MUSCLE User Guide

Multiple sequence comparison by log-expectation
by Robert C. Edgar

Version 2.1
January 2004

http://www.drive5.com/muscle
muscle (at) drive5.com

Table of Contents

1 Introduction

MUSCLE is a program for creating multiple alignments of amino acid sequences. A range of options is provided that give you the choice of optimizing accuracy, speed, or some compromise between the two. Default parameters are those that give the best average accuracy in our tests. Using versions current at the time of writing, my tests show that MUSCLE can achieve both better average accuracy and better speed than CLUSTALW or T-Coffee, depending on the chosen options.

1.1 Changes in version 2.1

There has been a substantial re-design of MUSCLE since 2.0. The speed of the algorithm has been greatly improved, and many "research" options have been removed.

2 Quick Start

The MUSCLE algorithm is delivered as a command-line program called *muscle*. If you are running under Linux or Unix that means you will need to be working at a shell prompt. If you are running under Windows, you should be in a command window (nostalgically known to us older people as a DOS prompt). If you don't know how to use command-line programs, you will need to get help—from now on, I will assume you know how to work with command-line programs.

2.1 Installation

Copy the *muscle* binary file to a directory that is accessible from your computer. That's it—there are no configuration files, libraries, environment variables or other settings to worry about. If you are using Windows, then the binary file is named *muscle.exe*. From now on *muscle* should be understood to mean "*muscle* if you are using Linux or Unix, *muscle.exe* if you are using Windows".

2.2 Making an alignment

Make a FASTA file containing some protein sequences. (If you are not familiar with FASTA format, it is described in detail later in this Guide.) For now, just to make things fast, limit the number of sequence in the file to no more than 50 and the sequence length to be no more than 500. Call the input file *seqs.fa*. (An example file named *seqs.fa* is distributed with the standard MUSCLE package). Make sure the directory containing the *muscle* binary is in your path. (If it isn't, you can run it by typing the full path name, and the following example command lines must be changed accordingly). Now type:

```
muscle -in seqs.fa -out seqs.afa
```

You should see some progress messages. If *muscle* completes successfully, it will create a file *seqs.afa* containing the alignment. By default, output is created in "aligned FASTA" format (hence the *.afa* extension). This is just like regular FASTA except that gaps are added in order to align the sequences. This is a nice format for computers but not very readable for people, so to look at the alignment you will want an alignment viewer such as Belvu, or a script that converts FASTA to a more readable format. You can also use the *–msf* command-line option to request output in MSF format, which is easier to understand for people. If *muscle* gives an error message and you don't know how to fix it, please read the Troubleshooting section.

2.3 Pipelining

Input can be taken from standard input, and output can be written to standard output. This is the default, so our first example would also work like this:

```
muscle < seqs.fa > seqs.afa
```

3 File Formats

MUSCLE uses FASTA format for both input and output. It also offers MSF, a more readable output format, which is selected by the *–msf* option.

3.1 Input files

Input files must be in FASTA format. These are plain text files (word processing files such as Word documents are not understood!). Unix, Windows and DOS text files are supported (end-of-line may be NL or CR NL). There is a maximum length of 16,000 characters per line in the current version (this limit is subject to change, and hopefully elimination, in future versions). There is no explicit limit on the length of a sequence, however if you are running a 32-bit version of *muscle* then the maximum will be very roughly 10,000 letters due to maximum addressable size of tables required in memory. Each sequence starts with an annotation line, which is recognized by having a greater-than symbol ">" as its first character. There is no limit on the length of an annotation line (other than the input line length limit), and there is no requirement that the annotation be unique. The sequence itself follows on one or more subsequent lines, and is terminated either by the next annotation line or by the end of the file. The standard single-letter amino acid alphabet is used. Upper and lower case is allowed, the case is not significant. The special characters X, B, Z and U are understood. X means "unknown amino acid", B is D or N, Z is E or Q. Nucleotide sequences (DNA and RNA) are not supported. If you give *muscle* a file containing letters AGCTU only, it will assume that they are amino acids, not nucleotides. U is understood to be the 21st amino acid Selenocysteine (three-letter abbreviation Sel; not to be confused with the RNA base Uracil which is represented by U in some alphabets). White space (spaces, tabs and the end-of-line characters CR and NL) is allowed inside sequence data. Dots "." and dashes "–" in sequences are allowed and are discarded unless the input is expected to be aligned (*–refine* option).

3.2 Output files

By default, output is also written in FASTA format. All letters are upper-case and gaps are represented by dashes "–". You can also request output in MSF format, which is more readable than FASTA, by using the *–msf* command-line option. It would be nice if more output formats were supported—please let me know what formats you would find useful.

4 Using MUSCLE

4.1 Command-line options

There are two types of command-line options: value options and flag options. Value options are followed by the value of the given parameter, for example *–in* <*filename*>; flag options just stand for themselves, such as *–msf*. All options are a dash (not two dashes!) followed by a long name; there are no single-letter equivalents. Value options must be separated from their values by white space in the command line. Thus, *muscle* does not follow Unix, Linux or Posix standards, for which we apologize. The order in which options are given is irrelevant unless two options contradict, in which case the right-most option silently wins.

4.2 Log file

You can specify a log file by using *–log* <*filename*> or *–loga* <*filename*>. Using *–log* causes any existing file to be deleted, *–loga* appends to any existing file. A message will be written to the log file when *muscle* starts and stops. Error and warning messages will also be written to the log. If *–verbose* is specified, then more information will be written, including the command line used to invoke *muscle*, the resulting internal parameter settings, and also progress messages. The content and format of verbose log file output is subject to change in future versions.

The use of a log file may seem contrary to Unix conventions for using standard output and standard error. I like these conventions, but never found a fully satisfactory way to use them. I like progress messages (see below), but they mess up a file if you re-direct standard error and there are errors or warning messages too.

I could try to detect whether a standard file handle is a *tty* device or a disk file and change behavior accordingly, but I regard this as too complicated and too hard for the user to understand. On Windows it can be hard to re-direct standard file handles, especially when working in a GUI debugger. Maybe one day I will figure out a better solution (suggestions welcomed).

I highly recommend using *–verbose* and *–log[a]*, especially when running *muscle* in a batch mode. This enables you to verify whether a particular alignment was completed and to review any errors and warning that occurred.

4.3 Progress messages

By default, *muscle* writes progress messages to standard error periodically so that you know it's doing something and get some feedback about the time and memory requirements for the alignment. Here is a typical progress message.

```
00:00:23     25 Mb  Iter   2  87.20%  Build guide tree
```

The fields are as follows.

`00:00:23`	Elapsed time since *muscle* started.
`25 Mb`	Peak memory use in megabytes (i.e., not the current usage, but the maximum amount of memory used since *muscle* started).
`Iter 2`	Iteration currently in progress.
`87.20%`	How much of the current step has been completed (percentage).
`Build...`	A brief description of the current step.

The *–quiet* command-line option disables writing progress messages to standard error. If the *–verbose* command-line option is specified, a progress message will be written to the log file when each iteration completes. So *–quiet* and *–verbose* are not contradictory.

4.4 Running out of memory

The *muscle* code tries to deal gracefully with low-memory conditions by using the following technique. A block of "emergency reserve" memory is allocated when *muscle* starts. If a later request to allocate memory fails, this reserve block is made available, and *muscle* attempts to save the current alignment. With luck, the reserved memory will be enough to allow *muscle* to save the alignment and exit gracefully with an informative error message.

4.5 Troubleshooting

Here is some general advice on what to do if *muscle* fails and you don't understand what happened. The code is designed to fail gracefully with an informative error message when something goes wrong, but there will no doubt be situations I haven't anticipated (not to mention bugs).

Check the MUSCLE web site for updates, bug reports and other relevant information.

> http://www.drive5.com/muscle

Check the input file to make sure it is in valid FASTA format. Try giving it to another sequence analysis program that can accept large FASTA files (e.g., the NCBI *formatdb* utility) to see if you get an informative error message. Try dividing the file into two halves and using each half individually as input. If one half fails and the other does not, repeat until the problem is localized as far as possible.

Use *–log* or *–loga* and *–verbose* and check the log file to see if there are any messages that give you a hint about the problem. Look at the peak memory requirements (reported in progress messages) to see if you may be exceeding the physical or virtual memory capacity of your computer.

If *muscle* crashes without giving an error message, or hangs, then you may need to refer to the source code or use a debugger. A "debug" version, *muscled*, may be provided. This is built from the same source code but with the DEBUG macro defined and without compiler optimizations. This version runs much more slowly (perhaps by a factor of three or more), but does a lot more internal checking and may be able to catch something that is going wrong in the code. The *–core* option specifies that *muscle* should not catch exceptions. When *–core* is specified, an exception may result in a debugger trap or a core dump, depending on the execution environment. The *–nocore* option has the opposite effect. In *muscle*, *–nocore* is the default, *–core* is the default in *muscled*.

4.6 Technical support

I am happy to provide support. But I am busy, and am offering this program at no charge, so I ask you to make a reasonable effort to figure things out for yourself before contacting me.

5 Command Line Reference

Value option	Legal values	Default	Description
`gapopen`	Floating point	[1]	The gap open score. Must be negative.
`in`	Any file name	standard input	Where to find the input sequences.
`log`	File name	None.	Log file name (delete existing file).
`loga`	File name	None.	Log file name (append to existing file).
`out`	File name	standard output	Where to write the alignment.

Flag option	Set by default?	Description
`msf`	no	Write output in MSF format (default is to use FASTA).
`nocore`	no in muscle, yes in muscled.	Catch exceptions and give an error message if possible.
`quiet`	no	Do not display progress messages.
`verbose`	no	Write parameter settings and progress messages to log file.