Lecture1: MUSCLE

Sanjay Tiwari

Introduction

- MUSCLE (**MU**ltiple **S**equence **C**omparison by **L**og **E**xpectation) is a program for creating multiple alignments of amino acid or nucleotide 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.
- Published tests show that MUSCLE can achieve both better average accuracy and better speed than CLUSTALW or T-Coffee, depending on the chosen options.

References

- Edgar, Robert C. (2004), MUSCLE: multiple sequence alignment with high accuracy and high throughput, *Nucleic Acids Research* **32**(5), 1792-97
- Edgar, Robert C (2004), MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, **5**(1):113

Install

- Go to: http://www.drive5.com/muscle/
- Downloads link there.
- Available for Linux, Windows, and MAC
- For windows: will install in directory muscle3.6

Windows

- Open Command Prompt Window (MS-DOS)
- Change directory to muscle3.6 (In MS-DOS, the command is: cd C:\pathname_of_muscle3.6
- Once there, type: muscle, or muscle –help
- You should see a number of options displayed.

Aligning Sequences

- Make a FASTA file containing some sequences.
- For now, 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.
- Now type:

  muscle -in seqs.fa -out seqs.afa

Large Alignments

- If you have a large number of sequences (a few thousand), or they are very long, then the default settings of may be too slow for practical use.
- A good compromise between speed and accuracy is to run just the first two iterations of the algorithm.
- On average, this gives accuracy comparable to T-Coffee and speeds much faster than CLUSTALW.
- This is done by the option *–maxiters 2*, as in the following example:
- muscle -in seqs.fa -out seqs.afa -maxiters 2

Faster Speed

- The *–diags* option enables an optimization for speed by finding common words (6-mers in a compressed amino acid alphabet) between the two sequences as seeds for diagonals. This is related to optimizations in programs such as BLAST and FASTA: you get faster speed, but sometimes lower average accuracy. For large numbers of closely related sequences, this option works very well.
- If you want the fastest possible speed, then the following example shows the applicable options for proteins.
- muscle -in seqs.fa -out seqs.afa -maxiters 1 -diags -sv -distance1 kbit20_3

Faster Speed (cont.)

- For nucleotides, use:
- muscle -in seqs.fa -out seqs.afa -maxiters 1 -diags
- According to the creator of the program, *muscle* with these options is faster than any other multiple sequence alignment program tested. The alignments are not bad, especially when the sequences are closely related. However, this blazing speed comes at the cost of the lowest average accuracy of the options that *muscle* provides.

File Formats

- MUSCLE uses FASTA format for both input and output. For output only, it also offers CLUSTALW, MSF, HTML, Phylip sequential and Phylip interleaved formats. See the following command-line options: *-clw*, *-clwstrict, –msf*, *–html*, *–phys*, *–phyi, -clwout*, *-clwstrictout, –msfout*, *–htmlout*, *–physout* and *–phyiout*.

The logic of MUSCLE

- The first step is to calculate a tree. In CLUSTALW, this is done as follows. Each pair of input sequences is aligned, and used to compute the pair-wise identity of the pair. Identities are converted to a measure of distance. Finally, the distance matrix is converted to a tree using a clustering method (CLUSTALW uses neighbor-joining). If you have 1,000 sequences, there are (1,000 ´ 999)/2 = 499,500 pairs, so aligning every pair can take a while

The logic (cont.)

- MUSCLE uses a much faster, but somewhat more approximate, method to compute distances: it counts the number of short sub-sequences (known as *k*-mers, *k*-tuples or words) that two sequences have in common, without constructing an alignment. This is typically around 3,000 times faster that CLUSTALW's method, but the trees will generally be less accurate. We call this step "*k*-mer clustering".

The logic (cont.)

- The second step is to use the tree to construct what is known as a progressive alignment. At each node of the binary tree, a pair-wise alignment is constructed, progressing from the leaves towards the root. The first alignment will be made from two sequences. Later alignments will be one of the three following types: sequence-sequence, profile-sequence or profile-profile, where "profile" means the multiple alignment of the sequences under a given internal node of the tree. This is very similar to what CLUSTALW does once it has built a tree.

The logic (cont.)

- Now we have a multiple alignment, which has been built very quickly compared with conventional methods, mainly because of the distance calculation using *k*-mers rather than alignments.
- The quality of this alignment is typically pretty good—it will often tie or beat a T-Coffee alignment on tests. However, on average, it can be improved by proceeding through the following steps.

The logic (cont.)

- From the multiple alignment, we can now compute the pair-wise identities of each pair of sequences. This gives us a new distance matrix, from which we estimate a new tree. We compare the old and new trees, and re-align subgroups where needed to produce a progressive multiple alignment from the new tree. If the two trees are identical, there is nothing to do; if there are no subtrees that agree (very unusual), then the whole progressive alignment procedure must be repeated from scratch.

The logic (cont.)

- Typically, the tree is pretty stable near the leaves, but some re-alignments are needed closer to the root.
- This procedure (compute pair-wise identities→estimate new tree→compare trees→re-align) is iterated until the tree stabilizes or until a specified maximum number of iterations has been done.
- This process is called "tree refinement", although it also tends to improve the alignment.

The logic (cont.)

- Next, the tree is kept fixed and the algorithm moves to a new procedure which is designed to improve the multiple alignment.
- The set of sequences is divided into two subsets (i.e., we make a *bipartition* (defined below) on the set of sequences).
- A profile is constructed for each of the two subsets based on the current multiple alignment. These two profiles are then re-aligned to each other using the same pair-wise alignment algorithm as used in the progressive stage.

The logic (cont.)

- If this improves an "objective score" that measures the quality of the alignment, then the new multiple alignment is kept, otherwise it is discarded. By default, the objective score is the classic sum-of-pairs score that takes the (sequence weighted) average of the pair-wise alignment score of every pair of sequences in the alignment.

The logic (cont.)

- *Bipartitions* (*i.e*. division into two groups) are chosen by deleting an edge in the guide tree, each of the two resulting subtrees defines a subset of sequences. This procedure is called "tree dependent refinement".
- One iteration of tree dependent refinement tries bipartitions produced by deleting every edge of the tree in depth order moving from the leaves towards the center of the tree.
- Iterations continue until convergence or up to a specified maximum.