MULTIPLE SEQUENCE ALIGNMENT: CSL

JOSHUA MURPHY AND JON WALZ
CSCI 5454: ALGORITHMS
CSL: 16 MAR 2011

1. Multiple Sequence Alignments: An Introduction

In its simplest form, the chief goal of Multiple Sequence Alginment (MSA) is to determine the levels of underlying similarity between three or more strings over a shared alphabet and thus to order them in some reasonable manner. MSA is most widely used in the field of Molecular Biology where these strings represent the primary structure of biologically significant macromolecules, specifically segments of Nucleic Acids (i.e. DNA/RNA) and Polypeptides (i.e. proteins).

The alignment of sequences is a fundamental part of almost all work in the field of Bioinformatics. Sequences can be compared and aligned in pairs to determine their relationship to each other, or they can be aligned in multiple sequences to extract more insightful information. The objects of the alignment are usually either proteins or nucleic acids (RNA and DNA). A multiple sequence alignment can be defined as a sequence of three or more of these biological sequences. There is a tremendous amount of information that biologists can extract from a set of aligned sequences.

1.1. Phylogenetic Analysis.

Phylogenetic Trees are used to discover information about evolutionary relationships between different organisms.[18] The Multiple Sequence Alignment (MSA) is the critical input to creating these trees. It is important to note that the quality of the MSA (the ability for it to correctly order sequences based on 'biological significance')[8] determines how useful, if at all, a Phylogenetic tree is to a researcher. Much effort has been put into determining not the mathematical optimum alignment, but the biologically optimum alignment.

1.2. Structure Prediction.

MSAs play a major role in predicting what a residue does in a given protein structure. Secondary structure predictions based on MSAs have a relatively high accuracy of 75% (as apposed to the 60% of predicting based on a single sequence) [18]. Tertiary structure prediction can also help in identifying 'correlated mutations.' [18]

2. Biology Background

First, we need to understand that there is a difference between *biological optimal* and *mathematically optimal* alignments, especially in the case of proteins. To see why, we need to talk about biology again for a second.

One of the main purposes of DNA is to contain instructions for the assembly of proteins. This is done through what is known as the genetic code. In short, any triple of nucleotides can be interpreted as an amino acid. For instance, the triple GGC represents Glycine while AGC represents Serine. Since there are four possible nucleotides (AGCU), you can all see that there are 64 possible codes for a triple. However, the genetic code only contains instructions for 20 amino acids – with a couple of additional triples representing other instructions. This means that there is redundancy, which provides a certain robustness against mutation. This means that we may want to score some mutations less harshly with looking at DNA sequences. It also means that we may want to score some mismatches differently, based on the number of mutations required to earn that mismatch, when looking at proteins.

As we discussed during the lectures on Phylogenetic Trees, mutations are not the only source of discrepancy in evolutionary development. Insertions and deletions are also inevitable. This means that we can characters $A[i]$ and $B[j]$ in our strings without requiring i to equal j. Our only requirements are that each character can be aligned with just one other character and that characters $A[x]$ and $B[y]$ cannot be aligned if they are on opposite sides of another aligned pair. That is, if $A[i]$ aligns with $B[j]$ and $A[x]$ aligns with $B[y]$, then, either $x > i$ and $y > j$ or $x < i$ and $y < j$. How we penalize gaps from insertions and deletions will play an important role in our resulting alignment scores.

Its worthwhile to briefly note that insertions and deletions are largely interchangeable. An insertion in A could be seen as a deletion in B and vice-versa. From here on, we will discuss insertions and deletions as being relative to A. So, if we have strings GATGGCA and GTATGCA, we could have the following alignment:

```
G-ATGGCA
GTATG-CA
```

Where - in the top string represents a prior deletion in A and a - in the lower string represents a prior insertion in A.

3. Multiple Sequence Alignments (MSAs)

Generically, MSA is a two phase algorithm beginning with an input of some set S containing 3 or more strings over some shared alphabet (again, typically either the language of nucleotides or amino acids). We then proceed as follows:

(1) Perform a pairwise alignment on each pair of strings in S and store the results.

(2) Using the results from 1, progressively align the strings in S by similarity.

The progressive alignment for MSAs is, by necessity, a heuristic, as the the optimal MSA alignment has been repeatedly shown to be an NP-Complete problem. [5] One of the most widely used methods of generating MSAs is the use of progressive alignment. This strategy, introduced by Thompson, Higgins and Gibson in 1994, aligns pre-scored pairwise alignments into a master multiple sequence alignment. This is done as follows:

(1) All Sequences are pairwise aligned with each other using either a global or local (and in some cases, both) alignment methods.

(2) A distance matrix is computed between all of the sequences.

(3) A Guide Tree is created using a clustering algorithm (every MSA algorithm has its favorite)

(4) The Guide tree is used in the alignment process to progressively align the sequences

(5) The closest two sequences on the guide tree are aligned first using dynamic programming pairwise alignment. This pair is fixed and any introduced gaps cannot be latter removed.

(6) The next closest two sequences are aligned, or, if the guide tree dictates, the next sequence is added to the existing alignment.

(7) The next two closest sequences (or group of pre-aligned sequences) are always joined.

(8) Continue aligning as above until all sequences are aligned.

One possible vision in pseudo code might look like the following:

```
MSA(array of sequences)
{
   allocate queue Alignment;
   pairwise align all sequences   //(N(N-1)/2 alignments)
   create distance matrix;        // ?
   build guide tree;              // Uses clustering algorithm

   //start alignment
   align = pairwiseAlign(Closest Two candidates on guide tree);
   push (Alignment, align)       //put the first alignment string in the queue

   while(sequences remain in tree)
   {
      if(guide tree indicates add string to alignment)
         push(Alignment, next sequence or pre-aligned pair);
      else
         pre-align next two closest sequences or join pairs;
         update tree to indicate pre-aligned pair;
    }
}
```

4. Dynamic Programming

Dynamic Programming is a programming technique that solves a problem by breaking it down into overlapping subproblems that are slightly smaller than the original. If subproblems are MUCH smaller than the original, we would use the "divide and conquer" method instead. In general, we solve a sub-problem, store the results, and then use those results to solve the next subproblem, and so on.

There are two basic requirements for being able to use dynamic programming on a problem. First, the problem, as stated above, must be sub-dividable into *overlapping* subproblems. Each overlapping subproblem should be *slightlysmaller* than the original, with a large amount of overlap. The power of dynamic program comes from the ability to re-use past results, and not have to re-calculate any portion of the problem. At the expense of a bit of extra storage space, we can gain large advantages in speed over the naive model.

The second requirement for dynamic programming is that the problem must contain an optimal substructure. This means, simply, that there must be an optimal solution to what we are seeking that is a substructure of the original problem. [wikipedia, CLRS]

5. Pairwise Alignment

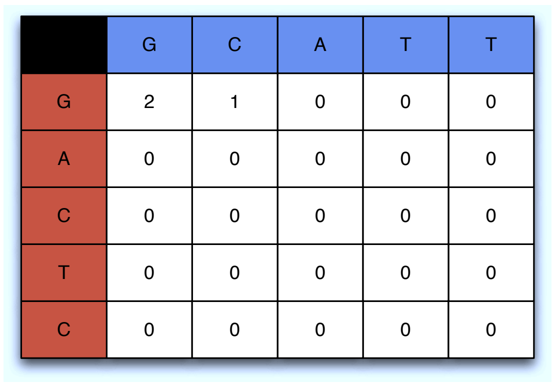
5.1. **Needleman-Wunsch.** The oldest and simplest pairwise alignment algorithm comes from Needleman and Wunsch back in 1969[17]. If you've ever read a paper in which Molecular Biologists attempt to describe an algorithm, though, you may not be surprised to discover that they didn't do a particularly good job of nailing down a particular approach. Instead, they described the basic framework of what has since been adapted into the family of what are called Global (Pairwise) Alignment Algorithms. These are a family of algorithms in which the entire alignment is scored, and penalties are ascribed for any gap (i.e. an insertion or deletion as opposed to a simple mismatch) that is assumed. It is a Dynamic Programming algorithm, which is to say, it operates bottom-up, using the result of previous sub-problems in the solution to subsequent steps. We can represent a generic, Needleman-Wunsch style approach in pseudocode as follows:

```
Needleman-Wunsch( A, B, g, weight() )
   M = new Matrix[|A|][|B|]
      for i in 1:|A|
         for j in 1:|B|
            aligned = ins = del = 0
            if(j > 0 && i > 0)
               aligned = weight(A[i], B[j]) + M[i-1][j-1]
               ins = M[i-1][j] + g
               del = M[i][j-1] + g
            else
               aligned = weight(A[i],B[j])
               if ( i > 0 )
                  del = M[i - 1][j] + g
               elsif( j > 0 )
                  aligned = score(A[i],B[j])
                end
            end
            M[i][j] = max(aligned, ins, del)
         end
      end
   return M[|A|][|B|]
end
```

In plain text, the algorithm works as follows: We begin by receiving two strings, A and B, over some shared alphabet along with their sizes, m and n. We also receive g, the penalty for a gap, some scoring function *weight*(), which takes two characters over the alphabet and returns some weight rating of their similarity. This is virtually always implemented as a lookup table so as to minimize time cost, and, in its simplest form, this would returns a 1 if the characters match and a 0 if they are different. However, as discussed previously, there may be times when, in order to better represent the underlying Biology, we may want a different function.

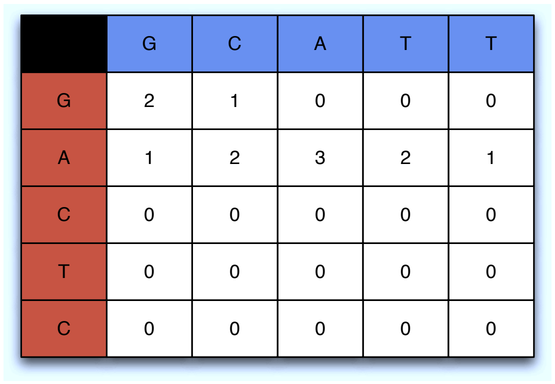
We begin by initializing a Matrix M which will, over time, be filled with the maximum alignment score for pairings such that $A[i], B[j]$ corresponds to $M[i][j]$. We then iterate through M looking for the highest possible score of an alignment at that pairing. We do this in the following manner: Each $M[i]$ is scored as the maximum of one of three possible results. First is the potential for an alignment at that location. This score is recorded as the sum of the scoring of the pair $A[i], B[j]$ and the previous entry in the diagonal. Second is the possibility that the value at $A[i]$ could be an insertion. This is represented by using the value at $M[i-1][j]$ plus whatever penalty we ascribe for a gap. Third is the possibility that there was a deletion at $A[i]$. This is scored by using the value at $M[i][j-1]$ plus our gap penalty.

It ought to be apparent from the pseudo-code that this approach requires $O(nm)$ time and $O(nm)$ space. Later on, we will discuss a variant approach that requires only $O(n+m)$ space. To get a better idea for the behavior of the algorithm and to begin to show its correctness, we will now follow a brief example of its execution over the strings GACTC and GCATT. We will try a scoring algorithm where a match is worth 2, all mismatches are worth 0, and the gap penalty is -1.



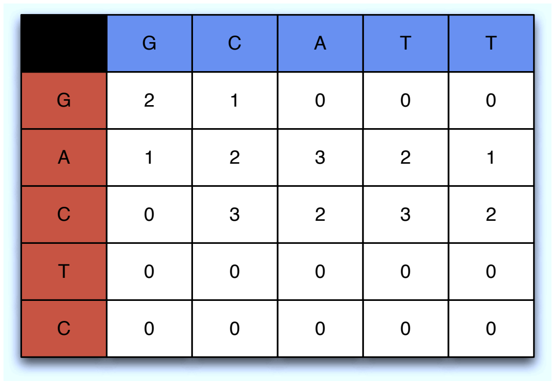
	G	C	A	T	T
G	2	1	0	0	0
A	0	0	0	0	0
C	0	0	0	0	0
T	0	0	0	0	0
C	0	0	0	0	0

In our initial pass, we see that our optimal result is to align $A[1]$ and $B[1]$ – logical, as they match. We also see that we can get some small gain by trying a deletion at $B[2]$.



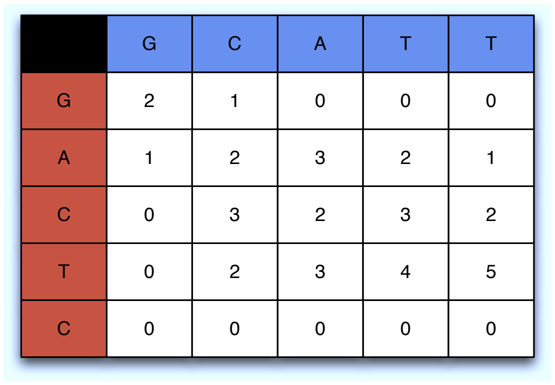
	G	C	A	T	T
G	2	1	0	0	0
A	1	2	3	2	1
C	0	0	0	0	0
T	0	0	0	0	0
C	0	0	0	0	0

In phase 2, it appears as though assuming the deletion may have been the right call, as it allows us to align the As at $A[3]$,$B[2]$ for a positive gain.



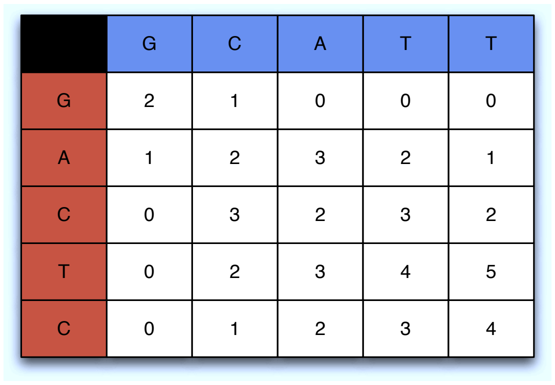
	G	C	A	T	T
G	2	1	0	0	0
A	1	2	3	2	1
C	0	3	2	3	2
T	0	0	0	0	0
C	0	0	0	0	0

On the other hand, as we continue, we can also see that $A[2]$ may have been an insertion, as assuming as much allows us to align the Cs at $A[3]$ and $B[2]$. We also get a sense of how no single alignment is assured by this scoring, as the 2 in $(3, 3)$ could be gained by aligning, inserting, or deleting at that location.



	G	C	A	T	T
G	2	1	0	0	0
A	1	2	3	2	1
C	0	3	2	3	2
T	0	2	3	4	5
C	0	0	0	0	0

Almost done.



	G	C	A	T	T
G	2	1	0	0	0
A	1	2	3	2	1
C	0	3	2	3	2
T	0	2	3	4	5
C	0	1	2	3	4

So we see that the maximum possible alignment here is worth 4 points, but also that there are several possible paths to it. Possible alignments are:

```
GAC-TC | G-ACTC | G-ACTC | GACTC
G-CATT | GCATT- | GCA-TT | GCATT
```

You may notice that the alignment score here is not the maximum score in the alignment. Indeed, the score of 5 found at $A[4], B[5]$ is the highest score. However, global alignment algorithms mandate that we align the entirety of both strings, requiring us to add the final gap in B and incur the resulting penalty. There are many who think that this sort of whole-string alignment is unnecessary, that what is most important is to simply align those segments with high similarity. This is where Local (Pairwise) Alignment Algorithms come into play.

5.2. **Smith-Waterman.** Now, to look at an algorithm that was designed for analyzing the more “locally oriented” cases of DNA and RNA. The major motivation for the development of the local alignment techniques was the inability of global techniques to to obtain a correct alignment of between strings of RNA or DNA that have low levels of similarity between distantly related species. [26] The main issue is that such sequences have a great deal of global “noise” added by mutations over the intervening generations. This noise had made it ineffectual to make a global alignment over these strings. It was necessary to look closer at the strings, and align the regions that do line up. The local alignments avoid noisy regions and focus instead on the local regions that generate positive ”scores”. Another motivation for the use of local alignments is that the alignments tend agree with statistical expectations for evolutionary history of a genome. The Smith-Waterman algorithm is based on the Needleman-Wunsch dynamic programming algorithm, but makes a few changes to “expose” these local similarities. Because of this algorithms run time ($O(n^2)$), and its space requirements (also $O(n^2)$), it is not used a great deal anymore, and faster, more space conscious heuristics are used instead. However, this makes a good introduction to local pairwise alignment.

The basic Smith-Waterman algorithm looks like the following. [26]

```
S-W(A, B)
{
   Allocate matrix [|A|] [|B|]

   //zero out the insertion/deletion rows
   For x, 0 to |A|
      matrix[0,x] = 0

   For x, 0 to |B|
         matrix[x,0] = 0

   //generate scores
   For x, 1 to |A| {
      For y, 1 to |B| {
         matrix[x,y] = compare(matrix, x, y, A[x], B[y])
   }}
  AlignPath(matrix);
  PRINT A;     //Show the alignment
  PRINT B;     //Show the alignment
}
```

First, we allocate a square array to the cardinality of the input strings. We then must zero out the 0 spot on the both axis (this is the insertion/deletion area). After this, we need to generate the score matrix (Dynamic programming), giving us the ability to find an alignment path. Finally, we need to calculate the alignment path. Thats it! The only detail missing is HOW we score the matrix. We do this with the $compare(matrix,\ x,\ y,\ A,\ B)$ function.

In the Smith-Waterman algorithm, we do not have gap penalties. This absence of gap penalties gives us the ability to align locally. This Dynamic programming algorithm will find the OPTIMAL local alignment of two strings (of the same length). We create the score matrix with the following rules: [26]

- $H(i,j)$ is computed by taking of the max of the following: $(1 \leq i \leq m, 1 \leq j \leq n)$

$$\begin{array}{ll} 0 & \\ H(i-1,j-1)+w(a_i,b_j) & \text{(Match/Mismatch)} \\ H(i-1,j)+w(a_i,'-') & \text{(Deletion)} \\ H(i,j-1)+w('-',b_j) & \text{(Insertion)} \end{array}$$

- a, b are strings in the alphabet Σ
- $m = length(a)$
- $n = length(b)$
- $H(i,j)$ is the maximum similarity-score between $a[1...i]$ and $b[1...j]$
- $w(c,d) :: c,d \in \Sigma \cup \{'-'\}$ is the gap scoring scheme

The scoring scheme for S-W is defined as follows:

- $w(match) = +2$
- $w(a,-) = w(-,b) = w(mismatch) = -1$

This function is defined as:

```
compare(matrix, x, y, A, B)
{
   if(A == B)
      return matrix[x-1,y-1] + w(match)
   else {
      returnValue = 0;
      if((matrix[x-1, y-1] + w(mismatch)) > returnValue
         returnValue = matrix[x-1, y-1] + w(mismatch);

      if((matrix[x,y-1] + w(a,-)) > returnValue)
         returnValue = matrix[x,y-1] + w(a,-);

      if((matrix[x-1,y] + w(-,b)) > returnValue)
         returnValue = matrix[x-1,y] + w(-,b);

      return returnValue;
}}
```

The final step to the alignment is to actually align the strings based on there scores. The $alignPath(matrix, A, B)$ algorithm below does just that.

```
alignPath(matrix, A, B, AO, BO)
{
   AO = blank output string;
   BO = blank output string;
   struct current = {x,y};
   current = max(matrix);

   while(current != {0,0} && matrix(current) != 0)
   {
      if A[current(x)] == B[current(y)]
      {
         mode = match;
         newPair = (current(x)-1,current(y)-1)
      }
      else
      {
         temp = matrix[current(x)-1, current(y)-1]
         mode = mismatch;
         newPair = (current(x)-1,current(y)-1)

         if(matrix[current(x)-1, current(y)] > matrix[newPair])
         {
            mode = insertion;
            newPair = {current(x)-1, current(y)};
         }

         if(matrix[current(x), current(y)-1] > matrix[newPair])
         {
            mode = deletion;
            newPair = {current(x), current(y)-1};
         }

         switch (mode)
         {
         case match:
             Add A[current(x)] to AO
             Add B[current(y)] to BO
             break;

         case mismatch:
             Add[current(x)] to AO
             Add[curent(y)] to BO
             break;
```

```
        case insertion:
            Add '-' to BO
            break;

        case deletion:
   Add '-' to AO
   break;
  }
  current = newPair;
}}
```

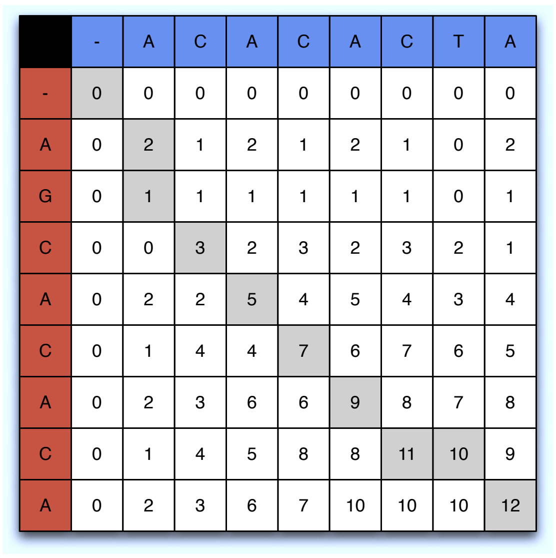
After the *alignPath* algorithm has run, the sequences are now aligned. The following example demonstrates the smith-waterson algorithm.

5.2.1. *smith-waterman example.* [26]

Sequence 1: $ACACACTA$
Sequence 2: $AGCACACA$

Score Function:

- $w(match) = +2$
- $w(a, -)\{insertion\} = w(-, b)\{deletion\} = w(mismatch) = -1$



	-	A	C	A	C	A	C	T	A
-	0	0	0	0	0	0	0	0	0
A	0	2	1	2	1	2	1	0	2
G	0	1	1	1	1	1	1	0	1
C	0	0	3	2	3	2	3	2	1
A	0	2	2	5	4	5	4	3	4
C	0	1	4	4	7	6	7	6	5
A	0	2	3	6	6	9	8	7	8
C	0	1	4	5	8	8	11	10	9
A	0	2	3	6	7	10	10	10	12

FIGURE 1. Example Score Matrix: $\{7{,}7\} \rightsquigarrow \{6{,}7\}$ is an *insertion*, and $\{1{,}2\} \rightsquigarrow \{1{,}1\}$ is a *deleteion*. Remember that the *pathAlign* algorithm runs backwards up and to the left through the matrix.

To create this score matrix, you need to iterate through all of the cells and calculate each of them via the S-W score function. Once you score the entire matrix, you then need to find the $matrix(max)$ value, and then start matching backward from there using the *alignPath* algorithm. Once the *alignPath* algorithm has finished, you have two locally aligned strings.

The final aligned strings are:

```
AO = A-CACACTA
BO = AGCACAC-A
```

To prove the runtime complexity of this algorithm, we must first note that it is dominated by the nested loop over the matrix. One loop is $O(|A|)$ and the other loop is $O(|B|)$, giving us a dominated run-time of $O(|A||B|)$. This far dominates the $O(max(|A|, |B|))$ of the path algorithm and the 0(1) of the compare algorithm. The actually total runtime would be $O(|A||B| + max(|A|, |B|))$.

As with all biological application algorithms, correctness is in the eyes of the beholder. The only way to formally prove that this algorithm is correct is to ask the question: “Does it match up to what we observe in nature?” The answer is “Pretty Close.”

6. Current State

6.1. **Global Alignments.**

6.1.1. *GLASS.* (2000) [24]
See section on addition readings for more information on this topic.

6.1.2. *AVID.* (2003)[24]
See section on addition readings for more information on this topic.

6.1.3. *LAGAN.* (2003)[24]
See section on addition readings for more information on this topic.

6.2. **Local Alignments.** [6, 13, 24, 16]

In the past couple of decades and enormous amount of biological information has come out of the large number of genome projects under way in the world. With these new data came a need for a new way of sequencing patterns. With such a large amount of data, the speed of the early local alignments was becoming a serious issue. It became far more important to the biologists to be able to align large number of sequences than it was to have absolutely perfect alignments. To this end, many local alignment heuristic algorithms have come about. There have been many entries into the arena: Here are some of the most recent.

6.2.1. *FASTA.* Uses seeds and look-up table for indexing the first sequence. (1997) [24]

FASTA first coarsely aligns the sequences to find regions of similarity, then uses the Smith-Waterman algorithm to fine-tune the alignment within the area of similarity.

- Starts by identifying “perfect match” seeds in both sequences using a lookup table
- Uses a diagnal analysis method to find seeds along a diagonal between the above two sequences
- All seeds are scored positive, intermediate regions are scored negative
- Seeds with high scores contribute more to alignment. FASTA saves top 10 best seeds.
- All good diagonals are (score above a threshold) are combined into single high-score alignment by creating directed weighted graph
- Maximum weighted graph is the best alignment.
- computes optimal local alignment in the identified band using Smith-Waterman

6.2.2. *BLAST.* (Basic Local Alignment Search Tool) (1990) [24]

BLAST uses lookup table to identify seeds, then chooses seeds that are above a certain threshold for Proteins, or seeds of exact match for DNA alignment. Default seed length is 3 for Protein, 11 for DNA. You can find good information in the “additional readings” section.

6.2.3. *BLAT.* (Blast Like Alignment Tool) (1991) [24]

See section on addition readings for more information on this topic.

6.2.4. *PatternHunter.* (2002) [24]

See section on addition readings for more information on this topic.

6.2.5. *BLASTZ.* Fastest algorithm in the BLAST series. (2003) [24]

See section on addition readings for more information on this topic.

6.2.6. *MASSA.* (Multiple Anchor Staged Alignment Algorithm) (2008) [24]

See section on addition readings for more information on this topic.

6.3. **GLOCAL Alignments.**]

As biologists attempt to sequence entire genomes, it has been necessary to find a hybrid method that can combine a global alignment with a local alignment. The glocal alignment was created in 2003 by Brudno, et. al. to handle this type of alignment activity. The official line is "a combination of global and local methods, where one creates a map that transforms one sequence into the other while allowing for rearrangement events."[14] This algorithm was developed in cooperation between the Stanford Dept. of Computer Science and the Lawrence Berkeley National Lab.

7. linear space algorithms

Looking at our algorithms for both global and local alignment, we can see that, at any given step, we never look farther back in M than at the previous row and/or column. This means that we can actually perform our scoring in $O(n+m)$ space as follows:

Instead of allocating an entire $|A|x|B|$ matrix M, we allocate 2 arrays of length $|B|$, one, C, to track the current row and one, P, to track the previous row. When we need to check for an insertion, instead of looking at $M[i-1][j]$, we look at $P[j]$. To check the diagonal, we look at $P[j-1]$, and to check for an insertion, we look at $C[j-1]$. At the end of each iteration of the inner loop, we copy C to P and zero out C again.

This, doesnt affect the asymptotic running time of the algorithm, and saves us space that becomes more valuable when dealing with the sorts of strings we get when analyzing DNA and protein segments. Unfortunately, it limits us to retrieving the score of an alignment and reduces the feasibility of reconstructing an actual alignment of the strings.

8. Future

In recent history, much work has been done with sequence alignments in an effort to parallelize the process. With the advent of GPU and Cell processor technology, scientists are starting to make inroads into this area of optimization. Historically the problem has been seen as "inherently sequential", but with the motivation of large amounts of data to process, new heuristics are being developed to make the problem much less sequential.

9. Additional Readings

The References section of these notes list some great resources for finding additional information on Sequence alignments and Multiple Sequence alignments. Most of these paper were referenced during the research for this presentation.

10. References

[1] S.F. Altschul and D.J. Lipman. Trees, stars, and multiple biological sequence alignment. SIAM Journal on Applied Mathematics, 49(1):197209, 02 1989.

[2] D.J. Bacon and W.F. Anderson. Multiple sequence alignment. Journal of Molecular Biology, 191(2):153161, 9 1986.

[3] V.Bafna, E.L.Lawler, and P.A. Pevzner. Approximation algorithms for multiple sequence alignment. Theoretical Computer Science, 182(1-2):233244, 8 1997.

[4] Y.Bilu, P.K. Agarwal, and R.Kolodny. Faster algorithms for optimal multiple sequence alignment based on pairwise comparisons. IEEE/ACM Trans. Comput. Biol. Bioinformatics 1545-5963, 3(4):408422, 2006.

[5] P.Bonizzoni and G.D. Vedova. The complexity of multiple sequence alignment with sp-score that is a metric. Theor. Comput. Sci. 0304-3975, 259(1-2):6379, 2001.

[6] R.Bundschuh. An anylitic approach to significance assessment in local sequence alignment with gaps. RECOMB 2000, 2000.

[7] J.H. CedricNotredame, Desmond G.Higgins. T-coffee: A novel method for fast and accurate multiple sequence alignment. Journal of Molecular Biology, 2000.

[8] S.Chan, A.Wong, and D.Chiu. A survey of multiple sequence comparison methods. Bulletin of Mathematical Biology, 54(4):563598, 1992-07-01.

[9] R.C. Edgar and S.Batzoglou. Multiple sequence alignment. Current Opinion in Structural Biology, 16(3):368373, 6 2006.

[10] D.-F. Feng and R.Doolittle. Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. Journal of Molecular Evolution, 25:351360, 1987. 10.1007/BF02603120.

[11] D.Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. Bulletin of Mathematical Biology, 55(1):141154, 1993-01-01.

[12] D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. Commun. ACM, 18(6):341343, 1975.

[13] D.G.H. Iain M.Wallace, OrlaOSullivan. Evaluation of iterative alignment algorothims for multiple alignment. Bioinformatics, 21(8):14081414, 2005.

[14] O.P. Julie D.Thompson, FredericPlewniak. A comprehensive comparison of mulitple sequence alignment programs. Nucleic Acids Research, 27(13):26822690, 1999.

[15] T.J.G. Julie D.Thompson, Desmond G.Higgins. 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):46734680, 1994.

[16] National Academy of Science. Multiple DNA and protein sequence alignment based on segment-to-segment comparison, volume93. Applied Mathematics, October 1996.

[17] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology, 48(3):443453, 3 1970.

[18] C.Notredame. Recent progress in multiple sequence alignment: a survey. Pharmacogenomics, 2002.

[19] A.Prakash and M.Tompa. Assessing the discordance of multiple sequence alignments. IEEE/ACM Trans. Comput. Biol. Bioinformatics 1545-5963, 6(4):542551, 2009.

[20] T.Riaz, Y.Wang, and K.-B. Li. Multiple sequence alignment using tabu search, 2004.

[21] I.Rutter and A.Wolff. Computing large matchings fast. ACM Trans. Algorithms 1549-6325, 7(1):121, 2010.

[22] G.S. Sadasivam and G.Baktavatchalam. A novel approach to multiple sequence alignment using hadoop data grids, 2010.

[23] J.Taheri and A.Y. Zomaya. A location based approach for solving the multiple sequence alignment problem. Int. J. Bioinformatics Res. Appl., 6(1):3757, 2010.

[24] B.R. WaqarHaque, AlexAravind. Pairwise sequence alignment algorthims - a survey. ISTA 09, 2009.

[25] Wikipedia. Multiple sequence alignment.

[26] Wikipedia. Smith-waterman algorithm.