Multiple Sequence Alignment with Dynamic Programming and CLUSTALW

Kevin Lin

Professor Amy Williams

CS4775: Computational Genomics - Final Project

# **Introduction**

The alignment of biological sequences is important because it allows the assessment of similarity between sequences. A given alignment can be used for gene-finding, function prediction, genome sequence assembly, and phylogenetic relationship prediction (1).

The standard algorithm for pairwise sequence alignment is the Needleman-Wunsch algorithm, which uses dynamic programming to infer the insertions, mutations, and identical positions between two sequences. Given a two-dimensional scoring matrix that represents the likelihood ratio of observing the pair of nucleotides from homologous versus non-homologous sequences and given a gap penalty, the Needleman-Wunsch algorithm calculates an alignment of the pair of sequences by recursively considering all the best sub-alignments prior to the current position (2). The result is the global maximum alignment because it has exhaustively considered all prior alignments leading up to the ends of both sequences. Local pairwise alignments can also be found with the Smith-Waterman algorithm using a slightly modified recurrence relation (2).

An important extension to pairwise sequence alignment is the multiple sequence alignment, when the number of sequences is greater than two. Example usage cases of this type of alignment include Felsenstein’s algorithm for reconstructing the ancestral tree of greater than two observations, and the assessment of conservation of certain protein sequences.

In this project, a direct extension to the dynamic programming algorithm is implemented. Next, a more efficient algorithm, CLUSTALW is implemented by heuristically and iteratively applying the pairwise sequence alignment algorithm to pairs of sequences until all sequences are aligned to each other.

# **Algorithms**

## Dynamic Programming

The pairwise sequence alignment recurrence relation for sequences and is:

where is the optimal solution for the alignment of sequences up until indices and , given score matrix and linear gap penalty . The running time for this algorithm will be , where is the length of and is the length of . This is because the dynamic programming matrix is of size , and the algorithm performs constant time calculations at each point in the matrix (2).

The multiple sequence alignment recurrence relation for sequences is:

were is the maximum score of the alignment up until subsequences ending with . The sum of pairs scoring function on a given alignment column is defined as:

where is the same scoring function for pairwise sequence alignment (2). This provides a simple extension from pairwise sequence alignment to multiple sequence alignment by aggregating all the pairwise scores. However, this extension is a simple approximation, and because these scores derive from log-odds of pairwise comparisons, it is only a rough measurement of the score of a column in a multiple alignment (2). The running time for this algorithm is , because the multi-dimensional dynamic programming matrix is now a hypercube of size , and at each point in this matrix, the algorithm must consider the scores from previous sub-alignments.

## CLUSTALW

The obvious flaw in the multidimensional dynamic programming approach is the inefficient and slow run time. Even for small inputs, the run time can get very large. For example, consider the case where the average sequence length , and the number of sequences . According to the big-O notation, this should take on order of operations, and considering a modern computer runs approximately operations per second, the algorithm would likely never terminate in an ordinary lifespan.

CLUSTALW comes from a class of multiple sequence alignment algorithms called progressive alignment algorithms that make use of heuristics to produce an alignment more efficiently. This method will not produce the global best alignment, but should still produce a reasonable alignment much more efficiently. These algorithms iteratively align pairwise sequences until one alignment remains. Some heuristics or decisions to make include the order of pairwise alignment, handling of sequence-alignment and alignment-alignment cases, tree-based versus linear alignment order (2).

The CLUSTALW algorithm is a widely-used algorithm that begins with a basic progressive alignment and implements a few heuristics. It works as follows (1):

1. A distance matrix is calculated by running the Needleman-Wunsch on all pairs of sequences. Distances are calculated by the percent identity excluding gap positions.
2. A guide tree is constructed by the neighbor-join algorithm.
3. Nodes are progressively aligned bottom-up in order of decreasing similarity, accounting for sequence-alignment and alignment cases using averaged sum of pairs calculations.

[example of distance matrix]

[example of guide tree]

[show insertion of gaps at column, and calculation of averaged sum of pairs, example of profile alignment]

## Further Heuristic Improvements

CLUSTALW furthermore uses other heuristics to improve the accuracy of the multiple alignment (1). They are as follows:

1. Sequence weighting. Given two sets of sequences to align, the scores are calculated by the summation of pairwise scores for each position. However, an initial unevenly sampled set of sequences can overweight the scores caused by similar sequences and underweight the scores caused by dissimilar sequences and vice versa (1). Therefore, the sequence weight calculated from the guide tree is applied to each score calculation to account for this.
2. Initial gap penalties. Gap opening penalties are altered based on the weight matrix, similarity of sequences, and the lengths of the sequences according to the formula (1): GOP -> [GOP + log[min(N,M)]] \* (average residue mismatch score) \* (per cent identity scaling factor) where N, M are the lengths of the two sequences. Gap extension penalties are altered based on the lengths of the sequences so that there are not too many gaps in the shorter sequence as such: GEP -> GEP \* [1.0 + log(N/M)].
3. Position-specific gap penalties. The penalty for opening gaps decreases if there are existing gaps at a specific position to make gaps more likely in positions where there are existing gaps (1). Increased opening gap penalties are set when the current position is not a gap, yet there are gaps within 8 positions (1). These forces gaps away from each other. Heuristics involving hydrophilic residues are not implemented because in [paper’s citation], they involve bases not in {A,C,T,G}.
4. Weight matrices. Based upon the distance between two sets of sequences, differing weight matrices are used to calculate scores. These range from strict matrices for similar sequences and soft matrices for dissimilar sequences (1).
5. Divergent sequences. The most divergent sequences have the least similarity to all the other sequences, and thus the insertions and deletions for these sequences should be aligned last (1). This is already implicitly taken care of by the guide tree constructed in the neighbor-join algorithm.

# **Results**

1. efficiency between DP vs progressive

# **Discussion**

# **Code**

# **References**

1. Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic acids research*, *22*(22), 4673-80.
2. Durbin, Richard, et al. *Biological Sequence Analysis: Probabalistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 2013.