 6 FIM
TYPE -1 FIM
ENDMODEL
```

and set the model length equal to 17 (5+10+2FIMs).

If domains are clipped from an alignment, converted to a model using `modelfromalign`, and then to a FIM model using `addfims`, it is best to model several positions on either side of the domain to prevent the FIMs from eating up the ends of the domain.

The FIM state's insert table (or the generic node's, if the FIM is not fully specified) has the distribution over characters to be used.

The program `addfims` will add free insertion models to both ends of a model. See Section 10.3 on page 122. Because the correct addition of a FIM requires changes in the transition probabilities to and from the FIM, it is recommended that users only add FIMs by hand for unspecified regularizers and models: those that only have a generic node and one or more type declarations. To add free

insertion modules to the ends of an existing model, for training or for motif searching, be sure to use the `addfims` or `modifymodel` program, or that `auto_fim` is set to its default value of 1.

Sometimes, in alignment or training the model may not be using FIMs as much as desired when, for example, there is a reasonably strong probability of using a specific internal insert state. The probability of a FIM modeling n characters is by default the product of the insert table probabilities for those characters. If this probability is too low, meaning that sequences are not using the FIM enough, it can be adjusted with the `fimstrength` parameter. Changing this parameter from its default 1.0 to 2.0, for example, make use of the FIM twice as likely as before. The value of `fimstrength` is also applied to simple null models, and if less than zero, the absolute value of `fimstrength` is applied to both FIMs and normal insertion states.

Similar to `fimstrength`, `fimtrans` can be used to adjust the cost of the insert to insert loop within the FIM, and possibly within insert states in the model. If set to a non-zero value, `fimtrans` becomes a multiplier of the geometric average of the match to match transitions in the model. Thus, if set to 1.0 (the default), the FIM insert-to-insert transition will cost the same as the average match to match transition. When set to a negative number, non-FIM insert-to-insert transitions are set to $p - (1 - f)p^2$, where p is the regularized and normalized frequency counts for the transition and f is the FIM insert-to-insert transition. The effect of this formula is to ensure that the FIMs remain cheap in comparison to any insert-to-insert transitions, something that can be critical in using `buildmodel` to train HMMs.

8.6 FIM, insert and match tables

To aid in the manipulation of insert tables, SAM provides several options to globally change the regularizer's match and insert states, and the initial model if it is generated from the regularizer. These options are controlled by `FIM_method_train` and `insert_method_train`. The default is to use the residue counts of the training sequences in both the insert and FIM states, as well as the match states of the GENERIC node.

- 0 Use the tables present in the model.
- 1 The relative frequencies of residues in the training sequences (from the `LETTCOUNT` node or the training sequences).
- 2 The relative frequencies of residues in model match states (from the `FREQAVE` node).
- 3 Uniform (flat) probability over all residues.
- 5 Amino acid background frequencies over all proteins (from the `GENERIC` node).
- 6 Geometric average of the match state probabilities.

The match state frequency average is only available when an existing model (with a `FREQAVE` node) is being trained.

If the FIM method number is negative, the change is used when FIMs are being added by the program, as when `SW` is 2 (local) or 3 (domain), or `auto_fim` is 2 (always add). When the FIM method number is 0, added FIMs use the geometric average as if it were -6 .

If `insert_method_train` is set, and a generic node exists in the regularizer or model, then both the match and the insert states of that generic node are changed. This is important because it means that match states in the initial model (before training commences) will be based on, for example if `insert_method_train` is 1, the letter counts of the current training set, assuming that there is no initial model or alignment.

If the `train_reset_inserts` variable is set, then after each re-estimation cycle in the training process, the insertion and FIM tables will be set according its value: 0, 1, 2, 3, 5 as above, or 6 (the default) to set to the geometric average of the match table probabilities in the newly trained model. See Section 10.2.1 on page 102.

9 The buildmodel estimation process

A detailed discussion of the estimation process can be found in “Hidden Markov models in computational biology: Applications to protein modeling,” mentioned in the Introduction. This section provides an overview of the mechanics of model estimation.

After the sequences have been divided into training and test sets, and the initial model or models have been created, `buildmodel` will iteratively train the model using expectation-maximization (EM). For each iteration, a comment line (beginning with a percent sign ‘%’) is written to the output file (e.g., `test.mod`) that includes the iteration number and the average NLL distance between the set of training sequences and the model. Iterations continue until either an iteration gains less improvement than the `stopcriterion` (and noise is less than 10% of its starting value) or `re-estimates` iterations have been performed. When multiple models are being trained (but not multiple subfamily models) training on each model is stopped individually when that model reaches the `stopcriterion` (provided noise is less than one tenth its initial value).

If surgery and multiple initial models are used, one model is selected for the surgery procedure, which will attempt to prune and grow the model as appropriate. After each surgery procedure (up to `nsurgery`), the re-estimation process is repeated. Once either the limit on the number of surgeries is reached, or the surgery parameters produce no model modifications, the training procedure is complete.

After the model has been trained, the NLL scores for the test set are computed and reported, and the final model is written to the output file. This model file may be used as an input file to further refine the model, perhaps by setting the `stopcriterion` to a smaller value.

9.1 Noise and annealing

It is possible to add noise to the initial model(s). By setting `initial_noise` to a positive number that amount of noise is added to a model in the beginning of the program. It serves the important purpose to make models differ, if the program runs many models simultaneously — each model will have a different noise added.

To try to avoid local minima, one can add noise to the models during their estimation, and decrease the noise level gradually in a technique similar to the general optimization method called

simulated annealing. The initial level of the noise in this annealing process is called `anneal_noise`. If `anneal_noise` is greater than 0, annealing is performed. (If `initial_noise` is also given, that will determine the noise for the first iteration, and `anneal_noise` the noise in the following iterations.) During the estimation process the annealing noise is decreased by a speed determined by `anneal_length`. There are two ways it can be done:

Linearly: If `anneal_length` is greater than or equal to 1, the noise is decreased linearly to zero in `anneal_length` iterations by the formula

$$\text{noise} = \text{anneal_noise}(1 - \text{number_of_iterations}/\text{anneal_length})$$

Exponentially: If `anneal_length` is less than 1, the noise is decreased exponentially by multiplying the noise with `anneal_length` in each iteration, which gives the noise

$$\text{noise} = \text{anneal_noise} * \text{anneal_length}^{\text{number_of_iterations}}$$

In the exponential schedule, noise injection is halted when the amount of noise reaches 10% of its initial value.

Once the noise level has been calculated, there are three possible ways noise can be added, as controlled by whether `randomize` is positive, negative, or zero.

positive A set of `randomize` random paths are calculated through the model according to the regularizer probabilities. Each of these sequences is weighted by the amount of noise. These sequences are added to the counts generated by the real frequencies, thus the noise setting is somewhat dependent on the number of sequences being trained.

negative A set of `-randomize` random paths are calculated through the model. With a weight of `-noise/randomize`, these counts are added to the *normalized* (probability, rather frequency) model, and then the model is renormalized. Thus, the noise generation is similar to that of the first case, but total noise added is independent of both the number of sequences and the `randomize` setting.

zero For each probability parameter in the model, a random number between 0 and the corresponding parameter in the normalized regularizer is found. This number is scaled by the level of the noise, given for instance by `initial_noise`, and added to the probability in the model. After doing this for the whole model, all the probabilities are normalized. For example, if the noise is a random number between 0 and 2, random pseudo counts corresponding to up to two sequences will be added to each transition and each match state. This is the fastest means of noise generation.

Note that the annealing schedules are *ad hoc*. Still, according to our experience even fast and crude annealing generally improves performance. By default, exponential noise at a ratio of 0.8 is used with no `initial_noise`, an `anneal_noise` of 5, and a `randomize` setting of 50 (corresponding to 50 random sequences). These values were chosen experimentally (see the Hughey and Krogh paper mentioned in the introduction).

After a model has been created, adding too much noise to the model may eliminate all the trained information. Therefore, if an initial model or an initial alignment is specified, noise (`initial_noise`

or `anneal_noise`) is reduced from the default setting by a factor of `retrain_noise_scale`, which has a default of 0.1. Thus, the effective noise during a retraining would be 0.5 rather than 5. The same is true of surgery iterations, discussed in the next section. In this case, the starting noise of the re-estimation process after a surgery, whether or not an initial model is specified, is the `anneal_noise` scaled by `surgery_noise_scale` parameter, which also has a default of 0.1.

A second annealing option, one based on slowing increasing the weights of the sequences being trained is discussed in Section 9.4.2.

9.2 Frequency-based Surgery

It is often the case that during the course of learning, some match states in the model are used by few sequences, while some insertion states are used by many sequences. *Model Surgery* is a means of dynamically adjusting the model during training.

Surgery will be `nsurgery` times: a full re-estimation process is performed including `reestimates` re-estimations, or until the `stopcriterion` is reached. By default, as in the tRNA example above, surgery is performed up to two times.

The basic operation of surgery is to delete unused match states and to insert match states in place of over-used insert states (the special node types described in Section 8.4.2 are never subjected to surgery modification). In the default case, any match state used by less than one half of the sequences is removed, forcing those sequences to use an insert state or to significantly change their alignment to the model.² Similarly, any insert state used by more than half sequences is replaced with a number of match states approximating the average number of characters inserted by that insert state.

The surgery heuristic is based on frequency counts of the training set. can be adjusted with one parameter or with three. In the first case, setting `mainline_cutoff` to a number other than the default 0.5 will indicate how much non-match, or main line, activity will be accepted. For example, a setting of 0.25 indicates that any match state used by less than one quarter of the sequences should be removed, while any insert state used by more than one quarter of the sequences should be expanded into a number of match states approximately equal to the average number of characters emitted by that state.

For finer tuning of the surgery process, the parameters `cutmatch`, `cutinsert`, and `fracinsert`, can be used. During surgery, any match state with a smaller portion of sequences than `cutmatch` is removed, and any insert state with a higher portion of sequences than `cutinsert` is replaced by the average number of characters emitted by that insert state multiplied by `fracinsert`. By default, `fracinsert` is 1.0, and `cutmatch` and `cutinsert` are both equal to `mainline_cutoff`.

These parameters can be set in ways that cause large amounts of surgery. For example, setting `cutmatch` to 0.5 and `cutinsert` to 0.25 will delete any match state used by fewer than half the sequences, and insert match states for any insert node used by greater than one quarter of the sequences. Typically, this will result in an oscillating model in several positions — those positions used by more than one quarter and less than one half of the sequences. Such excessive surgery can sometimes aid in forming a good model.

²To be more precise, any node that has a frequency count of less than one half the number of sequences is removed.

9.2.1 Sequence-based Surgery

If a specific sequence, such as one of known structure, is being modeling using SAM (or a SAM-target method such as SAM-T04), that sequence can be used to guide the surgery process. The process requires a guide alignment in a2m format, the FASTA style format in which lowercase letters indicate insertion states, uppercase match states, and hyphens deletes. See Section 7.2 on page 56.

For each surgery, the guide sequence will be aligned to the model being trained. This alignment and the guide alignment are then analyzed. The model is modified in wherever the guide sequence did not align to the model according to the desired alignment (unless special nodes like Keep or FIM are present). If the guide alignment specifies a delete state but the actual alignment has none, a node is inserted at that position. If the guide alignment specifies a character emission but the actual alignment has a delete, then that model node is removed. If the guide alignment specifies an insert and the actual alignment has a match, that match state is removed and counts are merged into the appropriate insert.

Sequence-based surgery has two parameters: `syncfile` and `syncweight` ('synch' works as well). A `syncfile` must be specified for sequence surgery, and should contain the guide sequence in a2m format (the `id` option can be used to select one aligned sequence among multiple). The `syncweight` parameter specifies the weight to be assumed for the guide sequence as the surgery procedure moves frequency counts around to reflect the changing model. The value of `syncweight` is added to the appropriate freq count of emit states, and also to the transitions into and out of that state. The `synchseq` is not subtracted from the frequency counts of the whole model.

9.3 Training statistics

In addition to the trained model, a report of the training procedure is included in `buildmodel`'s output. The comment sections of this file for the training example in the introduction is reproduced below.

```

% SAM: buildmodel v3.5 (July 15, 2005) compiled 07/15/05_11:25:30
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%           Sequence Alignment and Modeling Software System
%           http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% test  Host: peep    Fri Jul 15 13:44:57 2005
% rph   Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Internal sequence weighting method 1
% Regularizer FIM_method_train training letter frequencies (1)
% Regularizer Insert_method_train training letter frequencies (1)
% Model initial FIM_method_train training letter frequencies (1)
% Model initial insert_method_train training letter frequencies (1)
% Generic, Insert, and FIM tables dynamically reset to
%   train_reset_inserts geometric mean of match probabilities (6)
% All models generated from regularizer.
% Subsequence-submodel (local) (SW = 2)
% Simple scores adjusted by +1.5*ln(seq len) (adjust_score = 2)
% Track 0 FIMs added (geometric mean of match probabilities (6))
%
% Model lengths:      80  79  79
% Itera-   Average
% tion     distance
%   1  -127.420  -121.451  -121.122
%   2   92.146   92.922   93.749
%   3   93.191   91.936   92.998
%   4   91.954   91.229   92.048
%   5   91.148   91.389   91.788
%   6   91.023   91.038   91.014
%   7   91.316   91.283   91.188
%   8   89.859   91.487   90.998
%   9   88.451   90.918   91.147
%  10   84.391   90.674   90.794
%  11   78.354   90.067   88.379
%  12   73.763   87.607   84.589
%  13   71.775   78.423   80.192
%  14   69.969   72.381   76.562
%  15   64.308   67.253   72.399
%  16   63.440   66.310   71.728
%  17   63.244   65.898   71.588
%  18   63.188   65.678   71.567

```

```

%   19   63.188   65.556   71.567
%   20   63.188   65.489   71.567
% Model 0 (counting from 0) wins!!
% Surgery, New mod. length 79
%   D79
% Model modified by surgery (final length 79)
%   21   71.858
%   22   70.867
%   23   70.910
%   24   71.094
%   25   71.271
%   26   69.693
%   27   68.843
%   28   67.645
%   29   67.066
%   30   66.897
%   31   66.408
%   32   66.025
%   33   65.675
%   34   65.430
%   35   64.803
%   36   64.748
% Surgery, New mod. length 79
%   I131 D78
% Model modified by surgery (final length 79)
%   37   74.809
%   38   73.426
%   39   71.162
%   40   67.854
%   41   67.148
%   42   67.316
%   43   66.981
%   44   66.172
%   45   65.253
%   46   64.251
%   47   64.172
% Surgery, New mod. length 78
%   D78
% Model modified by surgery (final length 78)
%   48   72.597
%   49   72.030
%   50   72.905
%   51   70.537
%   52   67.432
%   53   66.213
%   54   66.584
%   55   65.625
%   56   65.356
%   57   64.895
%   58   64.202

```

Here, the initial information includes program version and run information. In this case, no initial models were specified, so `buildmodel` created 3 models (the default value of `Nmodels`) from the regularizer of the specified lengths. Next, all three models are trained, and the one with the best score is selected. In this case, the best model did not require any surgery, so the process was

complete. If the best model did need surgery, that single model would be further refined. Next, statistics on the scores of the training sequences (and test sequences, if present) and CPU time are presented, followed by non-default parameter settings. The `randseed` entry is commented out to prevent any future training iterations from reusing the old seed. Finally, the model, along with its generic, letter count, and frequency average nodes is printed. The model has not been included in the example.

If the `many_files` variable is set, then the results of `buildmodel` are broken up into three files: the `.stat` file contains the run statistics and parameter settings, the `.mod` file contains the final model, and the `.freq` file contains the frequency model if `print_frequencies` is set.

9.4 Weighted training

SAM is able to perform a variety of weighted training options. Sequence weighting is particularly important when, as normal, the sequence data given to SAM is biased toward some type or sub-family of sequences (for example, from those organisms that have been most studied). Prior to Version 2.0, the SAM software system did not include any internal sequence weighting schemes, but could use weights generated by some other program. Version 2.0 added two internal weighting methods described below, the first of which is turned on by default when external weighting is not used. Version 3.0 includes internal weighting of alignments with several algorithm choices, which is the preferred form of sequence weighting.

9.4.1 External weighting

For all external sequence weighting options, a sequence weights file is specified with either the `sequence_weights` variable (for `buildmodel` training sets) or the `alignment_weights` variable (for `buildmodel` or `modelfromalign` initial alignments). In this file, any line starting with a percent sign (%) is ignored as a comment. The first non-comment line is presumed to be a description of the weights file, for example including the program that generated the sequence weights. The next non-comment line contains two integers, the number of weighted sequences and the number of weighted subfamilies. Remaining uncommented lines consist of a sequence identifier, white-space, and floating-point sequence weights, one per family. Weights can be positive, negative, or zero, and need not sum to one. If a sequence does not have a corresponding weight, its weight is set to 1.0 and a message is printed. If a weight does not have a corresponding sequence, a message is printed. Sequences and weights do not need to be in the same order within their respective files.

For plain sequence weighting, the number of families is set to 1, and each sequence is assigned a single weight in the `sequence_weights` file. During the re-estimation cycle, the frequency counts for each sequence will be multiplied by its weight.

Sequence weighting is particularly important in database discrimination: without sequence weighting, the model may specialize to an over-represented subset of the sequences, meaning that family members that do not happen to be in that sub-family will receive low scores.

The file `gseg4.seq` contains the initial 70-character segments of each of 4 globins. The last three are quite similar.

```

;
BAHG$VITSP
mldqqtiniikatvpvlkehgvttttfyknlfakhpevrplfdmqrqesleqpkalamtvlaaaqnien
;
GLB$APLJU
alsaadagllaqswapvfansdangasflvalftqfpesanffndfkgksladiqaspklrdvssrifar
;
GLB$APLKU
slsaaeadlvgkswapvyankdadganfillsfekfpnnanyfadfkgksiadikaspklrdvssriftr
;
GLB$APLLI
slsaaeadlagkswapvfanknangadflvalfekfpdsanffadfkgksvadikaspklrdvssriftr

```

When a model is trained on this file without weighted training, the model is specialized to the latter group of sequences, resulting in the following scores:

GLB\$APLLI	70	-140.08	-132.49	1.15e-57
GLB\$APLJU	70	-140.16	-131.50	3.12e-57
GLB\$APLJU	70	-134.77	-127.94	1.09e-55
BAHG\$VITSP	70	-83.81	-77.65	7.53e-34

The following simple weight file is an attempt to correct this bias:

```

% gseg4.weights
Weight file for the simple globin segment example.
% 4 sequences and 1 family
4 1
BAHG$VITSP 2.0
GLB$APLJU 0.66
GLB$APLJU 0.66
GLB$APLLI 0.66

```

Note that in this weight file, to make the results of the two examples comparable, the weights were made to sum to 4. The reason for this is that in addition to sequences, the regularizer (provided the various confidence parameters are non-zero) shapes the model. Setting all sequence weights uniformly high (e.g., 100.0) will have a similar effect to setting all the regularizer confidences to 0.

With the simple weight file, the following scores are produced.

BAHG\$VITSP	70	-115.63	-109.50	1.12e-47
GLB\$APLJU	70	-110.64	-104.34	1.94e-45
GLB\$APLLI	70	-110.38	-103.67	3.81e-45
GLB\$APLJU	70	-105.85	-96.93	3.19e-42

Here, with the three similar sequences weighted less, the model better matches (perhaps too much) the dissimilar sequence.

The above example is definitely a toy problem: weights must be set using statistically and biologically valid means.

9.4.1.1 Multi-subfamily weighting *Warning: This feature is not completely available or completely debugged.*

A particularly interesting use of weighting schemes is when a family of sequences can be divided into several subfamilies. This special type of training is used whenever the number of families in a weights file is greater than one.

In this case, SAM will train one model per family in parallel so that each model can specialize to its subfamily. Although this sounds just like training each subfamily separately, there is an important difference. During the regularization procedure, the counts across all subfamily models are taken into account when re-estimating each subfamily's model. This means that, in the case of multiple alignments, a full-family multiple alignment can be generated by aligning each sequence to its appropriate subfamily model and combining the results. At the moment, subfamily modeling is not fully implemented and not recommended for use.

There are a few changes in the functionality of `buildmodel` when subfamily modeling is used. First, only a single suite of subfamily models is trained at a time, so multiple runs must be performed to match the functionality of starting with more than one model and selecting the best. Second, prior libraries must be used. Third, a more conservative approach to model surgery is taken. The subfamily models are always modified in parallel, and only if all the subfamily models agree on the surgery procedure (if all subfamily models believe inserting new model positions is appropriate, the minimum of all proposed insertion lengths is used). To encourage more surgery, users may wish to lower the surgery thresholds when training with multiple models. See Section 9.2 on page 76.

Also, model files are treated somewhat differently. The `many_files` option is always turned on. The subfamily models are writing to files named, for example, `runname.3.mod`, where the number indicates which subfamily (starting from zero) that model is for. It is possible to retrain a suite of subfamily models by setting `family_base_file` to the root name of the suite of models (i.e., `runname` in the above example).

The `hmmscore` program does not yet score against multiple models—to perform database search against a suite of models, `hmmscore` must be run independently for each subfamily, and then the results combined by, for example, classifying each sequence as a member of the subfamily with which it scored best.

9.4.2 Annealing with Weights

Sequence weights can also be an effective means of annealing (Section 9.1) during the training process. When using this option, the sequence weights are slowly increased over the first several re-estimation cycles. Thus, at first, the sequences will have little effect on the model for the next training iteration: the regularizer and prior library will dominate, though particularly strong signals in the sequences, such as strongly conserved regions, will show through. As the training continues, the sequence weight multiplier is brought up to its final value, giving full weight to the sequences.

The annealing schedule options are similar to that available with noise generation. The relevant parameters are `weight_length`, which indicates how long the annealing should last, and `weight_final`, indicating the final sequence weight multiplier (the default is 1.0). The sequence weight multiplier found by the formulas below is multiplied by the sequence weight (which is 1.0 if no weight file is used in training) to find each sequence's weight during the given re-estimation iteration.

Linearly: If `weight_length` is greater than or equal to 1, the weight multiplier is increased linearly

to `weight_final` in `anneal_length` iterations by the formula

$$\text{multiplier} = \text{weight_final} * \text{number_of_iterations} / \text{weight_length}$$

Exponentially: If `weight_length` is less than 1, the multiplier is increased exponentially until the multiplier reaches 90% of its final value as follows:

$$\text{multiplier} = \text{weight_final} * (1.0 - \text{weight_length}^{\text{number_of_iterations}}).$$

Sequence weights are never zero: the first re-estimation cycle uses the first non-zero value of the weight formula.

9.4.3 Internal weighting of alignments

Version 3.0 of SAM incorporates the weighting methods used in the SAM-T99 remote homology detection method. See Section 4 on page 24.. Relevant parameters include: `aweight_method`, for which 0 indicates no internal alignment weighting, 1 indicates Karplus relative weighting, 2 Henikoff relative weighting, and 3 flat relative weighting; `aweight_bits` indicating the target number of bits to save per column; and `aweight_exponent` indicating the weighting exponent. The Dirichlet `prior_library` and `alignfile` specified for `buildmodel` or `modelfromalign` are the remaining components of the method.

For the Henikoff and flat weighting schemes, if `aweight_bits` is set negative, the initial weighting is used, without adjusting the total weights to get a specified number of bits saved. The initial weighting in these schemes has a total weight of

$$\text{number_sequences}^{\text{aweight_exponent}}.$$

By default, internal weighting of alignments is turned on, with Henikoff relative weighting, 0.5 target bits per column, and a 0.5 exponent. If an `alignment_weights` file is specified, the external weights are used, and unless `aweight_method` is set to zero, a warning message is printed.

9.4.4 Internal weighting during training

Version 2.0 of SAM introduced two methods of internal weighting during `buildmodel` training. These methods are based entirely on the log-odds score of the sequence against the model being trained. Their invention was motivated by HMMer's maximum-discrimination training method. In general, these methods do not produce as good results as the alignment-based weighting.

Given a linear hidden Markov model M , a dynamic programming calculation can be used to calculate, for a given sequence, the probability that sequence a was generated by the model, $P(a|M)$. The question of interest is, however, does that model match the sequence? That is, is the sequence more likely to be generated by the hidden Markov model than some other, less structured null model, ϕ . Making the assumption that the models M and ϕ are *a priori* equally likely, this reduces to the log-odds probability of

$$P(M|a) = \frac{p(a|M)}{p(a|M) + p(a|\phi)}.$$

Typically, a log-odds score S is used instead [2]:

$$S_a = \ln \frac{P(a|M)}{P(\phi|a)}$$
$$P(M|a) = \frac{1}{1 + e^{-S}}$$

This score measures whether the probability that the sequence was generated by the model is greater than the probability it was generated by a null model. This log-odds score is used to calculate the weight of a sequence. As model training iterations proceed, sequences that poorly match the model (i.e. with poor log-odds scores) are given higher weight.

Internal weighting method 1 uses the following equation to calculate a sequence weight.

$$W = e^{(w-S)*(\frac{\ln K}{(w-b)})}$$

S is the log-odds score of the sequence. The program keeps track of the current worst score w and the current best score b and these are used to decide on the two extreme weights. The worst scoring sequence will have a weight of 1, while the best scoring sequence will have a weight of K , typically in the 0.01 to 0.1 range. K is a user-controlled parameter entered on the command line as *iweight*.

To use method 1 with $K=.1$, run `buildmodel` with the following arguments:

```
buildmodel runname -train train.seq -internal_weight 1 -iweight .1
```

Internal weighting method 2 is a variation of method 1. When using method 1, sequences with very poor scores may get excessively large weights. Method 2 modifies the weights of such outlier sequences. If a sequence scores so badly that it exceeds the median score by three standard deviations, it is weighted with a decreasing linear weight function, reaching a minimum of 1.0 for the sequence with the worst score.

Method 2:

$$W = e^{(w-S*\frac{\ln K}{(w-b)})}$$

is applied to scores no more than 3 deviations below the median, while

$$W = 1 - \frac{S - w}{b - w}$$

is applied to the remaining sequences.

To use method 1 with $K=0.05$, run `buildmodel` with the following arguments:

```
buildmodel runname -train train.seq -internal_weight 2 -iweight 0.05
```

The current SAM default is to not use internal weighting. If internal weighting is selected and no *iweight* parameter entered, SAM defaults to an *iweight* of 0.1.

Looking again at the toy problem example demonstrated in the previous section, we saw the following scores when a model was trained on 4 globins without weighting.

GLB\$APLLI	70	-140.08	-132.49	1.15e-57
GLB\$APLKU	70	-140.16	-131.50	3.12e-57
GLB\$APLJU	70	-134.77	-127.94	1.09e-55
BAHG\$VITSP	70	-83.81	-77.65	7.53e-34

The unweighted model is overspecialized.

Internal weighting method 1 produces these scores:

GLB\$APLKU	70	-116.19	-106.17	3.12e-46
GLB\$APLLI	70	-115.78	-104.93	1.07e-45
GLB\$APLJU	70	-109.16	-100.39	1.01e-43
BAHG\$VITSP	70	-64.82	-57.58	3.95e-25

Internal weighting method 2 produces these scores:

GLB\$APLKU	70	-114.58	-106.42	2.43e-46
GLB\$APLLI	70	-114.28	-105.78	4.61e-46
GLB\$APLJU	70	-107.72	-100.04	1.42e-43
BAHG\$VITSP	70	-81.04	-74.33	2.09e-32

When using internal weighting, you can inspect all sequence weights generated during each iteration of the model building process.

```
buildmodel runname -train train.seq -internal_weight 2 -iweight 0.05 -print_all_weights
1
```

The `print_all_weights` option when set to 1 will produce a weight output file once per iteration. The files are named `runname1.weightoutput`, where 1 is the iteration number.

By default, `print_all_weights` is set to off.

Continuing the example of Figure 14 on page 63, performing the two commands:

```
buildmodel train4w -train globins50.seq
    -randseed 0 -trainseed 0 -ntrain 4 -internalweight 2
buildmodel train4wreg -train globins50.seq
    -priorlibrary recode1.20comp
    -regularizerfile weak-gap.regularizer
    -randseed 0 -trainseed 0 -ntrain 4 -internalweight 2
hmm score train4w -i train4w.mod -db globins50.seq -sw 2
hmm score train4wreg -i train4wreg.mod -db globins50.seq -sw 2
```

results in the score histograms of Figure 15, in which the scores improve even further from the use of regularizers and Dirichlet mixtures.

9.5 Viterbi training

For increased speed and performance (and possible worse results), Viterbi training is now possible (though in some instances not robust). The `dpstyle` parameter should be set to 1 for Viterbi training, rather than the default 0, which indicates EM training. It can also be set to 4 or 5 to train based on the posterior-decoded alignments (these options will more than double computation time and are not recommended). See Section 10.1 on page 90. Two other values should only be used for

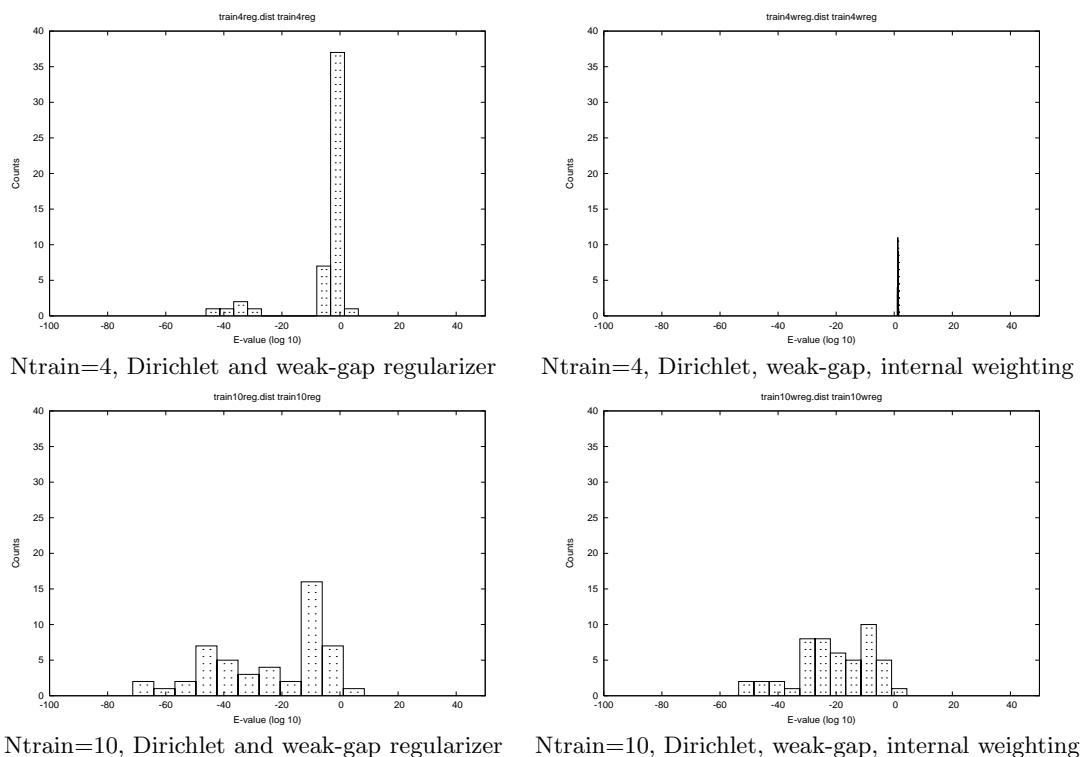


Figure 15: Weighting performance

scoring. Setting `dpstyle` to 2 will produce a very large dump of the posterior probabilities of the entire dynamic programming table in the file `runname.pdoc`. This option should be used in conjunction with the `grabdp` program. See Section 10.5 on page 124. Setting `dpstyle` to 3 will produce a frequency dump for each sequence against each model as scored or aligned, using the EM dynamic programming method (also for use with `grabdp`). Files will be of the form `runname.id.freq`.

The related `adpstyle` parameter can be used to choose between Viterbi (1), posterior-decoded alignment based on transitions and character emission posteriors (4), and posterior-decoded alignment based solely on emission posteriors (5) for alignments and multiple domain alignments. See Section 10.1 on page 90.

Future research will help determine the usefulness of these option. Fragment training (Section 10.1.2) with the `SW` variable can also be used with both the Viterbi and EM training algorithms, but has not been completely evaluated.

9.6 Global and semi-local training

Please first read forward to the discussions about fragment alignment and scoring. See Section 10.1.2 on page 92 and Section 10.2.4 on page 107.

As with `align2model` and `hmmscore`, there are various options for global or local training using the

SW variable. SW can be set to 0 (global training: all of the sequence matches all of the model), 1 (semi-local: part of the sequence matching the entire model), 2 (local: part of the sequence matching part of the model), or 3 (domain: part of the sequence matching all of the model). As with scoring, the default is 2, fully local..

As with `hmmscore` and `align2model`, the FIMs that SAM internally added to the model are not included in the output model.

9.7 Building models with constraints

SAM HMM constraints provide a mechanism for associating position in a sequence with a specific node in a model. During model training, a constrained position remains associated with its node, allowing the remaining portions of the sequence to align naturally to the model. This serves as a method of incorporating prior knowledge about the training sequence, such as structurally similar regions.

Currently, there is an inconsistency in scoring constraints, in that for EM-style scoring, the constrained score is reported, which will generally provide a very strong signal. For Viterbi or posterior-alignment scoring, constraints are used to find the best path, but the unconstrained score of that path is reported, and thus the score of a constrained sequence is similar to that of an unconstrained sequence.

9.7.1 Model Node Labels

The node a residue is constrained to is specified using a *node label*. A node label is a unique, positive integer assigned to a node. Often it is the same as the node number, however this is not required. This is useful for allowing model surgery to add or remove a node. It is specified using the `NODELABELS` declaration in a model (see Section 8.4.3).

Node labels must be increasing in value from from left to right in the model. Labels are normally specified as contiguous numbers for a set of contiguous nodes to allow for easier constraint definition for contiguous residues. Labeled nodes will not be removed by model surgery; it is possible to have nodes added between labeled nodes.

When creating a model from an alignment, SAM will define node labels automatically.

9.7.2 Constraints Definition File

Constraints for individual residues are specified in line-oriented constraint definition files separate from the sequence databases. By convention, they have an extension of `.cst`. These files are specified using the `-constraints` option, which may be specified multiple times. Certain SAM programs can create new constraint definition files using the `-constraints_out`.

Lines containing only whitespace and those where the first non-whitespace character is “#” are comments and ignored. For all types of lines, leading white spaces are ignored.

The file is divided into sections starting with a line containing only the section name followed by a colon. Currently there is only one type of section containing sequence constraints, starting with

```
constraints:
```

Multiple constraints sections may appear, allowing for the easy combining of files.

A constraint entry defines the model nodes to which residues of a sequence are constrained. Multiple entries may appear for a sequence, however a given residue of a sequence may be constrained only once. Constraints entries consists of lines in the form:

```
seqid constrdef, constrdef, ...
```

Where the `seqid` is any sequence identifier consisting of non-whitespace characters and `constrdef` defines a set of model node labels for a range of positions in the sequence in any combination of the following formats:

- `startpos-endpos: startlabel`
Specifies that consecutive residues starting at `startpos` and ending at `endpos` are constrained to consecutive node labels starting at `startlabel`.
- `seqpos: label`
Constrains a single position to a node label.
- `startpos-endpos`
Specifies that these residues are to be constrained, but doesn't actually associate a label with the positions. This is used when the labels are to be assigned from an alignment by `modelfromalign`.
- `seqpos`
Specifies a single residue to be constrained from an alignment.
- *empty*
An entry may be empty (zero or more white space) and is ignored. This can be useful when generating constraint files.

Positions in sequences are identified by one-based indices and node labels are positive numbers.

A simple example of a constraint file is shown here.

```
constraints:  
PROT1 10-30: 8, 80-82: 100  
PROT2 12-28: 10, 62: 100  
PROT3 16-32: 12
```

9.7.3 Using Constraints

To train a new model using constraints, specify one or more constraints file using the `-constraints` option to `buildmodel`. The node labels in the constraints file are assigned to nodes with the same node number. If necessary, the length of the model will be increased to allow for all labels. If

surgery is used, an unlabeled node may be added or removed, resulting in the labels and node numbers differing in the final model. When retraining a model using constraints, the model must already contain labels. Constraints work with both EM and Viterbi

When creating a model from a multiple alignment with `modelfromalign` or using the `-alignfile` parameter to `buildmodel`, constraints can be created for specific positions or for all match positions. Constrained positions are specified using `-constraints` option. Model labels need not be specified and are ignored if they are. The residues specified in the file will be constrained to the nodes to which they are aligned. Insert sequences positions will not be constrained. To constrain all match positions in an alignment without creating a constraints definition file, use `-constraints_from_align`. With either approach, a new constraints definition file maybe required as input to other SAM programs. A new constraints file for the alignment sequences can be created using the `-constraints_out` option.

Constraints can be used when creating multiple-domain alignments with the `hmmscore` program (See Section 10.2.5 on page 111.). The constraints for the sequence being aligned are specified with the `-constraints` option. The model must have node labels that correspond to the constraints file; thus the constraints file are normally the ones used to train the model or created when the model was defined from an alignment. If the extracted domains are to be processed by another SAM program, a constraint definition file for the domains can be written to a file specified by the `-constraints_out` option.

9.8 Reducing buildmodel runtime

Training a model can be a be a time-consuming process. Each re-estimation cycles through all sequences in the training set, performing a dynamic programming algorithm with operations proportional to the product of the total number of characters in the training set and the length of the model. Then, there can be large numbers of re-estimations, making some runs take overnight.

Firat, using Viterbi training speeds buildmodel by a factor of 5 over EM training, and is used in the SAM-T04 process.

Alternatively, shorter execution times (and possibly worse models or alignments) can be had in several: a hard limit can be placed on the number of `reestimates`, or the `stopcriterion` can be increased, though both of these can decrease model quality. Similarly, the number of surgeries can be reduced. One of the most effective ways to reduce runtime is to simply reduce the number of sequences in the training set. A small, well-chosen training set, in which close homologs have been eliminated, can produce better models than a larger, random training set.

If a run seems to be taking too long, it is possible to tell `buildmodel` to save the next model as a prelude to killing the program. The two UNIX signals, `SIGUSR1` and `SIGUSR2`, can be used to toggle the `print_surg_models` and `print_all_models` variables. In the first case, models are printed after each surgery procedure, and in the second, after each re-estimation cycle.

10 Related programs

10.1 align2model and prettyalign

After you have obtained a model of your sequence family, `align2model` can be used to give a multiple alignment of sequences. Often one is just interested in aligning the sequences that were also used to train the model, but in principle any sequence can be included in the alignment.

The multiple alignment is obtained by aligning each of the sequences to the model by the Viterbi algorithm. This has the advantage that it can be done for each sequence independently, and therefore it is very simple to add new sequences to the multiple alignment. Also, once the model is found, the multiple alignment is very fast and easy to produce.

The program to produce the basic alignment is called `align2model`. Calling it with no arguments gives a brief explanation. To align all the sequences in `trna10.seq` use the command:

```
align2model trna10 -i test.mod -db trna10.seq
```

This will put an intermediate form of the alignment in the file `trna10.a2m`. In this FASTA-compatible intermediate format deletions are shown as dashes ('-') and insertions (produced in the insert states of the model) are shown as lower case characters, while periods ('.') are used to fill in the sequences that did not have any insertions if `a2mdots` is set to 1, the default.

In case you only want to align a few sequences in a large file, you can specify the identifiers of these sequences on the command line. For instance

```
align2model trna2 -i test.mod -db trna10.seq -id TRNA1 -id TRNA9
```

will only align the two specified sequences. The output (in `trna2.a2m`) would look like this:

```
>TRNA1
ggGGAUGUAGCUCAG-.UGGUAGAGCGCAUGCUCUG.CAUGUAUGAGGcC
CCGGGUUCGAUCCCCGGCAUCUCCA-----
>TRNA9
cgGCACGUAGCGCAGCcUGGUAGCGCACCGUCCUGGgGUUGCGGGGU.C
GGAGGUUCAAUCCUCUCUGCCGACCA----
```

The `not_id` option can be used to exclude specific sequences from being aligned, and has precedence over the `id` option.

The Viterbi algorithm finds the single best path for the alignment — that which has the highest probability. The SAM system can also find the single most *likely* path. Internally, this corresponds to using the EM algorithm to evaluate all possible paths, calculating the probability of each residue appear in each model state and the probability of use for each transition (the posterior probabilities). A second (Viterbi-style) dynamic programming is then performed on these probabilities to find the most likely path (see, for example, Holmes and Durbin in Recomb98).

SAM includes two flavors of posterior-decoded alignment. When `adpstyle` is 4, posteriors are calculated using a *global* (SW is 0) forward calculation. Then, the minimizing path through the transition *and* character emission posteriors is calculated using the specified SW mode.

When `adpstyle` is 5, posteriors for character emissions are calculated using the specified global or local SW mode. Then, the minimizing path for character emission posteriors *only* is calculated using a global dynamic programming mode.

These alignment methods are much slower than standard Viterbi alignment, and have not yet been implemented for reduced memory use. The `maxposdecodemem` specifies the numbers of bytes that may be allocated in to this calculation. If the sequence is too long to perform this alignment calculation in the space specified by `maxposdecodemem`, a warning messages printed and the Viterbi algorithms used.

10.1.1 `prettyalign`

To get a nice display of the alignment produced by `align2model`, you can use the `prettyalign` program, which has several display options. The program reads from a file like the one made in the example above:

```
prettyalign trna10.a2m > trna10.pretty
```

which would give you an alignment similar to the one shown in the Section 3

`Prettyalign` does not follow SAM's normal commandline format. To see an explanation of the various options, run the program with some invalid option (like `prettyalign -h`). Some of the most useful options are

- f** Print in a FASTA-like format.
- i** Do not include sequence identifiers in front of each line.
- l *num*** Set the output line length equal to *num*.
- n** Toggle indexing the sequences, as well as labeling them.
- c** Toggle column numbering.
- m** Set maximum insertion length (longer insertions are printed as their length).
- I** I-G style alignment. Also sets maximum insertion length very high.
- b** Generate BLAST-style alignments.

The commands

```
align2model trna3 -i test.mod -db trna10.seq -id TRNA1 -id TRNA2 -id TRNA9  
prettyalign trna3.a2m -l 50 > trna3.pretty
```

gives the following output

```

          10          20          30
          |          |          |
TRNA1  ggGGAUGUAGCUCAG-.UGG..UAGAGCGCAUGCUCUG.CAUG
TRNA2  gcGGCCGUCGUCUAGU.CUGgauUAGGACGCUGGCCUCC.CAAG
TRNA9  cgGCACGUAGCGCAGCcUGG..UAGCGCACCGUCCUGGgGUUG
          40          50          60          70
          |          |          |          |
TRNA1  UAUG.AGGcCCCgGGUUCGAUCCCCGGCAUCUCCA-----
TRNA2  CCAGcAAU.CCCgGGUUCGAAUCCCCGGCGCGCGCA-----
TRNA9  CGGG.GGU.CGGAGGUUCAAAUCCUCUCGUGCCGACCA----

```

The `prettyalign` program can compress long insertions to only the initial segment of bases in the insertion plus digits representing the *total* length of the insertion. For example, the sequence `GacguacguG` could be printed out as `Ga8guG` if 4 was the largest number of insertions that was to be allowed (note that the character 8 is using up one of the positions). By default, insertions of up to length ten thousand are fully printed. This can be changed with the `-m` flag to `prettyalign`, which sets the maximum number of insertions that are printed. If set to zero, no insertions are printed, and no indication of the lack is given. If less than zero, insertion characters are not printed, and that number of digits is used to indicate the length of each insertion.

The `-I` switch will create a compatible IG-style alignment file which may be converted to other formats using the `readseq` package included as a subdirectory of SAM. The `-I` option automatically sets a high value for the insertion length parameter.

The `-b` switch generate a BLAST-style output, comparing the first sequence in the alignment (by default) to each subsequent sequence, and generating pairwise alignments with a middle line highlighting identical residues and conservative substitutions. These pairwise alignments do not include terminal inserts: These pairwise alignments run from the first to last residue aligned by either sequence. They do not include terminal inserts; they include internal inserts only if some sequence in the pairwise alignment is inserting residues. The `-R` option can be used to specify a different sequence to use as this reference sequence.

10.1.2 Aligning fragments and SW

Consider a model of 100 nodes and a fragment of 25 that very closely matches some contiguous section of the model. Even though that section would align very well, the overall alignment of the fragment could be quite poor because of its need to use 75 delete states in the model. The problem in this example of global alignment is that in addition to modeling conserved regions, the model also models the length of the conserved region.

SAM's `SW` variable can be used to control the alignment type. Global alignments can be forced by setting `SW` to 0. The default (2) is local alignment similar to Smith and Waterman method of sequence comparison, which will find the best alignment for any pair of subsequences within two sequences. The same can be done with models, allowing a submodel to match a subsequence. This type of dynamic programming specified with `SW 2`, the default. In this case, sequences can jump from the initial module (presumably a FIM, automatically added when `auto_fim` is set) into the match state of any module in the model, and can also jump out of the match state of any module within the model to the delete state of the next-to-last node. The first and next-to-last module are assumed to be FIMs, hence the rationale is that a sequence will use the FIM for some period of time to consume characters that do not match the model, then the sequence will jump to the model node

corresponding to the start of the fragment, use several model nodes, and then jump to the ending FIM to consume the rest of the sequence.

The probability of these jumps is set by the variables `jump_in_prob` and `jump_out_prob`, both of which have a default value of unity. That is, as in the sequence-to-sequence Smith and Waterman, there is no cost associated with jumping in and out of the model.

Setting `SW 1` specifies a semi-local alignment. This option allows a sequence to start matching the model at any location (rather than only the begin node) and end at any location (rather than only the end node). This will improve alignment for short sequences that match a segment of the model. (Internally, no FIMs are added to the model and jumps are allowed).

For domains, `SW` can be set to 3 to match the full model to part of the sequence. (Internally, FIMs are added to the model but jumps are not allowed).

The file `trna1frag.seq` contains several sequences that contain part or all of TRNA1. The sequence include TRNA1 (72 bases), TRNA1Long (the complete tRNA with additional characters), Long (58 base segment of TRNA1), Medium (34 base segment), Short (6 base segment TRNA1), and AAMediumA, an embedding of Medium within several segments of As to bring it to 176 characters. Additionally, the file contains several (obviously) non-tRNAs of various lengths, all of whose IDs begin with the word 'Not'.

When this file is aligned to the model `test.mod`, created above, the alignment of the sequence and fragments is reasonable, but the non-tRNAs still align the entire model and may even use internal insertion states. (As shall be seen in Section 10.2.4, the scoring of these fragments with the `SW` option off is not nearly so good as their alignments).

```

          10      20      30      40      50
          |      |      |      |      |
TRNA1      gg.....GGAUGUAGCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGcCCCCGGUU
TRNA1Long  aaa13aggGGAUGUAGCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGcCCCCGGUU
Short      .....-----CAUGC-----
ShortReverse u.....---CGUA-----
Medium     .....-----GAGCGCAUGCUCGCAUGUAUGAGG.CCCCCGGU
MediumReverse uug20uac-----GCUUCGUACGCGAG-----
AAMediumAA aaa70aaa-----AGAGCGCAUGCUCGCAUGUAUGAGG.CCCCCGGU
Long       .....-----GCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGcCCCCGGUU
NotShort   a.....-----
Not        a.....-----
NotLong    a.....-----
NotExtraLong a.....-----

          60      70
          |      |
TRNA1      CGAUC CCCGGCAUCUCCA-----
TRNA1Long  CGAUC CCCGGCAUCUCCA-----aaa11aaa
Short      -----u.....
ShortReverse -----c.....
Medium     -----u.....
MediumReverse -----
AAMediumAA UAAA-----aaa68aaa
Long       CGAUC CCCGGCAU-----
NotShort   -AAA-----aaa9aaaa
Not        -AAA-----aaa69aaa
NotLong    -AAA-----aa105aaa
NotExtraLong -AAA-----aa434aaa

```

Alignment with the SW option set to 1 is much the same (again, scoring will be improved), though the alignment procedure has managed to better isolate the AAMediumAA sequence's tRNA core, modeled by match states, from its prefix and postfix, modeling by internal insertion nodes.

```

                10         20         30         40
                |         |         |         |
TRNA1          ggG.....GAUGUAGCUCAG-UGG.....UAGAGCGCAUGCUCGCAUGUAUGAGGc
TRNA1Long      ..-aaa14gggGAUGUAGCUCAG-UGG.....UAGAGCGCAUGCUCGCAUGUAUGAGGc
Short          ..-.....-CAUGCU-----
ShortReverse   ..-.....-UCGUAC-----
Medium         ..-.....-GAGCGCAUGCUCGCAUGUAUGAGGc
MediumReverse  ..-.....-UUGGgccccgGAGUAUGUACGCUUCGUACGCGAG---
AAMediumAA     ..-.....-
Long           ..-.....-GCUCAG-UGG.....UAGAGCGCAUGCUCGCAUGUAUGAGGc
NotShort       ..-.....-AAAAAAAAAAAAA.
Not            ..-.....-
NotLong        ..-.....-
NotExtraLong   ..-.....-

                50         60         70
                |         |         |
TRNA1          CCCGGGUUCGAUCCCGGCAUCUCCA-----
TRNA1Long      CCCGGGUUCGAUCCCGGCAUCUCCAAAAAAaaaa...
Short          -----
ShortReverse   -----
Medium         CCCGGGUU-----
MediumReverse  -----
AAMediumAA     -----AAAAAaa171aaa
Long           CCCGGGUUCGAUCCCGGCAU-----
NotShort       -----
Not            -----AAAAAaaa68aaa
NotLong        -----AAAAAaaa104aaa
NotExtraLong   -----AAAAAaaa433aaa

```

When alignment is performed using the SW option set to 2, only the core tRNA segments are aligned: the non-tRNAs, as well as the prefix and postfix of AAMediumAA are aligned to the FIMs that have been automatically added to the model. The one problem is that the Short sequence has made some use of the end FIM because it is not long enough to make a really significant hit to the model's internal nodes.

```

                10      20      30      40      50
                |      |      |      |      |
TRNA1          gg.....GGAUGUAGCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGcCCCCGGUU
TRNA1Long     aaa13aggGGAUGUAGCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGcCCCCGGUU
Short         .....-----CAUGC-----
ShortReverse  u.....---CGUA-----
Medium        .....-----GAGCGCAUGCUCGCAUGUAUGAGG.CCCC GGU
MediumReverse uug20uac-----GCUUCGUACGCGAG-----
AAMediumAA    aaa70aaa-----AGAGCGCAUGCUCGCAUGUAUGAGG.CCCC GGU
Long          .....-----GCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGcCCCCGGUU
NotShort      a.....-----
Not           a.....-----
NotLong       a.....-----
NotExtraLong  a.....-----

                60      70
                |      |
TRNA1          CGAUC CCGGCAUCUCCA-----
TRNA1Long     CGAUC CCGGCAUCUCCA-----aaa11aaa
Short         -----u.....
ShortReverse  -----c.....
Medium        -----u.....
MediumReverse -----
AAMediumAA    UAAA-----aaa68aaa
Long          CGAUC CCGGCAU-----
NotShort      -AAA-----aaa9aaaa
Not           -AAA-----aaa69aaa
NotLong       -AAA-----aa105aaa
NotExtraLong  -AAA-----aa434aaa

```

A different alignment is produced when `SW` is set to 2 (fully local) and `adpstyle` is set to 4 (posterior-decoded alignment with transitions). As can be seen in the example below, this option does not neatly cut the sequences to their matching positions, but may produce a better core alignment.

```

                                10      20      30      40      50
                                |      |      |      |      |
TRNA1      gg.....GGAUGUAGCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGCcCC.GGGU
TRNA1Long  aaa13aggGGAUGUAGCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGCcCC.GGGU
Short      c.....-----
ShortReverse u.....---CGUA-----
Medium     .....-----GAGCGCAUGCUCGCAUGUAUGAGGC.CC.CGGG
MediumReverse uu.....-----GGGCCCGGA-----GUAUG.UA.CGCU
AAMediumAA aaa63aaa-----AAAAAAGAGCGCAUGCUCGCAUGUAUGAGGC.CCcGGGU
Long      .....-----GCUCAG-UGGUAGAGCGCAUGCUCGCAUGUAUGAGGCcCC.GGGU
NotShort   aaaa...-----
Not        aaa18aaa-----AAAAAAAAAAAAAAAAAAAAAAAAAA.AA.AAAA
NotLong    aaa52aaa-----AAAAA-----
NotExtraLong aa218aaa-----A-----

                                60      70
                                |      |
TRNA1      UCGAUCCCCGGCAUCUCCA-----.....
TRNA1Long  UCGAUCCCCGGCAUCUCCAAAAAAAAAaaaa....
Short      -----AUGCU.....
ShortReverse -----c.....
Medium     U-----u.....
MediumReverse UCGUACGC-----gag....
AAMediumAA UAAAAAAAAAAAAA-----aaa58aaa
Long      UCGAUCCCCGGCA-----u.....
NotShort   -----AAAAAAAA.....
Not        AAAAA-----aaa20aaa
NotLong    -----aaa52aaa
NotExtraLong -----aa219aaa

```

As well as when SW is set to 2 (fully local) and adpstyle is set to 5 (posterior-decoded alignment solely on character emission).

		10	20	30	40	50
TRNA1	ggg.....	-GAUGUAGCUCAG-	UGGUAGAGCGCAUGC	UUCGCAUGUAUGAGGC	cCC.GGGU	
TRNA1Long	aaa14ggg-	GAUGUAGCUCAG-	UGGUAGAGCGCAUGC	UUCGCAUGUAUGAGGC	cCC.GGGU	
Short	cau.....	-----	-----	-----	-----	-----
ShortReverse	ucg.....	-----	-----	-----	-----	-----
Medium	g.....	-----	AGCGCAUGC	UUCGCAUGUAUGAGGC	.CC.CGG-	
MediumReverse	uug18ugu-	-----	-----	-----	-----	-----
AAMediumAA	aaa70aaa-	-----	AGAGCGCAUGC	UUCGCAUGUAUGAGGC	.CCcGGGU	
Long	gc.....	-----	UCAG-UGGUAGAGCGCAUGC	UUCGCAUGUAUGAGGC	cCC.GGGU	
NotShort	aaaaaaaa.	-----	-----	-----	-----	-----
Not	aaa36aaa-	-----	-----	-----	-----	-----
NotLong	aaa54aaa-	-----	-----	-----	-----	-----
NotExtraLong	aa219aaa-	-----	-----	-----	-----	-----
		60	70			
TRNA1	UCGAUCCCCGGCAUCUCCA	-----			
TRNA1Long	UCGAUCCCCGGCAUCUCCA	AAAAA--	aaaaaaa.			
Short	-----	-----	gcu.....			
ShortReverse	-----	-----	uac.....			
Medium	-----	-----	guu.....			
MediumReverse	-----	-----	acg16gag			
AAMediumAA	UAAA-	-----	aaa68aaa			
Long	UCGAUCCCCGGC-	-----	au.....			
NotShort	-----	-----	aaaaaa..			
Not	-----	-----	aaa37aaa			
NotLong	-----	-----	aaa55aaa			
NotExtraLong	-----	-----	aa219aaa			

Here is the posterior-decoded (adpstyle 4) global alignment (SW is 0).


```

                                10      20      30      40      50
                                |      |      |      |      |
TRNA1      ggG.....GAUGUAGCUCAG-UGGUAGAGCGCAUGCUUCGCAUGUAUGAGGCcCCGGG
TRNA1Long  aa-aaa12gggGAUGUAGCUCAG-UGGUAGAGCGCAUGCUUCGCAUGUAUGAGGCcCCGGG
Short      ..-.....-CG-----
ShortReverse u.-.....-CG-----
Medium     ..-.....-GAGCGCAUGCUUCGCAUGUAUGAGGC. CCCGG
MediumReverse ..-.....-UU---GGGCCCGGAGU-----AUG. UACGC
AAMediumAA a.-.....AAAAAAAAAAAA---AAAAAAAAAAAAAAAAAAAAAAAAAAAA. AAAAA
Long      ..-.....-GCUCAG-UGGUAGAGCGCAUGCUUCGCAUGUAUGAGGCcCCGGG
NotShort   a.-.....-AA-----AA-----
Not        a.-.....AAAAAAAAAAAA---AAAAAAAAAAAAAAAAAAAAAAAAAAAA. AAAAA
NotLong    a.-.....AAAAAAAAAAAA---AAAAAAAAAAAAAAAAAAAAAAAAAAAA. AAAAA
NotExtraLong a.-.....AAAAAAAAAAAA---AAAAAAAAAAAAAAAAAAAAAAAAAAAA. AAAAA
                                60      70
                                |      |
TRNA1      UUCGAUCCCGGCAUCUCCA-----.....
TRNA1Long  UUCGAUCCCGGCAUCUCCAAAAAAAAAaaaa...
Short      -----CAUGC.....
ShortReverse -----UAC.....
Medium     G-----UU.....
MediumReverse UUCGUA-----CGCGAG.....
AAMediumAA AAAAA-----aa125aaa
Long      UUCGAUCCCGGCA-----U.....
NotShort   -----AAAAAAA.....
Not        AAAAA-----aaa20aaa
NotLong    AAAAA-----aaa58aaa
NotExtraLong AAAAA-----aa387aaa

```

10.2 hmmscore

Any sequence can be compared to a model by calculating the probability that the sequence was generated by that model. Taking the negative (natural) logarithm of this probability gives the NLL score. For sequences of equal length the NLL scores measures how ‘far’ they are from the model, and it can be used to select sequences that are from the same family. However, the NLL score has a strong dependence on sequence length and model length. `Hmmscore` provides several less biased means of scoring by reporting NLL scores as the difference between a null model and trained model NLL score (a log-odds score, as used in HMMER).

Null model scoring is discussed in more detail in the Barrett, Karplus, and Hughey paper mentioned in the introduction and available from the SAM WWW page. (<http://www.cse.ucsc.edu/research/compbio/papers/nullmod/nullmod.html>).

The program `hmmscore` can find NLL and NLL–NULL (log-odds) scores. E-values can be calculated for the reverse sequence null model. The most common operation is to calculate NLL–NULL scores for a large number of sequences. This can be done by supplying the name of the model file and one or more sequence database files on the command line, optionally followed by `hmmscore` parameter specifications. For instance, for the example files described earlier the NLL scores are found the following way

```
hmmscore test -insert test.mod -db trna10.seq -sw 2
```

Produces the file `test.dist` already displayed:

```
% SAM: hmmscore v3.5 (July 15, 2005) compiled 07/15/05_11:25:31
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%       Sequence Alignment and Modeling Software System
%       http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% test  Host: peep    Fri Jul 15 13:45:00 2005
% rph   Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Inserted Files:  test.mod
% Database Files:  /projects/kestrel/rph/sam32/demos/trna10.seq
%
% Subsequence-submodel (local) (SW = 2)
% Simple scores adjusted by +1.5*ln(seq len) (adjust_score = 2)
% Track 0 FIMs added (geometric mean of match probabilities (6))
% Single Track Model:  test.mod
% Score DP Method: forward all-paths (dpstyle = 0)
% Align DP Method: viterbi (adpstyle = 1)
% 10 sequences, 747 residues, 78 nodes, 0.01 seconds
%
% Sequence scores selected:  All (select_score=8)
%
% Simple: NLL=NULL using FIM probabilities
% Reverse: NLL=NULL for the reverse sequence NULL model
%   Calculated when Simple < simple_threshold (10000.00)
% E-value on N (=10) sequences:
%   N / (1 + exp(-(lambda(=1.0000) * Reverse)**tau(=1.0000)))
%   Calculated when Simple < simple_threshold (10000.00)
%   Rescale E-values or use -dbsize for multiple scoring runs.
%   WARNING:  E-VALUES ARE NOT CALIBRATED!
% Scores sorted by E-value, best first
%
% Sequence ID   Length   Simple   Reverse   E-value   X count
TRNA7          73      -36.19   -28.50    4.18e-12
TRNA2          76      -35.48   -28.13    6.08e-12
TRNA4          75      -35.66   -27.90    7.65e-12
TRNA9          77      -35.54   -27.82    8.29e-12
TRNA10         76      -35.21   -27.69    9.39e-12
TRNA1          72      -34.85   -27.32    1.37e-11
TRNA8          75      -36.32   -27.26    1.44e-11
TRNA5          73      -35.10   -26.36    3.58e-11
TRNA3          76      -34.43   -26.26    3.93e-11
TRNA6          74      -34.59   -23.81    4.56e-10
```

As discussed in Section 3.4, the score file contains six columns. The first is the sequence identifier,

followed by sequence length, the ‘NLL–NULL’ score using a simple null model. The next score column is either the raw NLL score if only the simple null model is calculated (the default), or the complex or reverse sequence null model’s ‘NLL–NULL’ if one of the more time-consuming null models is used, as discussed below. If E-values are calculated, they are listed after the two score columns. Last, the number of all-character wildcards in each sequence is listed for those that have any wildcards.

By default, `hmmscore` uses the EM scoring method, just as is used to train a model. If desired, scores can be based on exact alignment to the model, multiplying the probabilities along the best path rather than all paths. This method, which corresponds to the forward half of `align2model`, can be turned on by setting `viterbi` to 1. Viterbi scoring is appropriate for finding out how good a sequence’s best alignment to a model is.

The E-value computation is based on a simple, but somewhat unrealistic, assumption that the scores for the sequence and the reversed-sequence are independent draws from an extreme-value (Gumbel) distribution:

$$P(\text{score} < a) \approx 1 - e^{-ke^{\lambda a}} .$$

Subtracting the two scores (derivation not published yet) gives

$$P(\text{diff} < a) \approx \frac{1}{1 + e^{-\lambda a}} .$$

The E-value is the expected number of sequences with that good a score, so is simply the probability of seeing that negative a difference, multiplied by the number of sequences scored (or `dbsize`, if that is specified).

The only parameter that needs to be estimated is the natural scaling λ . Since we use natural logs in computing our probabilities, we set $\lambda = 1$, which seems to be correct experimentally. Thus we compute our E-values with no parameter fitting at all—it is based purely on theoretical considerations.

The `calibrate` option of `hmmscore` can be used to calibrate the lambda values for the E-value calculation. Once a model library has been created, it can be specified to `hmmscore` using the `model_library` directive (or `modlib` alias). See Section 10.2.10 on page 119.

The `hmmscore` program can also be used to select sequences according to various criteria.

Plots of the NLL–NULL scores can be used to visually look for a break between significant and insignificant matches. See Section 10.11 on page 137.

10.2.1 NLL–NULL scoring

SAM includes several possibilities for NULL model scoring. The null model can be a simple probability distribution, effectively a model with a single FIM. The null model can be any model specified in SAM format, with the key word ‘NULLMODEL’ (rather than, for example, ‘MODEL’ or ‘REGULARIZER’), or the first model in a file specified with the `nullmodel_file` parameter. The null model can be the reversed HMM, or equivalently, the score of the reversed sequence through the original HMM.

To report differences between the model NLL score and the simple null model score (possibly modified by `FIM_method_score`, see below), set the `subtract_null` variable to 1. To report differences

between two models (for example, one trained on positive family examples and one trained on negative examples of a family), set `subtract_null` to 3. To report differences between the score of the sequence and the score of the reversed sequence, which provides an automatic adjustment for compositional bias and allows the simple calculation of E-values, set `subtract_null` to 4 (the default). To report scaled reverse null model scores, set `subtract_null` to 5. The complex null model (previously `subtract_null 2`) is no longer supported, and reverts to the simple null model.

Because the difference between the sequence and reverse sequence scores automatically adjusts for model and sequence length, SAM is able to add E-values to the score file in this case. The command

```
hmmscore testrev -i test.mod -db trna10.seq -subtract_null 4 -sw 2
```

produces a score file that includes both simple and reverse-sequence null model scores:

```

% SAM: hmmscore v3.5 (July 15, 2005) compiled 07/15/05_11:25:31
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%           Sequence Alignment and Modeling Software System
%           http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% testrev  Host: peep    Fri Jul 15 13:45:01 2005
% rph      Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Inserted Files:  test.mod
% Database Files:  /projects/kestrel/rph/sam32/demos/trna10.seq
%
% Subsequence-submodel (local) (SW = 2)
% Simple scores adjusted by +1.5*ln(seq len) (adjust_score = 2)
% Track 0 FIMs added (geometric mean of match probabilities (6))
% Single Track Model:  test.mod
% Score DP Method: forward all-paths (dpstyle = 0)
% Align DP Method: viterbi (adpstyle = 1)
% 10 sequences, 747 residues, 78 nodes, 0.01 seconds
%
% Sequence scores selected:  All (select_score=8)
%
% Simple: NLL=NULL using FIM probabilities
% Reverse: NLL=NULL for the reverse sequence NULL model
%   Calculated when Simple < simple_threshold (10000.00)
% E-value on N (=10) sequences:
%   N / (1 + exp(-(lambda(=1.0000) * Reverse)**tau(=1.0000)))
%   Calculated when Simple < simple_threshold (10000.00)
%   Rescale E-values or use -dbsize for multiple scoring runs.
%   WARNING:  E-VALUES ARE NOT CALIBRATED!
% Scores sorted by E-value, best first
%
% Sequence ID   Length   Simple   Reverse   E-value   X count
TRNA7          73      -36.19   -28.50    4.18e-12
TRNA2          76      -35.48   -28.13    6.08e-12
TRNA4          75      -35.66   -27.90    7.65e-12
TRNA9          77      -35.54   -27.82    8.29e-12
TRNA10         76      -35.21   -27.69    9.39e-12
TRNA1          72      -34.85   -27.32    1.37e-11
TRNA8          75      -36.32   -27.26    1.44e-11
TRNA5          73      -35.10   -26.36    3.58e-11
TRNA3          76      -34.43   -26.26    3.93e-11
TRNA6          74      -34.59   -23.81    4.56e-10

```

We are presently experimenting with a scaled reverse null model score (`subtract_null` set to 5). This computation is inspired by the PSI-BLAST approach. The scaled score is the reverse null model score divided by the average simple null model score per character.

The NLL-NULL scores, especially for the simple null model, are most useful when the model has had free insertion modules (Section 8.5) added to it. Then, the null model and the FIMs will cancel out, and the score will be based primarily on the section of the sequence that matches the region that has been modeled. By default, `hmmscore` automatically adds FIMs to any model that does not already contain them when SW scoring is used. To change this, if for example you want to ensure that entire sequences are modeled, rather than simply subregions, change `auto_fim` from its default value of 1 to 0. The `auto_fim` variable has no effect when a user-specified null model is used.

Since NLL-NULL scores are negative logs, the lower the better. In the case above, all of the tRNAs have been positively identified as tRNAs. (Not surprising as they were all in the training set!)

New to Version 1.2 is the ability to adjust the null model scoring. Since this determines the probability that a sequence was randomly generated according to the residue insertion probabilities, these values should reflect knowledge of the problem domain. Five possibilities are offered. The flat distribution or the background distribution of amino acids over all proteins can be used. Both of these distributions are invariant over all families, and are thus a simplistic assumption. The distribution can also be the distribution of the residues in the training set or the average residue distribution over all columns (match states) modeled by the HMM. The advantage of these two, especially the latter, is the ability to partially correct for compositional bias in the sequences. Lastly, the insertion probabilities can reflect the residue distribution of the sequence currently being scored. This is the most pessimistic null model, as it demands not that the HMM model the sequence better than fixed background frequencies, but that it model the sequence significantly better than frequencies exactly matching the sequence's composition.

So the options available for `FIM_method_score` are

- 0 Use the tables present in the model.
- 1 The relative frequencies of residues in the training sequences (from the Lettercounts node or the training sequences).
- 2 The relative frequencies of residues in model match states (from the Frequency node).
- 3 Uniform (flat) probability over all residues.
- 5 Amino acid background frequencies over all proteins (from the Generic node).
- 6 Geometric average of the match state probabilities in the model.

The default setting, for experimental and statistical reasons, is the geometric average of the model match states (6). The insertion tables can be similarly modified with `Insert_method_score`, the default of which is no change (0).

As with the training methods, if the method value is negative instead, the FIMs and insert tables will only be modified if there is no initial model read in (an unlikely occurrence for `hmmscore`).

The FIM scoring methods are only applied to the primary track of multitrack models. Other tracks never have their insert or FIM tables changed, and if FIMs are added to the model either the character table of the generic node is used (when present) or the geometric match state average (when there is no generic node).

The FIM scoring methods are not applied to the user's null model.

A more detailed discussion of these issues can be found in (Barrett, Hughey & Karplus 1996), mentioned in the Introduction.

10.2.2 Reducing hmmscore runtime

Full evaluation of a hidden Markov model takes time. The dynamic programming algorithm underlying `hmmscore` and `buildmodel` require time proportional to the product of the length of the model and the total length of the sequences.

The choice of dynamic programming style (`dpstyle`) also has a dramatic effect on execution time. The default all-paths (EM) method (`dpstyle 1`) multiplies and adds probabilities represented as log-probabilities in each of the $O(n^2)$ dynamic programming cell calculations. Adding probabilities represented as log-probabilities is internally sped with table-lookup, but still takes time. The single-best-path (Viterbi) method (`dpstyle 0`) multiplies and maximizes probabilities represented as log-probabilities, where the operations become addition and minimization, and thus is faster. The posterior-decoded (PD) alignment and scoring methods (`dpstyle 5`) perform an EM pass followed by a Viterbi pass, and use far more memory, and are the slowest. EM scoring is approximately twice as slow as Viterbi scoring, and PD scoring is three times slower than EM because a full forward-backward algorithm (as used in training) is performed followed by the Viterbi algorithm. For training models with `buildmodel`, Viterbi is about X times faster than EM. The SAM-T04 iterative model building procedure uses Viterbi training and full EM multiple domain scoring with the reverse null model.

The default reverse null model means that two dynamic programming calculations are done for each sequence. The `simple_threshold` variable can be used to control when the second calculation is performed based on the simple null model. When set to less than 10000, the reverse (or user) null model will only be evaluated when the simple null model score is less than `simple_threshold`. Sequences for which the more time consuming null model was not calculated will have the score 10000 in the second score column of the distance file. The default value of `simple_threshold` is 10000 (off); a setting of 0 is a reasonable setting to halve the running time of database searches, though occasionally some positive E-values may be lost. It is suggested that users experiment on their own data to ensure results are not lost. Runtime for a large database search will be sped by approximately one half.

Independently, when EM or PD scoring is used, the `viterbi_threshold` variable may be used to first calculate a single-best-path (Viterbi) simple null model score, and then decide whether or not to perform the EM or PD scoring based on that score. As with `simple_threshold`, the default value is 10000, and lower values will activate this option. A setting of 0 is also reasonable. This can reduce PD execution time by about ZZ, and EM execution time by about YY.

10.2.3 Selecting sequences, scores, and alignments

Sequences can be selected by `hmmscore` and placed in a `.sel` file.

A selection mode is chosen by setting `select_seq`. If 0, no sequences are selected; if 1, sequences

are selected according to their simple null model scores and `NLLNull`; if 2, sequences are selected according to their column 2 score (user or reverse sequence null, or raw NLL score if `subtract_null` is 0 or 1) and `NLLcomplex`; if 4, sequences are selected according to their E-value and `Emax`; if 8, all sequences are selected. Selection criteria can be combined: 3 requires sequences to score better than `NLLnull` with the simple null model and `NLLcomplex` with the complex (user or reverse sequence) null model. Negative numbers indicate that sequences that do not pass the corresponding positive test should be selected.

The following will place labeled copies of all sequences scoring lower than -35 with the reverse sequence null model into `test.sel`.

```
hmmscore tests -i test.mod -db trna10.seq -select_seq 2 -NLLcomplex -35 -sw 2
```

Selected sequences are written out in the same order they are encountered in the database files, which may be different from the order they are listed in the score file if scores are sorted. The `sortseq` program can be used to write out the sequences in the same order they are listed in the score file. See Section 10.12.8 on page 145. The `sort` variable controls whether sequence scores are unsorted (0); sorted by E-value (4), column 2 of the distance file (2), or column 1 of the distance file (1). Sequences in the select file will be selected by other than the sorting criteria unless `sort` and `select_seq` are set to corresponding values.

The `select_score` variable can be set in the same manner as `select_seq`, in which case only scores of those sequences that match the specified criteria will be recorded in the distance file. This is particularly useful for database searches in which only sequence IDs are of interest.

The `select_align` variable can be used in a similar way to cause selected sequence alignments to be placed in the `runname.a2m` file. Note that all selection variables use the same `NLLnull`, `NLLcomplex`, and `Emax` thresholds, though different combinations of the thresholds can be used by the different parameters.

If the binary expansion of the `many_files` variable includes a '2' (e.g., 2, 3, 6, 7), the score information is sent to standard output. If the binary expansion of the `many_files` variable includes a '4' (e.g., 4, 5, 6, 7), the multiple domain score information is sent to standard output. (See Section 10.2.5 on page 111.) If the binary expansion of the `many_files` variable includes a '1' (e.g., 1, 3, 5, 7), `buildmodel` will create multiple files. See first two options enable UNIX pipe processing, such as:

```
hmmscore tests -i test.mod -db trna10.seq | awk '$1 !~ % { print $5 "\t" $1}'
```

to print a file with an evaluate and an identifier on each line after removing the comment lines. If both options are in force (e.g., 6, 7), both the multiple domain and the simple scoring values will be sent to standard output in an undefined order. In a terminal window, you will also see `hmmscore`'s standard error output of various diagnostic messages.

10.2.4 Scoring Fragments

As with alignments, the `SW` parameter can be used to set global (0), semi-local (1), local (2), and domain (3) scoring styles. See Section 10.1.2 on page 92.

The default local scoring is similar to Smith and Waterman method of sequence comparison, which will find the best score for any pair of subsequences within two sequences. The same can be done with models, allowing a submodel to match a subsequence. With local scoring, sequences can jump from the initial module (presumably a FIM) into the delete state of any module in the model, and can also jump out of the delete state of any module within the model to the delete state of the next-to-last node. The first and next-to-last module are assumed to be FIMs, hence the rationale is that a sequence will use the FIM for some period of time to consume characters that do not match the model, then the sequence will jump to the model node corresponding to the start of the fragment, use several model nodes, and then jump to the ending FIM to consume the rest of the sequence.

The probability of these jumps is set by the variables `jump_in_prob` and `jump_out_prob`, both of which have a default value of unity. That is, as in the sequence-to-sequence Smith and Waterman, there is no cost associated with jumping in and out of the model.

The file `trna1frag.seq` contains several sequences that contain part or all of TRNA1. The sequences include TRNA1 (72 bases, TRNA1Long (the complete tRNA with additional characters), Long (58 base segment) TRNA), Medium (34 base segment), Short (6 base segment TRNA1), and AAMediumA, an embedding of Medium within several segments of As to bring it to 176 characters. Additionally, the file contains several (obviously) non-tRNAs of various lengths, all of whose IDs begin with the word 'Not'.

When this file is scored using `hmmscore` with `SW` set to 0 for global alignment, the scores of the full sequence and the long fragment place them clearly as tRNAs. However, the score of the short and medium fragments are greatly penalized by the large number of delete states they must use.

TRNA1Long	94	-36.25	-28.73	4.01e-12
:TRNA1 with flanking As				
TRNA1	72	-34.85	-27.32	1.64e-11
:the original sequence				
Long	58	-27.64	-20.06	2.33e-08
:A long partial segment from TRNA1				
AAMediumAA	176	-16.81	-9.61	8.04e-04
:The medium segment with long flanking A regions				
Medium	34	-14.62	-7.63	5.84e-03
:A medium-length segment from TRNA1				
ShortReverse	6	-5.77	-0.11	5.68e+00
:A short reverse segment from TRNA1				
NotShort	13	-6.42	0.00	6.00e+00
:A short segment that is not TRNA1				
Not	73	-6.84	0.00	6.00e+00
:A string of As of equal length to TRNA1				
NotExtraLong	438	-6.90	0.00	6.00e+00
:A string of As much longer than TRNA1				
NotLong	109	-6.86	0.00	6.00e+00
:A string of As longer than TRNA1				
Short	6	-5.66	0.11	6.32e+00
:A short segment from TRNA1				
MediumReverse	34	-7.00	7.63	1.20e+01
:The reversal of medium				

(Several of the sequences had simple null model scores worse than `simple_threshold`, so their reverse null model scores were not calculated.)

When scoring is performed using the SW option set to 1, the following score file is generated, which places even the short fragment in the possible tRNA range.

TRNA1	72	-31.14	-37.43	6.69e-16
:the original sequence				
TRNA1Long	94	-19.56	-28.35	5.88e-12
:TRNA1 with flanking As				
Long	58	-24.76	-27.13	1.99e-11
:A long partial segment from TRNA1				
Medium	34	-12.10	-13.03	2.64e-05
:A medium-length segment from TRNA1				
Short	6	-1.47	-0.04	5.87e+00
:A short segment from TRNA1				
NotExtraLong	438	26.74	-0.00	6.00e+00
:A string of As much longer than TRNA1				
NotLong	109	10.22	-0.00	6.00e+00
:A string of As longer than TRNA1				
NotShort	13	0.58	0.00	6.00e+00
:A short segment that is not TRNA1				
Not	73	8.16	0.00	6.00e+00
:A string of As of equal length to TRNA1				
AAMediumAA	176	13.75	0.00	6.00e+00
:The medium segment with long flanking A regions				
ShortReverse	6	-1.43	0.04	6.13e+00
:A short reverse segment from TRNA1				
MediumReverse	34	0.93	13.03	1.20e+01
:The reversal of medium				

When scoring is performed using the SW option set to 2, the following score file is generated, which also picks up the sequence AAMediumAA, which is a segment of a tRNA embedded within a longer sequence.

TRNA1Long	94	-36.25	-28.73	4.01e-12
:TRNA1 with flanking As				
TRNA1	72	-34.85	-27.32	1.64e-11
:the original sequence				
Long	58	-27.64	-20.06	2.33e-08
:A long partial segment from TRNA1				
AAMediumAA	176	-16.81	-9.61	8.04e-04
:The medium segment with long flanking A regions				
Medium	34	-14.62	-7.63	5.84e-03
:A medium-length segment from TRNA1				
ShortReverse	6	-5.77	-0.11	5.68e+00
:A short reverse segment from TRNA1				
NotShort	13	-6.42	0.00	6.00e+00
:A short segment that is not TRNA1				
Not	73	-6.84	0.00	6.00e+00
:A string of As of equal length to TRNA1				
NotExtraLong	438	-6.90	0.00	6.00e+00
:A string of As much longer than TRNA1				
NotLong	109	-6.86	0.00	6.00e+00
:A string of As longer than TRNA1				
Short	6	-5.66	0.11	6.32e+00
:A short segment from TRNA1				
MediumReverse	34	-7.00	7.63	1.20e+01
:The reversal of medium				

The unclear separation in this file between the best non-tRNA and the worst tRNA segment is because local alignment allows sequences to use only the parts of the model that improve its score. Thus, a long random sequence will have a better score than a shorter random sequence. Better scores are gained by using the more expensive (to calculate) reverse sequence null model. Setting `subtract_null` to 4 will differentiate the tRNAs, excepting the 6-nucleotide sequence, from the non-tRNAs. (In this contrived example, the long sequences are composed of the single letter A, and thus the sequence and the reverse sequence score the same going through the HMM.)

TRNA1Long	94	-36.25	-28.73	4.01e-12
:TRNA1 with flanking As				
TRNA1	72	-34.85	-27.32	1.64e-11
:the original sequence				
Long	58	-27.64	-20.06	2.33e-08
:A long partial segment from TRNA1				
AAMediumAA	176	-16.81	-9.61	8.04e-04
:The medium segment with long flanking A regions				
Medium	34	-14.62	-7.63	5.84e-03
:A medium-length segment from TRNA1				
ShortReverse	6	-5.77	-0.11	5.68e+00
:A short reverse segment from TRNA1				
NotShort	13	-6.42	0.00	6.00e+00
:A short segment that is not TRNA1				
Not	73	-6.84	0.00	6.00e+00
:A string of As of equal length to TRNA1				
NotExtraLong	438	-6.90	0.00	6.00e+00
:A string of As much longer than TRNA1				
NotLong	109	-6.86	0.00	6.00e+00
:A string of As longer than TRNA1				
Short	6	-5.66	0.11	6.32e+00
:A short segment from TRNA1				
MediumReverse	34	-7.00	7.63	1.20e+01
:The reversal of medium				

Significance levels for the simple null model, especially for the second option, change greatly. In the first option, because sequences can start at any location (e.g., the initial FIM) and jump out of any location (e.g., also the initial FIM), no sequence will have scores worse than zero — even non-family members will have a negative NLL–NULL score. However, the significance level will be similar to that of standard scoring.

In the second option, the number of placements of a sequence to the model is essentially the number of starting points of the sequence plus the number of exit points once the sequence has started using the core of the model. That is, sequences start in the initial FIM, may at any time jump anywhere into the model, and then jump out again later. The effect seen in the significance level depends on both the sequence length and the model length, and for comparison between different models, must be done to the scores themselves as they are being generated. If the `adjust_score` variable is set to 1 and `SW=2`, all simple null model scores will have added to them the log of the sum of sequence and model length. If `adjust_score` is set to 2 (the default) and `SW=1` or `2`, then all scores will have added to them 1.5 times the log of the sequence length. In the future, the `adjust_score` parameter may be refined as we further explore the length dependence of scores. The adjustment is not sufficient for models that have internal FIMs. See Section 10.2.1 on page 102.

10.2.5 Selecting multiple domain alignments

The `hmmscore` program also can create multiple-domain alignments and score files from selected sequences. Prior to Version 2.1, this feature was called the `multidomain` program. To enable this option, the `select_mdalign` parameter is set in a manner similar to other selection parameters. See Section 10.2.3 on page 106.

For each selected sequence, the multiple domain search procedure will locate copies of a single motif

within each selected sequence. A user specified `select_md` selection variable, along with thresholds `mdNLLnull`, `mdNLLcomplex`, and `mdEmax` is the criterion by which a subsequence is judged to be a match to the model. If `select_md` is 1, whenever an `mdNLLnull` simple null model score or better is achieved, a match to the model has been found, and so forth. The default `select_md` value of 4 uses `mdEmax` (default of 0.01) to select subsequence matches. The `adpstyle` parameter determines whether Viterbi (1, default), posterior-decoded with transitions and emissions (4), or posterior-decoded with only character emissions (5) alignment is used.

Once this match is found, it is cut from the sequence and another match is looked for. The process terminates when no matches scoring better than the selection criteria are found. Note that the multiple domain scoring procedure always uses Viterbi scoring. Thus, it is theoretically possible for a sequence to be selected by `hmmscore` for multiple domain search but for no domain to be found even if the selection criteria are the same.

The output is similar to that of `align2model`, except that for each match the sequence ID is modified to indicate where in the sequence the match occurred. Additionally, all letters in the sequence that are part of the match are capitalized. Unlike `align2model`, multiple domain search sequence output does not include periods (‘.’) as spacers: `prettyalign` must be used to correctly space the multiple alignment.

As an example, the file `multtrna.seq` contains two sequences, each of which contains two tRNA motifs. The multdomain feature is used by selecting all sequences for a multiple domain

```
hmmscore multtrna -i testf.mod -db multtrna.seq -select_mdalign 8 -sw 2
prettyalign multtrna.mult -190 > multtrna.pretty
```

the file `multtrna.pretty` generated is

```

; SAM: prettyalign v3.5 (July 15, 2005) compiled 07/15/05_11:25:33
; (c) 1992-2001 Regents of the University of California, Santa Cruz
;
;       Sequence Alignment and Modeling Software System
;       http://www.cse.ucsc.edu/research/compbio/sam.html
;
; ----- Citations (SAM, SAM-T2K, HMMs) -----
; R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
;   Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
; K. Karplus, et al., What is the value added by human intervention in protein
;   structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
; A. Krogh et al., Hidden Markov models in computational biology:
;   Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
; -----

```

```

TRNA12_90:163  ugcuaaggggauguagcucagugguagagcgcaugcuucgcauguaugaggccccg
TRNA12_8:77    ugcuaagg.....
TRNA34_104:172 gcuagcguaggcccuguggcuagcuggucaaagcgcugucuaguaaacaggaga
TRNA34_20:87   gcuagcguaggcccugugg.....

```

```

TRNA12_90:163  gguucgauccccggcaucccagucugcguugc.....GGCCGUC
TRNA12_8:77    .....GGAUGUA
TRNA34_104:172 uccuggguucgaaucccagcggggccuccagcauaguuagcgggcga---AUA
TRNA34_20:87   .....-----

```

```

          10          20          30          40          50
          |           |           |           |           |
TRNA12_90:163  GUCUAGUCUGgauU.AGGACGCUGGCCUCCCAAGCCAGcA.AU.CCCGGGUUCGA
TRNA12_8:77    GCUCAG-UGG...U.AGAGCGCAUGCUCGCAUGUAUG.A.GGcCCCGGUUCGA
TRNA34_104:172 GUGUCAGCGG...G.AGCACACCAGACUUGCAAUCUGG.U.AG.GGAGGGUUCGA
TRNA34_20:87   --CUAGCUGG...UcAAAGCGCCUGUCUAGUAAACAGG.AgAU.CCUGGGUUCGA
          60          70
          |           |

```

```

TRNA12_90:163  AUCCCGGCGGCCGCA-----ucgcuu.....
TRNA12_8:77    UCCCGGCAUCUCCA-----guacugcguugcggccgucgucuagucuggau
TRNA34_104:172 GUCCUCUUUGUCCACCA----guacguagaucgcggc.....
TRNA34_20:87   AUCCAGCGGGGCCUCCAGC---auaguuagcgggcgaaauagugucagcgggag

```

```

TRNA12_90:163  .....
TRNA12_8:77    uaggacgcuggccuccaagccagcaauccggguucgaauccggcgccgcau
TRNA34_104:172 .....
TRNA34_20:87   cacaccagacuugcaaucugguaggaggguucgagucccucuuguccaccagu

```

```

TRNA12_90:163  .....
TRNA12_8:77    cgcuu.....
TRNA34_104:172 .....
TRNA34_20:87   acguagaucgcggc

```

This file shows the matching area of the sequence within several copies of the sequence. Even though there is an extra FIM state in the model, the required ending delete state is automatically removed from the alignment because `auto_fim` is at its default value of 1.

If the variable `alignshort` is set to zero or higher, matching segments of the sequence are clipped, with `alignshort` positions shown on either side. The IDs are the same as if complete sequences are printed, corresponding to the starting and ending points of the motif within the original sequence. Depending on how large `alignshort` is, the subsequences may overlap. For example, the commands

```
hmmscore multtrnas -i testf.mod -db multtrna.seq -alignshort 3
-select_mdalign 8 -sw 2
prettyalign multtrnas.mult -l90 > multtrnas.pretty
```

produce the alignment

```
; SAM: prettyalign v3.5 (July 15, 2005) compiled 07/15/05_11:25:33
; (c) 1992-2001 Regents of the University of California, Santa Cruz
;
;       Sequence Alignment and Modeling Software System
;       http://www.cse.ucsc.edu/research/compbio/sam.html
;
; ----- Citations (SAM, SAM-T2K, HMMs) -----
; R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
; Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
; K. Karplus, et al., What is the value added by human intervention in protein
; structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
; A. Krogh et al., Hidden Markov models in computational biology:
; Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
; -----
;
;              10          20          30          40
;              |          |          |          |
TRNA12_90:163  ugcGGCCGUCGUCUAGUCUGgauU.AGGACGCUGGCCUCCCAAGCCAGcA.AU.C
TRNA12_8:77   aggGGAUGUAGCUCAG-UGG...U.AGAGCGCAUGCUCGCAUGUAUG.A.GGcC
TRNA34_104:172 cga----AUAGUGUCAGCGG...G.AGCACACCAGACUUGCAAUCUGG.U.AG.G
TRNA34_20:87  ugg-----CUAGCUGG...UcAAAGCGCCUGUCUAGUAAACAGG.AgAU.C
;
;              50          60          70
;              |          |          |
TRNA12_90:163  CCGGGUUCGAAUCCCGGGCGCGCA-----ucg
TRNA12_8:77   CCGGGUUCGAAUCCCGGCAUCUCCA-----gua
TRNA34_104:172 GAGGGUUCGAGUCCCUUUGUCCACCA-----gua
TRNA34_20:87  CUGGGUUCGAAUCCCGGGGCCUCCAGC---aau
```

In addition to `multtrna.mult`, the file `multtrna.mstat` is produced. It reports the `NLL-NULL` score for each of the motifs listed in `multtrna.mult`. In this case, `multtrna.mstat` (or `multtrnas.mstat`) looks like

```

% SAM: hmmscore v3.5 (July 15, 2005) compiled 07/15/05_11:25:31
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%           Sequence Alignment and Modeling Software System
%           http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% multtrna  Host: peep    Fri Jul 15 13:45:02 2005
% rph       Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Inserted Files:  testf.mod
% Database Files:  /projects/kestrel/rph/sam32/demos/multtrna.seq
%
% Motifcutoff 0.500000
% Motifs selected from sequences into file multtrna.mult if better than:
% E-value (mdEmax 1.0e-02) (select_md=4)
% See related information in multtrna.dist
%
% Simple: NLL=NULL using FIM probabilities
% Reverse: NLL=NULL for the reverse sequence NULL model
%   Calculated when Simple < simple_threshold (10000.00)
% E-value on N (=2) sequences:
%   N / (1 + exp(-(lambda(=1.0000) * Reverse)**tau(=1.0000)))
%   Calculated when Simple < simple_threshold (10000.00)
%   Rescale E-values or use -dbsize for multiple scoring runs.
%   WARNING: E-VALUES ARE NOT CALIBRATED!
% Scores sorted by E-value, best first
%
% Sequence ID      Length      Simple      Reverse      E-value      X count
TRNA34_104:172    69          -32.06      -30.14       1.63e-13
TRNA12_90:163    74          -31.41      -29.78       2.34e-13
TRNA12_8:77      70          -30.25      -27.98       1.42e-12
TRNA34_20:87     68          -29.98      -27.46       2.37e-12

```

Because reliable results are only obtained if FIMs are added to the model, multiple domain searchers are best performed when `auto_fim` is set to 1 (the default).

If the binary expansion of the `many_files` variable includes a '4' (e.g., 4, 5, 6, 7), the multiple domain score information (`.mstat`) is sent to standard output instead. See Section 10.2.3 on page 106.

10.2.6 Multi-track HMM scoring

Suppose one wanted to model a group of protein sequences with associated secondary structure information. The secondary structure labels are important information that could lead to better modelling. To enable such analysis (without the need, for example, of a 60 character alphabet of each amino acid with each secondary structure type), SAM can process multi-track sequences and

multi-track HMMs.

A multi-track sequence is specified by a pair or set of sequence files. The first sequence file contains the first track of data (amino acid sequences in this case), while the second sequence file contains the second track of data (secondary structure) with each sequence corresponding exactly in length, identifier, and position to a sequence in the first file. The sequences are specified as a comma-separated list with the `db` parameter.

Multi-track HMMs are similarly paired or sets of SAM HMM files. The models are specified as a comma-separated list for the `-trackmod` filename list variable. During the dynamic programming calculation, all transition probabilities are taken from the first or primary track, while character emission probabilities are taken from the joint (product) probability of the first track character being emitted from the given first track HMM state, and the second track character being emitted from the given second track HMM state, and so on. All track models must, of course, be of the same length.

By default, the tracks are all given unity weight, so that, ignoring transition probabilities, a 2-track model will produce scores of about twice the magnitude of a 1-track model. To compensate for this, each track can also be given a character emission coefficient with `trackcoeff`. The scores (log-probabilities) of the match and insert states are scaled by this (≥ 0.0) value, so that a 2-track model with identical tracks and coefficients of 0.5 will produce the same scores (approximately, due to rounding) as a one-track model, such as with the following two commands:

```
hmmscore t2 -trackmod test.mod,test.mod -db trna10.seq,trna10.seq
-a RNA,RNA -trackcoeff 0.5,0.5
hmmscore t1 -i test.mod -db trna10.seq
```

if a mixture of user-defined and standard alphabets are used, the hyphen ('-') can be used in place of the normal character list in an `alphabetdef` statement to indicate a standard alphabet. For example:

```
hmmscore t2 -trackmod test.mod,test.mod -db trna10.seq,trna10.seq
-alphabetdef "RNA -","RNA -" -trackcoeff 0.5,0.5
```

Multi-track models are easy to create with external software scripts.

The additional tracks have the same SW mode preprocessing (i.e., possibly adding FIMs), but never have their insert or FIM character tables changed (i.e., `Insert_method_score` and `FIM_method_score` are ignored for tracks other than the first). If FIMs are added to the model either the character table of the generic node is used (when present) or the geometric match state average (when there is no generic node).

10.2.7 Distributed scoring

Scoring a large database with `hmmscore` can take several hours on even the fastest workstation. The scoring program includes primitive support for distributed scoring. If the `segments` variable is set to an integer larger than 1, `hmmscore` assumes that that many runs of `hmmscore` are being used to score a complete database. The `segment_number` variable is used to label each segment.

These parameters might be used as follows:

```

hmmscore test1 -i test.mod -db bigdatabase -segments 2 -segment_number 1 -sw 2&
rsh othermachine hmmscore test2 -i test.mod -db bigdatabase -segments 2 -segment_number
2 -sw 2
wait cat test1.dist test2.dist > test.dist

```

The associated parameter, `segment_size`, specifies that number of sequences that are read in at a time. At a value of 100, two segments would produce the effect that the first 100 sequences are processed by segment 1, the next 100 be segment 2, the next 100 by segment 1, and so on. Note that workload is partitioned according to the number of sequences rather than the number of residues, some segments may take longer to complete than other segments.

The `segment_size` parameter is important even if distributed scoring is not being performed to reduce memory consumption by `hmmscore`. Unfortunately, selection based on E-values requires knowledge of the database size. For this reason, if selection is being performed, segmented file reading will be performed twice; the first time to calculate the size of the database, in the second time to perform the scoring. The `dbsize` variable can be set to externally indicate the size of the database for calculating E-values. When set, the first reading of the database is not performed.

10.2.8 Single Sequence Smith & Waterman Scoring and Alignment

The `hmmscore` program can be used to perform Smith and Waterman scoring, alignment, and multiple domain alignment. This option is achieved by internally creating a SAM model based on a single query sequence, gap and continue penalties, and a scoring matrix. These models cannot be stored, and can only be used with Viterbi scoring. Local or global alignment can be performed on this model by appropriately setting `sw`.

The internal model zeros the appropriate transitions and uses an appropriately simplified dynamic programming calculation. Its dynamic programming speed is about the same as a dedicated Smith and Waterman program, though the default reverse-sequence e-value calculation doubles execution time.

Smith and Waterman mode is specified by including a `query` file on the command line. The first sequence in this file is taken to be the query sequence. Additional parameters include `gap` and `continue`, which are by default set to 12 and 1, respectively, and `matrix`, set to “blosum62.” Matrices are read in BLAST format. They are first looked for in the current working directory, then in the directory pointed to by `BLASTMAT`, if set. If not, the subdirectories `aa` and `nt`, depending on alphabet, of the environment variable `PRIOR_PATH` are checked, and finally, the same subdirectories of the default prior path compiled into the code. See Section 8.1.1 on page 60.

As with other forms of scoring, SAM is able to calculate e-values for Smith and Waterman queries based on the reverse sequence in all model. For this to be effective, the system must know the scaling factor for computing E-values based on how the scoring matrix has been formed. The value `swlambda` should be set to $\ln(b)/u$, where b is the base of the matrix and $1/u$ is the unit weight. For example, the Hennikoff & Hennikoff “blosum62” matrix distributed with BLAST and included with SAM is in units of 1/2 bit, so $b = 2$ (bits are base 2) and $u = 2$, since a ‘2’ in the matrix corresponds one unit (a bit, in this case). The default value of `lambda` is thus $\ln(2)/2 = 0.34657$.

```

hmmscore sw -query globins50 -db globins50 -matrix blosum62 -gap 12
-continue 1

```

will produce a score file using the first sequence of `globins50` as a query and the first 10 sequence of `globins50` as a database.

```
% SAM: hmmscore v3.5 (July 15, 2005) compiled 07/15/05_11:25:31
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%       Sequence Alignment and Modeling Software System
%       http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% sw  Host: peep    Fri Jul 15 13:45:02 2005
% rph  Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Inserted Files:
% Database Files:  /projects/kestrel/rph/sam32/demos/globins50.seq
%
% Smith & Waterman Query Sequence:  BAHG$VITSP
% Cost matrix:    /projects/kestrel/rph/sam32/lib/matrix/aa/blosum62
% Gap cost 12    Continue cost 1
% Score DP Method: viterbi single-path (dpstyle = 1)
% Align DP Method: viterbi (adpstyle = 1)
% 50 sequences, 7369 residues, 0 nodes, 0.08 seconds
%
% Sequence scores selected:  All (select_score=8)
%
% Simple: NLL=NULL using FIM probabilities
% Reverse: NLL=NULL for the reverse sequence NULL model
%   Calculated when Simple < simple_threshold (10000.00)
% E-value on N (=50) sequences:
%   N / (1 + exp(-(swlambda(=0.3466) * Reverse)**tau(=1.0000)))
%   Rescale E-values or use -dbsize for multiple scoring runs.
%   WARNING:  E-VALUES ARE NOT CALIBRATED!
% Scores sorted by E-value, best first
%
% Sequence ID   Length      Simple      Reverse      E-value      X count
```

The various `select` options (e.g., `selectalign`, `selectmdalign`, etc.) can also be used, according to Smith and Waterman score (`NLLnull` and `mdNLLnull`), reverse sequence (`NLLcomplex` and `mdNLLcomplex`, or e-value (`Evalue` and `mdEvalue`) score.

10.2.9 Scoring using the Kestrel parallel processor

The `hmmscore` program can use the UCSC Kestrel parallel processor to perform high-speed scoring of sequence databases.

Currently `hmmscore` only supports global or local EM scoring on Kestrel and models that have a final length, with added FIMs, of no more than 512 nodes. If any of these restrictions are not met, `hmmscore` reverts to the sequential algorithm, unless `kestrel_fallback` is set to 0.

Kestrel scoring is enabled using the `use_kestrel` parameter. Since a request queue mechanism is not currently available as part of the Kestrel runtime environment, a sleep and retry mechanism is implemented as part of `hmmscore`. Retrying is controlled using the `kestrel_retry_cnt` and `kestrel_retry_time` parameters. The `kestrel` runtime environment program is executed to access the Kestrel server and is found using the `PATH`.

Small models will generally score more quickly using the sequential algorithm than with Kestrel. A minimum model size for using Kestrel may be specified with the `kestrel_min_model_len` parameter. Models smaller than this will use sequential scoring.

Kestrel-format databases, created using `_kestrel_db`, are used by `hmmscore`. The original sequence databases are specified as the value of the `-db` arguments to `hmmscore`. The file names of the Kestrel-specific database will be derived from the `db` file names by appending the appropriate suffix. If `subtract_null` is 4, then `.krseq` is used, otherwise `.kseq`. Additionally, the `.kids` file must exist in the same directory as the `-db` file. The `.kseq` or `.krseq` are found either in the `-db` file directory or in the directory on the Kestrel server specified by `kestrel_remote_db_dir`. See Section 10.12.10 on page 147.

10.2.10 Calibration and Model libraries

A SAM model library is a SAM parameter file that has been divided into several sections, one for each item in the model library. For a model, each section includes either a `modelfile` or `trackmod`, though it is acceptable to actually include the model definition in the library file. The typical parameters completely specify the method of scoring, including dynamic programming mode (`dpstyle`, `sw`, `autoaddfims`) and scoring parameters (`subtractnull`, and `swlambda` or `enlambda`). Also, each entry includes a name and a comment.

Single-entry model libraries are created when the `calibrate` option of `hmmscore` is used on a `modelfile` or a `trackmod`. If a `modellibrary` is passed to `hmmscore`, the resulting `runname.mlib` file has the same number of models as the original model library. To create a multi-model library, individual libraries can be concatenated, though you may wish to remove the comments (lines beginning with `'%`) for a more compact form.

When a model library is scored with `rdb` output, a column is included for the model name. Distance `.dist` files, however, do not have a means of indicating model names. Therefore, the standard `.dist` output will create one distance file for each model in the model library. The files will have a three-part name, being the run name, a sequence number (starting at 1 for the first model in the model library), and the name of the model as indicated in the model library. For example, `test.1.firstmodel.dist` is the first score file created by `hmmscore` run `test` that used a model library in which the first model was named “firstmodel”. This format is used so that a model library can be scored multiple times (for example, with different databases) in the same directory. If the binary representation of `manyfiles` includes a 1 in the 8s place (for example, by setting `manyfiles` to any number between 8 and 15, inclusive), the distance file names will only use the model names specified in the model library, such as “firstmodel.dist”. This will create a problem, of course, if multiple models in the model library have the same name.

SAM model library output will use absolute path names if the `modlib_absolute` variable is set to 1. This should be done if model libraries created in different directories are to be combined, and absolute path names were not used on the command line.

All models in a model library must have the same number of sequence tracks to facilitate comparison against a sequence database.

Suppose that model library `test.mlib` contains:

```
MODLIBMOD first Test.mod with sw 1
modelfile test.mod
dpstyle 0
sw 1
ENDMODLIBMOD
MODLIBMOD second Test.mod with sw 2
modelfile test.mod
dpstyle 0
sw 2
ENDMODLIBMOD
```

This `test.mlib` file can then be calibrated using `hmmscore`:

```
hmmscore caltest -modellibrary test.mlib -calibrate 1
```

To produce the file

```

% SAM: hmmscore v3.5 (July 15, 2005) compiled 07/15/05_11:25:31
% (c) 1992-2001 Regents of the University of California, Santa Cruz
%
%           Sequence Alignment and Modeling Software System
%           http://www.cse.ucsc.edu/research/compbio/sam.html
%
% ----- Citations (SAM, SAM-T2K, HMMs) -----
% R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
% Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
% K. Karplus, et al., What is the value added by human intervention in protein
% structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
% A. Krogh et al., Hidden Markov models in computational biology:
% Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
% -----
% caltest  Host: peep    Fri Jul 15 13:45:27 2005
% rph      Dir:  /projects/kestrel/rph/sam32/SAMBUILD/peep/demos
% -----
% Inserted Files:
% Database Files:
%
MODLIBMOD first Test.mod with sw 1
% Calibration data for 1000 random sequences
% 1000 of 1000 scores > calibrate_threshold (-1e+20) used (default).
model_file test.mod
lambda 0.863821
tau 0.911306
sw 1
dpstyle 0
jumpinprob 1.000000
jumpoutprob 1.000000
subtractnull 4
FIMmethodscore -6
insertmethodscore 0
adjust_score 2
ENDMODLIBMOD
MODLIBMOD second Test.mod with sw 2
% Calibration data for 1000 random sequences
% 1000 of 1000 scores > calibrate_threshold (-1e+20) used (default).
model_file test.mod
lambda 2.319682
tau 0.792675
sw 2
dpstyle 0
jumpinprob 1.000000
jumpoutprob 1.000000
subtractnull 4
FIMmethodscore -6
insertmethodscore 0
adjust_score 2
ENDMODLIBMOD

```

As can be seen, model calibration has added additional information to the model library. The most important is the setting of the `lambda` parameter for E-value calculation.

If any of the parameters shown in the resulting model library are changed, calibration must be

performed again. It is important to note that models must be calibrated separately for all differences in scoring method.

The process of model calibration involves scoring thousands of sequences from either a database or a an internal random sequence generator (for protein sequences, a dirichlet mixture distribution is used). This can be a time-consuming process, but once calculated `hmm_score` will produce much more accurate E-values.

At present, we prefer scoring against a database of sequences. In this process, we assume a symmetric distribution, and remove all reverse null model scores less than (better than) zero to ensure that calibration is based on non-matching sequences. To perform database calibration, set `calibrate` to 1 to use the entire database, or a number larger than one, such as 1000, to use the first 1000 sequences. The evaluates reported as a result of a calibration run will be the newly calibrated evaluates. If you generally score small sets of sequence, you should calibrate your model against a large, non-redundent database, and then score the calibrated model against the smaller database.

For random sequence calibration, the only required parameter is `calibrate`, which specifies the number of random sequences to score (if `calibrate` is set to '1', an internal default is used). Optionally, `trackprior` specifies a list (one per track) of Dirichlet mixtures over sequence composition, `genprot_prior` indicates the default Dirichlet mixture prior for protein sequences, `genehl2_prior` indicates the default Dirichlet mixture prior for the EHL2 alphabet, `gs_mean_log_len` is the natural logarithm of the mean sequence length to generate, and `gs_sd_log_len` is the natural logarithm of the standard deviation of the synthetic sequence length distribution. If a Dirichlet mixture is not available for one or more of the sequence tracks, the characters are drawn according to SAM's internal default background frequencies with no variation in distribution between sequences.

We are reasonably happy with our single-track model calibration method. Our random sequence generators for multi-track model calibration are not appropriately linked, and thus do not effectively calibrate multi-track models, such as those with amino acid and secondary structure sequences. This is currently an active research area, and we hope to improve the calibration method for these models in the next release.

10.3 addfims

The `addfims` program is obsolete and may be removed from the SAM suite at a future date. With the addition of full-local and semi-local scoring, training, and alignment using the `SW` and `SWtrain` options, FIMs are automatically added to the beginning and end of the model internally (and latter removed) See Section 9.6 on page 86.

The `addfims` program can be used to add Free Insertion Modules to the beginning and end of a model. The program `modifymodel` can also be used for this purpose, and 'O' characters in `modelfromalign` also produce FIMs. See Section 8.5 on page 71. This is particularly useful if a model has been trained on a clipped sequence motif, and is to be used in analyzing full sequences.

For example, the file `trna10f.seq` is the same as `trna10.seq`, except that sequences 2–5 have had extraneous characters appended to one or both ends. Using the `test.mod` file previously generated, an alignment of the first 5 sequences looks like this:

```

                                10      20      30
                                |      |      |
TRNA1  gg.....GGAUGUAGCUCAG-UGG...U. AGAGCGCAUGCUCGCAUGU
TRNA2X  gcggccgucgucuagugc...GGCCGUCGUCUAGUCUGgauU. AGGACGUCGGCCUCCCAAGC
TRNA3X  cccggccugugg.....-----CUAGCUGG...UcAAAGCGCCUGUCUAGUAAAC
TRNA4X  gggcgaaauagugucagggcga----AUAGUGUCAGCGG...G. AGCACACCAGACUUGCAAUC
TRNA5X  gccggg.....-----AUAGCUCAGUUGG...U. AGAGCAGAGGACUGAAAAUC
    40          50          60          70
    |          |          |          |
TRNA1  AU.G.A.GGcCCCGGGUUCGAUCCCGGCAUCUCCA-----.....
TRNA2X  CA.G.cA.AU.CCCGGGUUCGAAUCCCGGGCGCCGCAC-----ggcggccgca.....
TRNA3X  AG.G.AgAU.CCUGGGUUCGAAUCCAGCGGGGCCUCCA----gggg.....
TRNA4X  UG.G.U.AG.GGAGGGUUCGAGUCCUCUUGUCCACCA----.....
TRNA5X  CUcG.U.GU.CACCAGUCAAUAUCUGGUU-----ccuggcaugguuccggca

```

This alignment is incorrect: the extra end characters do not all use end insert states. Rather, internal insert states are found to minimize the alignment cost for sequences 2X, 4X, and 5X.

The `addfims` program has an interface identical to that of `buildmodel`, though only the runname, model file (and its model, or if not present, its regularizer), and the alphabet are used:

```

addfims testf -insert test.mod
align2model testf -i testf.mod -db trna10f.seq
prettyalign testf.a2m -l90 > testf.align

```

```

                                10      20      30
                                |      |      |
TRNA1  gg.....GGAUGUAGCUCAG-UGG...U. AGAGCGCAUGCUCGCAUGU
TRNA2X  gcggccgucgucuagugc...GGCCGUCGUCUAGUCUGgauU. AGGACGUCGGCCUCCCAAGC
TRNA3X  cccggccugugg.....-----CUAGCUGG...UcAAAGCGCCUGUCUAGUAAAC
TRNA4X  gggcgaaauagugucagggcga----AUAGUGUCAGCGG...G. AGCACACCAGACUUGCAAUC
TRNA5X  gccggg.....-----AUAGCUCAGUUGG...U. AGAGCAGAGGACUGAAAAUC
    40          50          60          70
    |          |          |          |
TRNA1  AU.G.A.GGcCCCGGGUUCGAUCCCGGCAUCUCCA-----.....
TRNA2X  CA.G.cA.AU.CCCGGGUUCGAAUCCCGGGCGCCGCAC-----ggcggccgca.....
TRNA3X  AG.G.AgAU.CCUGGGUUCGAAUCCAGCGGGGCCUCCA----gggg.....
TRNA4X  UG.G.U.AG.GGAGGGUUCGAGUCCUCUUGUCCACCA----.....
TRNA5X  CUcG.U.GU.CACCAGUCAAUAUCUGGUU-----ccuggcaugguuccggca

```

10.4 fragfinder

The `fragfinder` program will, given a database (`db`) and a model (`modelfile`, `insert`, or `trackmod` and `trackcoeff`), find a specified number of gapless sequence fragments of a specified length commencing at each model state (excepting the ones at the end where fragments of the given length would not fit).

The program calculates the posterior matrix for each sequence — the values “what is the probability that this specific character is in this specific model state” using the forward-backward algorithm. Then, for each position in the model, the top `numpermatch` (default 5) scoring fragments of length `fraglen` (default 10) are saved. Fragment scores are the sum of the posterior scores for the fragment with a null model subtracted off. In this case, we use the posteriors of the reversed fragment as

the null model. This provides some compensation for the strong length-dependent signal present in the posteriors, even with fully-local alignment (characters near beginning the sequence are far more likely to be at the start of the model).

The program produces two output files: `runname.frag` and `runname.fstat`. The first is an a2m multiple alignment format of the fragments. If desired, a single sequence can be specified to be aligned against the model, and then be used as a guide, specified with `firstsequence`. The second file is a standard SAM score file with mangled names, sequence lengths, and fragment scores. The names are mangled by appending “.x.y” to each sequence ID, where ‘x’ is the starting position in the model and ‘y’ is the starting position in the sequences. Fragments will be listed in order of starting model node number, with `numpermatch` fragments for each model node.

10.5 grabdp

The `grabdp` program can be used to examine the dynamic programming table of the forward-backward dynamic programming calculation.

If used with no arguments, or a dynamic programming style `dpstyle` other than 2 (forward-backward with posterior saving), it will produce a `runname.seqid.freq` model file with a frequency model for every sequence within the `db` presented to `grabdp`. The frequency files are similar to those produced with `buildmodel`, and indicate the fractional number of sequences that have used each transition and each character generating state in the model.

When `grabdp` is run with `dpstyle 2`, it produces a full dump of the dynamic programming posterior matrix. The resulting `runname.pdoc` file is very large, as it includes the posteriors for the three transitions into match and the character posteriors for each for the index points in the dynamic programming matrix (i.e., approximately sequence length times model length records). The file first includes a copy of the sequence and of the model, followed by the posteriors. The following posteriors are for a sequence ‘AT’ against a 2-state model trained on the several copies of the sequence ‘AT’.

```
POSTERIORS
(Residue 2 DM+MM+IM) 46.0517 19.3068 18.9014 -7.92682e-05
(Char T, index 3, Match) 19.9366 13.342 0.00100001 19.5945
(Char T, index 3, Insert) 11.603 7.322 7.806 19.5945
(Residue 1 DM+MM+IM) 46.0517 13.3407 0.00114948 18.9014
(Char A, index 0, Match) 19.9366 0.00699997 9.183 19.5945
(Char A, index 0, Insert) 9.616 4.984 8.509 19.5945
(Residue 0 DM+MM+IM) 46.0517 0.00699997 9.18296 18.9014
```

Posteriors are output in reverse order. Each “Residue” line shows the log-probability that residue being in a given matchstate. Since there is no residue 2, the first line should be ignored. The second and third lines indicate negative log probabilities of the character T (which has character code 3 in the DNA alphabet) being in the match or insert states of each model node. As can be seen, the character is almost certainly in model node 2, as indicated by the ‘0.00100001’. The ‘Residue 1’ line indicates that the match state is again most likely; it shows the sum of all the transitions going into match states for that node. Residue 0, ‘A’ is similarly placed in model node 1.

Because of the size of these files, the `grabdp` program will only output the values for a single sequence and model, choosing the first sequence in a database. The program requires a model file, a database file, and optionally a single sequence identifier.

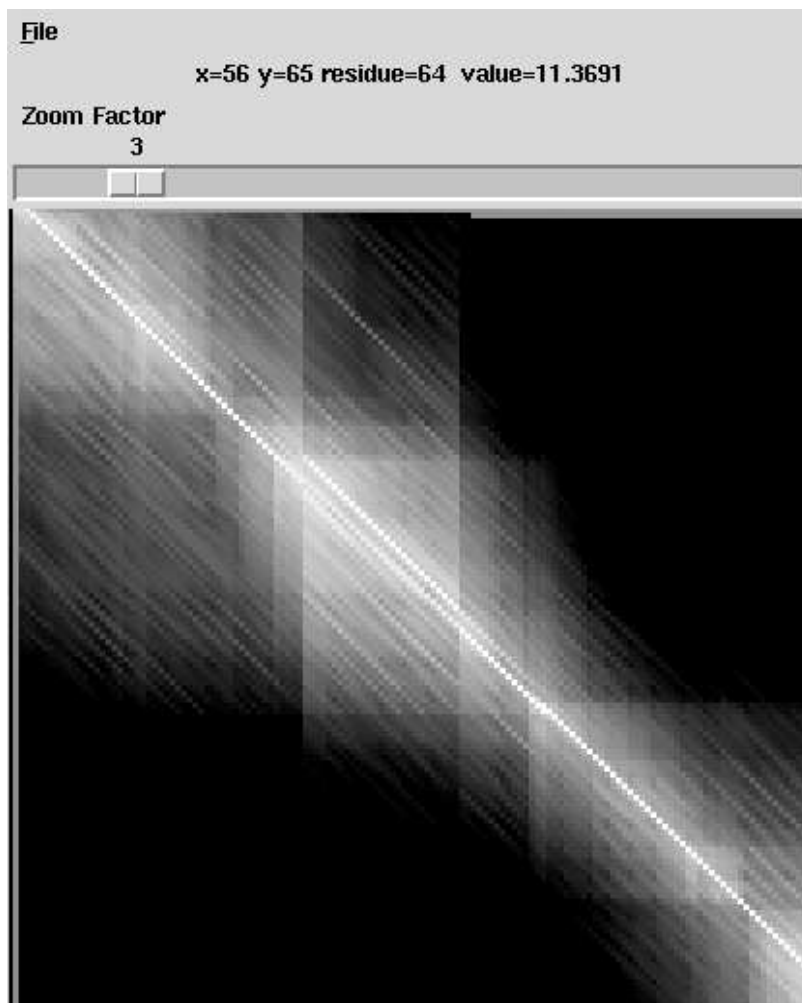


Figure 16: The `view_pdoc` visualization of posteriors calculated for a globin against the 1babA model.

The `view_pdoc` program is a viewer for the posterior decoding files. It enables zooming and coordinate location in regions of interest in the posterior decoding matrix. The program requires the `wish` simple windowing shell, part of the Tk suite. It may be necessary to edit the `view_pdoc` script to include path names for `xwd`, the X window dump program, and `ppmtogif`, the image conversion program. An example visualization is shown in Figure 16.

For diagnostic purposes, it can sometimes be useful to view the amino acid priors and posteriors. When the `dump_match_probs` flag is set, `grabdp` will generate a RDB file listing the posteriors for each node, and listing the priors under the label “FREQAVE”.

10.6 get_fisher_scores

The `get_fisher_scores` program scores sequences with a model and outputs *Fisher score vectors* for input an external discriminative learning program as described in:

T. Jaakkola and M. Diekhans and D. Haussler A discriminative framework for detecting remote protein homologies, *Journal of Computational Biology* 7(1):2000

The `hmm_score` arguments specifying the model, sequences, and scoring parameters (e.g. `-dpstyle`, `-sw`) are valid. The `-fisher_feature` parameter specifies which features are used to calculate the Fisher score vectors. Use the `-null_score_weight_scale` parameter to weight sequences by their NLL-NULL score.

As the Fisher score vectors can be large, they are written as a stream to `stdout`, allowing piping directly to another program without creating an intermediate file. The output stream maybe in ASCII or binary format, selected by the `-binary_output8.4.5` option. The first line of each record indicates the record format and sequence id. ASCII records start with a line in the form

```
>A seqid1
```

and binary start with

```
>B seqid1
```

In both formats, the record header is terminated by a newline.

The first number in a record indicates the total number of features in the record. It is followed by that many floating point numbers. In an ASCII stream, the numbers are whitespace separated (both spaces and newlines are used) designed to be read by `fscanf`. In the binary format, the feature count is an `int` and the features are `floats`.

```
>A seqid1
3
0.1 0.2
>A seqid2
3
1.2 -0.3 1.0
```

The features may also be written to a file in RDB format, specified with the `-rdb`, creating a file names `runname.fisher-rdb`. The `-write_dist` parameter can be used to generate a `runname.dist-rdb` score file, which maybe useful in analyzing the results.

For example, the following command will derive *Fisher scores vectors* from the match states and Dirichlet mixture used to train the model:

```
get_fisher_scores unused -fisher_feature match_prior -i test.mod -db prots.seq -sw 2
```

10.7 modelfromalign

`Modelfromalign` takes a multiple alignment and converts it to a model. As a base model, the program starts with the default regularizer (or a specified regularizer, as for `buildmodel`), and then calculates node frequencies according to the given multiple alignment. Sequences in the alignment can be weighted according to the `alignment_weights` file, discussed in Section 9.4.

If a trustworthy hand alignment is available, this is often the best way to build a model: create one from an alignment, and then refine it using `buildmodel`. If some sections of the alignment are particularly important, it may be desirable to make them fixed nodes, as described in Section 8.4.2.

The `alignfile` parameter can also be used with `buildmodel` to specify a seed alignment. See Section 8.3 on page 65.

The `modelfromalign` program will read any `readseq` format, but has a few special interpretations. It follows the `align2model` convention that lowercase letters are insertions, hyphens are deletions, and dots are simply filler for insertions in other sequences. Additionally, the letter ‘O’ is converted into a FIM, following a convention used in some multiple alignment formats (the other SAM programs will convert ‘O’ to the ‘X’ wildcard). If all sequences do not have the same number of uppercase letters and hyphens, then `modelfromalign` will try treating all characters as uppercase and all periods as hyphens (i.e., it will try modeling each character as a match column).

The program has one required parameter that must be set on the command line or in an inserted or `.samrc` file: the `alignfile` is the name of the file with the alignment. For example,

```
modelfromalign trna2 -alignfile trna2.align
```

will produce a model from the `trna2.align` alignment. All `buildmodel` parameters dealing with regularization and prior libraries are used in the conversion from column frequencies to a model. Prior libraries are particularly helpful when converting a small protein alignment to a model.

If the alignment is of a motif, setting the `align_fim` variable to 1 will cause FIMs to be added to the model before printing it out.

10.8 pathprobs

The `pathprobs` program can be used to generate the posterior probabilities of each of the characters in an alignment. This is a small subset of the information generated by the `grabdp` program. In effect, `pathprobs` uses the alignment to trace through the posterior matrix and decide which character emission probabilities are desired. Optionally, `pathprobs` can convert all match states with a worse posterior than some threshold to insert states, which can improve alignments.

The program can be run using, for example,

```
pathprobs 1babA-50p -modelfile 1babA-t2k-w0.5.mod -alignfile 1babA-50.a2m
```

which will produce tab-separated RDB output alignment probability file `1babA-50p.apr` along the lines of:

```

SequenceID SeqIndex ModIndex ModState NegLnPosterior T0:protein
10s 8n 8n 8s 9n 10s
1babA 0 0 I 0.347000 X
1babA 1 2 M 0.461000 M
1babA 2 3 M 0.185000 E
1babA 3 4 M 0.027000 L
1babA 4 5 M 0.007000 S
1babA 5 6 M 0.003000 P
1babA 6 7 M 0.001000 A

```

Here, the first line identifies the columns. In this case, a single-track model and sequences were used. The second line is field width and type specifiers. The remaining lines show, for each character in the labeled sequence, the state that it was generated by according to the alignment (a model index and an 'M' for match or 'I' for insert; 'D' never occurs because it does not generate any characters), the negative natural log posterior probability according to the forward-backward algorithm of the character appearing in that state, and the (possibly multi-track) character itself.

The program will also generate the 1babA-50p.pa2m file that interleaves sequences and digits corresponding to negative log posterior probabilities multiplied by ppscale (default = 1.0). The values are rounded to a single digit, so 9.0 and higher is reduced to the digit 9. With the default ppscale of $1.0/\ln(e) = 1.0$, 0 indicates a completely certain position ($p = 1.0$), 1 indicates 1 NAT of probability ($p = 0.368$), 2 indicates 2 NATS ($p = 0.136$), and so on. For example, here is a prettyaligned excerpt of a .pa2m file for several globins.

```

; SAM: prettyalign v3.5 (July 15, 2005) compiled 07/15/05_11:25:33
; (c) 1992-2001 Regents of the University of California, Santa Cruz
;
;           Sequence Alignment and Modeling Software System
;           http://www.cse.ucsc.edu/research/compbio/sam.html
;
; ----- Citations (SAM, SAM-T2K, HMMs) -----
; R. Hughey, A. Krogh, Hidden Markov models for sequence analysis:
; Extension and analysis of the basic method, CABIOS 12:95-107, 1996.
; K. Karplus, et al., What is the value added by human intervention in protein
; structure prediction, Proteins: Structure, Function, Genetics 45(S5):86--91, 2001.
; A. Krogh et al., Hidden Markov models in computational biology:
; Applications to protein modeling, JMB 235:1501-1531, Feb 1994.
; -----
;
;           10           20           30           40           50
;           |           |           |           |           |
1babA      x.....-MELSPADKTNVKAAWGKV....GA.HAG.....EY...GAEALERMFLSFPTTKTYFPHF.DLS....
1babA      x.....-0000000000000000....00.000.....00...0000000000000000000000.000....
BAHG$VITSP m.....--LDQQTINI I KATVPVL....KE.HGV.....TI...TTTFYKNLFAKHPEVRPLFD--...
BAHG$VITSP m.....--0000000000000000....00.000.....00...000000000000000000--...
GLB$APLJU  a.....--LSAADAGLLAQSWAPV....FA.NSD.....AN...GASFLVALFTQFPESANFNDF.KGKslad
GLB$APLJU  a.....--0000000000000000....00.000.....00...00000000000000000000.000slad
GLB$APLKU  .....--SLSAAEADLVGKSWAPV....YA.NKD.....AD...GANFLLSLFEKFPNNANYFADF.KGKsiad
GLB$APLKU  .....--0000000000000000....00.000.....00...00000000000000000000.000siad

```

The pathprobs program can also trim alignments (Cline, Karplus, & Hughey, Bioinformatics 18(2):306-314, 2002) according to a specific posterior threshold indicated with pptrim. All match states with negative log probabilities equal to or above this threshold will be turned into insertions.

It is of course critical that the alignment and the model are of the same length. A typical use of the

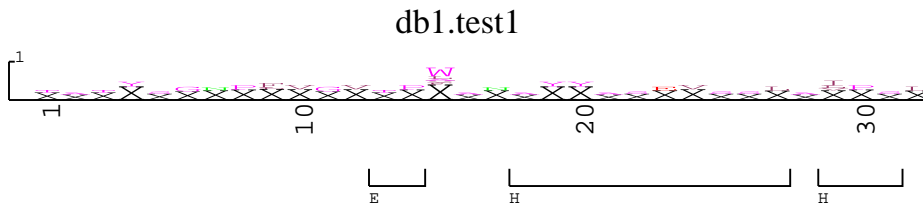


Figure 17: A model and its predicted secondary structure (`makelogo db1.test1.eps -modelfile db1.test1.mod -logo_captionf db1.test1.ptrack -logo_sections 1-32`).

program would be to generate a Viterbi or Posterior-Decoded alignment and then use `pathprobs` to determine how strongly that alignment is reflected in the posterior matrix.

10.9 predict_track

The `predict_track` program can be used to predict the secondary structure of a sequence based on a protein model and a set of sequences of known secondary structure. The program can either be provided with models for all tracks or (in the 2-track case) create a model for the second track based on a database of 2-track sequences. The program thus requires a base model or set of base models, a prediction sequence, a multi-track alphabet definition, and (when it generates its own second-track model) a 2-track database. See Section 10.2.6 on page 115.

For output, the program will produce a `.ptrack` file of per-sequence structure predictions or if `rdb` is set, a `.rdb` file. If the program created a model for the second track, it will be written to a `.mod` file, and can be used, for example, with multi-track HMM scoring and alignment.

For example, the command:

```
predict_track db1.test1 -sw 2 -subtractnull 4 -i db1.test1.mod
-a protein,ELH -db predict.db1.seq,predict.db1.2d
-predictseq db1.test1.seq
```

will produce as output `db1.test1.ptrack`, a predicted secondary structure sequence for the first sequence in the `db1.test1.seq` file based on the protein model and the primary and secondary structure sequences provided to `predict_track`, and `db1.test1.mod`, a secondary structure model. The model is created by aligning the `predict.db1.seq` sequences to `db1.test1.mod`, weighting the sequences, and then tabulating the frequencies of each letter in each model node according to the generated sequence weights. The frequencies are regularized with the ELH default regularizer, and transitions are copied from `db1.test1.mod`. See Section 9.4.3 on page 83.

The model and the prediction can be visualized with `makelogo` using the `logo_captionf` option (Figure 17).

In creating a secondary track model, `FIM_method_train` and `Insert_method_train` are used to

set the FIM and Insert tables. It is highly recommended to set these to 0 to use the regularizer (equivalent to the generic node) rather than the `buildmodel` letter-count defaults.

10.10 Model manipulation

10.10.1 `checkmodel`

The `checkmodel` program reports various information about a SAM HMM, including alphabet, name, and length, as well as the number of each type of special node the model includes. Also, the total number of labels are reports (See Section 9.7 on page 87.) and the number of environment codes (See Section 11.1 on page 147.). Its output is to `stdout`, but a run name must be provided anyway:

```
checkmodel test -modelfile test.mod
```

10.10.2 `drawmodel`

The `drawmodel` program is a means of generating a postscript drawing from a model, regularizer, or frequency count data (created with the `print_frequencies` option). The program is run with two arguments, the first being a model file, and the second the output file.

```
drawmodel model.mod drawing.ps
```

The program will scan the file, looking for models, regularizers, and frequency counts, and query whether or not each one should be printed, after presenting a line from the file.

There are two drawing options: overall and local. Overall is the correct option for frequency counts — the outgoing transitions of each state are drawn in different styles depending on the fraction of all sequences that use that transition. The circular delete states show the node number, while the diamond insert states show average number of characters, rounded up, inserted by each sequence that used the insert state.

In the local option, suitable for models and regularizers, transitions from a given state are drawn according to what percentage of sequences in that given state take each transition. Also, the diamond insert states have the percent of sequences which, once within the insert state, remain in the insert state. (See Figure 4 on page 21.)

The `drawmodel` program has several command-line options: `-landscape` to draw models in landscape format, and `-scale num`, to change the scale of the drawing to an arbitrary floating-point number. The default is portrait mode with a scale 0.235, which fits six row of 19 protein model nodes (plus one ghost node, the first model of the next row) on each page. Larger scale settings increase the size of the model nodes, and cause fewer to be placed on each line. For additional customization, the postscript file, which is readable, can be modified (e.g., to change print on a different size of paper).

If `-mod n`, `-freq n`, or `-reg n` is specified on the command line, the *n*th model, frequency count,

or regularizer will be selected for printing, and the interactive queries on which model to print will not occur.

Europeans and other people who like the A4 paper size can change `‘/A4 false def’` to `‘/A4 true def’` near the bottom of the header commands in the output of `drawmodel`.

10.10.3 `hmmconvert`

The `hmmconvert` program takes a model file as input and outputs a new model file in the ‘opposite’ format (binary or text) from the original.

The command syntax is as follows:

```
hmmconvert runname -model_file test.mod
```

If `test.mod` is in text format, `runname.mod` will be in binary format and vice versa. If you wish to destroy your original model file, use a runname of the file’s name without the `.mod` extension.

```
hmmconvert test -model_file test.mod
```

If `test.mod` is in text format, this command will create a new file called `test.mod` in binary format. If `test.mod` is binary, the new `test.mod` will be text.

To preserve your original model file, enter a new runname.

```
hmmconvert new -model_file test.mod
```

This will create a file called `new.mod` in the opposite format of `test.mod`. `test.mod` will be left alone.

10.10.4 `makelogo`

Sequence logos (Schneider and Stephens, NAR 18:6097–6199, 1990) are an excellent means of viewing model match state distributions (Figures 3 and 17). Each HMM match state is represented by a bar. The bar height corresponds to the negative entropy in bits of that match state distribution. Each bar is composed of letters in the target alphabet and the letter ‘X’. The size of the letters correspond to the relative frequencies of the letters; letters of low frequency are lumped together into the ‘X’ character. FIMs are represented with a large letter ‘O’. The `makelogo` program has default internal colors for protein, nucleotide, and secondary structure alphabets.

The simplest way to run the program is with just a model file to create the postscript file `test.eps`, as:

```
makelogo test -model_file test.mod
```

The logos are colored according to the alphabet of the model; new color schemes can be specified with the `logo_color` file name. The internal defaults are in the example files `protein.colors`,

3chy primary structure model

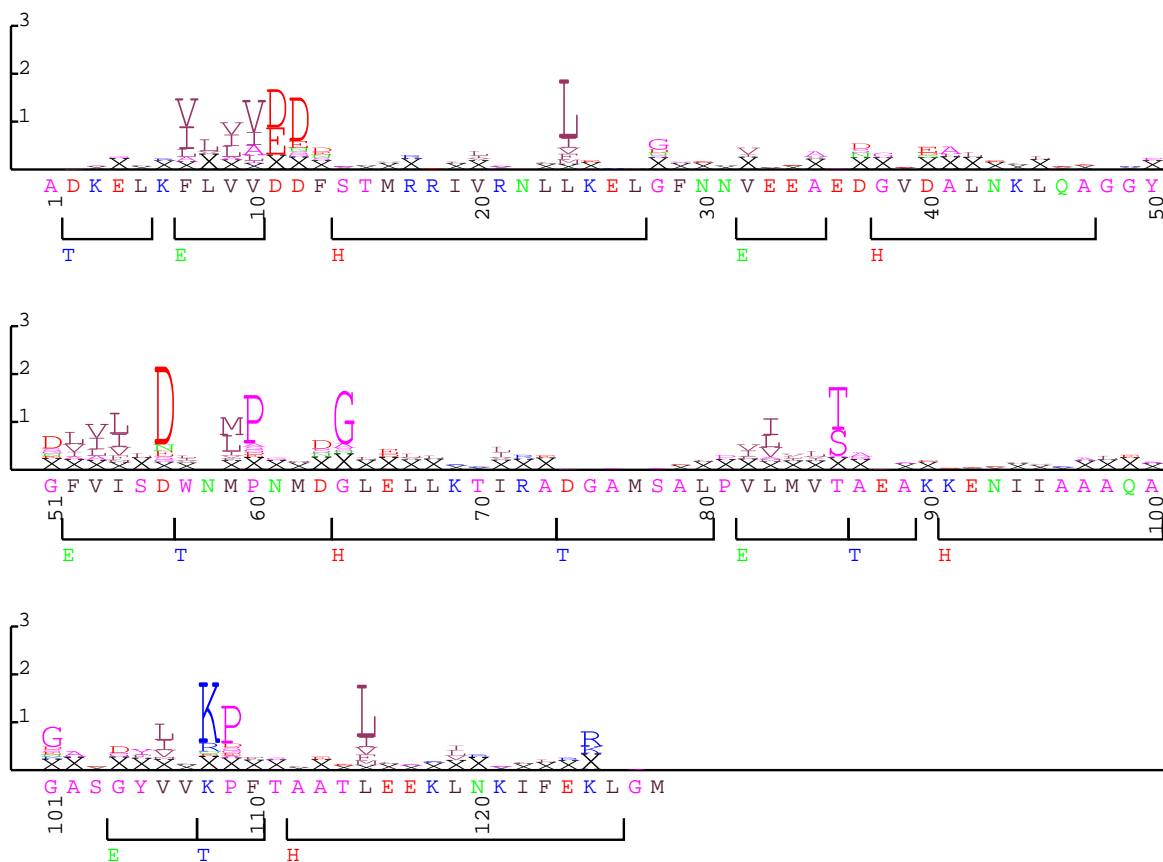


Figure 18: Primary structure model of 3chy with sequence and secondary structure

`nucleotide.colors`, and `stride.colors` included in the SAM distribution, and can be modified to create new color schemes. Each line of the file is either a comment (preceded by an asterix) or includes a single letter and a triple of RGB values between 0.0 and 1.0. The `makelogo` file will look for color files in the current directory as well as the `PRIOR_PATH`. See Section 11.1 on page 147.

The logo diagram can be annotated with a title (`logo_title`), and also with sequences in three ways. First, a single sequence (with dots, but not lower-case letters, ignored), such as a protein that the model is based on, can be displayed under the graph using `logo_under_file` (and the color file `logo_under_color` if black is not desired). If a sequence (or sequences, when `logo_captiofnf_numseq` is set to a number larger than 1) of structure is specified with `logo_captiofnf` (and the corresponding `logo_captiofnf_color`), repeated occurrences of the secondary structure character will be indicated as a range of positions with that the character for each sequence in the file. Finally, a file that alternates lines of start and end positions, and annotations, can be specified with `logo_caption`. All three of these options can be used simultaneously.

For example, Figure 18 shows a SAM-T2K (the pre-release successor to SAM-T2K) model for the

3chy secondary structure model

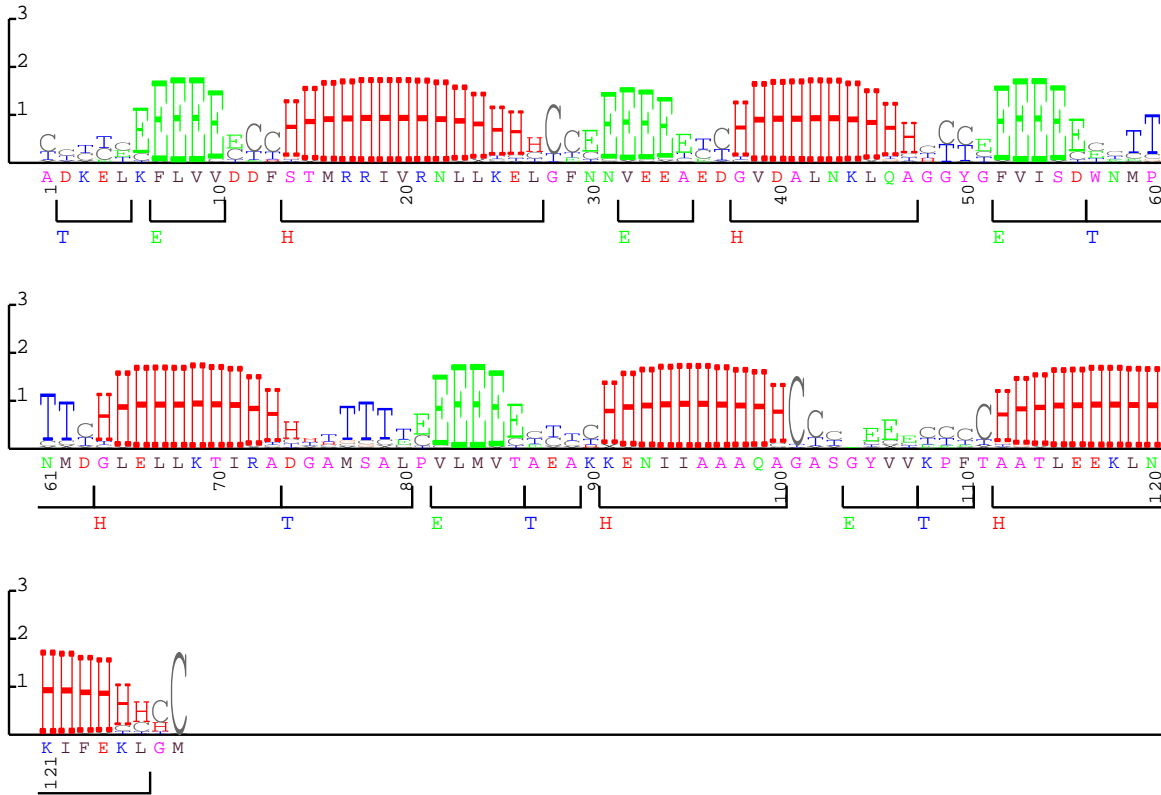


Figure 19: Predicted secondary structure model of 3chy with sequence and true secondary structure

PDB sequest 2chy. The logo diagram includes the 2chy sequence under the sequence, as well as the secondary structure, both input from separate FASTA-format files. The following Figure 19 shows a SAM-T2K secondary structure prediction HMM calculated without the aid of the secondary structure sequence for 2chy. The logo again includes the 2chy sequence as a `logo_under_file`, and the true secondary structure as a `logo_captionf`.

The program has a number of other options (all beginning with `logo_`). See Section 12 on page 154.

10.10.5 `modifymodel`

`modifymodel` is a utility program for modifying SAM models. It has an interactive type in interface as well as command line arguments. This program will grow over time to meet the needs of users. It contains extensive built in help with examples and a question mark (?) will list the commands:

```

AddNodes nodeNumber count #adds generic nodes after specified
Change subCommand...      #has sub commands, change ? to see
DeleteNodes nodeSpec      #deletes the specified nodes
DuplicateNode nodeNumber count #copies the specified node
NewModel modelLength      #make new model from default generic
PrependToCurrentModel fileName #reads model from file, pastes on front
PostpendToCurrentModel fileName #reads model from file, puts on end
ReadModel fileName
Show subCommand...
WriteModel [fileName] #defaults to name of file last read in
Quit
Help

```

Most (at the moment all) operations act on the current model: which is the last read in model. The basic usage is to read a model in, do what you want to it (add nodes, change types, paste models together, etc.) and write it back out.

The currently supported functions are

- Changing node types.
- Adding/duplicating/deleting nodes.
- Cutting models into smaller parts.
- Pasting models together.

To print all the types of the nodes (use this to see how long the model is):

```
show type all
```

To change all nodes that do not have a type into KEEP nodes,

```
change type - K ALL
```

To change node 9 into a FIM:

```
change type + F 9
```

To shorten a model by deleting nodes 4 through the end:

```
delete 4,end
```

Below is an example of reading a model, deleting the first 20 nodes, adding a new node to the front and changing it to a FIM, then pasting it on to the end of a second model, and finally writing the modified model out.

```

read one.mod
delete 1,20
add 0 1
change type + F 1
prepend two.mod
writemodel both.mod
quit

```

10.10.6 sam2psi and psi2sam

The `sam2psi` program converts the character probabilities in the match states to a PSI-BLAST (Altschul *et al.*, NAR 25:3389-3402, 1997) profile. This file can then be loaded into the PSI-BLAST program for searching.

The program creates two files: a PSI-BLAST checkpoint profile, based on the format used by version 6.60 (5/30/2003) of `posit.c`, the PSI-BLAST module of `blastpgp`. The checkpoint is called `runname.ckp`. PSI-BLAST also requires a query sequence that exactly matches the sequence specified in the checkpoint. SAM HMMs do not have a single sequence that can describe the generation of the model. For these profiles, `sam2psi` creates a `runname.cks` checkpoint sequence file for a synthetic sequence based on the most likely amino acid to be generated by each match state. This sequence is also present in the checkpoint file.

The commands are thus:

```
sam2psi test -modelfile test.mod
blastpgp -i test.cks -d nrp -R test.ckp
```

The `psi2sam` program will read a PSI-BLAST checkpoint file and produce a SAM model. It will use the default or specified transition regularizer, and the current `train_reset_inserts` to set the insertion state probabilities. See Section 8.6 on page 73. ‘X’ states begin as the background distribution of the regularizer (the PSI-BLAST checkpoint files do not include probability tables for ‘X’ positions), and are similarly changed according to `train_reset_insert`. The program will output (to stdout) the PSI-BLAST query sequence for verification.

```
psi2sam testp -modelfile test.ckp
```

10.10.7 Conversion between SAM and HMMer

SAM and HMMer are the two most widely used HMM sequences alignment and modeling systems. Even though they both employ hidden Markov models, their file formats are vastly different. Provided with this release are the conversion programs `sam2hmm` and `hmm2sam` that make almost complete conversion between SAM and HMMer v1.7 possible.

Martin Madera and Julian Gough have written a perl converter between SAM and HMMer 2.0 formats, available from <http://www.mrc-lmb.cam.ac.uk/genomes/julian/convert/convert.html> and the SAM WWW site. Use of this program is strongly suggested.

The conversion is “almost” because there is some information loss, and this information loss is due to the structural differences between the two systems. For those users who wish to use the conversion utilities, we discuss these issues below.

10.10.7.1 Internal HMM Structure Both SAM and HMMer use the same linear model of nodes that contain 3 states – match, delete, insert. The number of transitions per node is the same, 9. The internal difference is in how transitions between nodes are viewed. SAM views transitions INTO a node, thus each node contains the transitions FROM the previous node. HMMer is the opposite, transitions are TO the next node. HMMer’s convention makes more sense because each

triple of transitions stored in a node sums to one. SAM's convention, slightly helpful in the internal dynamic programming loop, is maintained for historical reasons. This becomes a problem at the beginning and end of the model.

The begin node of HMMer has a valid transition probability distribution out of the MATCH state (in that the transition probabilities sum to one), but has the null distribution for matching characters. This means HMMer always starts in the insert state, and the begin node match state is not used. The begin node of SAM can start in either non-character-generating match or an insert. So the SAM to HMMer conversion is fine, but HMMer to SAM conversion leaves SAM's begin node match state character table zero. The net result is that SAM is forced to start in the insert state.

The end node of HMMer also has a zero match state character table.

The two programs are used as follows:

```
hmmmer2sam hmmmerfile.hmmmer  newsamfile.mod
sam2hmmmer newhmmmerfile -i samfile.mod
```

Above, the output of `sam2hmmmer` is stored in the file `newhmmmerfile.hmmmer`.

10.10.7.2 Extra Information in HMMer HMMer allows other information on a per node basis. This information seems to be solvent access and consensus information. This information cannot be used in SAM at this time, and is silently dropped. A note of the global presence of this extra information is noted as a comment in the SAM model, solely for documentation purposes.

10.10.7.3 Extra Information in SAM SAM has constructs in the model that are not supported in HMMer, and these constitute the greatest difference in the two systems. SAM has per node "types" that control learning and evaluation parameters. The problematic type is the Free Insertion Module (or FIM). This special type allows SAM to have a single node give equal cost for all characters by effectively matching one or more characters. These FIMs can be positioned anywhere in a model, but are most commonly positioned at the ends (the begin node and the node before the end node). HMMer has no such concept, and handling FIMs in a conversion is a problem. FIMs at the begin and end are not too critical, because HMMer's system always treats the begin and end just like they are free inserts (also, HMMer re-normalizes all nodes so any strange distributions are "fixed"). FIMs in the middle of a model can NOT be converted in an exact manner. Due to the structure in SAM, some probabilities in a FIM node do not sum to one and the match state transition probabilities are all zero. During conversion, these nodes have to be "fixed" so that valid (sum to one) probability distributions are given to HMMer.

One way to fix up a FIM is to remove it: it maps to nothing in a HMMer model. This method is not too bad if the FIM does not usually match many characters, but is very bad if the FIM is used a lot. A second way is fabricate some normal cheap insert states and hope that they match the average length of the FIM. This method is hard, since there is not enough information to guess the number of insert nodes to add. (If the frequencies are present in the SAM model, then this can be done. However this information is not always present.). `sam2hmmmer` utilizes the former option. The transition probabilities for the match state are computed from the exit probabilities of the FIM (a chain of FIMs in a row are removed as they were one). A SAM FIM has exits from the insert and delete states, and the previous non-FIM node has a transition from match to insert. The new transitions are computed from these numbers by breaking the match to FIM transition into two

parts that have the same ratio as the exit probabilities of the FIM. This same computation is done to compute the new transitions from delete and insert.

It is not possible to record any extra information from SAM into a HMMer model as HMMer does not support comments in the file.

10.11 Plotting Programs

SAM has several plotting programs to assist in data analysis. The programs require **gnuplot** http://www.cs.dartmouth.edu/gnuplot_info.html to be in the user's path. In general, the programs create one or more **.data** files, a **.plt** file of **gnuplot** commands, and a postscript file.

Options common to the programs include

plotcolumn — Column of score file to use in calculating plots. Length (0), simple null model (1), complex or reverse null model (2), or Evaluate (3, the default). If Evaluates are requested but not present, then column 1 is used.

plots — Creates a postscript file **runname.ps** using **gnuplot** if set to 1, 2, or 3. When set to 0, only a **.plt** file and one or two **.data** files are generated. A setting of 1 generates a plot in **gnuplot**'s default rectangular shape, while a setting of 2 generates a square plot. For options 1 and 2, the **.data** and **.plt** files used to create the postscript file are deleted. When set to 3, the postscript file is generated and the **.data** and **.plt** files are retained. The default setting is 1.

plotleft — Lowest X axis value on a graph generated by **gnuplot**. The X axis is calculated internally if **plotleft=plotright**. The default setting is 0.0.

plotright — Highest X axis value on a graph generated by **gnuplot**. The X axis is calculated internally if **plotleft=plotright**. The default setting is 0.0.

plotmax — Highest Y axis value on a graph generated by **gnuplot**. The Y axis is calculated internally if **plotmax=plotmin**. The default setting is 0.0

plotmin — Lowest Y axis value on a graph generated by **gnuplot**. The Y axis is calculated internally if **plotmax=plotmin**. The default setting is 0.0.

plotline — Creates a vertical line at this value in a graph generated by **gnuplot** if **plotline** is nonzero. The default setting is 0.0.

plotnegate — Negates the scores on a graph generated by **gnuplot** if set to 1. The default setting is 0 (off).

10.11.1 makehist

The **makehist** program can be used to make postscript histograms from one or two **.dist** or **.mstat** distance files created by **hmmscore**. Histograms are most useful to discover and examine the separation between family members and non-family members during a database search. The **makehist**

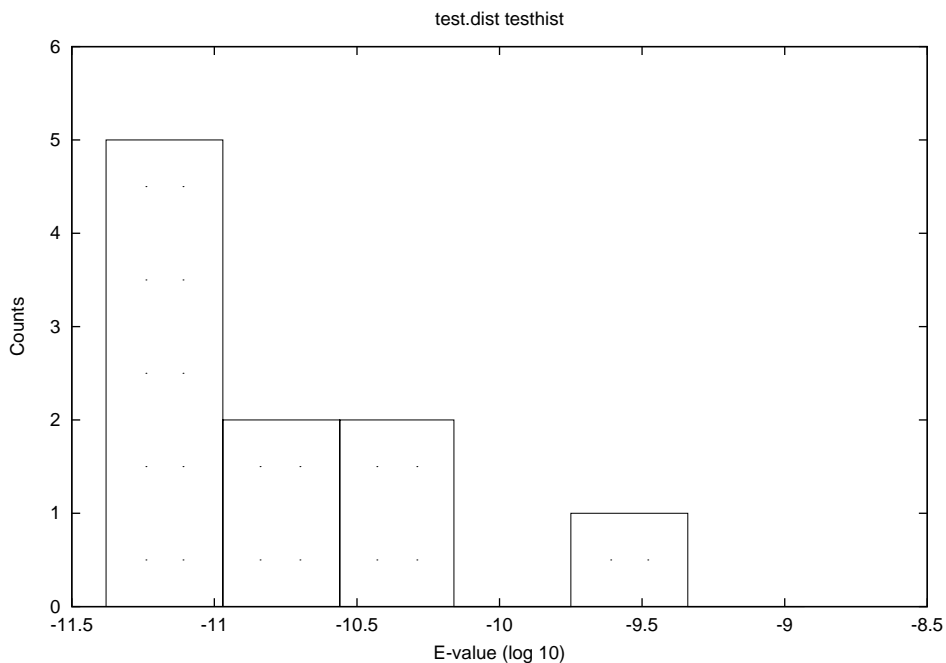


Figure 20: A histogram generated by `makehist` and `gnuplot`.

program requires a `distfile` as an argument, in addition to the runname. For example, to make a histogram from `test.dist`, the following command would be used:

```
makehist test -distfile test.dist
```

Then, to view the histogram, use `ghostview` or a similar postscript viewer to view `test.ps`. If you would like to use `gnuplot` directly, set `plotps` to 3 to keep all the intermediate files. Then, run the `gnuplot` program and enter the command `load "test.plt"` to view the results. The `test.plt` file may be modified to change the graph. If the two marked lines in the file are uncommented, a postscript file will be generated.

If a second file is specified, using the `distfile2` option, a second histogram is placed above the first one. The `makehist` program has an optional argument, `histbins`, which can be used to set the number of bins between which scores should be divided. The histogram in Figure 20 was generated using five bins.

10.11.2 makeroc

The `makeroc` program generates a postscript graph file using the `gnuplot` plotting package. It takes two `.dist` distance files created by `hmmscore` as input and plots false negatives/ false positives on the axis Score vs. Counts where Score is the NLL-NULL score of a sequence and Counts is the number of sequences in the files with a particular score.

To plot false negatives vs. false positives in two distance files `globins.dist` and `nonglobins.dist`,

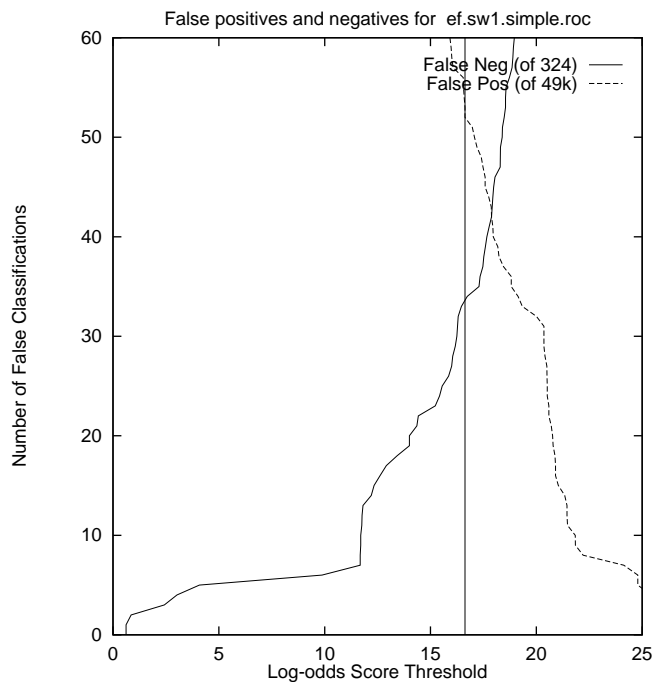


Figure 21: A plot generated by `makeroc` and `gnuplot`.

the following command would be used:

```
makeroc test -distfile globins.dist -distfile2 nonglobins.dist
```

The program will output a file `test.ps` which you can view with `ghostview` or other postscript viewer.

10.11.3 `makeroc2`

The `makeroc2` program generates a postscript graph file using the `gnuplot` plotting package. It takes two `.dist` distance files created by `hmmscore` as input and plots false negative vs. false positive scores, each of which are a function of the setting of threshold score. At each integral point on, for example, the false positive axis, the value of the false negative coordinate is a linear interpolation between the two nearest integral false negative values based on the scores of the two false negative points and the false positive point in question. This means, for example, that if two negative examples score 10 and 20, and two positive examples score 19 and 21, the point (1,1.1) will be plotted for the score of 19, indicating that if the threshold is set at 19, there will be one false negative (the sequence that scores 21) and a little over one false positive (the sequence scoring 19 is 1.1 false positive examples).

To plot false negatives vs. false positives in two distance files `globins.dist` and `nonglobins.dist`, the following command would be used:

```
makeroc2 test -distfile globins.dist -distfile2 nonglobins.dist
```

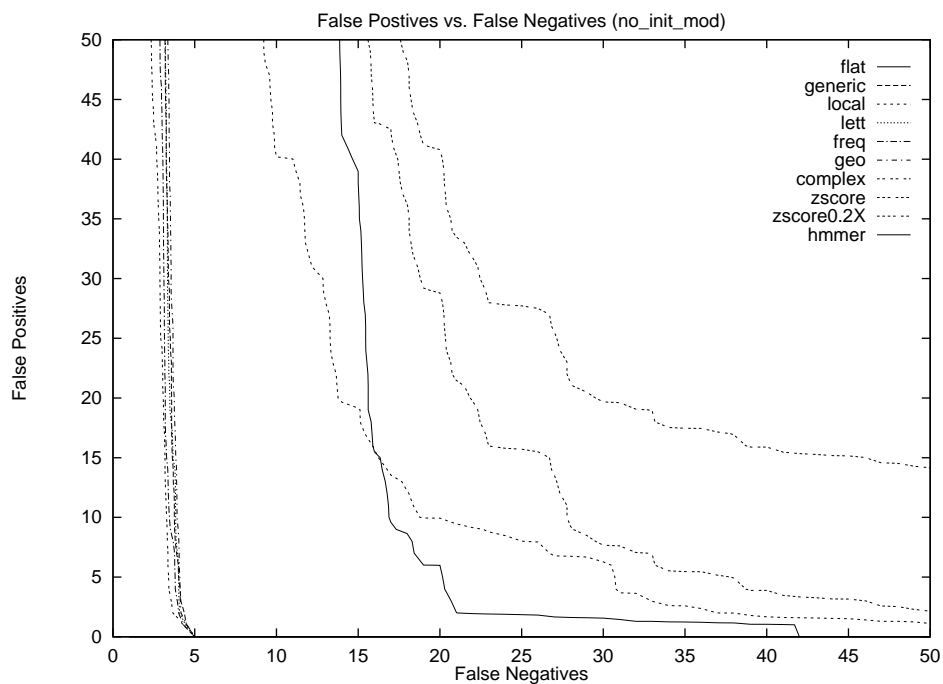


Figure 22: A plot generated by `multi_roc2.pl`, `makeroc2` and `gnuplot`.

The program will output a file `test.ps` which you can view with `ghostview` or other postscript viewer.

To plot multiple pairs of distance files on the same axis, use the perl script `multi_roc2.pl`.

```
multi_roc2.pl true1.dist false1.dist title1 true2.dist false2.dist title2
true3.dist false3.dist title3 . . .
```

The titles appear in a legend in the upper right hand corner of the plot and identify the individual curves.

10.11.4 makeroc3

The third of the series is `makeroc3`. This program is given two data files, as with `makeroc2` and prints numerical information about a positive distance file and a negative distance file. With the command

```
makeroc2 test -distfile true.dist -distfile2 false.dist
```

the program will output a `test.dat` file of the form

```
flips between x:2.000000, y:2.280622 score:-36.376999 and x:3.000000, y:2.263981
score:-36.743000
3 sequences at FP=FN
7 FN at FP=0
2 FP at FN=0
```

10.12 Sequence manipulation

The programs described in this section can be used to perform many helpful tasks.

10.12.1 listalphabets

The `listalphabets` program reports default information for the internal alphabets. If run with only an (ignored) `runname`, the program will list all alphabets, valid characters, and default regularizers. If the program is provided an alphabet argument, such as

```
listalphabets test -a protein,EHL
```

it will provide information about the multi-track alphabets listed, as shown in Figure 23. The program provides entries for all possible tracks, showing the maximum number of sequence tracks that can be used at one time. The entries listed under “Convert” are pairs of letters, where the first one is automatically converted to the second, such as the merging of various types of coils, including ‘L’, into the single ‘C’ class of the (slightly) misnamed EHL alphabet. See Section 7.1 on page 49 and Section 8.1 on page 59.

10.12.2 checkseq

The `checkseq` program will read a sequence file and list various pieces of information about it. For example,

```
checkseq unused_runname -db trna10.seq
```

will produce the output

```
Alphabet: RNA
Input Format: pearson
Alignment: no
AlignType: unaligned
AlignColumns: 0
Num Sequences: 10
Average Length: 74.70
Max Length: 77
Min Length: 72
Total Length: 747
```

SAM internally recognizes several types of alignments. The check proceeds as follows: if the file is an HSSP file, it is considered an alignment. Otherwise, if the file is an a2m format file with the same number of upper case (match) and hyphen (delete) characters in each line, it is an a2m format alignment. Otherwise, if all characters have the same total number of upper case, lower case, hyphen, and period characters, it is an “all positions alignment” — if all positions are regarded as match columns, the file can be viewed as an alignment.

If the `sort` variable is set on the command line, `checkseq` will indicate whether or not all IDs are unique and whether or not all sequences are unique. This can be time consuming for large data files,

```

##### Track 0 alphabet #####
#InternalID:      3
Name:             protein
Comment:          Protein with B=N/D and Z=Q/E
Length:           20
WildCards:        3
FLength:          20
FWildCards:       3
Letters:          ACDEFGHIKLMNPQRSTVWYBZX
Convert:          <none>
MatchRegularizer:
  A 0.087130  C 0.033470  D 0.046870  E 0.049530
  F 0.039770  G 0.088610  H 0.033620  I 0.036890
  K 0.080480  L 0.085360  M 0.014750  N 0.040430
  P 0.050680  Q 0.038260  R 0.040900  S 0.069580
  T 0.058540  V 0.064720  W 0.010490  Y 0.029920
# Total match regularizer weight is 1.000000
InsertRegularizer:
  A 0.087130  C 0.033470  D 0.046870  E 0.049530
  F 0.039770  G 0.088610  H 0.033620  I 0.036890
  K 0.080480  L 0.085360  M 0.014750  N 0.040430
  P 0.050680  Q 0.038260  R 0.040900  S 0.069580
  T 0.058540  V 0.064720  W 0.010490  Y 0.029920
# Total insert regularizer weight is 1.000000
TransitionRegularizer:
  IntoDeleteFrom: D 1.886984 M 0.254944 I 0.376488
  IntoMatchFrom:  D 1.819972 M 15.521340 I 3.764209
  IntoInsertFrom: D 0.225758 M 0.265967 I 4.006562
##### Track 1 alphabet #####
#InternalID:      5
Name:             EHL
Comment:          Secondary structure alphabet
Length:           3
WildCards:        1
FLength:          3
FWildCards:       1
Letters:          EHCX
Convert:          GH,LC,TC,SC,BC,IC
MatchRegularizer:
  E 0.104000  H 0.178000  C 0.214100
# Total match regularizer weight is 0.496100
InsertRegularizer:
  E 0.104000  H 0.178000  C 0.214100
# Total insert regularizer weight is 0.496100
TransitionRegularizer:
  IntoDeleteFrom: D 1.886984 M 0.254944 I 0.376488
  IntoMatchFrom:  D 1.819972 M 15.521340 I 3.764209
  IntoInsertFrom: D 0.225758 M 0.265967 I 4.006562
##### Track 2 alphabet #####
##### Track 3 alphabet #####
##### Track 4 alphabet #####
##### Track 5 alphabet #####

```

Figure 23: Output of listalphabets test -a protein,EHL.

which is why `checkseq` will ignore the default setting of `sort` if it is non-zero.

10.12.3 `genseq`

The `genseq` program generates random sequences from either a standard 1-component regularizer, such as SAM's default per-alphabet regularizers, or a Dirichlet mixture regularizer. In this case, the Dirichlet mixture must be a mixture of sequence compositions, not a mixture of alignment column distributions, as is used in `buildmodel`.

```
genseq rseqs -nseq 20 -a protein
```

The SAM system includes default sequence generation multi-component regularizers for the protein alphabet (`rsdb-comp2.32comp.gen`) the EHL2 secondary structure alphabet (`t99-2d-comp.9comp.gen`), and the EBGHTL secondary structure alphabet (`t99-ebght1-comp.6comp.gen`), which are the default settings of the `genprot_prior`, `genehl2_prior`, and `genebght1` variables. If a `track_prior` is not specified as a single prior file (`genseq` cannot generate multi-track sequences) and alphabet is protein or EHL2, the `genprot_prior` or `genrh12_prior` is used. Otherwise, the `regularizer_file` is used, or the default internal regularizer.

The length distribution of the generated sequences is based on the normal distribution. The mean and standard deviation are presented in (natural) log format as `gs_mean_log_len` (default 5.4151) and `gs_sd_log_len` (default 1.03632564). The default values are intended for protein sequences.

The sophisticated distribution sampling source code (designed and written by Kevin Karplus) at the core of these routines is available at:

<http://www.cse.ucsc.edu/research/compbio/dirichlets/index.html>

10.12.4 `permuteseq`

The `permuteseq` program requires a run name and a database file. Its output will be a file of sequences that are permuted copies of the sequences in the database file. If `Nseq` is specified, the sequence file will be looped through multiple times until the requested total number of sequences are created. That is, in a file of 10 sequences, if `Nseq` is 15, the output `.seq` file will contain 10 random sequences, one for each original sequence, followed by 5 random sequences, one for each of the first 5 sequences in the file.

The `permuteseq` program can be used to verify scoring results. For example, the command

```
permuteseq permuted -db trna10.seq -Nseq 8 -id TRNA1 -id TRNA2 -id TRNA3  
hmmscore permuted -i test.mod -db permuted.seq -sw 2
```

produces the following distance file, which shows all the permuted tRNA sequences as having poor scores.

Rand5-TRNA2	76	-7.31	-0.34	3.32e+00
Rand7-TRNA1	72	-7.94	-0.28	3.44e+00
Rand1-TRNA1	72	-7.94	-0.27	3.47e+00
Rand2-TRNA2	76	-7.78	-0.24	3.51e+00
Rand4-TRNA1	72	-7.03	0.21	4.41e+00
Rand3-TRNA3	76	-7.47	0.31	4.61e+00
Rand6-TRNA3	76	-7.48	0.67	5.29e+00
Rand8-TRNA2	76	-7.22	0.83	5.57e+00

10.12.5 randseq

The `randseq` program can be used to randomly sample a database file. It requires a database file (`db`), and a number of sequences (`Nseq`), and can optionally be provided a random number seed (`trainseed`). For example,

```
randseq rseqs -db nrp -nseq 1000
```

would place 1000 sequences randomly selected from `nrp` into the file `rseqs.seq`.

The `randseq` program performs the random selection in memory.

10.12.6 splitseq

The `splitseq` program can be used to split a database according to a sequence length threshold. It requires a database file (`db`), and a maximum sequence length (`max_seq_length`). For example,

```
splitseq split -db nrp -max_seq_length 1000
```

would place sequences of length less than or equal to 1000 in `splitA.seq` and sequences of length greater than 1000 in `splitB.seq`.

The `splitseq` program performs the selection in memory. It is particularly useful with posterior-decoded alignment styles because they are not yet implemented with a reduced space algorithm.

10.12.7 sampleseqs

The `sampleseqs` program will, given a model, randomly generate sequences according to the model's probabilities. For example,

```
sampleseqs sample -i test.mod -Nseq 5
```

will produce the following `sample.seq` file of synthetic tRNAs:

```

>SampleSeq0
AUUAGCCCAUUUGCGACCGCACCGGACUCGCGAGCCUGUAACGCAGGUUCGAAUCCCCCGGGCCACCAAUGACCGGUGUG
>SampleSeq1
GUCAGUAGCUUACGUCGUACCAUACUGCUGCGUGAACCCUCGUGUCGAGUCCCUUCGACUCGCGCAAUUCUGCGGGCGGCU
>SampleSeq2
GUCCCCCGUGGGCGACCGGUAAGACACUCGCGUUGCAUUCGGGAAGUACGUGGGUUUACAUCGCCGAGGGCCACUUCGGCCCGUUGUCGGUGUUGACCUGUC
>SampleSeq3
GCGGUGUGGGCGACGGUGUACAGAACAUGUCCCGCAAUCUAGUAUCCACGGUUCAAUUCUGCGUAGCUGAAUCCACGG
>SampleSeq4
CCGUAGCCUAACUGAGCAGGCUUUAGCUCGCGUAUUGGGCAGGCAGGACUCGCGCGUUAACCCCAAUCGGGGCGCGAUC
>SampleSeq5
UCGUGGGUUAGGGUCGAGGGCUGGCUUGUCAGGCUGCAGUCGCAAGGAGACAGGCUCACCGGGGCCUCCAAUCG
>SampleSeq6
CACGUCGUUCAGUGUCAACACCUUGGACUUGCAAUCGGGAACCACCGGUUCAAGGCCUCCGAGUCCAGGACGCU
>SampleSeq7
GUCCGAGCCUAGGUCGUACGGUAGUCUCGCUCUUUCGGAAGCACGGUUUCGAUCCUGUGGCCCGCGCGCCGG
>SampleSeq8
CGGGGAUUAAGGAGGGAGCGCGCAAGCUUAGCAAGGGGGCUGCCGGGUCAAUCCUGCCGUGUCCCCCAUAUCC
>SampleSeq9
CACAGAGCGCCGGGUUAUAGCAUCGGUUUUCAAACCAGGGCGCUCGGUCGAAACCGGCUCGCCGCCCUUCCAGCAAGCG

```

Run with no arguments for a usage message.

10.12.8 sortseq

The `sortseq` program takes as input a database file(s) of sequences (`*.seq` or `*.seq`) or an input `-alignfile` (`*.a2m` or `*.mult`) and a distance file (`*.dist`) of scores generated by `hmmscore`. It outputs a file of sequences or alignment (depending on input) in the same order as they occur in the distance file. `Sortseq` provides an enhancement to `hmmscore`. It outputs a `.seq` or `.a2m` file that contains the actual sequences in sorted order while `hmmscore` outputs a file only containing sorted sequence id's and scores.

For example,

```
sortseq sort -db trna10.seq -distfile test.dist
```

will produce the following `sort.seq` file of sequences:

```

>TRNA7
GGGCACAUGGCGCAGUUGGUAGCGCGCUUCCUUGCAAGGAAGAGGUCAUCGGUUCGAUCCGGUUGCGUCCA
>TRNA2
GCGGCCGUCGUCUAGUCUGGAUAGGACGCGUGGCCUCCCAAGCCAGCAAUCCCGGGUUCGAAUCCCGCGGCCGCA
>TRNA4
GGGCGAAUAGUGUCAGCGGGAGCACACCAGACUUGCAAUCUGGUAGGGAGGUUCGAGUCCUCUUUGUCCACCA
>TRNA9
CGGCACGUAGCGCAGCCUGGUAGCGCACCGUCCUGGGUUGCGGGGUCGAGGUUCAAUCCUCUCGUGCCGACCA
>TRNA10
UCCGUCGUAGUCUAGGUGGUAGGAUACUCGGCUUUCACCCGAGAGACCCGGGUUCAAGUCCCGGCGACGGAACCA
>TRNA1
GGGGAUGUAGCUCAGUGGUAGAGCGCAUGCUUCGCAUGUAUGAGGCCCCGGGUUCGAUCCCGGCAUCUCCA
>TRNA8
GGGCCCGUGGCCUAGUCUGGAUACGGCACCGGCCUUCUAAGCCGGGAUCGGGGUUCAAAUCCUCCGGGUCGG
>TRNA5
GCCGGGAUAGCUCAGUUGGUAGAGCAGAGGACUGAAAAUCCUCGUGUCACCAGUUCAAAUCUGGUUCCUGGCA
>TRNA3
GGCCUUGUGCUAGCUGGUCAAAGCGCCUGUCUAGUAAACAGGAGAUCCUGGGUUCGAAUCCAGCGGGCCUCCA
>TRNA6
GGGGCCUAGCUCAGCUGGGAGAGCGCCUGCUUUGCAGCAGGAGGUCAGCGGUCGACCCGCUAGGCUCACCA

```

10.12.9 uniqueseq

The `uniqueseq` program takes as input a database file(s) of sequences (`*.seq` or `*.se1`) or a single alignment file. It sorts the sequences and outputs a file containing every sequence with a unique ID (i.e. no dupes are copied to the output file). It also outputs messages to the user informing of the following conditions in the database file: same id for different sequences; same sequence for different ids; and duplicate id found.

If an alignment file is specified the fractional `percent_id` of the sequence's match characters as aligned are present in another sequence. If two sequences mutually satisfy this threshold, the one that is more identical to the other is removed, or the one that has the fewest total number of match columns if their relative percentages are the same. The order in which this is performed is undefined, so that when `percent_id` is less than the default 1.0, it is possible to get to different results with different orderings of an alignment due to transitivity considerations. If `percent_id` is negative, a message is printed about each sequence that is dropped.

If an alignment file is specified and `aligncheckonly` is cleared to 0 (from the default 1), then duplicate sequences are removed from the alignment based only on the sequence data (not the alignment) prior to thinning the alignment.

If the optional `-train mytrainfile.seq` is used, the program checks the sequences in the `trainfile` and writes any sequence not in the database file to the output file. In this case, the following user information messages are output: training sequence (ID number) is in database with different ID (ID number); training sequence (ID number) is in database with same ID; training sequence (ID number) not in database (note: the program currently checks only one training file).

For example,

```

uniqueseq unique -db testme.seq -train trna10.seq
uniqueseq unique1 -alignfile testme.a2m

```

will produce `unique.seq` and `unique1.a2m`, respectively.

Messages about which sequences are dropped are printed to standard error.

10.12.10 `mk_kestrel_db`

The `mk_kestrel_db` command creates databases for use with Kestrel `hmmscore`. These databases are in a format optimized for scoring with the Kestrel processor and parallel the standard sequence databases. For each `db` parameter specified, `mk_kestrel_db` will create a `.kids` file and either a `.kseq` file or a `.krseq` if `subtract_null` is set to 4. These files will be in the same directory as the `db` file and will have the same name, with the extension appended. The `runname` argument is ignored.

11 System installation

The SAM system runs on a variety of Unix workstations (we have checked installation on workstations including DEC DECstation and Alpha, HP 715, IBM RS6000, SGI Onyx Reality Engine, Sun Sparc, Intel Pentium with the Linux operating system, and the UCSC Kestrel parallel processor.

The distribution includes an `INSTALL` file that discusses installation procedures.

The `gnuplot`, `gunzip`, and `uncompress` programs should be in the user's path, and other programs should be available as required by SAM-T2K. See Section 4.11 on page 43.

11.1 Environment variables

The SAM system has several environment variables, typically set in `csh` using a command of the form `setenv PRIOR_PATH /projects/compbio/lib`.

PRIOR_PATH Directory with prior libraries, regularizers, and `makelogo` color definitions.. See Section 8.1.1 on page 60 and Section 10.10.4 on page 131.

BLASTMAT Directory for BLAST scoring matrices. See Section 10.2.8 on page 117.

11.2 Runtime statistics

At the end of each run of `buildmodel`, a line of statistics is printed out, such as the line

```
-218.36 -217.00 -217.68 0.96 22 0 149
```

mentioned in Section 3. These numbers are quite useful for quick comparison of results when, for example, running the program many times using a shell script. The numbers are: minimum NLL-NULL score, maximum score, average score, sample deviation of scores, number of re-estimates,

number of surgeries, and the length of the final model. In the above case, the scores are for the training set: if a test set were specified (Section 7.4), the minimum, maximum, average, and sample deviation for the test set would be reported after the model length, followed by the ratio of the average test set score to the average training set score (ideally, this value should be close to unity — larger values may indicate overfitting of the model to the training set).

11.3 Manipulating large collections of data

The underlying SAM programs were originally written to work with single models and single databases of sequences. Many current users, including UCSC, use SAM to create vast libraries of models and to analysis many independent, small files of sequences.

If this is also appropriate for your application, you will undoubtedly find yourself designing scripts, Makefiles, and databases in a manner similar to any other large project.

Several features in SAM may help you with this design

- If you have many models, you can place them in a model library, and score the group of them against one or more databases all at the same time. See Section 10.2.10 on page 119.
- If you have large database files and would like faster results, split the database into smaller segments, and run multiple `hmmscore` jobs at the same time. See Section 10.2.7 on page 116.
- If you have a large number of database files, and get tired of endlessly long `db` commands (or if you exceed system limits on commandline length), you can put all those `db` commands into a single file, and then use the insertion `i` command to include this database list, and score all the databases.

11.4 Future Features

There are many future features we would like to include in SAM. The following list will also point out some of the things you currently cannot do using the system. The items are of varying difficulty.

- Position-specific regularizer strengths to extend the special node concepts between entirely fixed and entirely free.
- Model learning and combining using genetic algorithms.
- A coarse-grain (MPI) parallel implementation.
- A version that can run on the Kestrel programmable parallel processor:

<http://www.cse.ucsc.edu/research/kestrel>

11.5 Prior versions

11.5.1 Version 2.2.1

- Corrections to MSF and FSSP file reading.
- Addition of SW 3 option for modeling a domain within a longer protein. See Section 8.5 on page 71.

11.5.2 Version 2.2

July, 1998

- Release of 2.1.1 and 2.1.2.
- More efficient implementation of Dirichlet mixture priors.

11.5.3 Version 2.1.2

June, 1998.

- Implementation of posterior-decoded alignments and output of posterior-decoded values of the dynamic programming operation. The `viterbi` variable has been renamed `dpstyle`. An alias to `viterbi` is currently included. See Section 9.5 on page 85, Section 10.1 on page 90, and Section 10.5 on page 124.
- Implementation of local and semi-local training. See Section 9.5 on page 85.
- Change in the definition of FIM probabilities: all outgoing arcs from a FIM node now have (unnormalized) unity probability (zero cost), as the internal delete to insert and insert to insert transitions have always had. FIMs continue to not have match states. Optionally, the FIM insert to insert transition can be set with `fimtrans`. See Section 8.5 on page 71.
- Change in the implementation of jumps for local and semi-local scoring. Jumps are now from the delete node after the initial FIM to an internal match state and from an internal match state to the delete node of the final FIM. Previously, internal delete states were used. See Section 10.1.2 on page 92.
- The extra delete state in alignments induced by automatically-added FIMs is now automatically removed for both alignments and multiple-domain alignments. If FIMs are present in the model and `auto_fim` is set, the delete state is also removed.
- The `prettyalign` FASTA-like format no longer includes semicolon-delimited comments.

11.5.4 Version 2.1.1

April, 1998.

- User-defined alphabets for sequences. See Section 7.1.1 on page 52.
- Negative `fimstrength` values will adjust both insert and FIM states. See Section 8.5 on page 71.

11.5.5 Version 2.1

February, 1998.

- The `multdomain` program has been removed and its function has been merged into `hmmscore`. The `mdNLLminusNULL` parameter has been renamed `mdNLLnull`. The `multdomainshort` parameter has been renamed `alignshort`. The old names are currently aliased to the new names for these two parameters. See Section 10.2.5 on page 111.
- The `hmmscore` program can now print selected sequence alignments and selected sequence multiple domain alignments during scoring. See Section 10.2.3 on page 106 and Section 10.2.5 on page 111.
- The interactive mode of `hmmscore` has been removed. See Section 10.2 on page 100.
- The scored sequence letter counts null model has been removed. Null model scores can now be calculated based on the reverse sequences. The `simple_threshold` variable determines when complex null model calculations, such as the reverse null model or the user's null model, should be performed in terms of the simple null model score. See Section 10.2.1 on page 102.
- The content of score files has changed, as has the use of `select_seq`, `select_score`, `sort`, and `subtract_null`. See Section 10.2 on page 100.
- The `uniqueseq` program has been updated. See Section 10.12.9 on page 146.
- The `checkseq` program has been updated. See Section 10.12.2 on page 141.
- Scoring examples in this manual have been changed to use fully-local scoring and `hmmscore` now prints a warning whenever fully-local scoring is not used. See Section 10.2.4 on page 107.
- The `protein_prior` and `nucleotide_prior` variables can be used to specify default prior libraries. The Dirichlet mixture `recodel.20comp` is now used by default with protein sequences. See Section 8.1 on page 59.
- Internal weighting in `buildmodel` is now by default turned on with `internal_weight` set to 1. If an external weight file is specified and `internal_weight` is not explicitly set on the command line, internal weighting will be turned off. See Section 9.4.4 on page 83.
- The `sequence_models` variable, when set, causes `buildmodel` to create initial models from random sequences in the training set which are then regularized. The each single sequence is given a weight equal to the value of `sequence_models`. This option is recommended and

is expected to become default behavior in a future release. It can both reduce runtime by providing an initial starting point when an alignment is not available and increase modeling performance. See Section 8.3 on page 65.

- The `seed_runs` parameter has been removed from `buildmodel`.

11.5.6 Version 2.0

November, 1997.

- A complete rewrite of the inner dynamic programming loop to save memory (see the Grice, Hughey, and Speck, and the Tarnas and Hughey papers mentioned in the introduction) and allow local and semi-local scoring and alignment, as well as Viterbi-based training. Memory use is now proportional to the product of model length and the square root of the sequence length rather than the model length and the sequence length. See Section 10.1.2 on page 92 and Section 10.2.4 on page 107.
- HSSP-based structural transition regularizer. See Section 8.1.2 on page 62.
- The `multdomain` program now performs scoring adjustments identical to those of `hmm_score` when `SW` is set. See Section 10.2.4 on page 107.
- Internal sequence weighting inspired by HMMer's Maximum Discrimination method has been implemented. It significantly increases discrimination performance in the presence of biased training sets. See Section 9.4.4 on page 83.
- The `a2mdots` variable can be cleared to avoid printing dots in `a2m` files, leading to an at times considerable space reduction. See Section 10.2 on page 100.
- The `hmm_score` program can partition a database to aid in distributed scoring. See Section 10.2.7 on page 116.
- SAM will now read its input from compressed (`.gz` or `.Z`) files. See Section 5 on page 45.
- The alphabet code has been rewritten to make it simpler for those with source licenses to modify the code.
- Weight files for `buildmodel` initial alignments and `modelfromalign` alignments can be specified with the `alignment_weights` parameter. See Section 9.4 on page 80.
- A broader collection of Dirichlet mixture and transition regularizers is included with this version. See Section 8.1 on page 59.
- The MasPar implementation is no longer supported.

11.5.7 Version 1.4

August, 1996.

- The geometric average of the match state probabilities is now available for use (and the default) with simple and complex null models. Complex null models are now built from the transition and insert probabilities of the model, and the geometric average of all the model's match tables in the match table. See Section 10.2.1 on page 102.
- The `train_reset_inserts` variable causes `buildmodel` to, at the completion of re-estimation cycle, reset all the insertion and FIM tables to (by default) the geometric average of the match states. Set to 0 to turn off. See Section 8.6 on page 73.
- If no IDs are specified, the `hmmscore` and `multdomain` programs will read in sequences a few at a time, instead of all at once, saving a tremendous amount of memory. If this is done, sequence output by `hmmscore` is no longer sorted by score, though the score file can still be sorted. Given a sorted score file and unsorted sequence file, the new `sortseq` program will sort the sequences according to the score file. See Section 10.2 on page 100, Section 10.2.5 on page 111, and Section 10.12.8 on page 145.
- The `uniqueseq` program will eliminate sequences with duplicate IDs from a file. The `checkseq` program will read a sequence file and print information about it. See Section 10.12.9 on page 146.
- Training noise is reduced by `retrain_noise_scale` (default 0.1) whenever an initial model or alignment is provided. Noise is also reduced between the first and successive surgery iterations by `surgery_noise_scale` (default 0.1). See Section 9.1 on page 74.
- Models can be edited using the new utility program, `modifymodel`. See Section 10.10.5 on page 133.
- The program `makehist` will turn one or two `.dist` score file into a histogram, `makeroc` will turn two `.dist` score files into a false positive/false negative plot showing score vs. counts (number of sequences with the score) and `makeroc2` will turn two `.dist` score files into a plot of false positive vs. false negative as a function of threshold score. All three programs require `gnuplot`. See Section 10.11 on page 137.
- Weight file reading is more robust. We plan to implement a WWW weighting server which, given a multiple alignment, will return sequence weights under a variety of weighting schemes.
- The `a2mallcaps` variable has been removed: `modelfromalign`, `buildmodel`, and other alignment reading routines will first check to see if the file is an HSSP file, if not, the a2m format will be checked, and if that does not result in every sequence having the same number of columns, all characters will be treated as uppercase. See Section 8.3 on page 65.
- Binary model output. Models can be printed in human unreadable binary form. This reduces file size to about one quarter, and greatly increases model reading speed. See Section 8.4.5 on page 71.

11.5.8 Version 1.3

May, 1996.

- Weighted training. See Section 9.4 on page 80.
- Sequence weight annealing. See Section 9.4.2 on page 82.

- The ability to use files as model type specifiers rather than keywords such as `REGULARIZER`. See Section 5 on page 45.
- The ability to print scores of only those sequences doing better than some threshold. See Section 10.2 on page 100.
- When multiple models are trained, training is stopped for each model individually according to the `stopcriterion`. Previously, training was stopped when the average score difference reached the `stopcriterion`. See Section 9 on page 74.
- `Modelfromalign` can be told to treat all letters as match columns, and turns the letter ‘O’ (capital ‘o’) into a FIM. See Section 10.7 on page 127.
- Efficiency improvements to model reading.
- SAM alignments have undergone significant changes. `Align2model` output is now in a normal sequence format, though still with uppercase, lowercase, ‘.’ and ‘-’ meanings. `Prettyalign` can read any `readseq` format, with lower-case letters indicating insertions. `Prettyalign` can no longer be used as a pipe. `Modelfromalign` can read any `readseq` format. The `buildmodel` program can be given an initial alignment. See Section 10.7 on page 127.
- The method for specifying multiple database files or multiple sequence IDs has changed. Multiple `db` or `id` declarations on the command file or a parameter file will add to a list of database files or id files.
- The command lines for many programs has changed. Except for `prettyalign`, all programs now take arguments in the form of a run name followed by variable name and value pairs. See Section 6 on page 47.
- The `modelfromalign` program now uses prior libraries. See Section 10.7 on page 127.
- FIM normalization has been moved to another place in the code, and can be avoided if desired. See Section 9.6 on page 86.

11.5.9 Version 1.2

March, 1996.

- The ability to globally apply various FIM and insertion table settings to the regularizer during training and to the model during scoring. This reflects a general cleaning up of the log-odds scoring introduced in 1.1. The defaults are to use training set letter counts in both FIMs and insert states for training, and match state frequency averages for FIMs during scoring (with no change to the insert states). See Section 8.6 on page 73. See Section 10.2.1 on page 102.
- The ability to score sequences according to the difference between two models, such as models trained on positive and negative family examples. See Section 10.2.1 on page 102.
- By default, `hmmscore` will add FIMs to a model before scoring it. See Section 10.2.1 on page 102.
- By default, SAM will start with three models of random length, and then pick the best model for surgery and further re-estimation. This will increase runtime from Version 1.1, but improves model generation.

- Several new parameters exist. See Section 12 on page 154.
- An interface is provided between HMMer and SAM. See Section 10.10.7 on page 135.
- Non-default initial models are no longer printed in model files. This led to too much confusion about which model was the real one, as well as those pesky “non-default model being replaced” messages.
- The use of Dirichlet mixture priors has been updated to reflect in our most recent work (<http://www.cse.ucsc.edu/research/compbio/dirichlets/index.html>). The format of prior libraries has changed slightly, and only one prior library (`uprior9.plib`) is included in the distribution. **Mixtures in the earlier format may crash the program.** Mixture priors are particularly useful in database search from a small set of training examples. See Section 8.1 on page 59.

11.5.10 Version 1.1

November, 1995.

- The default protein regularizer has been changed from having the uniform distribution in the insert states to having the background distribution. This generally helps discrimination experiments, though may hurt sequence alignment. See Section 8.1 on page 59.
- The default scoring has been changed from calculating Z-scores using length bins to NULL model subtraction, which accounts for both sequence length and wildcards. For scoring, FIMs must be added to a model before it is scored for this to produce valid results. This corresponds to the log-odds scoring used in Sean Eddy’s HMMER. See Section 10.2.1 on page 102.
- Scoring and training with wildcards has been modified so that sequences with many wildcards can be properly scored with null models.
- An iterative program for finding multiple motifs in a single sequences is part of SAM. See Section 10.2.5 on page 111.

11.5.11 Version 1.0

January, 1995.

- First general release.

12 Parameter descriptions

This section alphabetically explains all the parameters that can be specified in an init file. Where appropriate, the type of the parameter and any default value is listed. The default values are automatically used by the program if the user does not specify any alternative setting. The `dump.parameters` option can be used to verify the default values. See Section 6 on page 47.

The `drawmodel` and `prettyalign` programs do not use parameter files.

The SAM-T2K parameters are discussed elsewhere. See Section 4 on page 24.

The parameter reading routines will accept variations in capitalization and the presence or absence of underscores.

SAM supports reading compressed input files. If any of the file name arguments to the options end in a `.gz` or `.Z` extension. SAM will read the files using the appropriate decompression program. If an input file does not exist and does not have a `.gz` or `.Z` extension is not found, SAM will try to read from a compressed file with one of these extensions.

`a2mdots <0 or 1> [1]`: By default (1), `align2model` will place dots in the sequence alignment to fill space need for other sequences' insertions. If set to 0, these dots are not printed. See Section 10.1 on page 90.

`adjust_score <0, 1 or 2> [2]`: If set, and local or semi-local scoring is performed, simple null model scores are adjusted according to the log of the model and sequence length (1) or log of the sequence length (2), so that final scores are somewhat independent of sequence length and/or model length. This parameter is used by `hmmscore`. See Section 10.2.4 on page 107.

`adpstyle <1, 4, or 5> [1]`: The dynamic programming style for alignment and multiple domain alignment used by `hmmscore` and `align2model`. Setting to 1 indicates Viterbi alignment, 4 indicates posterior-decoded alignment on transitions and character emissions, 5 indicates posterior-decoded alignment on only character emissions. If `adpstyle` is set for other than 1, 4, or 5, it is changed to 1 (Viterbi). See also the description of `dpstyle`. See Section 9.5 on page 85, Section 10.1.2 on page 92, and Section 10.2.5 on page 111.

`aligncheckonly <0 or 1> [1]`: If set and an `alignfile` is provided, `uniqueseq` will only thin based on `percentid` calculated according to the alignment, rather than first eliminating duplicate sequences and then thinning based on `percentid`. See Section 10.12.9 on page 146.

`alignfile <string>`: A file containing an alignment of sequences for use with `modelfromalign`, `uniqueseq`, and `sortseq`, or as an initial model for `buildmodel`. See Section 10.7 on page 127. See Section 10.2 on page 100.

`align_fim <0 or 1> [0]`: Add FIMs to the ends of a model generated by `modelfromalign` or an `alignfile` in `buildmodel`. See Section 10.7 on page 127.

`alignment_weights <string>`: A file containing sequence weights for alignments used to form initial models with `buildmodel` or models with `modelfromalign`. The external weights have precedence over the internal ones, but a warning message will be generated unless `aweight_method` is set to zero. See Section 9.4 on page 80.

`alignshort <integer> [-1]`: When less than 0 (default), multiple domain search produces an alignment file that copies the entire sequence for each copy of the domain occurring within the sequence. When 0, only the region matching the model is printed. When greater than zero, that many characters to the left and the right of the domain are also printed to the file. In both cases, sequence IDs in the new file can be used to locate where the `hmmscore` found copies of the model. See Section 10.2.5 on page 111.

`alphabet <string> [protein]`: This system supports a variety of alphabets, the most common being DNA, RNA, and protein (use the `listalphabets` program to see the remaining predefined

alphabets). The protein alphabet is the default, and does not need to be specified. The abbreviation `a` may be used in place of `alphabet`. If unset, the first `train`, `test`, or `db` file is checked to see if the `alphabet` can be determined from the data. A comma-separated list of alphabets is required for using multi-track HMMs. The `alphabet_def` command may be used to define an alphabet. See Section 7.1 on page 49, Section 10.2.6 on page 115, and Section 10.12.1 on page 141.

`alphabet_def` <string>: The `alphabet_def` variable can be used to define an alphabet of 2 to 25 letters plus a (require) all-matching wildcard character. In the quoted string argument, both an alphabet name and the list of characters, with the wildcard last, must be specified. See Section 7.1.1 on page 52.

`alphbackfile` <string>: The `alphbackfile` can be used to specify a background probability file for an alphabet. Multiple files can be specified, each preceded by the `alphbackfile` designator. See Section 7.1.3 on page 55.

`anneal_length` <float> [0.8]: Indicates the speed with which noise should be decreased to zero. If greater than 1, decrease linearly over `anneal_length` re-estimates. If less than one, decrease exponentially. See Section 9.1 on page 74.

`anneal_noise` <float> [5]: Amount of noise to add to the model (decreased linearly or exponentially according to `anneal_length`). See Section 9.1 on page 74.

`auto_fim` <0 or 1> [1]: Cause `hmmscore` and `align2model` to automatically add FIMs to the model (and to the user's null model, if used) before scoring when null model subtraction is used or fully local scoring (`SW` is 2) is used. Also, in alignments, the FIM-induced delete state is automatically removed whenever `auto_fim` is set, regardless of whether or not FIMs were originally present in the model. See Section 10.2 on page 100.

`aweight_bits` <float> [0.5]: Target bits per column to save in determining alignment sequence weighting. See Section 9.4.3 on page 83.

`aweight_exponent` <float> [0.5]: Exponent in determining alignment sequence weighting. See Section 9.4.3 on page 83.

`aweight_method` <0, 1, 2, or 3> [1]: Internal weighting method for initial alignment provided to `buildmodel` or `modelfromalign`. 0 (none), 1 (karplus), 2 (henikoff), or 3 (flat). If non-zero and an `alignment_weights` file is specified, the external weights are used and a warning message is printed to standard error. See Section 9.4.3 on page 83.

`binary_output` <0 or 1> [0]: Tells model-generating programs to write models in text format if set to 0 or a binary format if set to 1. Default is text or 0. See Section 8.4.5 on page 71.

`calibrate` <integer> [0]: Perform model calibration in `hmmscore` either with random sequences or, when `db` is used, to a database of sequences. With a database, all sequences in the database are used for calibration with any non-zero setting of `calibrate`. With random sequences, if `calibrate` is set to 1, an internal default number of sequences is used; otherwise `calibrate` sequences are used. If `calibrate` is less than zero, that random or database sequences are scored, but calibration is not performed. With a database, the E-values reported are the newly-calibrated E-values. See Section 10.2.10 on page 119.

`constraints` <string>: Specify a constraints definition file to be read. This option may be specified multiple times. See Section 9.7 on page 87.

constraints_out <string>: Specify the name for a constraints definition file to create. See Section 9.7 on page 87.

constraints_from_align <0 or 1> [0]: If a true value is specified, constraints will be created for all aligned positions when a model is created from an alignment. See Section 9.7 on page 87.

cutinsert <float> [0.5]: If this fraction of sequences use an insert state, surgery will be replaced with one or more match states. See Section 9.2 on page 76.

cutmatch <float> [0.5]: When fewer than this fraction of sequences use a match state, surgery will delete the state. See Section 9.2 on page 76.

db <string>: A file containing a database of sequences, to be scored against a model in **hmm_score** or aligned to a model in **align2model**, or for other purposes. Multiple instances of the **db** variable add to the list of database files, rather than replacing the previous **db** file name. A comma-separated list of parallel databases, one per track, is required for multi-track HMMs. The **id** and **not_id** options can be used multiple times to restrict processing to specific sequences. Probabilistic sequences are created if HMMs are provided as the **db** files, possibly associated with **dbguide** guide sequences. See Section 10.2 on page 100, Section 10.2.6 on page 115, and Section 7.3 on page 58.

dbguide <string>: A file containing sequences of the same length as probabilistic sequences specified with **db**. This sequence will be used when generating alignments rather than the sequence of most probable letters of the probabilistic sequence. See Section 7.3 on page 58.

dbsize <integer> [0]: When greater than 0, this value is used in the calculation of E-values rather than the number of sequences that are read in to **hmm_score**. This is useful for correctly calculating E-values when multiple scoring runs are performed, and to avoid having to perform a complete reading of the database twice, once to calculate the number of sequences, and second time to score the sequences. See Section 10.2 on page 100.

del_jump_conf <float> [1.0]: Confidence in the regularizer for transitions leaving a delete state. The regularizer's transition values are multiplied by this number. See Section 8.1 on page 59.

distfile <string>: File with already-calculated sequence distances for use with the **makehist**, **makeroc**, **makeroc2**, and **sortesq** programs. See Section 10.2 on page 100, Section 10.11 on page 137, and Section 10.12.8 on page 145..

distfile2 <string>: A second file with already-calculated sequence distances for use with the **makehist**, **makeroc** or **makeroc2**. See Section 10.11 on page 137.

dpstyle <0, 1, 2, 3, 4, or 5> [0]: Flavor of internal dynamic programming for scoring and training. 0 indicates forward-backward (see **adpstyle** for alignment). (EM) sum-of-all-paths, 1 indicates Viterbi single best path, 2 indicates EM with the posteriors saved in a **.pdoc** file, 3 (for use with **grabdp**, but presently identical to 0) indicates EM outputting the frequency counts of each sequence in its own **.freq** file, 4 indicates most probable alignment (posterior-decoded alignment on transitions and emissions), and 5 indicates most probable alignment (posterior-decoded alignment on emissions only). See Section 9.5 on page 85, Section 10.2 on page 100, Section 10.1 on page 90, and Section 10.5 on page 124.

dump_match_probs <0 or 1> [0]: When the **grabdp** program is run with this flag set, it generates an RDB file listing the amino acid posterior probabilities for each node, and the amino acid priors. Priors are listed with the node label "FREQAVE".

dump_parameters <0, 1, or 2> [0]: Normally, only modified parameters are printed to the output file. If this is set to 1, all parameters are printed. If 2, and specified alone on the command line, **buildmodel** and **align2model** will dump parameters and exit. Because in this case an alphabet is not specified and a regularizer not created, a setting of 2 will *not* reveal the default regularizer. See Section 8.4 on page 65.

E_{max} <float> [0.001]: When a selection variable includes 4 in its binary representation, **E_{max}** is used to determine what sequences are selected. Also, when **select_score/seq**=4, sequences with an E-value better than **E_{max}** are selected. See Section 10.2 on page 100.

family_base_file <string>: If non-null, and **sequence_weights** and **family_specific** are specified, initial models are read in from the files whose names are created by appending **.i.mod**, where **i** is an integer corresponding to the family number. For example, if there are three families and the base name is **test**, the family models will be read in from **test.0.mod**, **test.1.mod**, and **test.2.mod**. The first model in the file (of any type, including MODEL, REGULARIZER, NULLMODEL, and FREQUENCIES) is used. An error will result if the models are of different lengths. See Section 9.4 on page 80.

FIM_method_train <0, 1, 2, 3, 5, 6> [-1]: During the model building process, one may employ an initial model that contains FIMs. The table probabilities can readily be changed to reflect different distributions. Negative values only cause changes to the tables when models are created by the program, rather than being read in. The default setting of -1 uses the letter frequencies in the training set when generating new models. See Section 8.6 on page 73.

FIM_method_score <0, 1, 2, 3, 5, 6> [-6]: Similar to **FIM_method_train**, except that the insert probabilities in the FIMs are changed before sequences are scored against the model. Negative values only cause changes when FIMs are added to the model. When set to 0, it is treated as -6 for adding FIMs. The default method of -6 uses the geometric average of match state probabilities. See Section 10.2.1 on page 102.

fimstrength <float> [1.0]: A factor by which to multiply the FIM letter emission probabilities. If set to 2.0, for example, each letter will have twice the probability of being generated as in the normalized insert state. This can be used to encourage the use of FIMs. The value is also applied to simple null models. When set to a value less than 0, the absolute value of **fimstrength** is applied to all insert states, FIM or otherwise. See Section 8.5 on page 71.

fimtrans <float> [1.0]: When 0.0, the FIM's insert to insert probability is 1.0. When greater than 0.0, a factor by which to multiply the model's geometric average match to match probability to produce the FIM's insert-to-insert probability. When less than 0.0, the FIM is adjusted as according to the absolute value of **fimtrans**, and the non-FIM insert-to-insert probability is set to $p - (1 - f)p^2$, where p is the regularized and normalized frequency counts for the transition and f is the FIM insert-to-insert transition. See Section 8.5 on page 71.

firstsequence <integer>: Name of a sequence database, the first sequence of which will be aligned to the model as a guide sequence in **fragfinder**. See Section 10.4 on page 123.

fisher_feature <string> [match_prior simple]: Specifies the type of *Fisher score vector* features the **get_fisher_scores** will be generated. The following value are valid. This parameter maybe specified multiple times.

- **trans** - Compute gradients from the model transition parameters.
- **insert** - Compute gradients from the model insert parameters.
- **match** - Compute gradients from the model match parameters

- `match_prior` - Compute gradients from a mixture decomposition of the model match parameters with the prior library use to train the model.
- `simple` - Include the simple NLL-NULL score as a feature.

If this parameter is not explicitly specified, `match_prior` and `simple` are used. See Section 10.6 on page 126.

`fracinsert` <float> [1.0]: When an insert state is being replaced, surgery will replace it with the average number of characters generated by the insert state multiplied by this number. See Section 9.2 on page 76.

FREQUENCIES: A model structure that has frequency counts rather than probabilities. Output by `buildmodel` if the `print_frequencies` parameter is set to 1. The `drawmodel` program is the only program that can use frequencies as input. See Section 8.4 on page 65..

`fraglen` <integer> [10]: Length of fragments for `fragfinder` to produce for each model match state. See Section 10.4 on page 123.

`genprot_prior` <string> [rsdb-comp2.32comp.gen]: Dirichlet mixture for randomly generated protein sequences output by `genseq` and used in HMM model calibration. See Section 10.12.3 on page 143 and Section 10.2.10 on page 119.

`genehl2_prior` <string> [t99-2d-comp.9comp.gen]: Dirichlet mixture for randomly generated secondary structure alphabet EHL2 sequences output by `genseq` and used in HMM model calibration. See Section 10.12.3 on page 143, Section 10.2.10 on page 119, and Section 7.1 on page 49.

`genebghtl_prior` <string> [t99-ebghtl-comp.9comp.gen]: Dirichlet mixture for randomly generated secondary structure alphabet EHL2 sequences output by `genseq` and used in HMM model calibration. See Section 10.12.3 on page 143, Section 10.2.10 on page 119, and Section 7.1 on page 49.

`gs_mean_log_len` <float> [5.4151]: Logarithm of mean sequence length used for randomly generated protein sequences output by `genseq` and used in HMM model calibration. See Section 10.12.3 on page 143 and Section 10.2.10 on page 119.

`gs_sd_log_len` <float> [1.03632564]: Logarithm of standard deviation of sequence length used for randomly generated protein sequences output by `genseq` and used in HMM model calibration. See Section 10.12.3 on page 143 and Section 10.2.10 on page 119.

`histbins` <integer> [10]: Number of bins used by the `makehist` program. See Section 10.11.1 on page 137.

`id` <string>: A sequence identifier, used to restrict `align2model` or `hmmscore` to only considering specific sequences. Multiple occurrences of the `id` parameter are added to the list of sequence identifiers, rather than replacing the value of `id`. The `not_id` command is used to exclude sequences, and has precedence over `id`.

`initial_noise` <float> [-1.0]: When greater than zero, amount of noise to add for the first iteration. See Section 9.1 on page 74.

`ins_jump_conf` <float> [1.0]: Confidence in the regularizer for transitions leaving an insert state. The regularizer's transition values are multiplied by this number. See Section 8.1 on page 59.

insconf <float> [10000]: Confidence in the regularizer for character probabilities in an insert state. The high default means that the regularizer will overpower the actual counts determined by aligning sequences to the model. The regularizer's character insert values are multiplied by this number. See Section 8.1 on page 59.

insert <string>: Insert another parameter file. The single character **i** may be used in place of **insert**. See Section 6 on page 47.

insert_file_dna <string>: Insert another parameter file if the current alphabet has been set to DNA. This is particularly useful for alphabet-specific regularizers. See Section 6 on page 47.

insert_file_protein <string>: Insert another parameter file if the current alphabet has been set to protein. This is particularly useful for alphabet-specific regularizers. See Section 6 on page 47.

insert_file_rna <string>: Insert another parameter file if the current alphabet has been set to RNA. This is particularly useful for alphabet-specific regularizers. See Section 6 on page 47.

Insert_method_train <0, 1, 2, 3, 5> [-1]: Similar to **FIM_method_train** except that the insert probabilities are changed in the nodes that are not FIMs. Negative values only cause changes to the tables when models are created by the program, rather than being read in. The default method -1 uses the letter frequencies in the training set when generating models. If the model or regularizer includes a GENERIC node, then its match and insert tables are also filled in with these values. See Section 8.6 on page 73.

Insert_method_score <0, 1, 2, 3, 5, 6> [0]: Similar to **FIM_method_score** except that the insert probabilities are changed in the nodes that are not FIMs. Negative values only cause changes to the tables when models are created by the program, rather than being read in. The default method 0 is to not change the insert tables during scoring. See Section 10.2.1 on page 102.

internal_weight <0, 1, 2> [1]: Use internal maximum discrimination sequence weighting. Automatically turned off if not explicitly set and external weights are used. See Section 9.4.4 on page 83.

jump_in_prob <float> [1.0]: The probability cost of jumping into the center of the model when the SW option is set. See Section 10.2.4 on page 107.

jump_out_prob <float> [1.0]: The probability cost of jumping out of the center of the model when the SW option is set. See Section 10.2.4 on page 107.

keepannotations <0, 1> [1]: Keep sequence annotations for sequence output. Short versions of the annotations are included in distance files as well. Annotations are frequently longer than the sequences themselves, so setting this to 0 can save considerable amounts of memory.

kestrel_fallback <0 or 1> [1]: Enables or disables fallback into sequential mode if a Kestrel is board is not available after the specified number of retries or if features not implemented on Kestrel are requested. See Section 10.2 on page 100.

kestrel_min_model_len <integer> [0]: Specifies the minimum model length to use with Kestrel implementation of **hmmscore** EM scoring. Models smaller than this value will be scored using the sequential algorithm. This is useful as small models may be slower on Kestrel. See Section 10.2 on page 100.

kestrel_remote_db_dir <integer>: Specifies the remote directory containing the sequence databases in Kestrel format. This should be in Windows-NT syntax, for example `\\merlin\data`. See Section 10.2 on page 100.

kestrel_retry_cnt <integer> [0]: Specifies the number of times to retries if a Kestrel board is not available. See Section 10.2 on page 100.

kestrel_retry_time <integer> [0]: Specifies the number of seconds to wait between retries when Kestrel board is not available.

kestrel_use_simulator <integer> [0]: Use the Kestrel simulator. This is useful when debugging the SAM Kestrel code. The number of simulated PEs will be set to the minimum required to hold the model. See Section 10.2 on page 100.

kestrel_num_extra_pe <integer> [0]: When using the Kestrel simulator for debugging SAM, increase the number of simulated PEs by the specified number. See Section 10.2 on page 100.

kestrel_dual_mapping <integer> [-1]: Used for debugging Kestrel dual mapping code. When set to -1, dual mapping is selected if the model will fit in Kestrel PE array. If 0, then the dual mapping algorithm is never used. If 1, then the dual mapping is always used with an error generated if the model will not fit. See Section 10.2 on page 100.

lambda <float> [1.0]: Scaling factor for HMM e-value calculations. Best set using the **calibrate** option of **hmmscore**. For **pathprob** output scaling use **ppscale**, and for Smith & Waterman E-value scoring, use **swlambda**. See Section 10.2.10 on page 119.

logo_auto_size <0/1> [0]: Set to 1 to resize the logo to a single 8.5 by 11 page. See Section 10.10.4 on page 131.

logoBars_per_line <integer> [0]: Number of bars per line. Default setting of 0 allows variance for cosmetic reasons. See Section 10.10.4 on page 131.

logo_bw <0/1> [0]: If set to 1, make a black and white logo, ignoring color information. See Section 10.10.4 on page 131.

logo_captions <filename>: A file of captions. Each caption has two integers (start and end bars) on one line, and the caption on the following line. Captions can overlap, in which case they will be stacked on the final logo. See Section 10.10.4 on page 131.

logo_captionf <filename>: Captions as above, but taken from a sequence file. Each of the first **logo_captionf_manyseq** sequences has its own line on the logo. After these lines, captions from any **logo_captions** file will be displayed. Each run of identical characters will be collapsed into a single caption. Intended for secondary structure (see **logo_captionf_ignoreCL** and **logo_captionf_numseq**, below). See Section 10.10.4 on page 131.

logo_captionf_ignoreCL <integer> [1]: If non-zero, ignore ‘C’ and ‘L’ characters in the sequence-based caption file (see **logo_captionf**. See Section 10.10.4 on page 131.

logo_captionf_numseq <integer> [1]: Number of sequences from the **logo_captionf** file to process. If 1, the default, only the first sequence is used to label the logo. See Section 10.10.4 on page 131.

logo_captionf_color <filename>: Colors for the caption FASTA format file **logo_captionf**. Normally installed files include **protein.colors**, **nucleotide.colors**, and **stride.colors**. If not specified, black will be used. See Section 10.10.4 on page 131.

logo_color_file <filename>: Color file for the characters. Internal defaults exist for protein, nucleotide, and secondary structure. Each line has a character and 3 RGB numbers between 0.0 and 1.0, inclusive, but, not more than 9 characters. The pound sign (#) is a comment indicator. See Section 10.10.4 on page 131.

logo_font <fontname> [Courier]: Postscript font for logo letters. See Section 10.10.4 on page 131.

logo_savings <filename>: Use a bit savings file for logo creation rather than a SAM modelfile. Savings files have one line per bar of: (bits label percentage label percentage ... newline) where 'bits' is the total bar height, labels are single characters, and percentages are of the bar height. See Section 10.10.4 on page 131.

logo_scale <float> [20.0]: Vertical scale in points per bit of the logo. See Section 10.10.4 on page 131.

logo_sections <string>: A string (e.g. "3-8,12,15") indicating which bars to display. See Section 10.10.4 on page 131.

logo_sig_height <float> [1.0]: Height in points of smallest character other than X to display. See Section 10.10.4 on page 131.

logo_start_num <integer> [1]: The sequence index of the first bar in the logo. HMM files (**modelfile**), **logo_captionf**, and **logo_under_file** are all automatically adjusted to start with this index. Bar indices in **logo_sections**, **logo_caption**, and **logo_savings** are not adjusted by this amount. See Section 10.10.4 on page 131.

logo_title <string> [runname]: Title of the logo. Default is the current runname. See Section 10.10.4 on page 131.

logo_title_font <fontname> [Times-Roman]: Postscript font for the logo title. See Section 10.10.4 on page 131.

logo_under_file <filename>: A file with one FASTA-format sequence, to provide labels, such as amino acid names in a sequence, one character per logo bar. Useful for visually comparing a sequence and a model. See Section 10.10.4 on page 131.

logo_under_color <filename>: A color file for use with the underfile sequence. Normally installed files include **protein.colors**, **nucleotide.colors**, and **stride.colors**. If not specified, black will be used. PRIOR_PATH is searched for the file if not in the current directory. See Section 10.10.4 on page 131.

max_seq_length <integer> [2500]: Used to specify maximum sequence length for Kestrel database processing and the maximum sequence length for **splitseq** database splitting. See Section 10.12.6 on page 144.

mainline_cutoff <float> [0.5]: Changing this value will set both **cutmatch** and **cutinsert** to the new value. See Section 9.2 on page 76.

many_files <0-15> [0]: When zero, all the output of **buildmodel** is sent to the **.mod** file. If the binary expansion of **many_files** includes a '1' (e.g., is odd), **buildmodel** will create multiple files for the probability model, frequency model, and the run statistics. If the binary expansion of **many_files** includes a '2' (e.g., 2, 3, 6, 7, 10, 11, 14, 15), the **hmmscore** score information (**.dist**) is sent to standard output. If the binary expansion of **many_files** includes a '4' (e.g., 4, 5, 6, 7, 12, 13, 14, 15), the **hmmscore** multiple domain score information (**.mstat**) is sent to standard output. If the binary expansion of **many_files** includes an '8' (e.g., 8, 9, 10, 11, 12, 13, 14, 15), **hmmscore** will name model library distance files using only the model name listed in the model library, rather than a combination of the runname, model name, and position within the model library. See Section 5 on page 45, Section 10.2.3 on page 106, Section 10.2.5 on page 111, and Section 10.2.10 on page 119.

match_jump_conf <float> [1.0]: Confidence in the regularizer for transitions leaving a match state. The regularizer's transition values are multiplied by this number. See Section 8.1 on page 59.

matchconf <float> [1.0]: Confidence in the regularizer for character probabilities in a match state. The regularizer's character match values are multiplied by this number. This variable is ignored if a prior library is used. See Section 8.1 on page 59.

maxinserts <integer> [100]: In `buildmodel`, the maximum number of states inserted after any node by the surgery. See Section 9.2 on page 76.

maxmem <integer> [0]: Maximum size of dynamic programming array to use for training and alignment. See Grice, Hughey, and Speck, and Tarnas and Hughey CABIOS papers for more information on the algorithm used. Depending on system configuration, performance may increase with higher values. If set to zero (the default), SAM will always use the smallest possible amount of space.

maxmodlen <integer> [0]: When starting with multiple, randomly generated models, the longest model to use. If set to 0 (the default), the value is calculated as 10% above the average sequence length when needed. See Section 8.4.1 on page 68.

maxposdecodemem <integer> [100000000]: Maximum amount of memory, in bytes, to allocate for posterior-decoded alignments. These alignments require space proportional to the product of the sequence length and the HMM length. Sequences that are too long will be aligned using the Viterbi algorithm. See Section 10.1 on page 90.

mdNLLnull <float> [-10.0]: Criterion by which subsequences are judged to be matches to a single motif (model) during a multiple domain alignment if there is a 1 in the bit pattern of `select_md`. All occurrences for which `NLL-NULL` is better than the specified value are considered matches. See Section 10.2.5 on page 111.

mdNLLcomplex <float> [-10.0]: Criterion by which subsequences are judged to be matches to a single motif (model) during a multiple domain alignment if there is a 2 in the bit pattern of `select_md`. All occurrences for which `NLL-NULL` user's null or reverse null model score is better than the specified value are considered matches. See Section 10.2.5 on page 111.

mdEmax <float> [0.01]: Criterion by which subsequences are judged to be matches to a single motif (model) during a multiple domain alignment if there is a 4 in the bit pattern of `select_md`. All occurrences for which reverse sequence null model e-value is better than the specified value are considered matches. See Section 10.2.5 on page 111.

minmodlen <integer> [0]: When starting with multiple, randomly generated models, the shortest model to use. If set to 0 (the default), the value is calculated as 10% below the average sequence length when needed. See Section 8.4.1 on page 68. See Section 8.4.1 on page 68.

MODEL: Specify an initial model. See Section 8.4 on page 65..

model_abort_length <integer> [10000]: In `buildmodel`, if the initial model length is greater than this number, an error message is printed and the program is aborted. This is to avoid giant models that will never complete training because of their memory or execution time requirements.

model_file <string>: If non-null, this file is read for an initial model. The first model in the file (of any type, including `MODEL`, `REGULARIZER`, `NULLMODEL`, and `FREQUENCIES`) is used. This will override any models present in inserted files. See Section 5 on page 45.

modellength <integer> [-1]: When greater than 0, sets the model length to a specific value in **buildmodel**. (overridden if a model or regularizer without a **GENERIC** node is present). If equal to 0 and **maxmodlen** is less than 1, all model lengths are set to the average length of the training sequences. If less than 0, model length(s) are set to a random value between **minmodlen** and **maxmodlen** according to **randseed**. These two bounds will default to 90% and 110% of average sequence length if **maxmodlen** is less than 1. See Section 8.4.1 on page 68.

model_library <string>: Specify a library of models for scoring or calibration. See Section 10.2.10 on page 119.

modlib_absolute <0/1> [1]: Use absolute path names when creating a model library file. See Section 10.2.10 on page 119.

Motifcutoff <float> [0.5]: In multiple motif search, fragments which are smaller than this fraction of the model length are not considered for further processing. Further, processing stops if a fragment of length less than the square of **Motifcutoff** is the best match (this is needed when using SW scoring with weak thresholds). See Section 10.2.5 on page 111.

NLLnull <float> [-10.0]: If a selection variable is odd, this value is checked against a sequence's simple null model score. See Section 10.2 on page 100.

NLLcomplex <float> [-10.0]: If a selection variable includes 2 in its binary representation, this value is checked against a sequence's user or reverse sequence null model score. See Section 10.2 on page 100.

NLLfile <string>: Alias for **distfile**.

NLLfile2 <string>: Alias for **distfile2**

Nmodels <integer> [3]: Multiple initial models can be trained simultaneously, with the best one being used for surgery and further training. See Section 8.4.1 on page 68.

not_id <string>: A sequence identifier, used to eliminate a specific sequence from consideration by **align2model** or **hmmscore**. Multiple occurrences of the **not_id** parameter are added to the list of sequence identifiers, rather than replacing the value of **not_id**. The **not_id** command is used to exclude sequences, and has precedence over **id**.

NscoreSeq <integer> [100000]: Maximum number of sequences to be read by the **hmmscore** or **align2model** program.

Nseq <integer> [10000]: Maximum number of sequences to be read from any of the up to four sequence files or a database files in **buildmodel**. See Section 7.4 on page 58.

nsurgery <integer> [3]: Maximum number of surgeries to perform. Each surgery will result in a full EM cycle until **stopcriterion** or **reestimates** is reached.

Ntrain <integer> [0]: Number of sequences to train on. If zero, all sequences that were read from the files **train** and **train2** (up to a limit of **Nseq** per file) form the training set. If **Ntrain** is greater than the number of sequences read in from the files **train**, **train2**, **test**, and **test2**, all sequences are used for training. If **Ntrain** is less than the total number of sequences read in from the four files, all the sequences are randomly partitioned (using **trainseed**) into the training set with **Ntrain** sequences, and of the remaining sequences (i.e., whether or not a sequence occurred in a training file or a test file is ignored). See Section 7.4 on page 58.

nucleotide_prior <string>: The prior library to use if the RNA or DNA sequences are being modeled and **prior_library** has not been set. See Section 8.1 on page 59.

NULLMODEL: Identifies a user defined null model in a model file. The parameter **subtract_null** must be set to 3 to use this null model. See Section 10.2 on page 100.

nullmodel_file <string>: If non-null, this file is read for a user's null model. The first model in the file (of any type, including MODEL, REGULARIZER, NULLMODEL, and FREQUENCIES) is used. This will override any null models present in inserted files. To use this null model, **subtract_null** must be set to 3. See Section 5 on page 45.

null_score_weight_scale <float> [10.0]: Used with **get_fisher_scores** to weigh a sequences Fisher scores by how closely they match the model. If non-zero weigh the gradients according to the sequence NLL-NULL score, scaled by this parameter. Each gradient is multiplied by **sigmoid(-score/null_score_weight_scale)**, where **sigmoid** is the logistic function. See Section 10.6 on page 126.

numpermacth <integer> [5]: Number of fragments for **fragfinder** to produce for each model match state. See Section 10.4 on page 123.

percent_id <float> [1.0]: For alignments passed to **uniqueseq**, specifies fraction identity to use for deleting sequences. If **percent_id** is negative, a message is printed about each sequence that is dropped. See Section 10.12.9 on page 146.

plotcolumn <integer> [3]: Column of score file to use in calculating plots. Length (0), simple null model (1), complex or reverse null model (2), or Evaluate (3). See Section 10.11 on page 137.

plotleft <float> [0.0]: Lowest X axis value on a graph generated by **gnuplot**. The X axis is calculated internally if **plotleft=plotright**. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

plotline <float> [0.0]: Creates a vertical line at this value in a graph generated by **gnuplot** if **plotline** is nonzero. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

plotmax <float> [0]: Highest Y axis value on a graph generated by **gnuplot**. The Y axis is calculated internally if **plotmax=plotmin**. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

plotmin <float> [0]: Lowest Y axis value on a graph generated by **gnuplot**. The Y axis is calculated internally if **plotmax=plotmin**. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

plotnegate <int> [0]: Negates the scores on a graph generated by **gnuplot** if set to 1. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

plotps <int> [1]: Creates a postscript file **runname.ps** if set to 1. When set to 0, only a **.plt** file is generated. A square plot postscript file is generated for a setting of 2. For options 1 and 2, the **.data** and **.plt** files used to create the postscript file are deleted. When set to 3, the postscript file is generated and the **.data** and **.plt** files are retained. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

plotright <float> [0.0]: Highest X axis value on a graph generated by **gnuplot**. The X axis is calculated internally if **plotleft=plotright**. Used in conjunction with **makehist**, **makeroc** and **makeroc2**. See Section 10.11 on page 137.

ppscale <float> [1.0]: Scaling factor for **pathprobs** output scaling. For **hmmscore** E-value scoring use **lambda** or **swlambda**. See Section 10.8 on page 127.

pptrim <float> [0.0]: If greater than 0, posterior threshold for turning match states into insert states in the creation of a trimmed alignment (.ta2m) in the **pathprobs** program. See Section 10.8 on page 127.

print_all_models <0 or 1> [0]: When set, models are printed after each iteration of the forward-backward procedure. Models are printed to files of the form **runname.a.mrrr.mod**, where 'mrrr' is the catenation of the number of the model (or 1 if only one model is being estimated at a time) and the re-estimate number. This variable can be toggled at runtime by sending a **SIGUSR2** signal to the program, providing a means to look at intermediate results while the program is running or checkpointing a program run.

print_all_weights <0 or 1> [0]: When set, a weight output file is generated after each iteration of the forward-backward procedure. Weights are printed to files of the form **runname1.weightoutput**, where '1' is the number of the iteration.

print_frequencies <0 or 1> [0]: If this option is set, the frequency counts for each state will be printed as well as the model.

print_surg_models <0 or 1> [0]: When set, models are printed after each surgery (surgery occurs after a sequence of EM re-estimates). Models are printed to files of the form **runname.s.rr.mod**, where 'rrr' is the re-estimation index for the run. When surgery is used, a single winning model is automatically selected after the first EM re-estimation loop if multiple initial models are used. This variable can be toggled at runtime by sending a **SIGUSR1** signal to the program.

prior_library <string>: When set, use Dirichlet mixture priors to regularizer the models. Transition costs and insert states are still regularized by the default (or specified) regularizer, but match states are regularized with Dirichlet mixtures. The **matchconf** variable is ignored if a prior library is used, in favor of the **prior_weight** variable. If **prior_library** is not set and **protein_prior** or **nucleotide_prior** is set, the indicated prior library is used. If neither is set, and proteins are being modeled, and internal default prior library is used. See Section 8.1 on page 59.

prior_weight <float> [1.0]: Weight of the prior library, if it is used. See Section 8.1 on page 59.

protein_prior <string>: The prior library to use if the proteins are being modeled and **prior_library** has not been set. If not set, and proteins are being modeling, an internal default will be used. See Section 8.1 on page 59.

query <string>: Sequence file, the first sequence of which is used as a Smith and Waterman query. A Smith and Waterman model is built from this single sequence and scored against the database in **hmmscore**. See Section 10.2.8 on page 117.

randseed <integer> [-1]: Random seed for noise generation and for selection of initial model lengths if **modellength** is less than one. The default value causes the process's pid to be used, which will then be printed to the output file to enable replication of results.

rdb <0 or 1> [0]: Create the score file in RDB format with the extension **.dist-rdb** rather than the standard **.dist** format. See Section 10.2 on page 100.

randomize <integer> [5]: Determines how noise is added to the model. See Section 9.1 on page 74.

read_smooth <0 or 1> [0]: Tells **hmmscore** whether or not to read a smooth curve from **smooth_file**, or its default (**runname.smooth**). See Section 10.2 on page 100.

reestimates <integer> [40]: Maximum number of re-estimates to perform after a surgery. Generally, this should be set higher than the number of iterations that have noise. See Section 9 on page 74.

reglength <integer> [-1]: Similar to **modellength**, sets the length of the regularizer. Usually not needed. See Section 8.4.1 on page 68.

REGULARIZER: Specify an initial regularizer. See Section 8.4 on page 65.

regularizer_file <string>: If non-null, this file is read for a single-component regularizer. The first model in the file (of any type, including MODEL, REGULARIZER, NULLMODEL, and FREQUENCIES) is used. This will override any regularizers present in inserted files. See Section 5 on page 45.

rerun <integer> [-1]: The program optimizes **Nmodels** models until the first ‘surgery’, and then continues with the best one. Sometimes it is interesting to see how the second best would have done. If the second best is number 4 (starting from 0!), a setting this parameter to 4 would optimize that model. Models can also be accessed using one **print.all.models**.

retrain_noise_scale <float> [0.1]: If an initial model or alignment is passed to **buildmodel**, **initial_noise** (or **anneal_noise** if **initial_noise** is unspecified) is scaled by this multiplier, which must be between 0.0 and 1.0. See Section 9.1 on page 74.

segments <integer> [1]: Number of segments **hmmscore** should logically split database into. Segmentation is based on number of sequences. See Section 10.2.7 on page 116.

segment_number <integer> [1]: Segment number among segments. See Section 10.2.7 on page 116.

segment_size <integer> [1000]: Number of sequences read in at a time and given to one of the segments. See Section 10.2.7 on page 116.

select_align <integer> [0]: Tells **hmmscore** what selection criteria should be used for placing aligned sequences into the file **runname.a2m**. If 0, no sequences are selected; if 1, sequences are selected according to their simple null model scores and **NLLnull**; if 2, sequences are selected according to their complex, user, or reverse sequence null model score and **NLLcomplex**; if 4, sequences are selected according to their E-values and **E_{max}**; if 8, all sequences are selected. Selection criteria can be combined: 3 requires sequences to score better than **NLLnull** with the simple null model and **NLLcomplex** with the complex (user’s or reverse sequence) null model. Negative numbers indicate that sequences that do not pass the corresponding positive test should be selected. See Section 10.2 on page 100.

select_mdalign <integer> [0]: Tells **hmmscore** what selection criteria should be used for performing a multiple domain check on a scored sequence. Sequences that pass the **select_mdalign** criteria are analyzed and recorded if they pass the **select_md** criteria during the mult-domain Viterbi alignment pass. Once sequences have been selected using the **select_mdalign** parameter, the multiple domain search procedure is controlled by **select_md** and related parameters. Set to 8 to perform a multiple domain alignment for every sequence. See Section 10.2 on page 100.

select_md <integer> [1]: Tells **hmmscore** what selection criteria should be used treating a multiple domain alignment as found, in which case the alignment is written to **runname.mult** with scores in **runname.mstat**. The multiple domain search procedure is only performed on sequences that have satisfied the **select_mdalign** criterion in **hmmscore**. Functions as with **select_align** with the variables **mdNLLnull**, **mdNLLcomplex**, and **mdE_{max}**. Only sequences that pass the

selection criteria (which is always based on Viterbi scores) are recorded in the files. The default is to require passing the simple null model test. It does not make sense to set this parameter to 8. See Section 10.2.5 on page 111.

select_score <integer> [8]: Tells **hmmscore** what selection criteria should be used for listing sequence scores in the file **runname.dist**. Functions as with **select_align**. See Section 10.2 on page 100.

select_seq <integer> [0]: Tells **hmmscore** what selection criteria should be used for placing sequences in the file **runname.sel**. Functions as with **select_align**. See Section 10.2 on page 100.

sequence_models <float> [0.0]: Build initial models from randomly-selected sequences in the training set when greater than zero. Value indicates the weight the sequence should have when combined with the regularizer. See Section 8.3 on page 65.

sequence_warning <integer> [0]: Primarily for debugging. Set to -1 to print out all sequences in which a ‘wrong’ letter was found, or to -2 to print out all sequences.

sequence_weights <string>: File to read for sequence weights. See Section 9.4 on page 80.

simple_threshold <integer> [0]: User and reverse sequence scores will not be calculated by **hmmscore** unless the simple null model score is less than this number. Set to 10000 to require all scores to be calculated. See Section 10.2.1 on page 102.

sort <integer> [4]: Indicates whether or not sequence scores should be sorted by **hmmscore**. With a value of 1, sequences are sorted by column 1 (simple null model score). With a value of 2, sequences are sorted by column 2 (other null model selections; see **subtract_null**). With a value of 4, sequences are sorted by E-value if available or by column 1. When negative, scores are sorted in reverse order, worst first. When 0, scores are not sorted.

Sort also indicates whether or not **checkseq** should sort sequence IDs and sequences to check for uniqueness. This sorting requires storing all sequences in memory, so can be quite time consuming.

See Section 10.2 on page 100 and Section 10.12.2 on page 141.

stopcriterion <float> [0.1]: The re-estimation loop will stop whenever the improvement in the NLL score is less than this number (provided noise is less than 10% of its original value for that iteration), or when the maximum number of **reestimates** is reached. See Section 9 on page 74.

subtract_null <integer> [4]: In **hmmscore** and other programs, decides the type of null model to be used. In score files, this will be the second score column (the first is always the simple null model). When set to 0, raw scores are reported in the second column. Setting to 1 provides simple null model scores; to 2, issues a warning and uses the simple null model; to 3, user’s input null model; to 4, the reverse sequence null model; and to 5, the scaled reverse sequence null model.

surgery_noise_scale <float> [0.1]: After the first surgery, **anneal_noise** is scaled by this multiplier, which must be between 0.0 and 1.0. See Section 9.1 on page 74.

surgfile <string>: A sequence file in a2m alignment format to be used for guiding model surgery. On each surgery step, the sequence is aligned to the model, and model nodes are deleted or inserted to ensure to match the alignment given in **surgfile**. Optionally, **id** may be used to specify a specific sequence within **surgfile**, otherwise the first sequence will be used. See Section 9.2 on page 76.

SW <integer> [2]: When set to 0, `hmmscore`, `buildmodel`, and other programs use model to sequence (global) dynamic programming. When set to 1, SAM programs use submodel to sequence (semilocal) scoring. When set to 2, SAM programs use submodel to subsequence (local) scoring. When set to 3, SAM programs use model to subsequence (domain) scoring. Ignored in Smith and Waterman (`query`) mode. See Section 10.2.4 on page 107.

swlambda <float> [0.34657]: Scaling factor for Smith and Waterman e-value calculation and for `pathprobs` single-digit scores. Default is $\ln(b)/u$, for base 2 (bit) scoring matrices for which a unit indicates a half-bit. Do not confuse with `ppscale` or `lambda`. See Section 10.2.8 on page 117.

syncfile <string>: In `buildmodel`, perform sequence-based surgery using the first sequence in this file (or an alternate if `id` is specified). The sequence should be an a2m alignment. See Section 9.2.1 on page 77.

syncweight <float> [1.0]: The sequence weight of a synchronization sequence. During the sequence surgery procedure, emission and transition counts are moved in the model. This variable indicates how much count should be treated as corresponding to the guide alignment during the surgery procedure. See Section 9.2.1 on page 77.

test <string>: A file to read test sequences from. See Section 7.4 on page 58.

test2 <string>: A second file to read test sequences from. See Section 7.4 on page 58.

trackcoeff <string>: A comma-separated list of floating-point track coefficients used to compute character emission scores in multi-track HMMs. See Section 10.2.6 on page 115.

trackmod <string>: A comma-separated list of model files specifying a multi-track HMM. See Section 10.2.6 on page 115.

trackprior <string>: A comma-separated list of Dirichlet mixture priors to be used in model calibration of a single or multi-track HMM. Multiple tracks are generated independently. See Section 10.2.6 on page 115 and Section 10.2.10 on page 119.

trainseed <integer> [-1]: Random seed for partitioning the sequences into the test set and the training set. The default value causes the process's pid to be used, which will then be printed to the output file to enable replication of results. See Section 7.4 on page 58.

train <string>: A file to read training sequences from. See Section 7.4 on page 58.

train2 <string>: A second file to read training sequences from. See Section 7.4 on page 58.

train_reset_inserts <0,1,2,3, or 6> [6]: At the end of `buildmodel` training, all insert and FIM character tables are set according to this variable, which takes on the same meanings as `FIM_method_train`. The default setting is to set all insert and FIM tables to the normalized geometric average of the match state costs. See Section 8.6 on page 73.

trans_priors <string>: The name of the structure-specific transition prior library to use when structural information for transition probability estimation is to be used for HMM estimation. See Section 8.1.2 on page 62.

transweight <float> [1.0]: A multiplier that affects the influence of the pseudocounts generated by the structure-specific transition priors. See Section 8.1.2 on page 62.

template <string>: For use with the structure-specific transition prior library. A three-column file (amino acid sequence, secondary structure, accessibility) that is used during HMM estimation to assign a structural environment to each model node. See Section 8.1.2 on page 62.

`use_kestrel` <0 or 1> [0]: If 1, use the Kestrel implementation of the `hmmscore` scoring algorithm. See Section 10.2 on page 100.

`verbose` <0 or 1> [0]: If set to 1, more diagnostic status messages are printed in some programs, including `hmmscore` and `uniqueseq`.

`viterbi_threshold` <integer> [10000]: If changed from the default 10000, and the dynamic programming style is not Viterbi, sequences must first pass the `viterbi_threshold` for the Viterbi NLL-NULL score before being scored with the selected method. Because Viterbi scoring can be 2–5 times faster than EM-style scoring, this can lead to a considerable execution time savings. A typical use would be to set to, for example, 5 greater than the `simple_threshold` used for calculating the reverse null model score. See Section 10.2.1 on page 102.

`weight_final` <float> [1.0]: The final (steady-state) multiplier of sequence weights. The default (1.0) means that, if no sequence weight file is used, each sequence is weighted as being one sequence. If a weight file is used, all values in that file are multiplied by this value. See Section 9.4 on page 80 and Section 9.1 on page 74.

`weight_length` <float> [0]: An annealing schedule for the sequence weight multiplier. If greater than 1.0, the weight multiplier is increased from zero linearly over `weight_length` re-estimates. If less than one, increase exponentially. See Section 9.4 on page 80 and Section 9.1 on page 74.

`write_dist` <0 or 1> [0]: If a non-zero value is specified, `get_fisher_scores` will generate a `run-name.dist-rdb` score file. See Section 10.6 on page 126.