Available online at http://www.journalcra.com

International Journal of Current Research
Vol. 6, Issue, 11, pp.10004-10008, November, 2014

INTERNATIONAL JOURNAL OF CURRENT RESEARCH

ISSN: 0975-833X

REVIEW ARTICLE

DNA SEQUENCE ALIGNMENT USING PROGRAMME BY ALGORITHM

Sujana, G. and *Harinatha Reddy, A.

Jawaharlal Nehru Technological University Anantapur, Anantapur, A.P. India

ARTICLE INFO

Article History:
Received 17th August, 2014
Received in revised form
06th September, 2014
Accepted 21st October, 2014
Published online 30th November, 2014

Key words:

Dynamic Programming,
DNA Sequence alignment.

ABSTRACT

The code has two classes, the first one named Dynamic Programming.cs and the second named Cell. cs. I will discuss the details of Dynamic Programming.cs class in the following lines because it describes the main idea of my article. The first class contains three methods that describe the steps of dynamic programming algorithm. The first method is named Intialization_Step, this method prepares the matrix a(i,j) that holds the similarity between arbitrary prefixes of the two sequences. The algorithm starts with shorter prefixes and uses previously computed results to solve the problem for larger prefixes. The second method named Get_Maxcomputes the value of the cell (j,i) by the Equation 1. The third method is named Traceback_Step. This method will produce the alignment by traversing the cell matrix (N-1,M-1) back towards the initial entry of the cell matrix (1,1).

INTRODUCTION

The technique of dynamic programming can be applied to produce global alignments via the Needleman-Wunsch algorithm, and local alignments via the Smith-Waterman algorithm. In typical usage, protein alignments use a substitution matrix to assign scores to amino-acid matches or mismatches, and a gap penalty for matching an amino acid in one sequence to a gap in the other. DNA and RNA alignments may use a scoring matrix, but in practice often simply assign a positive match score, a negative mismatch score, and a negative gap penalty. (In standard dynamic programming, the score of each amino acid position is independent of the identity of its neighbors, and therefore base stacking effects are not taken into account. However, it is possible to account for such effects by modifying the algorithm). A common extension to standard linear gap costs, is the usage of two different gap penalties for opening a gap and for extending a gap. Typically the former is much larger than the latter, e.g. -10 for gap open and -2 for gap extension. Thus, the number of gaps in an alignment is usually reduced and residues and gaps are kept together, which typically makes more biological sense. The Gotoh algorithm implements affine gap costs by using three matrices.

Dynamic programming can be useful in aligning nucleotide to protein sequences, a task complicated by the need to take into account frame shift mutations (usually insertions or deletions). The frame search method produces a series of global or local pair wise alignments between a query nucleotide sequence and a search set of protein sequences, or vice versa. Its ability to evaluate frame shifts offset by an arbitrary number of nucleotides makes the method useful for sequences containing large numbers of indels, which can be very difficult to align with more efficient heuristic methods. In practice, the method requires large amounts of computing power or a system whose architecture is specialized for dynamic programming. The BLAST and EMBOSS suites provide basic tools for creating translated alignments (though some of these approaches take advantage of side-effects of sequence searching capabilities of the tools). More general methods are available from both commercial sources, such as *Frame Search*, distributed as part of the Accelrys GCG package, and Open Source software such as Genewise.

The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function; however, identifying a good scoring function is often an empirical rather than a theoretical matter. Although dynamic programming is extensible to more than two sequences, it is prohibitively slow for large numbers of or extremely long sequences (Sniedovich, 2010)

Sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences (http://www.cs.berkeley. edu/~vazirani/algorithms.html). Aligned sequences of nucleotide or amino acid residues are typically represented as rows within a matrix. Gaps are inserted between the residues so that identical or similar characters are aligned in successive columns.

**Corresponding author: Harinatha Reddy, A.*
Jawaharlal Nehru Technological University Anantapur, Anantapur, A.P. India.

Interpretation

If two sequences in an alignment share a common ancestor, mismatches can be interpreted as point mutations and gaps as indels (that is, insertion or deletion mutations) introduced in one or both lineages in the time since they diverged from one another. In sequence alignments of proteins, the degree of similarity between amino acids occupying a particular position in the sequence can be interpreted as a rough measure of how conserved a particular region or sequence motif is among lineages. The absence of substitutions, or the presence of only very conservative substitutions (that is, the substitution of amino acids whose side chains have similar biochemical properties) in a particular region of the sequence, suggest that this region has structural or functional importance. Although DNA and RNA nucleotide bases are more similar to each other than are amino acids, the conservation of base pairs can indicate a similar functional or structural role (Eddy, 2004).

Alignment methods

Very short or very similar sequences can be aligned by hand. However, most interesting problems require the alignment of lengthy, highly variable or extremely numerous sequences that cannot be aligned solely by human effort. Instead, human knowledge is applied in constructing algorithms to produce high-quality sequence alignments, and occasionally in adjusting the final results to reflect patterns that are difficult to represent algorithmically (especially in the case of nucleotide sequences). Computational approaches to sequence alignment generally fall into two categories: *global alignments* and *local alignments*. Calculating a global alignment is a form of global optimization that "forces" the alignment to span the entire length of all query sequences. By contrast, local alignments identify regions of similarity within long sequences that are often widely divergent overall. Local alignments are often preferable, but can be more difficult to calculate because of the additional challenge of identifying the regions of similarity. A variety of computational algorithms have been applied to the sequence alignment problem. These include slow but formally correct methods like dynamic programming. These also include efficient, heuristic algorithms or probabilistic methods designed for large-scale database search, that do not guarantee to find best matches (Nocedal and Wright, 2006).

Representations

Alignments are commonly represented both graphically and in text format. In almost all sequence alignment representations, sequences are written in rows arranged so that aligned residues appear in successive columns. In text formats, aligned columns containing identical or similar characters are indicated with a system of conservation symbols. As in the image above, an asterisk or pipe symbol is used to show identity between two columns; other less common symbols include a colon for conservative substitutions and a period for semi conservative substitutions. Many sequence visualization programs also use color to display information about the properties of the individual sequence elements; in DNA and RNA sequences, this equates to assigning each nucleotide its own color. In protein alignments, such as the one in the image above, color is often used to indicate amino acid properties to aid in judging the conservation of a given amino acid substitution. For multiple sequences the last row in each column is often the consensus sequence determined by the alignment; the consensus sequence is also often represented in graphical format with a sequence logo in which the size of each nucleotide or amino acid letter corresponds to its degree of conservation (Cormen *et al.,* 2001).

Sequence alignments can be stored in a wide variety of text-based file formats, many of which were originally developed in conjunction with a specific alignment program or implementation. Most web-based tools allow a limited number of input and output formats, such as FASTA format and GenBank format and the output is not easily editable. Several conversion programs that provide graphical and/or command line interfaces are available, such as READSEQ and EMBOSS. There are also several programming packages which provide this conversion functionality, such as BioPerl and BioRuby.

Global and local alignments

```
Global FTFTALILLAVAV
       F--TAL-LLA-AV

Local  FTFTALILL-AVAV
       --FTAL-LLAAV--
```

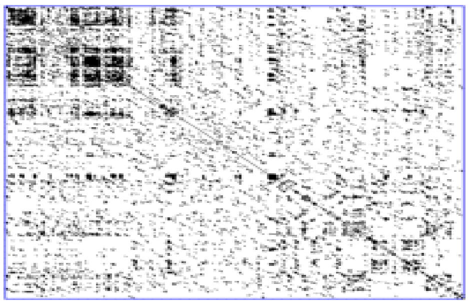
Illustration of global and local alignments demonstrating the 'gappy' quality of global alignments that can occur if sequences are insufficiently similar

Global alignments, which attempt to align every residue in every sequence, are most useful when the sequences in the query set are similar and of roughly equal size. (This does not mean global alignments cannot end in gaps.) A general global alignment technique is the Needleman–Wunsch algorithm, which is based on dynamic programming. Local alignments are more useful for dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context. The Smith–Waterman algorithm is a general local alignment method also based on dynamic programming. Hybrid methods, known as semiglobal or "glocal" (short for global-local) methods, attempt to find the best possible alignment that includes the start and end of one or the other sequence. This can be especially useful when the downstream part of one sequence overlaps with the upstream part of the other sequence. In this case, neither global nor local alignment is entirely appropriate: a global alignment would attempt to force the alignment to extend beyond the region of overlap, while a local alignment might not fully cover the region of overlap (Sniedovich, 2006).

Pairwise alignment

Pairwise sequence alignment methods are used to find the best-matching piecewise (local) or global alignments of two query sequences. Pairwise alignments can only be used between two sequences at a time, but they are efficient to calculate and are often used for methods that do not require extreme precision (such as searching a database for sequences with high similarity to a query). The three primary methods of producing pairwise alignments are dot-matrix methods, dynamic programming, and word methods;[1] however, multiple sequence alignment techniques can also align pairs of sequences. Although each method has its individual strengths and weaknesses, all three pairwise methods have difficulty with highly repetitive sequences of low information content - especially where the number of repetitions differ in the two sequences to be aligned. One way of quantifying the utility of a given pairwise alignment is the 'maximum unique match' (MUM), or the longest subsequence that occurs in both query sequence. Longer MUM sequences typically reflect closer relatedness (Denardo, 2003).

Dot-matrix methods



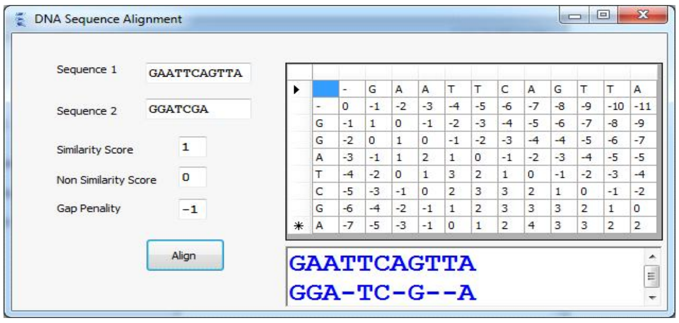
A DNA dot plot of a humanzinc fingertranscription factor (GenBank ID NM_002383), showing regional self-similarity. The main diagonal represents the sequence's alignment with itself; lines off the main diagonal represent similar or repetitive patterns within the sequence. This is a typical example of a recurrence plot.

The dot-matrix approach, which implicitly produces a family of alignments for individual sequence regions, is qualitative and conceptually simple, though time-consuming to analyze on a large scale. In the absence of noise, it can be easy to visually identify certain sequence features—such as insertions, deletions, repeats, or inverted repeats—from a dot-matrix plot. To construct a dot-matrix plot, the two sequences are written along the top row and leftmost column of a two-dimensional matrix and a dot is placed at any point where the characters in the appropriate columns match—this is a typical recurrence plot. Some implementations vary the size or intensity of the dot depending on the degree of similarity of the two characters, to accommodate conservative substitutions. The dot plots of very closely related sequences will appear as a single line along the matrix's main diagonal. Problems with dot plots as an information display technique include: noise, lack of clarity, non-intuitiveness, difficulty extracting match summary statistics and match positions on the two sequences. There is also much wasted space where the match data is inherently duplicated across the diagonal and most of the actual area of the plot is taken up by either empty space or noise, and, finally, dot-plots are limited to two sequences. None of these limitations apply to Miropeats alignment diagrams but they have their own particular flaws. Dot plots can also be used to assess repetitiveness in a single sequence. A sequence can be plotted against itself and regions that share significant similarities will appear as lines off the main diagonal. This effect can occur when a protein consists of multiple similar structural domains.

MATERIALS AND METHODS

Sequence alignment is the procedure of comparing two (pair-wise alignment) or more (multiple alignment) sequences by searching for a series of characters that are in the same order in all sequences. Two sequences can be aligned by writing them across a page in two rows. Identical or similar characters are placed in the same column, and non identical ones can either be placed in the same column as a mismatch or against a gap (-) in the other sequence. Sequences that are aligned in this manner are said to be similar. Sequence alignment is useful for discovering functional, structural, and evolutionary information in biological sequences. Consider the following DNA Sequences GACGGATTAG and GATCGGAATAG. Notice that when we align them one above the other:

GA-CGGATTAG
GATCGGAATAG



Sequence 1: GAATTCAGTTA
Sequence 2: GGATCGA
Similarity Score: 1
Non Similarity Score: 0
Gap Penality: -1

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

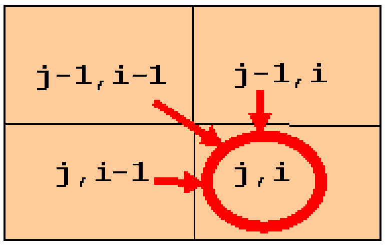
GAATTCAGTTA
GGA-TC-G--A

The only differences are marked with colors in the above sequences. Observe that the gap (-) is introduced in the first sequence to let equal bases align perfectly. the goal of this article is to present an efficient algorithm that takes two sequences and determine the best alignment between them. The total score of the alignment depends on each column of the alignment. If the column has two identical characters, it will receive value +1 (a match). Different characters will give the column value -1 (a mismatch). Finally a gap in a column drops down its value to -2 (Gap Penalty). The best alignment will be one with the maximum total score. The above alignment will give a total score: $9 \times 1 + 1 \times (-1) + 1 \times (-2) = 6$. These parameters match, mismatch and gap penalty can be adjusted to different values according to the choice of sequences or experimental results.

One approach to compute similarity between two sequences is to generate all possible alignments and pick the best one. However, the number of alignments between two sequences is exponential and this will result in a slow algorithm so, Dynamic Programming is used as a technique to produce faster alignment algorithm. Dynamic Programming tries to solve an instance of the problem by using already computed solutions for smaller instances of the same problem. Giving two sequences Seq1 and Seq2 instead of determining the similarity between sequences as a whole, dynamic programming tries to build up the solution by determining all similarities between arbitrary prefixes of the two sequences. The algorithm starts with shorter prefixes and uses previously computed results to solve the problem for larger prefixes.

Let M =size of Seq1 and N= size of Seq2 ,the computation is arranged into an (N+1) × (M+1) array where entry (j,i) contains similarity between Seq2(1.....j) and Seq1(1.....i). The algorithm computes the value for entry(j,i) by looking at just three previous entries:

(j-1,i-1) Diagonal Cell to entry (j,i)
(j-1,i) Above Cell to entry (j,i)
(j,i-1) Left Cell to entry (j,i)
j-1,i
j,i
j-1,i-1
j,i-1



The value of the entry (j,i) can be computed by the following equation:

$$\left\{\begin{matrix} a[j,i-1]+Gap \\ a[j-1,i-1]+p(j,i) \\ a[j-1,i]+Gap \end{matrix}\right\} \qquad \text{Equation (1.1)}$$

where p(j,i)= +1 if Seq2(j)=Seq1(i) (match Score) and p(j,i)= -1 if Seq2(j)!=Seq1(i).

The maximum value of the score of the alignment located in the cell (N-1,M-1) and the +\algorithm will trace back from this cell to the first entry cell (1,1) to produce the resulting alignment . IF the value of the cell (j,i) has been computed using the value of the diagonal cell, the alignment will contain the Seq2(j) and Seq1(i). IF the value has been computed using the above cell, the alignment will contain Seq2 (j) and a Gap ('-') in Seq1(i). IF the value has been computed using the left cell, the alignment will contain Seq1(i) and a Gap ('-') in Seq2(j). The resulting alignment will produce completely by traversing the cell (N-1,M-1) back towards the initial entry of the cell (1,1).

RESULTS AND DISCUSSIONS

Using the Code

My code has two classes, the first one named *Dynamic Programming.cs* and the second named *Cell.cs*. I will discuss the details of *Dynamic Programming.cs* class in the following lines because it describes the main idea of my article. The first class contains three methods that describe the steps of dynamic programming algorithm. The first method is named Intialization_Step, this method prepares the matrix a(i,j) that holds the similarity between arbitrary prefixes of the two sequences. The algorithm starts with shorter prefixes and uses previously computed results to solve the problem for larger prefixes.

```
public static Cell(,) Intialization_Step
            (string Seq1, string Seq2,int Sim,intNonSimilar,int
Gap)
     {
int M = Seq1.Length;//Length+1//-AAA
int N = Seq2.Length;//Length+1//-AAA

Cell(,) Matrix = new Cell(N, M);
        //Intialize the first Row With Gap Penalty Equal To
i*Gap
for (int i = 0; i <Matrix.GetLength(1); i++)
        {
Matrix(0, i) = new Cell(0, i, i*Gap);
        }

        //Intialize the first Column With Gap Penalty Equal To
i*Gap
for (int i = 0; i <Matrix.GetLength(0); i++)
        {
Matrix(i, 0) = new Cell(i, 0, i*Gap);
        }
        // Fill Matrix with each cell has a value result from
method Get_Max
for (int j = 1; j <Matrix.GetLength(0); j++)
        {
for (int i = 1; i <Matrix.GetLength(1); i++)
          {
Matrix(j,   i)   =   Get_Max(i,   j,   Seq1,   Seq2,
Matrix,Sim,NonSimilar,Gap);
          }
        }
return Matrix;
     }
```

The second method named Get_Maxcomputes the value of the cell (j,i) by the Equation 1.1 .

```
public static Cell Get_Max(int i, int j, string Seq1,
           string          Seq2,          Cell(,)
Matrix,intSimilar,intNonSimilar,intGapPenality)
     {
        Cell Temp = new Cell();
intSim;
int Gap = GapPenality;
if (Seq1(i) == Seq2(j))
Sim = Similar;
else
Sim = NonSimilar;
int M1, M2, M3;
 M1 = Matrix(j - 1, i - 1).CellScore + Sim;
 M2 = Matrix(j, i - 1).CellScore + Gap;
 M3 = Matrix(j - 1, i).CellScore + Gap;
int max = M1 >= M2 ? M1 : M2;
intMmax = M3 >= max ? M3 : max;
if (Mmax == M1)
{ Temp = new Cell(j, i, M1, Matrix(j - 1, i - 1),
            Cell.PrevcellType.Diagonal); }
else
        {
if (Mmax == M2)
{ Temp = new Cell(j, i, M2, Matrix(j, i - 1),
Cell.PrevcellType.Left); }
else
        {
if (Mmax == M3)
{ Temp = new Cell(j, i, M3, Matrix(j - 1, i),

            Cell.PrevcellType.Above); }
          }
        }
return Temp;
    }
```

The third method is named Traceback_Step. This method will produce the alignment by traversing the cell matrix(N-1,M-1) back towards the initial entry of the cell matrix (1,1).

```
public static void Traceback_Step(Cell(,) Matrix,
string Sq1, string Sq2, List<char> Seq1, List<char> Seq2)
    {
      //List<char> Seq1 = new List<char>();
      //List<char> Seq2 = new List<char>();
      Cell CurrentCell = Matrix(Sq2.Length - 1, Sq1.Length -
1);

while (CurrentCell.CellPointer != null)
        {
if (CurrentCell.Type == Cell.PrevcellType.Diagonal)
          {
Seq1.Add(Sq1(CurrentCell.CellColumn));
Seq2.Add(Sq2(CurrentCell.CellRow));
          }
if (CurrentCell.Type == Cell.PrevcellType.Left)
          {
Seq1.Add(Sq1(CurrentCell.CellColumn));
Seq2.Add('-');
          }
if (CurrentCell.Type == Cell.PrevcellType.Above)
          {
Seq1.Add('-');
Seq2.Add(Sq2(CurrentCell.CellRow));
          }
CurrentCell = CurrentCell.CellPointer;
        }
    }
```

The second class in my code is named *Cell.cs*. This class manipulates the cell of the matrix. Each cell has:

- A location indicated by the index of the row and index of the column
- A value that is represented by the score of the alignment
- A pointer to a previous cell that is used to compute the score of the current cell (Note: Pointer value "Diagonal, Above and Left").

REFERENCES

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Stein, C. 2001. Introduction to Algorithms (2nd ed.), MIT Press and McGraw–Hill, ISBN 0-262-03293-7. pp. 327–8.

Dasgupta S., C.H. Papadimitriou, and U.V. Vazirani, 'Algorithms', p 173, available at http://www.cs.berkeley.edu/~vazirani/algorithms.html

Denardo, E.V. 2003. *Dynamic Programming: Models and Applications*, Mineola, NY: Dover Publications, ISBN 978-0-486-42810-9

Eddy, S. R. 2004. What is dynamic programming?, *Nature Biotechnology,* 22, 909–910.

Nocedal, J, Wright, S. J. 2006. *Numerical Optimization, page 9, Springer.*

Sniedovich, M. 2006. "Dijkstra's algorithm revisited: the dynamic programming connexion" (PDF), *Journal of Control and Cybernetics,* 35 (3): 599–620. Online version of the paper with interactive computational modules.

Sniedovich, M. 2010. *Dynamic Programming: Foundations and Principles*, Taylor and Francis, ISBN 978-0-8247-4099-3.

****