

Lobster User Guide

Experimental release
November 21st, 2002
<http://www.drive5.com/lobster/index.htm>

Robert C. Edgar
lobster@drive5.com

Citation for SATCHMO:

Robert C. Edgar and Kimmen Sjolander (2003), SATCHMO: Sequence alignment and tree construction using hidden Markov models, *Bioinformatics* **19**(11), 1404-1411.

© Copyright 2002 Robert C. Edgar
The Lobster software is a copyrighted work. Please refer to the license agreement. Permission is hereby granted to make and distribute copies of this manual provided that the text is unchanged; including in particular that this copyright and permission notice is clearly visible and readable.

1 Contents

1 CONTENTS	2
2 QUICK START	3
2.1 What is Lobster?	3
2.2 What is SATCHMO?	3
2.3 What is COACH?	3
2.4 What files are in the Lobster package and how do I install them?	3
2.5 Quick start for SATCHMO	3
2.6 Quick start for COACH	4
2.7 Where are the papers? How do I cite Lobster?	5
2.8 Is the source code available?	5
2.9 Technical support and contacting the author	5
3 SATCHMO	6
3.1 Input to SATCHMO	6
3.2 Output from SATCHMO	6
3.3 Parameters to the SATCHMO algorithm	6
3.4 Execution time and memory requirements	7
3.5 Progress messages	8
4 SATCHMOVIEW	9
4.1 SatchmoView basics	9
4.2 Alignment display	9
4.2.1 Upper case letters	9
4.2.2 Lower case letters	10
4.2.3 Dashes	10
4.2.4 Dots	10
4.2.5 Colors	10
4.3 Tree display	10
4.4 Affinity histogram and smoothed affinity graph	11
4.5 Per-residue match state scores	11
4.6 Navigation	11
4.7 Saving an alignment	11
4.8 Saving a tree	12
4.9 Multiple document interface	12
5 COACH	13
5.1 Building a profile HMM from an alignment	13
5.2 Aligning an alignment to a profile HMM	13
5.3 HMM scoring	13
5.4 Sequence weighting	13
6 OTHER FEATURES	15
6.1 Log file	15
6.2 Comparing two alignments	15
7 FILE FORMATS	16
7.1 Unaligned FASTA (.fasta or .fa)	16
7.2 Aligned FASTA (.fasta or .fa)	16
7.3 HMM parameter files (.hmm)	16
7.4 SatchmoView files (.smo)	17
8 ACKNOWLEDGEMENTS	18

2 Quick Start

2.1 What is Lobster?

Lobster¹ is software for analyzing protein sequences. Its primary goal in this first, experimental release is to implement two algorithms: SATCHMO and COACH. It includes tools designed to assist users of these algorithms and fundamental research on the algorithms themselves. These algorithms make extensive use of profile hidden Markov models (HMMs).

2.2 What is SATCHMO?

SATCHMO stands for Simultaneous Alignment and Tree Construction using Hidden Markov mOdelS. The algorithm was developed in collaboration with Kimmen Sjölander, UC Berkeley. Given a set of protein sequences, SATCHMO produces a tree and a set of multiple sequence alignments. The tree is designed to bring sequences with closely related structures together. Initial tests suggest that SATCHMO trees are a good approximation to classifications such as SCOP or CATH that are created by experts. The sequence alignments predict which positions are structurally alignable, and which are not. SATCHMO thus differs from conventional algorithms like CLUSTALW, which in effect predicts *all columns* to be alignable, and conventional profile HMM methods, which attempt to identify those positions that are alignable across *all sequences*. SATCHMO predicts more positions to be alignable in more closely related sequences.

2.3 What is COACH?

COACH stands for Comparison Of Alignments by Constructing HMMs. It is used to align two multiple sequence alignments to each other. This alignment produces a score that can be used as a relatedness measure. The basic ideal behind the method is to construct a profile HMM from one alignment and align the other to that HMM. This method has been shown to produce more accurate alignments of remote homologs than the traditional method of aligning a sequence to a profile HMM constructed from an alignment built for the other sequence. I am hopeful that it is also capable of more accurate homolog detection through scoring, though this has not been validated at the time of writing.

2.4 What files are in the Lobster package and how do I install them?

All Lobster algorithms are contained in a single binary executable file named *lobster.exe* on Windows and *lobster* on Linux. In addition, there is the *sv.exe* binary: this is the SatchmoView graphical interface for viewing the output produced by SATCHMO. It is currently available only on Windows. (I will be glad to make the source code available to anyone interested in porting to other platforms.)

Installation is very simple: just copy the appropriate executable file(s) for your platform anywhere you like. You don't need to worry about setting up configuration files, registry entries, environment variables, shared libraries, DLLs...

2.5 Quick start for SATCHMO

To run SATCHMO, you need a FASTA file containing some protein sequences. FASTA is a widely-used plain-text file format for storing one or more sequences—I'm going to assume that you know how to create a FASTA file. Here is a simple example you could try:

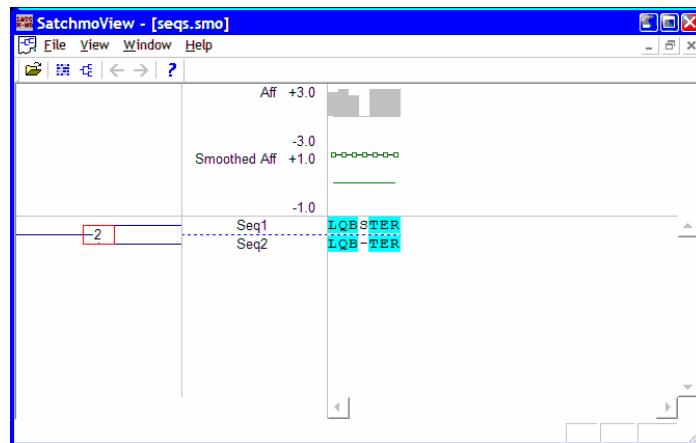
```
>Seq1
LQBSTER
>Seq2
LQBTER
```

¹ The name "Lobster" is an obscure bi-lingual pun. When I started creating Lobster, I spent a lot of time looking at Sean Eddy's HMMER package to see how he had implemented various HMM methods (thanks, Sean, for making the source code readily available). I mistakenly believed the name was pronounced "hummer" (correct is "hammer"). In many Scandinavian dialects, "hummer" means "lobster". So now you know.

Be sure that the > characters are in the first column. Save this in a file called *seqs.fasta*. To run SATCHMO, use the following command line:

```
lobster -satchmo seqs.fasta -out aln.fasta -sv seqs.smo
```

This should produce two output files: *aln.fasta*, which is a FASTA file containing an alignment of all the input sequences, and *seqs.smo*, which contains the alignment plus other information needed for SatchmoView. Start SatchmoView (*sv.exe*) on a Windows machine and use the *File, Open* menu command to open the file *seqs.smo*. You should see something like the following.



2.6 Quick start for COACH

To use COACH, you start with two multiple sequence alignments each in their own FASTA file. I'll assume those files are called *aln1.fasta* and *aln2.fasta*. The first step is to create an HMM from one of the alignments; this can be done with the following command line.

```
lobster -msa2hmm aln1.fasta -hmm aln1.hmm
```

This creates a file *aln1.hmm* containing parameters for a profile HMM constructed from the alignment. The next step is to align both alignments to that hmm, creating a combined alignment of the two original alignments. The command line is as follows.

```
lobster -align aln2.fasta -hmm aln1.hmm -tpl aln1.fasta -out combined.fasta
```

This produces a file *combined.fasta* with the combined alignment. The arguments are:

- align FASTA file containing the alignment to be aligned to the HMM.
- hmm HMM parameter file.
- tpl FASTA file containing the alignment that was used to construct the HMM (the "template" alignment).
- out Name to use for the FASTA output file containing the combined alignment.

2.7 Where are the papers? How do I cite Lobster?

At the time of writing, there are no published papers on COACH.

Citation for SATCHMO:

Robert C. Edgar and Kimmen Sjolander (2003), SATCHMO: Sequence alignment and tree construction using hidden Markov models, *Bioinformatics* 19(11), 1404-1411.

2.8 Is the source code available?

Sure. Just send me an e-mail.

2.9 Technical support and contacting the author

You are welcome to contact me to request technical support or to suggest fixes and enhancements that will help your work. I welcome reasonable comments, suggestions and questions. You can reach me at:

`lobster@drive5.com`

Notice that I always provide my e-mail address as a graphic rather than in text or as a *mailto:* hyperlink. This is to avoid webcrawlers used by spammers to collect e-mail addresses, and I will be grateful if you don't publish my address in text form on the Internet. As an alternative, use this hyperlink:

<http://www.drive5.com/lobster/index.htm>

3 SATCHMO

3.1 Input to SATCHMO

Input to SATCHMO is a single FASTA file containing unaligned sequences. If there are indel characters, i.e. dots or dashes, in the file, SATCHMO will ignore them. Letters may be specified in upper- or lower-case, SATCHMO treats them as equivalent. The standard 20-letter alphabet for amino acids is understood, as are the special characters X (unknown residue), B (D or N) and Z (E or Q). The input file is specified following `-satchmo` in the command-line.

3.2 Output from SATCHMO

SATCHMO can produce two kinds of output file: a FASTA file for the alignment at the root of the tree, and a file with other information which can be displayed by the SatchmoView program (`sv.exe`).

The root alignment file name may be specified by using the `-out` option.

The SatchmoView file name may be specified by using the `-sv` option, and by convention should be given the `.smo` extension.

One or both of `-out` and `-sv` must be given; it is an error not to specify some kind of output file.

Extra information may optionally be included in the SatchmoView file: the HMM parameters, and the match state scores (i.e., per-residue affinities). As this can lead to very large files (hundreds of megabytes), this information is excluded by default. The relevant options are `-savehmm`s (include HMM parameters), and `-savemss` (include match state scores).

3.3 Parameters to the SATCHMO algorithm

Two parameters to the SATCHMO algorithm control the method used to predict which columns of a combined alignment represent truly alignable positions. This involves using a sliding window to smooth the "affinity", defined to be the match state score of a column relative to the background probabilities of the residues in that column. The two parameters are the length of the window and the minimum score that predicts alignability. The length of the window is a number of HMM nodes (roughly, a number of alignment columns) and defaults to 7. It must be an odd integer ≥ 1 . The minimum smoothed affinity to predict alignment is specified in units of bits ($\log_2 P/Q$ for some probabilities P and Q), and defaults to 0.01. A value greater than zero can be interpreted as "more likely than the background probabilities", a value less than zero can be interpreted as "less likely than the background probabilities". Using a lower value for the minimum affinity will cause more positions to be aligned; a higher value will cause fewer positions to be aligned. Typical match state bit scores range between about 1.5 and -1.5; values outside this range are rare. A value of -2 is therefore approximately minus infinity, a value of (say) -9 can safely be assumed to cause all positions to align, regardless of the window length. The parameters are set using the `-minaff` and `-window` options, respectively. For example,

```
lobster -satchmo seqs.fasta -sv seqs.smo -window 13 -minaff -0.5
```

So, what values should you set for these parameters? The defaults often work well. If you have sequences with low pair-wise identity, then affinity values will tend to be low, and you may need to set a lower value for the minimum affinity. If you believe your sequences may be alignable but you are only seeing short motifs (or nothing at all) aligning, it might be worth trying `-minaff -9`, which will align as many positions as possible. If this appears to over-align, try an intermediate value between -1 and 0. The effect of the window length is less clear—I honestly don't have any guidelines for you. Improving the smoothing algorithm is an active area of research; we may be able to dispense with the sliding window and/or automatically detect the best smoothing parameters (I hope so).

By default, SATCHMO uses the full COACH algorithm to align an alignment to a profile HMM. This can be expensive in memory as this part of the algorithm requires the calculation of several so-called "break matrices" which are size $L \times L$ where L is the number of columns in the input alignment. To avoid re-calculating these matrices every time they are needed, SATCHMO caches them for all outer-most clusters as the tree is being built. As an example, this would need about 2.5 GB memory given 400 sequences of length 400 as input. As this exceeds the capacity of most Windows and Linux PCs currently available, Lobster provides a space-optimized approximation to the full COACH algorithm that often (but not always) gives results of comparable accuracy. This is the "no break matrices" optimization, and is requested by using the `-nobm` command-line argument. By default, break matrices are enabled, you can confirm this by using the `-bm` option (which has no effect as this is the default). The `-nobm` option saves memory and also reduces execution time, typically by 25% or so. However, I recommend that you use `-nobm` only when compelled to do so by lack of memory as this option sometimes degrades the accuracy of alignments and trees. The following section explains how to estimate the memory and time requirements for a given set of sequences.

You can further save memory and time by specifying the `-v1` option, which enables version 1 of the SATCHMO algorithm. If you do so, you should also specify `-nobm` as break matrices are not needed. This algorithm is smaller and faster (see next section), and sometimes produces alignments of comparable quality to the default, which is version 2 (which be confirmed by the `-v2` option). However, you can expect the results to be less accurate than `-nobm`, so you should use `-v1` only when compelled to do so by lack of computing resources.

3.4 Execution time and memory requirements

Given N sequences of length L , the time complexity of SATCHMO is $O(N^2L^2 + N^3L)$. The N^3L term is typically small, so the time taken is roughly a constant factor multiplied by N^2L^2 . At the time of writing, I use 2.5 GHz Pentium 4 Windows and Linux PCs. Using these computers, I found that execution times were reasonably well approximated by the following expressions.

Options	Time (Windows)	Time (Linux)
<code>-v2 -bm (defaults)</code>	$3.0 \times 10^{-6} N^2L^2$ s	$2.0 \times 10^{-6} N^2L^2$ s
<code>-v2 -nobm</code>	$2.3 \times 10^{-6} N^2L^2$ s	$1.5 \times 10^{-6} N^2L^2$ s
<code>-v1 -nobm</code>	$1.0 \times 10^{-6} N^2L^2$ s	$0.7 \times 10^{-6} N^2L^2$ s

Actual times vary by $\pm 30\%$ or more, these should be used as rough guides only. If you are using a different model of Intel CPU, and/or a different clock speed, a correction factor should be applied.

These numbers show execution on Windows to be about 50% slower, a result I found surprising. On Windows I used the Microsoft Visual Studio version 7.0 (.Net) compiler with the default optimization settings for a Release build, and on Linux I used the g++ compiler with `-O3`, loop unrolling and inlining enabled. Generally I have found the Microsoft compiler to optimize pretty well, but I have little experience with floating-point code (which is heavily used in Lobster). If anyone has any insight into this, including perhaps suggestions on how to improve performance under Windows, I would be interested to hear from you.

By default, break matrices are enabled, and they may take considerable amounts of memory. An estimate of the maximum (peak) total memory required during a SATCHMO run is $40 NL^2$ bytes. Given 500 sequences of length is 500, this is 5 GB, which exceeds the capacity of most PCs. If break matrices are disabled by using the `-nobm` option, the memory required is typically well under 100 MB.

Lobster can estimate the time and memory requirements of a given set of sequences. Use the `-dbstats` (database statistics) option to specify a FASTA file name containing your sequences, for example:

```
lobster -dbstats seqs.fasta
```

Lobster will write something like the following to standard output.

```
358 seqs, average length 402
Memory -bm -v2 2314 MB
Time -bm -v2 17:15:35
Time -nobm 13:13:57
Time -nobm -v1 05:45:11
```

Times are given as hours:minutes:seconds. This is an estimate; your mileage may vary.

You can apply a correction factor to this estimate by setting the CPUGHZ environment variable. If you have a 1.0 GHz Pentium 4, then you should set CPUGHZ=1.0. On Windows:

```
SET CPUGHZ=1.0
```

On Linux (depending on your shell):

```
CPUGHZ=1.0
export CPUGHZ
```

Lobster will adjust its time estimates by a factor of 2.5/CPUGHZ. If you are using a different model of CPU, you can measure some execution times to see how they compare with the predicted times and set CPUGHZ to adjust accordingly.

3.5 Progress messages

SATCHMO can take hours or even days to run. It therefore writes some progress messages to standard output so that you can see that something is happening and get some idea of when it will finish. A typical progress message looks like this:

```
Cluster 237/443 98.3 MB 00:56:12 / 01:01:44
```

The format of this message is:

Stage number_done/total_number peak_memory elapsed_time / est_time_remaining

SATCHMO has three stages: constructing leaf nodes ("Leaf"), all-against-all scoring of leaves ("All-all") and clustering ("Cluster"). In each stage, the progress message shows the number of units completed and the total number of units in the stage. "Units" are leaves, leaves and clusters respectively. The peak memory in MB, the elapsed time so far and the estimated time remaining are also shown. The estimated time remaining is mostly calculated from the initial estimate (as described in the previous section) minus the elapsed time so far. After a few clusters have been created, the time remaining is re-estimated based on the assumption that each cluster takes the same time; this estimate is then updated after each cluster is complete. You may therefore see some big changes in the estimated time remaining when clustering begins, and you may find that the new estimate is quite pessimistic (quite often clustering gets significantly faster closer to the root as aligned regions get shorter).

If you don't want to see the progress messages (e.g., because you're running SATCHMO as a background process on your console), you can re-direct standard output to a file or to the null device.

4 SatchmoView

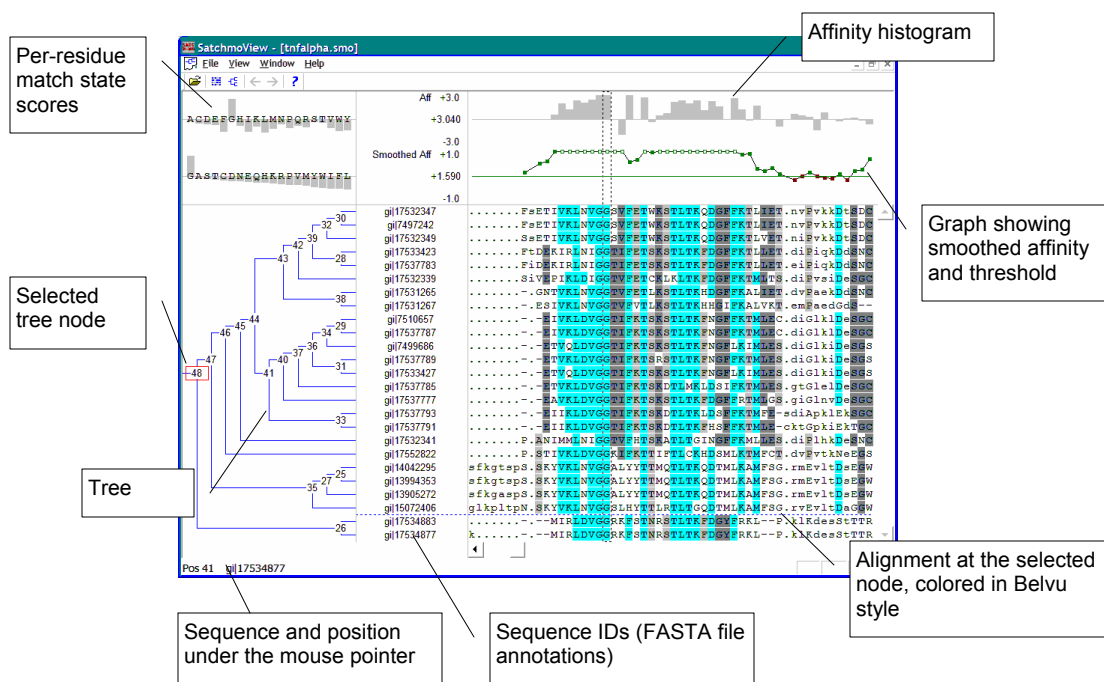
4.1 SatchmoView basics

SatchmoView is a Windows application. It was developed and tested on Windows XP, but I expect it to work under earlier versions (2000, NT) also. Install it by copying the *sv.exe* file; there are no registry entries or other configuration steps to worry about. Run it as you would any other Windows application, e.g. by double-clicking on *sv.exe*.

SatchmoView reads and displays a *.smo* file as produced by SATCHMO. Use the *-sv* command-line option to specify the file name, e.g.:

```
lobster -satchmo seqs.fasta -sv seqs.smo
```

Once SatchmoView has been started, you can use the *File, Open* menu command (Ctrl+O) to open and display a *.smo* file. The following illustration shows a typical display.



4.2 A Re-sizing display areas

The horizontal and vertical gray lines that separate the different display areas can be dragged to adjust the screen layout.

4.3 Alignment display

Select an alignment by clicking on a tree node. When SatchmoView first opens a file, it automatically selects the root node. The selected node is indicated by a red box around the node number.

4.3.1 Upper case letters

SATCHMO uses upper-case to flag columns that are predicted to be alignable. These columns, and only these columns, are used to construct match states for the profile HMM at the selected node. Columns are predicted to be alignable when (a) the column in the target alignment aligns to a match state in the template

alignment, and (b) the smoothed affinity of that column exceeds the $-minaff$ parameter (which defaults to 0.01).

4.3.2 Lower case letters

A column is displayed in lower case if it is not predicted to be alignable across all sequences. This arises if the smoothed affinity is too low, if the column was assigned to an insert state, or if the columns in the target and / or template alignments were predicted to be not aligned in a child node.

4.3.3 Dashes

Dash characters (-) are used to indicate deletions, i.e positions assigned delete states in the HMM.

4.3.4 Dots

Dot characters (.) are used to indicate inserts, i.e. positions where some other sequence visited an insert state more times than the current sequence; dots are then added to line up the alignable columns, causing all sequences to have the same length.

4.3.5 Colors

Positions are colored using the scheme defined by the Belvu alignment editor. Upper-case columns are scored using Blosum62 (lower-case columns remain uncolored). The Blosum62 score determines the color (if any) used in a column:

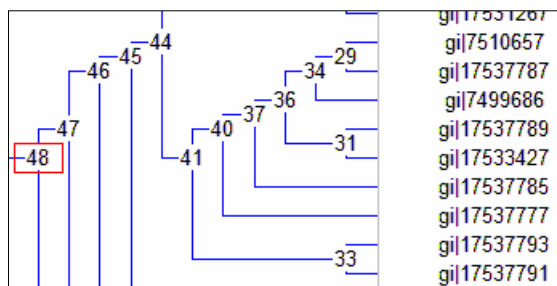
Cyan	Highest score (≥ 3)
Blue	Medium score (≥ 1.5)
Gray	Moderate score (≥ 0.5)
None	Low score (< 0.5)

Each letter in the column is scored against the most highly conserved residue type in the column. If that score for a letter is at least the column score, it is colored; otherwise it is not colored.

In summary, no more than one color is used in a column; this color indicates the Blosum62 score of the entire column. High-scoring residues are colored; any low-scoring residues remain uncolored.

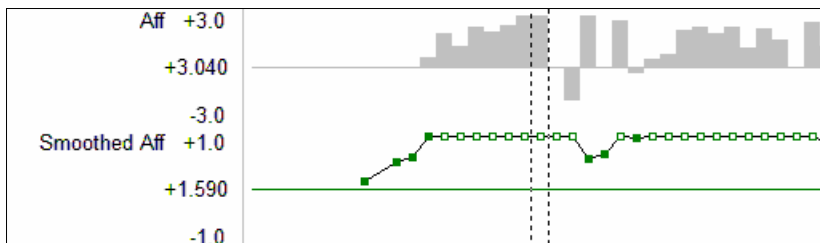
4.4 Tree display

The following shows a typical tree display.



A node is selected by clicking its number with the mouse pointer. Edges in the selected sub-tree are highlighted using blue lines, other edges are shown in gray. Edges at the leaves point to the sequence IDs and to their row in the alignment display.

Edge lengths in the tree are chosen for convenience in the display; they are not informative.

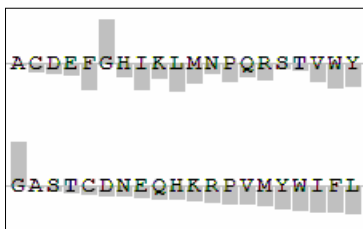


4.5 Affinity histogram and smoothed affinity graph

The histogram (upper) shows the match state affinity for each node in the HMM. The smoothed graph (lower) shows this value averaged over the window length specified by the `-window` option (default 7). The green horizontal line through the graph shows the minimum affinity threshold (as given by the `-minaff` option, default `0.01`). Values at or above the threshold are shown as green dots, values below the threshold are shown as red dots. When the dot is shown as an open circle, this means that the value is above `1.0` or below `-1.0`. The dashed lines indicate the selected column, the affinity and smoothed affinity values for that column are shown on the y axis (`+3.040` and `+1.590` respectively in the above screenshot).

4.6 Per-residue match state scores

The per-residue match state score display is found in the top-left corner of the SatchmoView window. It shows the match state score for the match state in the template HMM node used to construct the currently selected column (as opposed to the match state constructed from the selected column).



The same scores are displayed twice: sorted alphabetically (upper) and by score (lower).


By default, this display will not appear. Saving the required scores adds considerably to the size of a `.smo` file. Use the `-savemss` command-line option if you want to see this display.

4.7 Navigation


You can click on a tree node to view the alignment in that node. You can also click on a letter displayed in lower-case to find the last alignment (i.e., alignment closest to the root) in which that letter is in an aligned column. The cursor changes to a hand icon (as in a Web browser) when it is over a click-to-navigate position.

The toolbar has left- and right-arrow buttons `← →`. These navigate forwards and backwards in the history list in a similar way to a Web browser.


4.8 Saving an alignment

You can save the alignment currently being viewed by using the `File, Save Alignment As FASTA` menu command (`Ctrl+F`) or by clicking on the  toolbar button.

4.9 Saving a tree

You can save the currently selected sub-tree in PHYLIP (also called Newick) format by using the *File, Save Tree as PHYLIP* menu command (Ctrl+Y) or clicking on the  toolbar button.

4.10 Multiple document interface

SatchmoView conforms to the Windows Multiple Document Interface (MDI) standard. This means that you can open multiple files and open multiple windows on the same file. To manipulate these windows, use the *Window* menu and these buttons  (below the main title, not the similar but larger buttons on the main window title bar).

Microsoft is moving away from the MDI standard. Older versions of Office used MDI; more recent releases open a new copy of the whole application for each new document. If you are not familiar with or don't like MDI, you can open more copies of SatchmoView if you wish to see multiple views at one time. You can open the same file more than once without causing problems.

5 Coach

5.1 Building a profile HMM from an alignment

The following command line is used to construct an HMM.

```
lobster -msa2hmm alignmentfilename -hmm modelfilename
```

The alignment file must be in aligned FASTA format. Match states will be constructed from those columns that appear in upper-case letters.

The model file name is given the *.hmm* extension by convention.

Example:

```
lobster -msa2hmm kinase.fasta -hmm kinase.hmm
```

5.2 Aligning an alignment to a profile HMM

To align an alignment to an HMM, the following command line is used.

```
lobster -align target -hmm model -tpl template -out combined
```

The target file name must be in aligned FASTA format. Upper case letters indicate aligned columns; the COACH algorithm will not assign columns in lower-case letters to match states.

The model file name specifies an *.hmm* file, and the template file name specifies the alignment used to create that HMM.

The *-out* option specifies the file name of a FASTA file to store the combined alignment.

Example:

```
lobster -align aln.fasta -hmm kinase.hmm -tpl kinase.fasta -out both.fasta
```

5.3 HMM scoring

By default, COACH uses a simple null model as defined by HMMER. The Viterbi score relative to this null model is written to standard output (and to the log file, if applicable) when creating an alignment. To compute a SAM-style reverse score, use the *-rev* option, e.g.:

```
lobster -align aln.fasta -hmm kns.hmm -tpl kns.fasta -out both.fasta -rev
```

Using reverse scoring approximately doubles the execution time.

Reference:

Barrett,C., Hughey,R., Karplus,K. (1997), Scoring Hidden Markov Models, CABIOS **13**(2), 191-199.

5.4 Sequence weighting

The most expensive operation, both in constructing an HMM and aligning an alignment to an HMM, is sequence weighting. Lobster therefore provides a feature for storing sequence weights in an alignment file. This is useful when a given alignment file will be used repeatedly, for example to be scored against many different HMMs. This is done by using the following command line:

```
lobster -weight alignmentfile -out weightedalignmentfile
```

For example,

```
lobster -weight kinase.fasta -out kinase_weighted.fasta
```

The weights are stored by appending "`|weight=weight`" to the annotation line of each sequence. For example,

```
>gi|1207765|weight=0.15545
```

6 Other features

6.1 Log file

Upon request, Lobster will create a log file. The log file records the start and end time, all command-line options, and any error or warning messages produced during the run. Use `-log` to specify the log file name (overwriting any existing file) or `-loga` (to append to an existing file if one is found).

6.2 Comparing two alignments

Lobster can compare two alignments, producing three scores:

- Cline Shift score (CSS).
- Sum-of-pairs (SP) score, as defined for BAliBASE.
- Total columns (TC) score, as defined for BAliBASE.

The command line is as follows:

```
lobster -score testaln -ref refaln
```

Here, *testaln* is an aligned FASTA file containing the test alignment and *refaln* is an aligned FASTA file containing the reference alignment. In both files, upper-case letters indicate aligned columns, lower-case letters indicated non-aligned columns. The scores are written to standard output and, if specified, to the log file.

To compare a pair-wise alignment extracted from each alignment file, use the following command line:

```
lobster -score testaln -ref refaln -1 seq1 -2 seq2
```

Here, *seq1* and *seq2* are the sequence names, defined as the annotation line up to the first white-space character.

References:

Cline, M., R. Hughey, and K. Karplus, Predicting reliable regions in protein sequence alignments. *Bioinformatics*, 2002. **18**(2): p. 306-14.

Thompson, J.D., Plewniak, F. and Poch, O. (1999b), A comprehensive comparison of multiple sequence alignment programs, *Nucleic Acids Res.*, **27**(13), 2682-90.

7 File formats

7.1 Unaligned FASTA (.fasta or .fa)

Unaligned FASTA (or just FASTA) contains sequence data. White space (tabs, spaces and new-lines) and indel characters (dots and dashes) are ignored within sequence data.

Each sequence begins with an annotation line, which must have a greater-than (>) character in the first column. Lobster considers the sequence ID to be all the characters in the annotation line up to the first white-space character.

The alphabet is the standard 20-letter amino acid alphabet plus the special characters X (unknown residue), B (D or N) and Z (E or Q). Upper-case and lower-case letters are equivalent.

7.2 Aligned FASTA (.fasta or .fa)

Aligned FASTA is the same as unaligned FASTA, with the following differences.

Each sequence must have the same length. Indel characters (dots and dashes) are included as appropriate. Upper-case letters indicate aligned positions, lower-case letters unaligned positions. Dashes should be used as indels in aligned positions, dots otherwise. At a given position, all sequences must either be aligned or all be unaligned. The following is therefore an invalid aligned FASTA file.

```
>Seq1
AcD
>Seq2
ACD
```

The second column is marked as unaligned in Seq1 but aligned in Seq2: this is not allowed.

7.3 HMM parameter files (.hmm)

Lobster has its own HMM parameter file format. For various reasons, Lobster uses its own format. It would not be too hard to write a converter between Lobster and SAM or HMMER formats; this has not been done at the time of writing.

Following is a small *.hmm* file to use as an example. It is a text file that can be read using an editor. Some lines have been truncated for readability.

```
SatchmoHMM
{
length 3
entry -0.0628 -4.46
trans
; Node      MM      MD      MI      DM      DD      DI      IM      ID      II
0 -0.0423 -6.01 -5.95 -1.11 -1.06 -4.12 -1.11 -4.44 -1.02
1 -0.0423 -6.01 -5.95 -1.11 -1.06 -4.12 -1.11 -4.44 -1.02
2 -0.0603 -5.18 -5.95 -1.11 -1.06 -4.12 -1.11 -4.44 -1.02
match
; Node      A      C      D      E      ...
0 -0.687 -0.685 -1.62 -1.35 ...
1 -0.349 -1.04 0.118 0.216 ...
2 -0.258 -1.53 1.74 0.948 ...
insert
; Node      A      C      D      E      ...
0 0 0 0 0 ...
1 0 0 0 0 ...
2 * * * * ...
}
```

The file must begin with *SatchmoHMM*. Parameters are enclosed within curly braces { ... }. Comments are permitted, they are specified by a line beginning with a semi-colon and may appear anywhere except as the first line in the file. The number of nodes is specified by the *length* parameter. The scores for entering the

model via the first match state and first delete state follow the *entry* parameter. Transition scores for each node then follow the *trans* parameter. The first field is the node index (starting with zero), the order in which the scores are given is indicated by the comment. Then match and insert emission scores are given using a similar convention.

Following a convention inspired by HMMER, scores are stored as bit-scores relative to the simple HMMER null model. If you need to understand this, contact me or read the HMMER manual for a longer discussion. An asterisk (*) indicates a score of minus infinity (probability zero). You will notice that insert emission scores are all zero; this is because Lobster assumes background probabilities for all insert states (i.e., equal to the null model).

White space in the file is not significant except as needed to separate tokens.

7.4 SatchmoView files (.smo)

A SatchmoView file contains the binary tree built by SATCHMO. I won't reproduce an example here as the files are rather long. They are text files and can be read into an editor. The file is parsed into tokens. Curly braces *{* and *}* are special cases, they are tokens on their own, and text between curly braces is not parsed. All other tokens are terminated by white space. Text between curly braces is not intended for the top-level parser, it will have formatting rules of its own—it might be alignment data or HMM parameters, for example. Each binary tree node contains an alignment, and is marked as being a leaf or internal (i.e., non-leaf) node. A minimal node looks like this:

```
[
  type
  index index
  alignment
  {
    Aligned FASTA data
  }
]
```

Here, *type* is *leaf* or *internal*; *index* is the index number of the node. Index numbers range from 0 to $N-1$ for leaves and from N to $2N-2$ for internal nodes. The root node has index $2N-2$.

An internal node must have attributes:

```
leftchild index
rightchild index
```

Nodes may have other attributes. An attribute is specified as an attribute name followed by a value. A parser should ignore attributes it doesn't recognize. An attribute value is either a single token (a "short" attribute), or is enclosed within curly braces *{ ... }*, e.g. *alignment* (a "long" attribute). Reading and writing of long attribute values is delegated to functions specializing in its format (e.g., aligned FASTA). Right square brackets (*]*) and right curly brace characters (*}*) are not permitted within long attributes (to allow top-level parsers to skip parts of the file if desired).

The order nodes are written to the file is determined by the progress of the SATCHMO algorithm: a node is written when it is combined with another node so that the data in that node is not needed for future iterations. This produces something like a tree prefix order, but unfortunately not exactly. In prefix order, a left-to-right interpretation using a stack can be used to reconstruct the tree. In a *.smo* file, this is not possible; the *leftchild* and *rightchild* attributes must be used to determine branching order. It is guaranteed that a child node is always written before its parent.

A *.smo* file is terminated by a period (*.*).

8 Acknowledgements

I am greatly indebted to Kimmen Sjölander for sharing her ideas with me, teaching me something about protein informatics, and for a wonderful scientific collaboration. SATCHMO was Kimmen's idea.

Melissa Cline, Sean Eddy and Kevin Karplus contributed helpful ideas and discussions, showing patience and good humor in dealing with a newcomer to the field. Sean Eddy's HMMER package and its source code was an invaluable resource for me in learning practical implementation techniques for HMMs. Without Melissa Cline's encouragement, I would not have pursued COACH as far as I did and might not have discovered its applications outside SATCHMO—thanks, Melissa (and I highly recommend her PhD thesis to anyone interested in evaluating sequence alignment methods).

Thanks also to Wayne Christopher for providing a subroutine that optimizes the calculation of the log-Gamma function; this significantly improved the speed of SATCHMO.