

**jMOTU 1.0 by Anisah Goorah, Martin Jones and Mark Blaxter
May 2010 build 1.0.5 <http://www.nematodes.org/>
jMOTU defines molecular operational taxonomic units (MOTU) from
DNA barcode data using megablast and Needleman-Wunsch algorithms.**

jMOTU User Guide

Martin Jones, Anisah Ghoorah and Mark Blaxter

Institute of Evolutionary Biology

University of Edinburgh

<http://www.nematodes.org/bioinformatics/jMOTU>

10 May 2010

version 1.1

One-page overview

jMOTU is a software package for clustering barcode DNA sequence data into molecular operational taxonomic units (MOTU). **jMOTU** does the following:

- reads input sequences in FASTA or NEXUS format
- calculates distances between pairs of sequences using a combination of BLAST and the Needleman-Wunch exact global alignment algorithm
- clusters input sequences into MOTU using various cutoff measures

jMOTU can process large datasets and uses a series of filtering steps to minimise the number of exact global alignments that must be performed. When calculating pairwise distances, **jMOTU** ignores gaps and counts only nucleotide mis-matches, making it relatively robust to sequencing errors caused by homopolymer runs. Clustering is carried out using a greedy algorithm that is not dependent on input sequence order. **JMOTU** has been tested on raw datasets of around 50,000 input sequences, and can cluster larger datasets by performing preclustering of subsets of data before combining them for a global analysis.

jMOTU produces several types of output:

- text files describing the mapping of input sequences to MOTUs, and vice versa
- charts and comma separated value (CSV) files describing the numbers of MOTU defined at different cutoffs
- charts and comma separated value (CSV) files giving summary statistics of the input sequences
- a text file containing SQL statements which can be used to generate a relational database containing the relationships between input sequences and MOTUs. We provide example SQL queries for the database.

MOTU generated by **jMOTU** can be annotated with taxonomic information using another software package, **Taxonnerator**. See the **Taxonnerator** user guide for more details.

Quick Start Guide

Optimal use of jMOTU will be achieved by reading and following the main part of the guide that follows. However, to facilitate your getting started with the program, here is a 'quick start guide' that will take you through the process rapidly.

- (a) make a fasta file that contains a FEW sequences (we would suggest under 100) for your gene of interest. You can do this by copying them to a new directory, or by catenating them into a new file.
- (b) find out and note down where on your system the two components of the BLAST suite that jMOTU uses are located: you will need to know the paths to formatdb and megablast.
- (c) launch jMOTU (either using the command `java -jar jMOTU_1_0_5.jar` or by double clicking on the .jar file (LINUX) or the jMOTU application file (Mac OSX)).
- (d) under the 'Executables' tab, enter the locations of formatdb and megablast. The locations must include the name of the executable (megablast and formatdb on linux/mac, megablast.exe and formatdb.exe on Windows)
- (e) click the 'Load Dataset' button, and navigate to where you have generated your test dataset. 'choose' your dataset. The data will be imported and a histogram of sequence lengths displayed.
- (f) Click the 'Preprocess' tab if you want to filter any of the sequences out because they are too short. If not, proceed directly to the next step.
- (g) Click on the 'Parameters' tab. Here you enter the MOTU base-difference parameters to use. For this first run, type "1-10" in the 'Value' box, and press the "+" button. The values 1,2, ..., 10 will be displayed in the field to the right.
- (h) Type "97" in the 'Low BLAST identity filter' box.
- (i) Leave the boxed in the 'Sequence alignment Overlap' and 'Other' panels at their defaults.
- (j) Choose a MOTU set name in the 'MOTU set names and file locations' panel (something like 'test' is best for now) and use the interactive browser (indicated by the hand-lens button) to choose a directory on your computer to save these data in.
- (k) Click the 'Run' button. The program will first compare your sequences to identify the unique-sequence subset, then use megablast to compare these representative sequences, and then perform the relevant set of pairwise alignments using the Needleman-Wunsch algorithm. The final clustering of sequences into MOTU follows.
- (l) The program will, when the analysis is complete, switch to the 'Visualise' tab, showing a text view of the results. This can also be displayed graphically, and the image saved if required (but the same graphs are generated and saved in the next step).
- (m) Click on the 'Export' tab, and select the outputs required. This will write the selected files to (in this example case) to 'test_output', and also importantly write the jMOTU.conf file to your home directory. The jMOTU.conf file allows the program to 'remember' between runs where formatdb, megablast, and your last visited file directory are.

Table of Contents

1 Installation.....	4
1.1 Distribution.....	4
1.2 Dependencies.....	4
1.2.1 Java.....	4
1.2.2 Megablast & formatdb.....	4
1.2.3 Postgresql.....	5
1.3 Hardware.....	5
2 Usage.....	6
2.1 Input.....	6
2.2 Preprocessing.....	6
2.3 Parameters.....	6
2.3.1 “Parameters to use for MOTU definition” panel.....	6
2.3.2 “Sequence alignment overlap” panel.....	7
2.3.3 “MOTU set names and file locations” panel.....	7
2.3.4 “Other” panel.....	7
2.3.5 Recommended parameter settings.....	8
2.3.6 Running jMOTU.....	9
2.3.6.1 Preclustering.....	9
2.3.6.2 Complete MOTU definition run	9
2.4 Results.....	9
2.5 Visualise.....	10
2.6 Export.....	10
2.7 Exit.....	10
2.8 Databasing.....	11
3 Algorithms used in jMOTU.....	12
3.1 Preclustering.....	12
3.2 Megablast.....	12
3.3 Global alignment.....	14
3.4 MOTU clustering.....	14
4 Database.....	15
4.1 jMOTU database schema.....	15
4.2 jMOTU example SQL queries.....	16
4.2.1 List MOTUs and their representative sequences.....	16
4.2.2 Get representative sequence for single named MOTU.....	16
4.2.3 Get number of MOTU defined for each cutoff.....	16
4.2.4 Get number of sequences in each motu for a given cutoff.....	16
4.2.5 Get length of representative sequence for each MOTU for a given cutoff along with number of sequences.....	16
4.2.6 Calculate GC content of representative sequences for each MOTU.....	16
5 Implementation	17

1 Installation

jMOTU is distributed as a compiled Java application, so installation is minimal. However, jMOTU relies on a number of external programs and files to run.

1.1 Distribution

jMOTU is distributed as a compressed .tar.gz file. Extract the contents of this file to a new directory. Inside the newly-created directory should be:

- *jMOTU.jar* file
- *lib* directory

To run jMOTU, double-click on the jMOTU.jar file, or from the command line run

```
java -jar jMOTU.jar
```

This is the preferred way to run jMOTU as it ensures that the working directory will be the correct one.

1.2 Dependencies

To run jMOTU you need a recent version of Java and a local copy of the **megablast** command-line tool. If you want to turn your results into a relational database, you will also need a relational database management system (RDBMS), preferably **Postgresql**.

1.2.1 Java

jMOTU has been tested using Java 1.5 and 1.6. To run jMOTU you need the Java Virtual Machine (JVM) or Java Runtime Environment (JRE). You do **not** need the Java Development Kit (JDK). There may be slight differences in performance and appearance when you run jMOTU with different versions of java. Java can be downloaded here:

<http://www.java.com/en/download/>

jMOTU has been tested with both the 32-bit and 64-bit version of Java. However, note that the maximum amount of memory that can be allocated with 32-bit Java is around 1.6 Gigabytes (GByte). The amount of memory allocated to jMOTU will limit the maximum number of input sequences that can be processed; for this reason the 64-bit version of Java is strongly recommended if you will be processing large numbers of input sequences.

The best way to control the amount of memory available to jMOTU is by passing the Xmx command-line parameter to Java. Running this command:

```
java -Xmx8000m -jar jMOTU.jar
```

will allocate 8000 Megabytes (8 Gbyte) of memory to jMOTU. We recommend that you keep this parameter as high as possible in order to maximise the number of input sequences that can be processed.

1.2.2 Megablast & formatdb

The command-line tools **megablast** and **formatdb** are used to identify highly similar pairs of input

sequences. They are part of the BLAST tools collection which can be downloaded here:

<ftp://ftp.ncbi.nlm.nih.gov/blast/executables/LATEST/>

You can set the paths for **megablast** and **formatdb** by using the *executables* menu from within jMOTU. The paths must include the names of the executable files. E.g.

/usr/bin/megablast (on Linux / Mac OS)

c:\blast\megablast.exe (on Windows)

jMOTU was developed with version 2.2.20 of the BLAST tools collection. jMOTU is not currently compatible with the BLAST+ suite of advanced tools from NCBI.

1.2.3 Postgresql

If you want to turn the results from jMOTU into a relational database, you will need a relational database management system (RDBMS) installed. jMOTU has been tested with Postgresql 8.3 and should work with future versions. Postgresql can be obtained from

<http://www.postgresql.org/>

Since the output produced by jMOTU consists of standard Structured Query Language (SQL) statements, the file should be easily adapted for use in other SQL-compliant databases.

1.3 Hardware

jMOTU is designed to be fast at the expense of being memory-hungry. Because all pairwise distances between sequences are stored in memory, clustering at multiple cutoff values can proceed very rapidly once distance have been generated (see *Algorithms* for more details). However, this means that large amount of memory are required for jMOTU. We have had best results with between 8 and 15 Gbyte of memory allocated to jMOTU. The amount of memory available determines the number of input sequences that can be processed.

jMOTU can take advantage of computers with multiple processors by instructing **megablast** to use multiple processors when identifying highly similar sets of sequences. However, with very large numbers of input sequences, the execution time of jMOTU is dominated by the global pairwise alignment step (see *Algorithms* for further details) which cannot take advantage of multiple processors.

2 Usage

The jMOTU window is divided up into six tabs that deal with different stages of the clustering workflow. The tabs are explained in order in this section.

2.1 Input

This tab allows you to add input sequences to jMOTU. Input sequences can be in either FASTA or NEXUS format, and can be in a single multi-sequence file or split over multiple files. If the sequence header names contain spaces, they will be replaced with underscores. NEXUS format files should end with

```
end;
```

rather than

```
endblock;
```

To load a dataset click on the *input* tab then click on the *load dataset* button. After selecting the file format you will be presented with an *open file* dialogue. You can choose either a file or a directory. Depending on the number of sequences, it might take a few seconds to load the dataset. The name of each dataset is displayed alongside the time it was loaded on the left-hand side of the window. Multiple datasets can be loaded. To view a summary of a dataset, select it from the list of datasets on the left-hand side of the window. Sequence length distribution for the selected dataset will be displayed along with the minimum, maximum and mean length of the sequences and the standard deviation.

2.2 Preprocessing

This tab allows the user to filter out short sequences before carrying out clustering. If you want to filter out short sequences, enter the minimum sequence length in base pairs and click the *Filter* button. The summary panel will display the new set of summary statistics for the filtered data. You can use the *save* panel to generate a FASTA format file containing either the excluded or retained sequences. To cancel filtering click the *Reset* button.

2.3 Parameters

The parameters tab allows you to define the parameters for the clustering process. See the description of the clustering process in the Algorithms section for details of how they are used. You can reset all parameters by clicking the *Reset* button.

2.3.1 “Parameters to use for MOTU definition” panel

The upper-left panel *Parameters to use for MOTU definition* allows you to set the cutoff values at which clustering will be carried out. The cutoff value for a given clustering is the maximum number of base pairs difference between two sequences that will be grouped into the same cluster. To carry out clustering at a single cutoff value, select *Single* and enter the cutoff in the Value box. To add multiple cutoffs select the *Multiple* radio button and enter a set of values separated by spaces (e.g. 1 2 4-8), and click on the *Add* button. You can add as many cutoffs as you want. You can also remove a single or multiple cut-off from the list by selecting it (or them) and clicking on the *Remove* button.

Because of the way jMOTU is structured, performing the clustering is very rapid and requires only a small amount of time relative to the computationally-intensive process of generating pairwise distances. In practical terms, jMOTU takes roughly the same amount of time to run on a given dataset regardless of the number of cutoff values. Therefore, it is recommended that you enter a wide range of cutoffs (1-20) in order to avoid having to repeat the computationally intensive part of the analysis.

The *Low BLAST identity filter* is a 'gathering parameter' that allows jMOTU to assess suboptimal megablast matches during the Needleman-Wunch alignment step. Setting this value too low will dramatically increase the number of exact global alignments carried out (and consequently the run time of the program) but may not affect the results of clustering. A value of 95% or 97% (i.e. pairs of sequences scoring 95 or 97% of the expected minimum score will be assessed) is suggested.

2.3.2 “Sequence alignment overlap” panel

jMOTU's clustering algorithm groups pairs of sequences together only when the overlap between them is greater than some specified length. In the *Sequence alignment overlap* panel (upper right) there are three different ways to specify this length:

To enter an absolute number of base pairs, select *Minimum alignment length in base pairs* and enter the value in the box.

To set the minimum overlap to 60% of the length of the shortest sequence in the input dataset, select *60% of minimum sequence length*. The box will be pre-populated with the correct value. This value generally gives good results.

To express the minimum overlap in terms of a percentage of the length of the shortest sequence in the input dataset, select *Percentage of minimum sequence length* and enter the value in the box.

2.3.3 “MOTU set names and file locations” panel

In the *MOTU set names and file locations* panel (lower left) enter a unique name for the set of MOTU to be generated. This name will be for the directory for temporary files (see Other section), for the directory of output files (see Export section) and in the names of the MOTU that are generated. You should also select which directory on your system you would like this directory created in. The directory path and file name must not contain spaces. jMOTU can automatically delete the temporary directory after a run: just select the tick box to activate this feature. Note that if you are performing Preclustering you should NOT delete this directory.

2.3.4 “Other” panel

In the *Other* panel (lower right) you can set miscellaneous parameters. Number of processors to use in megablast determines the value of the -a parameter passed to the megablast command line tool, which determines how many processors it will use. For best performance, set this parameter equal to the number of processor cores on your machine.

The file selector box lets you specify where the temporary output files are to be stored. The default value is your home directory. Make sure that this directory name does not contain spaces.

The check box lets you specify whether the temporary directory should be deleted after the run is complete. The default behaviour is to keep the directory as the temporary files can sometimes be useful. However, if you are carrying out multiple analyses and want to conserve disk space, you may want to check this box.

2.3.5 Recommended parameter settings

We suggest the following parameter settings for jMOTU. Obviously you may wish to explore the effects of different parameter settings for your particular datasets.

The cutoff settings should reflect the expected divergence between the significant operational taxonomic units in your data. For nuclear small subunit ribosomal RNA data (nSSU), most species have unique sequences, and a low percentage difference between sequences is likely to indicate a biologically significant split. For mitochondrial cytochrome oxidase I (mCOI) data, the between-species divergence ranges from <1% to >10% for different sister taxa across Metazoa. jMOTU calculates operational taxonomic units based on exact base differences rather than percentage differences, and does not 'correct' for homoplastic substitution (as such homoplasy is unlikely to be significant in species level analyses).

The MOTU inference part of the jMOTU process is computationally trivial compared to the sequence comparison and pairwise alignment parts. We therefore recommend that you choose a wide range of cutoffs for your analyses (e.g. from 1 to 30 bases in 1 base intervals). This analysis will take the same time as one that only asks for MOTU to be defined at cutoffs of 1 and 30 bases.

If you have a set of DNA barcode data, for example from the 'Folmer' region of mCOI, or the 5' end of the nSSU, where you expect that all the sequences will span the same segment of the barcoding locus, the %overlap in ***Sequence alignment overlap*** settings can be made quite high (say 90%).

If your data derive from a more variable locus, or the individual sequences are of different lengths, the default settings are probably sufficient (60% of the length of the shortest sequence).

The *Low BLAST identity filter* is a method to include megablast-defined matches that fall just below the expected bit score for the maximal base difference cutoff requested. This filter allows the program to retrieve matches that megablast scores as being worse than required, but that the exact pairwise alignment of Needleman-Wunch might be able to align to give a match that falls within the cutoff. Normally it should be set to a high value (95% or greater), as lower values may result in a requirement for many-fold more NW alignments and thus increase the running time of the analysis significantly. Lower values might be advisable when the sequences being analysed are highly divergent or likely to contain large indels (such as nuclear ribosomal ITS sequences).

We have used jMOTU for up to ~50,000 Roche 454 nSSU sequences. Memory requirements for the base difference matrix mean that such large datasets require computers with large installed RAM (8 Gbyte and greater). It is possible to process larger datasets, or perform analyses on computers with limiting RAM, by:

- (1) splitting the dataset into segments that can each individually be preclustered (for example, a 200,000 sequence dataset could be split into eight segments of 25,000 sequences each)
- (2) performing preclustering on each segment independently (see below for details)
- (3) catenating the resulting *motusetname_preclusters_representatives.fsa* files into one fasta file
- (4) performing full MOTU clustering on this file

Unless you are sampling a hyperdiverse community, each segment (of, for example, 25,000 sequences) should result in a *preclusters_representatives.fsa* file with many fewer sequences (in our experience, environmental samples usually yield between 10 and 20% unique preclustered sequences). Catenating these files (for example using the 'cat' command in a terminal window) will generate a new fasta file for clustering with a manageable number of sequences.

2.3.6 Running jMOTU

2.3.6.1 Preclustering

You can run just the *preclustering* part of jMOTU by clicking on the Precluster button. This instructs the program to carry out the first step of the MOTU definition process, the reduction in redundancy of the dataset. You still need to fill in the values in the **MOTU set names and file locations** panel (see above). Once the preclustering step is complete, three files are written to the output directory:

Output file	Format	Description
<i>motusetname_preclusters_representatives.fsa</i>	FASTA	a fasta file of the longest representative sequence of each precluster.
<i>motusetname_preclusters_map.txt</i>	<i>text</i>	a text file mapping each sequence to its longest representative sequence
<i>motusetname_preclusters_counts.txt</i>	<i>text</i>	a file listing the names of the longest representative sequences and the number of sequences they represent.

These files are also created during a full run of jMOTU. After preclustering the program exits.

2.3.6.2 Complete MOTU definition run

When you have entered all necessary parameters, click the run button. A progress bar will appear at the bottom of the window and keep track of the progress of the run (see Algorithms for details of the various stages). If you just want to carry out the first part of the algorithm (preclustering of exact subsequences; see Algorithms for details) then click Precluster. Runs can be cancelled at any time by clicking Cancel.

2.4 Results

The Results panel will automatically be given focus when the run is complete and displays the results of clustering in several different ways.

To view the members of each MOTU as a list, select Motu Id - Fasta Id. This will show each MOTU name (in colour) next to a list of all the input sequences that are in that MOTU. To view the MOTU to which each input sequence belongs (i.e. the reverse mapping) select Fasta Id - Motu Id. For ease of comparison, MOTU name colours are consistent between these two views. If you have selected multiple cutoff values in the Parameters tab (see Parameters section) you can select a value from the Cutoff drop-down box to see results for that cutoff. For a given cutoff, the number of input sequences and the number of MOTU are displayed in bold.

To view the relationship between cutoff and number of MOTU defined, select Cutoff - No. Motu. The results are displayed here as a table, but can be better viewed in the Visualize tab (see Visualize section) or in separate charting software following export of the data (see Export section).

2.5 Visualise

The *Visualize* tab shows the relationship between cutoff and number of MOTU as a chart. On the Y-axis is the number of MOTU defined at a given cutoff. The X-axis shows the cutoff values, either as an absolute number of base pairs, or as a percentage of mean length. Use the *View* radio buttons to switch between the two representations. The graphs can be exported as .png files, or saved during the Export process (see below).

2.6 Export

The Export tab lets you save the results of a jMOTU run for further analysis. Files will be saved in a directory called *name_output*, where *name* is the MOTU set name that was specified in the *Parameters* tab. You can use the file chooser to select where the output directory should be created.

There are several different options for output files; use the check boxes to select which ones you want to create, or tick the *Select all* box to generate all output files:

Output file	Format	Description
Length of sequences used and their statistics	Plain text	Summary statistics and sequence lengths for the input sequences
All the parameters used to define MOTU (incl.) sequences	Plain text	Summary of the parameters used in MOTU clustering
Unused sequences	FASTA	Contains the sequences that were excluded from minimum length filter
Motu Id – Fasta Id	Tab-separated values	Mapping of input sequences -> MOTU (one file per cutoff value)
Fasta Id – Motu Id	Tab-separated values	Mapping of MOTU -> input sequences (one file per cutoff value)
Cutoff – No. of MOTUs	Tab-separated values	Table showing the number of MOTU defined at each cutoff value
Sequence Length Distribution	PDF	Chart showing the distribution of lengths of input sequences
Cutoff versus No. MOTU defined	PDF	Chart showing number of MOTU defined at each cutoff value
Cutoff percentage versus No. MOTU defined	PDF	As above but cutoff expressed as % mean sequence length
SQL statements for database import	SQL text	SQL statements to create relational database
Matrix of pairwise base differences	GZipped comma separated values	The distance matrix calculated by Needleman-Wunch alignment

2.7 Exit

When you exit from jMOTU, any results not Exported (see above) will be lost.

2.8 Databasing

If you have exported SQL statements as described above, you can use the generated file to create a relational database containing your MOTU cluster data. Assuming you have the postgresql command-line client **psql** installed, create an empty database called *motu* (you can of course use any sensible name you choose):

```
createdb motu
```

then from the output directory run:

```
psql motu <output.sql
```

This will populate the database MOTU with your cluster data. See the *Database* section for a schema and example SQL queries

Algorithms used in jMOTU

jMOTU uses pairwise distances to cluster input sequences into MOTU. Distances between pairs of sequences are calculated by aligning the sequences using the Needleman-Wunch algorithm as implemented in BioJava, then counting the number of mismatches. The Needleman-Wunch algorithm is guaranteed to find the best global alignment between a pair of sequences, but is very computationally intensive. To minimize the number of alignments that must be carried out, jMOTU uses a three-step process. An overview, with examples, is shown in figure 1 below.

2.9 Preclustering

In this step, any sequences that are exact sub-sequences of another sequence are clustered together (referred to as 'preclustering'). Because the distance between a given sequence and its subsequence will always be 0, two such sequences will always appear in the same MOTU regardless of the cutoff value. Therefore, this preclustering does not affect the final clusters, but reduces the number of sequence similarity searches and pairwise alignments that need to be carried out.

In the figure there are 9 input sequences. All red sequences are subsequences of **a**, and all blue sequences are subsequences of **d**. During preclustering, **c** and **b** are clustered with **a**, and **f** is clustered with **d**. Sequences **c**, **b** and **f** are effectively removed from the input dataset during subsequent steps, and are added back to MOTUs when the results are shown. During the remaining steps, **a** is effectively being used as a representative sequence for the precluster that includes **a**, **b** and **c**. Input sequences **e**, **g**, **h** and **i** have no subsequences and are not themselves subsequences of any other input sequence and they do not take part in the preclustering step

2.10 Megablast

To further reduce the number of exact global alignments that must be carried out, jMOTU uses the **megablast** command-line tool to identify highly similar pairs of sequences. Exact global alignments are then carried out only for pairs of sequences with more than a given proportion of identical bases (the *Gathering parameter* described in the *Parameters* section).

To identify highly similar sets of sequences, a BLAST database is generated consisting of the preclustered input sequences. Each input sequence in turn is then searched against this database to generate a table of similarities (see figure). In the table, coloured letters represent the input sequences that are representing preclusters. The number in each cell is the percentage of identical

bases between the two sequences. Hence, sequences **a** and **g** are 96% identical.

Digression: The table could also be generated by carrying out an all-vs-all **megablast** search with the preclustered input sequences as both the subjects and queries. Such an approach would possibly be marginally faster than BLASTing each sequence separately, as less time would be spent writing to disk. However, the speed increase would be insignificant compared to the time required to carry out the **megablast** searches and subsequent global alignments, and would make it more difficult to monitor progress.

After the table of identity has been constructed, highly similar pairs of sequences can be identified. In the figure, 95% has been selected as the Gathering parameter, so the following pairs of sequences are selected for global alignment: a+d, a+e, a+g, a+h, d+e, d+g, d+h, e+g, e+h, g+h. The following pairs of sequences are not similar enough to be globally aligned: a+i, d+i, e+i, g+i, h+i. In the figure, identity scores that are above the threshold are coloured black, while those under the threshold are grey.

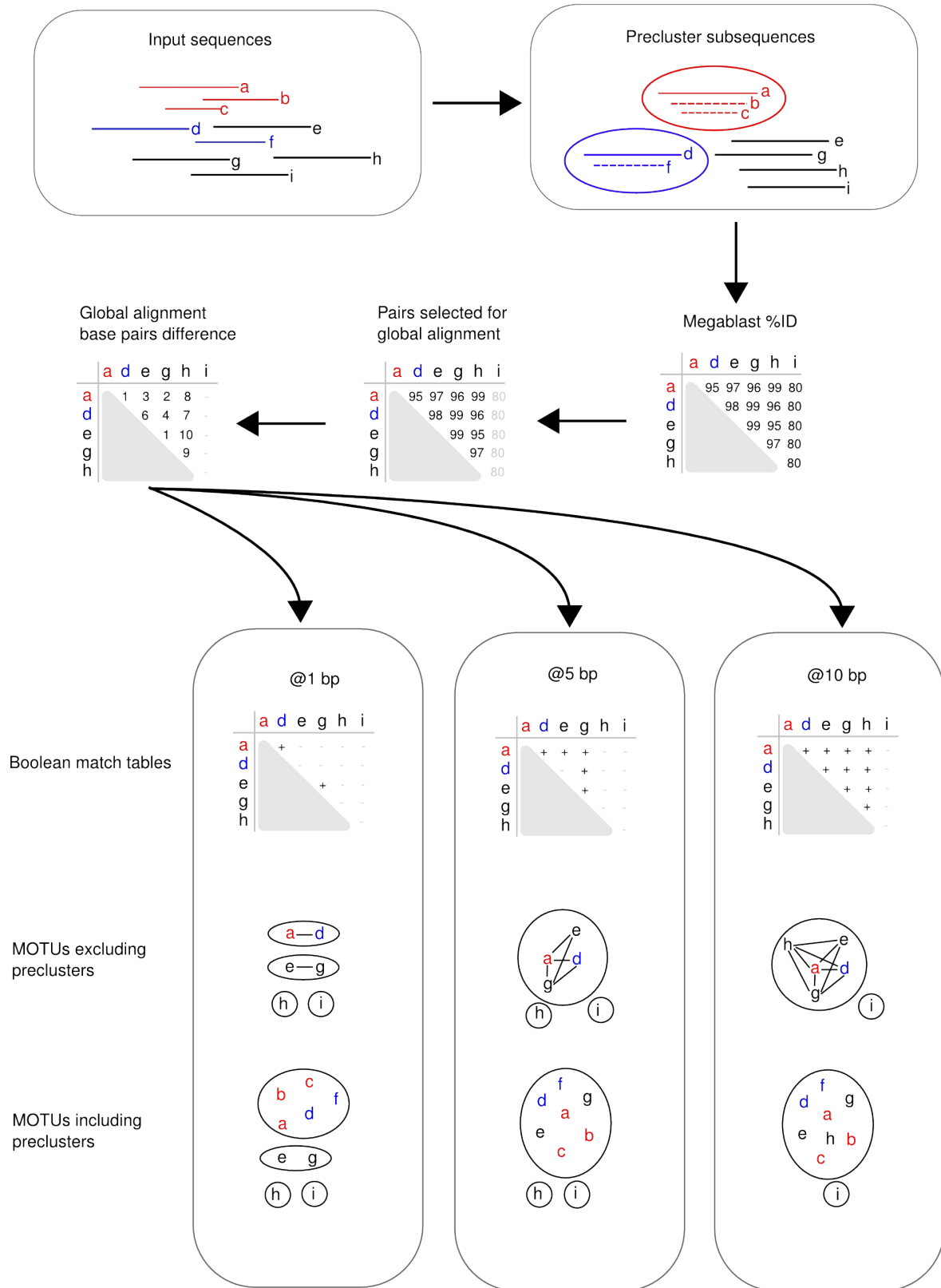


Figure 1: Cartoon of the jMOTU method

2.11 Global alignment

For clustering sequences into MOTU, we are interested in the exact number of mismatches between sequence pairs. The Needleman-Wunch algorithm is a well-characterised alignment algorithm that is guaranteed to find the best global alignment of any given pair of sequences. This is in contrast to **megablast** which considers only local regions of similarity.

To generate a table of pairwise distances, highly similar pairs that were identified in the previous step are sequentially aligned using the BioJava implementation of the Needleman-Wunch algorithm. The result is a table like that shown in the figure, where the distance score in each cell is the number of mismatches between a pair of sequences. Empty cells represent the case where the similarity between a pair of sequences (calculated in the previous step) was below the threshold. For example, **a** and **e** have 3 mismatches when globally aligned, whereas the cell for **a** and **i** is empty.

When calculating mismatches, gaps are ignored - only nucleotide-nucleotide pairs of characters are counted. This should limit the degree to which the mismatch score is affected by sequencing error due to homopolymer runs (which tends to generate insertions or deletions relative to the true sequence) and ensures that the distances reflect real genetic distance. This, in turn, will ensure that sequences are eventually clustered on the basis of shared sequence, and not shared sequencing error.

2.12 MOTU clustering

The clustering algorithm used in jMOTU is greedy; for a randomly picked starting sequence A, it gathers a set of sequences whose distance to A is less than or equal to the cutoff value. Then for each member of that set, it repeats the process, gathering all sequences whose distance to the member is less than or equal to the cutoff and adding them to the set. This process repeats until the first iteration where no new sequences have been added to the set. The set is then complete; the members of that set are clustered together as a MOTU and a new starting sequence is picked from the pool of unclustered sequences.

To allow rapid calculation of these sets, the clustering process for a given cutoff begins with the construction of a table, similar to those previously described for identity score and mismatch number, but where each cell is a boolean value recording whether the distance between a pair of sequences is less than or equal to the cutoff.

Examples are shown in the figure for cutoffs of 1, 5 and 10 base pairs. Each vertical group of illustrations shows the boolean table, the results of the MOTU clustering process (with links between sequences shown by solid lines), and the results after the addition of the subsequences that were removed in the first step.

The 5 base pair example illustrates an important point: because the clustering algorithm is greedy, the choice of starting sequence does not affect the results. Consider two potential starting sequences chosen to generate the first cluster.

If **a** were chosen as the starting sequence, then **e**, **g** and **d** would be added to the set on the first iteration. On the second iteration no new sequences would be added to the set (since **e**, **g** and **d** only have links to each other and to **a**) and the cluster would be complete.

If **e** were chosen as the starting sequence then **a** and **g** would be added to the set on the first iteration. **d** would not be added, since there is no link at this cutoff between **d** and **e**. However, it would be added on the second iteration, since there is a link between **d** and **a** and between **d** and **g**. Either starting sequence leads to the same outcome.

3 Database

jMOTU generates SQL commands which can be used to create a relational database of MOTU clusters (see *Export* section). The SQL commands include those necessary both to create the schema and to populate it with data. It is not necessary to create a relational database to view your clustering results, but it does allow you to carry out flexible queries on your data. If you want to annotate your results using the **Taxonnerator** tool, you do need to create a database since **Taxonnerator** uses it to read and store information (see **Taxonnerator** user guide for more information). We have used standard SQL statements in the *output.sql* file as far as was practical, except for one (the use of *cascade* for deleting table entries), and so the *output.sql* file should be easily edited for use with other platforms such as MySQL.

3.1 jMOTU database schema

The jMOTU database schema is simple, and is shown below.

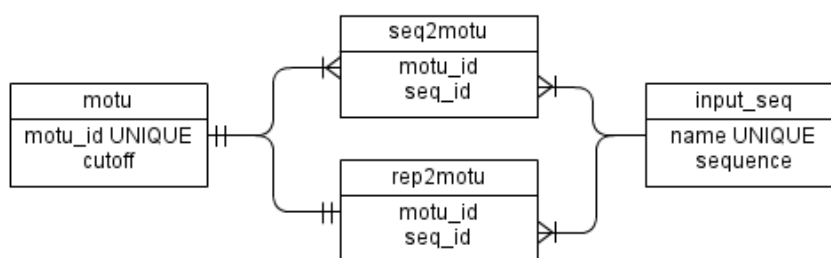


Figure 2: The jMOTU database schema

Each input sequence is represented by a row in the *input_seq* table, which records its unique name and its DNA sequence. Each MOTU is represented by a row in the *motu* table, and has a unique id and a cutoff value.

The *seq2motu* table records which input sequences belong to a given MOTU. Each MOTU has multiple corresponding rows in the *seq2motu* table - one for each of its member sequences. Note that each input sequence can belong to multiple MOTU - one for each cutoff value.

The *rep2motu* table stores the representative sequence for each MOTU - the longest sequence that belongs to that MOTU. Note that each MOTU is represented by only a single row in the *rep2motu* table, because each MOTU only has a single representative sequence (although it may have many shorter sequences).

3.2 jMOTU example SQL queries

Below are a number of examples of SQL queries which can be run against a jMOTU database, showing how useful information can be extracted. Postgresql has many built-in string processing functions which allow complex queries to be built.

3.2.1 List MOTUs and their representative sequences

```
SELECT motu.motu_id, input_seq.seq FROM motu, input_seq,
rep2motu WHERE rep2motu.motu_id=motu.motu_id;
```

3.2.2 Get representative sequence for single named MOTU

```
SELECT name, seq FROM input_seq, rep2motu WHERE input_seq.name =
rep2motu.seq_name AND rep2motu.motu_id='ast_0bp_MOTU0048';
```

3.2.3 Get number of MOTU defined for each cutoff

```
SELECT cutoff, count(*) FROM motu GROUP BY cutoff ORDER BY
cutoff ASC;
```

3.2.4 Get number of sequences in each motu for a given cutoff

```
SELECT motu.motu_id, count(seq2motu.seq_name) FROM motu,
seq2motu WHERE motu.cutoff=1 AND motu.motu_id=seq2motu.motu_id
GROUP BY motu.motu_id ORDER BY count DESC;
```

3.2.5 Get length of representative sequence for each MOTU for a given cutoff along with number of sequences

```
SELECT motu.motu_id, count(seq2motu.seq_name),
length(input_seq.seq) FROM motu, seq2motu, rep2motu, input_seq
WHERE motu.cutoff=1 AND motu.motu_id=seq2motu.motu_id AND
motu.motu_id=rep2motu.motu_id AND input_seq.name =
rep2motu.seq_name GROUP BY motu.motu_id, input_seq.seq ORDER BY
length DESC;
```

3.2.6 Calculate GC content of representative sequences for each MOTU

```
SELECT motu.motu_id, length(translate(input_seq.seq, 'gc',
''))::real / length(seq) AS gc_content FROM input_seq, motu,
rep2motu WHERE input_seq.name=rep2motu.seq_name AND
motu.motu_id=rep2motu.motu_id ORDER BY at_content DESC;
```

4 Implementation

jMOTU is written in Java. It uses the BioJava library to parse input files and carry out global alignments; the JFreeChart library to generate charts, and the iText library to generate PDF output.

jMOTU is available under the GPL.

The core of jMOTU was written by Anisah Goorah in 2008 and updated by Martin Jones and Mark Blaxter in 2010.