**The Implementation of Nth dimensional lattice Needleman-Wunsch Multiple Sequence Alignment (MSA) algorithm and other improvements base on its characteristics of Dynamic Programming (DP)**

Instructor, Professor Sorin Istrail, Jeff Du

**Abstract**. This work mainly summarizes the major idea for the dynamic programming (DP) approach of multiple sequence alignment (MSA), demonstrates the implementation of the original idea plus several improvements that are brought up from the very basic characteristics of DP and exam their effectiveness base on 3 main rules. The DP approach for MSA is not practical for large databases due to the high space/time complexity while calculation, but the heuristic way this algorithm manipulates to produce the result is intuitive enough thus worth studying.

**Key words**. Multiple sequence alignment, dynamic programming, heuristic algorithm

**Introduction**

Ever since the molecular sequences such as DNA and amino acid chain were discovered and described by scientists, the study in molecular biology had started to face the difficulties to produce optimal results given a large-scale database. Multiple Sequence Alignment (MSA) is one of these NP-complete problems. MSA involves finding optimal alignment among sequences number more than 2, which are assumed to have evolutionary relationships. Various series of methods base on completely different concepts were established to deal with MSA, including Needleman-Wunsch Algorithm[[1]](#endnote-1), a classic algorithm based on dynamic programming in computer science that was initially brought up to globally align 2 sequences. By assuming that the principle of optimality can be applied to sequence alignment problem, Needleman-Wunsch Algorithm is able to guarantee an optimal result given a realistically acceptable scoring criterion. In the following, we will go through the implementation of the Needleman-Wunsch algorithm that align N sequences, some optimization base on the fundamental properties of this algorithm. Evaluation data is obtained and analyzed then to give insight about how well the algorithm is optimized.

**The Implementation of Nth Dimensional Needleman-Wunsch Algorithm**

To better understand the idea of Needleman-Wunsch in Nth dimension, some basic concepts are provided below:[[2]](#endnote-2)

The alphabet can be denoted as:

[[3]](#endnote-3)

The target sequence of k is then . noticeable the characters inside this sequence is a subset of the original alphabet:

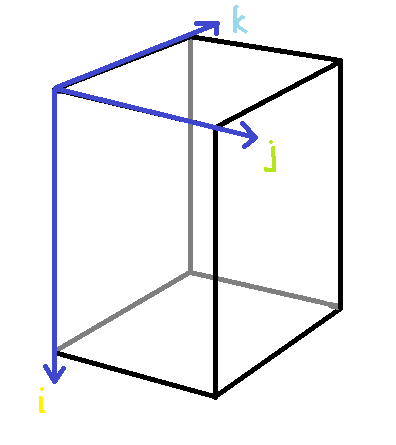
Thus, the alignment for N sequences in the same way can be:

[[4]](#endnote-6)

A path between the given set of N sequences of length will be associated with the N -dimensional Lattice, which consists of all the possible Cartesian products. If we name the corners of the starting point of any alignments from the given sequences original corners, and the corners of the ending point of the alignments end corners, then all the possible alignment results are represented as paths, combinations of neighborhood broken lines obtained inside the lattice from the starting corners to the end corners.[[5]](#endnote-7)

A big assumption is made during the whole computation: Each sub-optimum-finding-step does not affect other steps, but instead they aggregate a general optimum result together.

**The conceptual/implementational workflow of the algorithm**

**Initialization.**

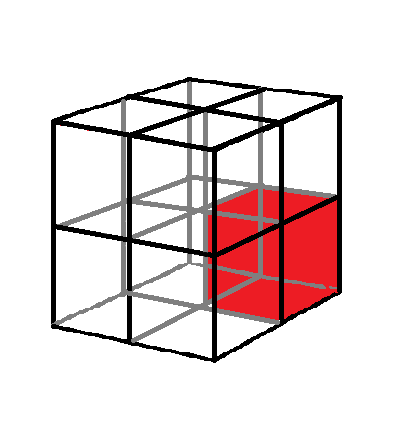
**Fig 1 The initial Lattice**

At this step, we initialize the scoring Lattice. But, as the original Needleman-Wunsch algorithm initialize a two-dimension scoring matrix for two sequences, our algorithm was supposed to initialize an N-dimensional Lattice for N sequences alignment. Each sequence is represented by one axis on the upper edge starting from the original corner. If we take a three-sequence alignment lattice for demonstration, the initialized lattice will be like figure 1. i, j, k represents the three sequences. All the points in this lattice will be given some initial values, including their coordination, their initial score and their matching components (which is the segments from target sequence on each axis at this point).

Notably, the initial scores that are required to start the dynamic programming will be only the scores from the points that are right on the axis, which are similar to those at the original Needleman-Wunsch algorithm.

The flow chart for code implementation is shown below:

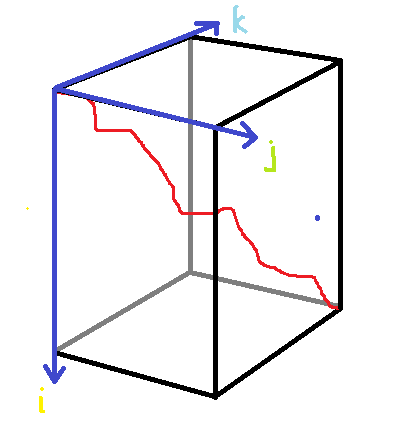
 

**Scoring.**

An N-dimensional pointer will be created at this step to go through all the points at this N-dimensional lattice. The key idea of this particular process is to find the maximum possible score for every position that are not on any axis (these points are already initialized and are fundamental to this scoring process). At each point in the lattice, this algorithm will consider in total 2n-1 directions. In a three-dimension example to find out the maximum score at each point (emphasized by red) inside the lattice, there will be 7 directions lead to seven previous points, shown as Fig 2. If denotes the three sequences that are required to align and denotes the alignment score at point Pi, j, k. The highest score S at point Pi, j, k is obtained from the formula:   
To make sure all the lattice is traversed and scored instead of having wrong score because some calculation involves previous points without initialization/scoring, we will need to traverse the lattice as the coordination increase but not as the index of the array increase.

**Fig 2 The directions for one 3D point (red)**

**Trace-back process.**

**Fig 3 The final path obtained (red)**

After the lattice is traversed, we are ready to trace back. The trace back process involving finding adjacent points backwards in the lattice that matches exactly as the formula given in the scoring process. Using recursion to implement this algorithm can realize a depth first search (DFS) for these possible paths to find all results.



**Improvements base on the characteristics of Dynamic programming**

Dynamic programming is an important approach to many problems. But at the same time, a DP solution to a problem may attain high space complexity, defecting the overall efficiency of the solution. The time/space complexity of DP for N sequences with length k will be , which is certainly a exponential growth towards infinity.

In fact, when using DP to solve NP-complete problems like MSA, the way to reduce the redundancy in space is the key to reach a faster solution. To mitigate the problem, some optimization modification according to the characteristics.[[6]](#endnote-9)

**The Time Complexity Analysis for Dynamic Programming.** DP successfully deals with time complexity mainly because it reduces the unnecessary repetition solution to some certain part of problem by dividing the problem into sub-problems and solve all sub-problems once. But, there is redundancy existing inside DP solutions, which are the solution to meaningless sub-problems. A simple equation gives us an insight about the time/space complexity of this algorithm. We regard the algorithm as a finite automata.

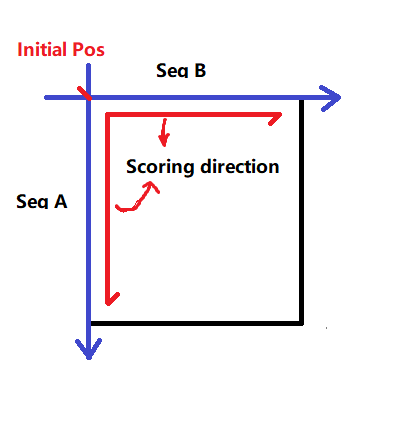
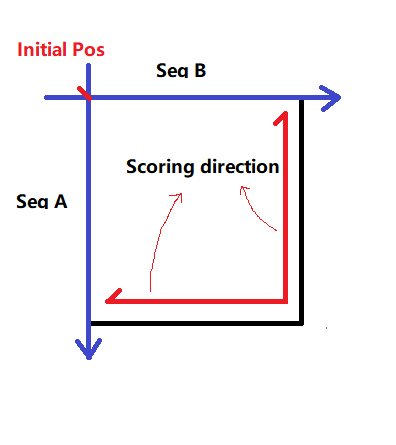
If N stands for the number of the subject and T stands for the time for the subject:

Time/Space Complexity=NStates×NDecisions/States×TState Transition [[7]](#endnote-10)

Its very clear that the optimization for DP is going to focus on these three components of the time complexities. Based on these analysis, improved versions of Needleman-Wunsch algorithm were designed, including Eugene W. Myers and Webb Miller’s method[[8]](#endnote-11) in two-dimension and Carrillo Lipman algorithm in N-dimension. We will majorly talk about how the Carrillo Lipman algorithm reduce time/space complexity in the following aspects. Before that, here are the overview of Carrillo Lipman algorithm.

**Carrillo Lipman Algorithm**

The key idea of Carrillo Lipman algorithm is to intelligently cut useless branches from overall the space-search strategy by setting an upper bound on costs (or a lower bound on scores) and terminate all the unqualified paths, which is a representation of using information from lower dimension to predict information from higher dimension. This algorithm is actually base on another algorithm called Ukkonen algorithm in two-sequence alignment[[9]](#endnote-12).

 Firstly, what does it mean to have a point that is on the optimum solution path for sequence alignment? The related point for two-sequence alignment must be on both of the optimum paths either starting from its front (prefix direction) or starting from its back (suffix direction). Take two-sequence as an example, a point Pij on the optimum path divide the two align sequence into four parts: a (0, i), a (i, n), b (0, j), b (j, n)[[10]](#endnote-13). according to the previously discussed property, any related point should attain optimum in both direction-scoring matrix. A sum matrix C will be created by using both the forward matrix and the backward matrix, and the path of the highest score will be the optimum path in this two-dimension alignment.

**Fig 4 Backward Matrix**

**Fig 5 Forward Matrix**

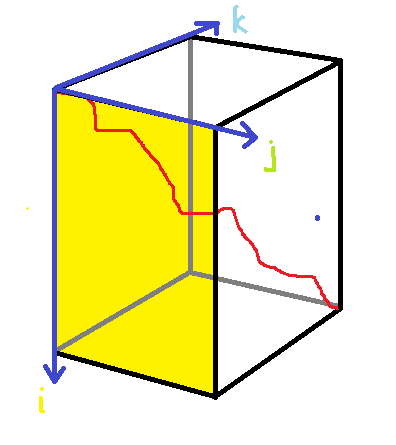
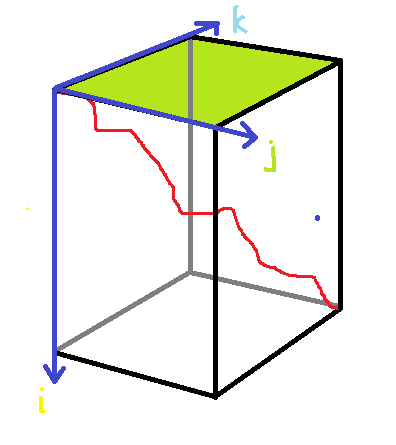
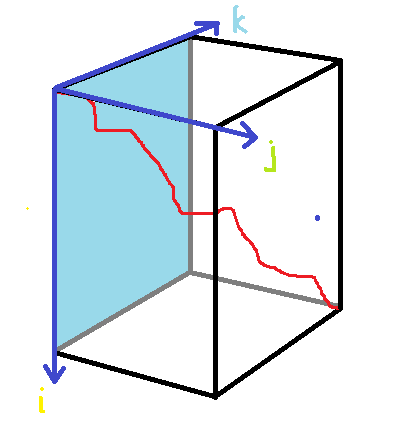
But when we extend the alignment to N-dimension, its unreasonable to say that the result of pair-wise alignment can be directly converted into the result of multiple alignment, but the relationship between the multiple alignment result and the pairwise alignment result is revealed in some way according to a large quantity of mathematical denotations and proves.[[11]](#endnote-14) This relationship can be presented as the following formula:

Let L be a lower bound of score difference between the pairwise alignment and the multiple alignment result at a point ( is the optimum result of N given sequences, i and j indicate the ith and jth sequence)

And

Then from the sum matrix C we can indicate the related/unrelated points and their range. A three-dimension example is shown below:

First, we score every pairwise score and find the sum matrix (represented as the colored side of the lattice).

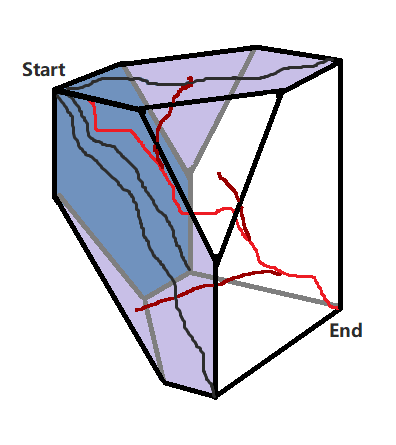
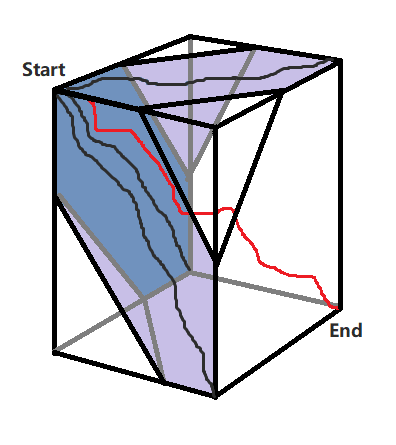
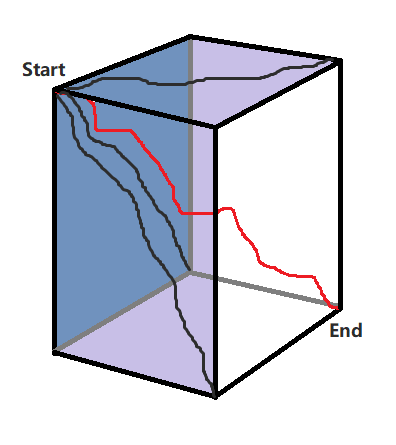


**Fig 6 The j, k side scoring**

**Fig 7 The i, j side scoring**

**Fig 8 The i, k side scoring**

Then, we find out the optimum path of the pairwise alignment and the related range according to L value, shown as Fig 9. And the points with unrelate-labeled index can ignored. The final reduced lattice in three-dimension will be like the third illustration in Fig 9.



**Figure 9 The Optimized Lattice**

**Decrease the amount of unnecessary States.**

The Carrillo Lipman algorithm successfully decrease the overall number for unnecessary points by label the states as related/unrelated and only score the related points.

**Decrease the amount of unnecessary Decisions.**

The Carrillo Lipman algorithm also successfully decrease the overall number for unnecessary decisions that will be made at the edges of the related point-cluster.

**Approximation and Other Heuristic Algorithms**

There are other better ideas in solving MSA in much efficient ways, other heuristic methods based on phylogenetic trees such as T-coffee and CLESTALW, or iterative algorithms such as Hidden Markov Models (HMM) or Annealing algorithm are doing great job finding the approximate optimal solutions.

**Data and Discussions**

Will be continued

**Conclusion**

In the study and reproduction of this classic algorithm for MSA problems, we gain the inspiration of using division and approximation for solving NP-complete problem. Although the time/space complexity of this classic algorithm may be uncompetitive comparing to the other progressive alignment algorithms or iteration algorithms, the Carrillo Lipman algorithm is suitable for most of the case in biological MSA because the input sequences are limited in quantities and mostly believed to be similar in high percentage.

The preliminaries for one algorithm to be meaningful is that:

1. It successfully materialize/quantify the criteria in realistic conditions to some degree and the result of this quantification is acceptable.
2. It optimizes the result according to the materialized criteria.
3. It saves resources while computing the result so that its practical in use.

I believe that the Carrillo Lipman algorithm can satisfy the above requirements and be practical in MSA problem solving.

1. Needleman SB, Wunsch CD. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. [↑](#endnote-ref-1)
2. Humberto Carrillo, David Lipman (1988). The Multiple Sequence Alignment Problem in Biology, 1. Introduction [↑](#endnote-ref-2)
3. H. Carrillo, D. Lipman (1988). The Multiple Sequence Alignment Problem in Biology, 2. Sequences, denotation 1 [↑](#endnote-ref-3)
4. H. Carrillo, D. Lipman (1988). The Multiple Sequence Alignment Problem in Biology, 3. Alignment, formula 3 [↑](#endnote-ref-6)
5. H. Carrillo, D. Lipman (1988). The Multiple Sequence Alignment Problem in Biology, 4. Paths, denotation 1, 2 [↑](#endnote-ref-7)
6. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Introduction to Algorithm, Third Edition, Chp4 advance design and analysis techniques, DP [↑](#endnote-ref-9)
7. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Introduction to Algorithm, Third Edition, Chp4 advance design and analysis techniques, DP [↑](#endnote-ref-10)
8. Eugene W. Myers & Webb Miller (1988). Optimal alignments in linear space [↑](#endnote-ref-11)
9. Esko Ukkonen (1983). On Approximate String Matching [↑](#endnote-ref-12)
10. H. Carrillo, D. Lipman (1988). The Multiple Sequence Alignment Problem in Biology, 7. Estimated Paths [↑](#endnote-ref-13)
11. H. Carrillo, D. Lipman (1988). The Multiple Sequence Alignment Problem in Biology, 5. Optimal Paths, 6. Constraints on multiple alignments, 7. Estimated Paths [↑](#endnote-ref-14)