Proceedings of the 5th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems, Madrid, Spain, February 15-17, 2006 (pp117-122)

A Recursive Algorithm for Global Alignment with Gap Consideration in a pair of Sequences

Muneer Ahmad, Shahbaz Pervez, Iftikhar Ahamed, Adeel Akram, Nadeem Daud Potta.
Comsat Institute of Information Technology,
Abbottabad, NWFP
PAKISTAN

Abstract: **-** The Sequences either Protein or DNA contain precious Biological Information of certain species for finding / knowing various characteristics of living organisms [5]. If a pair of sequence is not properly aligned, we cannot get exact information to study about these species. It is important to find the degree of similarity and no. of gaps in sequences. Sequences not properly aligned with or without consideration of gaps cannot provide a true picture of what information they owe and what characteristics they have. [1, 2]

A recursive approach is being proposed in this paper that would not only find the Global Alignment for a pair of protein / DNA sequence but also provides means for consideration of gaps between them. The algorithm will calculate the degree of similarity and bounds / extends of gaps to bring sophisticated results. The input variables (e.g. Strains) of program are user dependent and internal calculations are being performed in recursive fashion to make the input Strains smart enough. [4]

Key-Words: **-** DNA Strain→ Sequence of Nucleotide Characters (A, T, G, C), DSDR→ Algorithm for Duplicate Sequence Detection and Removal, Applet→ Java program that runs on web.

1. Introduction

Sequence Alignment is a procedure of comparing two or more sequences by searching for a series of characters or a pattern of characters that are in same order in sequences. [4]

There are two types of Sequence Alignment. [2]

i. Local Alignment
ii. Global Alignment

1.1) Local Alignment

In local alignment, the alignment is made at regions of identity and does not include neighboring regions. [2]

e.g.

```
----------ATGTT----------
          |||
----------ACGTT----------
```

1.2) Global Alignment

The Global Alignment contrary to the Local Alignment is stretched over the entire sequence length to find more matches at different regions of Strains. [2]

e.g.

```
----TGC----ATGCT---ATGCCC
    |  |    |  |  |   | |
--AAGC--TAGCT--ATGCAT
```

1.3) The Proposed Technique

The Recursive Technique works for better solutions at Global Alignment but can be modified a little to be used for Local Alignment also [1]. The Internal functionality of the algorithm is competent enough to deal with both kinds of alignments with a small change. Recursion brings fast and accurate results [3] that is why this algorithm can be considered to be at high level of efficiency and reliability.

2) Previous Work

Following techniques are being used for the alignment of two sequences. [1, 2]

i. DOT MATRIX Analysis.
ii. The Dynamic Programming Algorithm.
iii. WORD or k-tupple methods.

2.1) DOT MATRIX Method

The DOT MATRIX method is useful only when sequences are known to be very much alike because it displays any possible sequence alignment as diagonals on the matrix. It may be used for insertion / deletion and direct / inverted repeats of characters of sequences. The Major limitation of this method is that most DOT MATRIX programs don't show an actual alignment. [2]

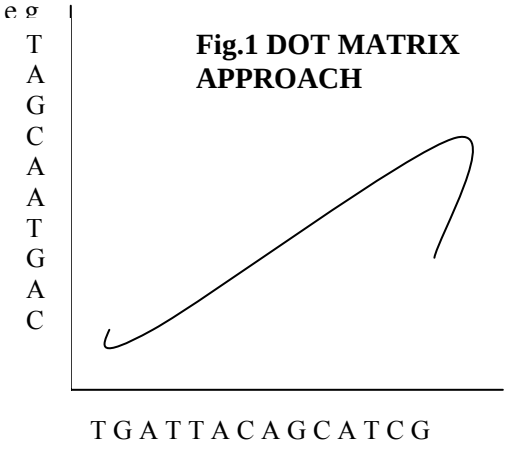

Fig.1 DOT MATRIX APPROACH

2.2) Dynamic Programming Method

The Dynamic Programming Method is mostly used for Global Alignment of sequences devised by Needleman and Wunsch (1970), this method was also used for Local Alignment by Smith and Waterman (1981). The procedure starts by attempting to match all possible pairs of characters [5] between sequences and by following a scoring scheme for matches, mismatches and gaps. Although this method is widely used for both kinds of alignments but it has also a major drawback that it can also be slow due to very large no. of computational steps, which increase approximately as square / cube of sequence lengths. Thus utilization if this method for large sequences is hard. [1]

2.3) WORD or K-Tupple Methods

The WORD or K-Tupple Methods are used by the FASTA and BLAST algorithms [1, 2]. They align two sequences very quickly by first searching for identical parts of sequences and then joining them for alignment purpose by Dynamic Programming Methods. Although these methods are reliable enough in a computational and statistical sense but as they use Dynamic Programming Technique so bring the result accurately but slowly

3) Our Work

The Algorithm works by taking input of two Strains, analysis the two Strains recursively and generates results about degree of similarity, non similarity and usage of GAPS for Global Alignment of Sequences.

-----START PROCEDURE-----

Variables:

GAPE_ONE,GAPE_TWO→ Calculate Gaps in two sequences.

MATCH→Find no. of pairs matched.

MMATCH→Find no. of pairs miss matched.

Block→One Block out of several blocks in sequences.
Gape1, Gape2→Calculate Gaps in individual blocks
match→No. of pairs matched in a Block
mmatch→ No. of pairs miss matched in a Block
N→ No. of characters in Strain
n→ No. of characters in a Block

1) [Outer Iteration for no. of Blocks]
Repeat for Block = n to Block ≤ N with Block = Block +n
2) [Initialize Block Variables]

Set
Gape1 = Gape2 = 0
Match = mmatch = 0
Var y = Block-n

3) [Call Recursive Function]

CALL Recursive ()
Set
MATCH = MATCH + match
GAPE_ONE = GAPE_ONE + Gape1

GAPE_TWO = GAPE_TWO + Gape2
MMATCH = MMATCH + mmatch

4) [Print individual Block Information]

Print "No. of Matches in Block"
Print "No. of Miss Matches in Block"
Print "No. of Gaps used in Sequence1"
Print "No. of Gaps used in Sequence2"

5) [Function Definition Recursive]

IF S_1[y] equals GAP then
Gap1 = Gap1 + 1
END IF
IF S_2[y] equals GAP then
Gap2 = Gap2 + 1
END IF

IF (S_1[y] Not equals GAP OR S_2[y] Not Equals GAP) AND S_1[y] equals S_2[y] then
Match = match +1
END IF

IF S_1[y] Not Equals S_2[y] then
mmatch = mmatch + 1
END IF
y = y + 1

6) [Check Recursion Condition]

IF y < Block
CALL Recursive ()

7) [Overall Result for a pair of Sequences]

Print "No. of Matches "
Print "No. of Mismatches"
Print "No. of Gaps used in Sequence 1"
Print "No. of Gaps used Sequence 2"

------ END PROCEDURE-------

3.1) Functionality of Algorithm

The Algorithm uses a recursive procedure for its operation. It inputs two Strains and divides them into equal length blocks. The individual blocks are made responsible to provide information about matching, miss matching pairs, gaps and percentage match of two Strains. The iterations are performed recursively to move into all blocks. The final information is obtained by integrating the information got from individual blocks.

3.2) Complexity of Algorithm

Complexity of Algorithm is N/n log (n)

Where N→No. of characters in Sequence.
n→No. of characters in individual block.

3.3) Sample Run using C++ (Test 1)

The functionality of the algorithm was tested to be quite satisfactory and brought the desired results, here is the sample C++ code described below.

```
#include<iostream.h>
#include<stdio.h>
#include<string.h>
#include<conio.h>

void rec(void);

int GAP1,GAP2,MATCH,MMATCH;
//capitals for whole sequences

int gap1,gap2,match,mmatch,bl;
char s1[40],s2[40];
int y;


void main()
{
clrscr();

GAP1=GAP2=MATCH=MMATCH=0;

int n=10;       //no. of chars in a block

cout<<"\nEnter 1st sequence ";
gets(s1);
cout<<"\nEnter 2nd sequence ";
gets(s2);
int N=strlen(s1);       //Total no of chars

for(bl=n;bl<=N;bl+=n)
{
gap1=gap2=match=mmatch=0;

y=bl-n;         //initiate from 1st char of block
rec();

MATCH+=match; //global calculation
```

```
 GAP1+=gap1;          //for the whole
 sequences.
 GAP2+=gap2;
 MMATCH+=mmatch;

 cout<<"\nThe no of gaps in seq.1 in block#
 "<<bl/n<< "="<<gap1;
 cout<<"\nThe no of gaps in seq.2 in block#
 "<<bl/n<< "="<<gap2;
 cout<<"\nThe no of MATCHES in block#
 "<<bl/n<< "="<<match;
cout<<"\nThe no of Mismatches in block#
 "<<bl/n<< "="<<mmatch;

cout<<"\nThe %age of matches in block#
 "<<bl/n<<          "="<<float(match*100)
 /n<<"%";

cout<<"\n\nPress any key to proceed\n";
getch();

}       //end for

cout<<"\n\n\n\t\tThe whole sequence is
 finished"
   <<"\n\t\t  Press any key to proceed\n\n";
getch();

cout<<"\nThe final result for whole
 sequences"
   <<"\n-----------------------------------\n\n";

cout<<"\nThe no of gaps in seq.1   "<<
 "="<<GAP1;
cout<<"\nThe no of gaps in seq.2   "<<
 "="<<GAP2;
cout<<"\nThe no of MATCHES         "<<
 "="<<MATCH;
cout<<"\nThe no of Mismatches      "<<
 "="<<MMATCH;
cout<<"\nThe %age of matches       "<<
 "="<<float(MATCH*100) /N<<"%";

getch();
}       //end main

void rec(void)
{

//gaps

   if(s1[y]==' ')
     gap1++;
   if(s2[y]==' ')
     gap2++;

        //no of matches & mismatches

if(s1[y]==s2[y])
  match++;
if(s1[y]!=s2[y])
  mmatch++;
y++;
if(y<bl)
rec();

}
```

3.4) Sample Run using Java (Test 2)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class analysis extends Frame
implements ActionListener
{
int
rem,quo,a,b,t,match,mismatch,gap1,gap2,t9;
float per;
String temp1,temp2;
String[] arr1;
String[] arr2;
Label l1=new Label("Enter first string:
");
TextField t1=new TextField(25);
Label l2=new Label("Enter second string:");
Button b1 =new Button(" Calculate ");
TextField t2=new TextField(25);

public analysis()
{
super("Analysizer");
setLayout(new FlowLayout());
gap1=0;gap2=0;
t=0;
match=0;
mismatch=0;
per=5;
add(l1);
add(t1);
add(l2);
add(t2);
add(b1);

b1.addActionListener(this);
        setSize(400,400);
        setVisible(true);
        addWindowListener
        (
        new WindowAdapter()
```

```
{
public void
windowClosing(WindowEvent e)
{
System.exit(0);
}
}
);
}
void match(char ch1,char ch2)
{
if(ch1!=' '&&ch2!=' ')
{
if(ch1==ch2)
match++;
else
mismatch++;
}
}
//end of function

 //cons

public void
actionPerformed(ActionEvent e)
{

if(e.getSource()==b1)
{

temp1=t1.getText();
temp2=t2.getText();
quo=temp1.length()/10;
rem=temp1.length()%10;
arr1=new String[quo];
arr2=new String[quo];
if(temp2.length()<temp1.length()||tem
p2.length()>temp1.length())
{
JOptionPane.showMessageDialog(nul
l,"Both strings must be of equal
length","Stop",JOptionPane.ERROR_
MESSAGE);
//JOptionPane.showMessageDialog(n
ull,"Length of t1 is
"+temp1.length(),"Stop",JOptionPane.
ERROR_MESSAGE);
}
else
{
//parsing code
for(a=0;a<temp1.length();a++)
{
if(temp1.charAt(a)==' ')
gap1++;
if(temp2.charAt(a)==' ')
gap2++;
}
for(a=0;a<quo;a++)
{
arr1[a]=temp1.substring(t,t+10);
arr2[a]=temp2.substring(t,t+10);
t+=10;
for(b=0;b<10;b++)
match(arr1[a].charAt(b),arr2[a].charA
t(b));
//dialogs
per=match*10;
JOptionPane.showMessageDialog(nul
l,"Number of matches : "+match+"\n
Number of mismatches:
"+mismatch+"\n Number of gapes in
string1  "+gap1+"\n Number of gapes
in string2 "+gap2+"\n Percentage of
matches "+per,"Result of block#
"+(a+1),JOptionPane.INFORMATIO
N_MESSAGE);
match=0;
mismatch=0;
per=0;

}//end of parsing

//JOptionPane.showMessageDialog(n
ull,"rem=
"+rem,"asdas",JOptionPane.INFORM
ATION_MESSAGE);

for(b=0;b<rem;b++)
{
match(temp1.charAt(t+b),temp2.char
At(t+b));

}
JOptionPane.showMessageDialog(nul
l,"Number of matches : "+match+"\n
Number of mismatches:
"+mismatch+"\n Number of gapes in
string1  "+gap1+"\n Number of gapes
in string2 "+gap2+"\n Percentage of
matches "+per,"Result of block#
"+(a+1),JOptionPane.INFORMATIO
N_MESSAGE);

per=2/4;

System.out.println(per);

}//end of else
}
```

```
}
public static void main(String[] args)
{
analysis ana =new analysis();
}
} //end of class
```

4) Conclusion

This recursive algorithm works best for accurate and prompt results. It partitions the sequences into blocks and gets information of matches, mismatches, gaps and percentage matches in blocks. Then integrates the information from individual Blocks to display a final summary about two Strains.
Also it not only finds the Global Alignment for a pair of protein / DNA sequence but also provides means for consideration of gaps between them. The input variables (e.g. Strains) of program are user dependant and internal calculations are being performed in recursive fashion to make the input Strains smart enough.

6) References:

1. Genetics and Genome, Bioinformatics Research and Genetic Algorithms (visit bioinformaticsonline.org)
2. Bioinformatics Sequence and Genome Analysis (http://www.bioinformaticsonline.org)
3. Fast and Accurate Probe Selection Algorithm for Large Genome (Wing-Kin Sung, Wah-Heng Lee) IEEE-2003
4. Statistical Inference for well-ordered Structure in Nucleotide Sequence(Shu-Yun Le, Jih-H. Chen) IEEE-2003
5. SMASHing regulatory sites in DNA by Human-mouse sequence comparisions (Mihaela Zavolan, Nicholas D. Socci, Nikolaus Rajewsky, Terry Gaasterland) IEEE-2003
6. Genotype Discrimination: The complex case for some legislative protection. Henry T. Greely. 149 U. Pa. L. Rev. 1483 (May 2001)
7. Towards Cystic Fibrosis Gene Therapy by John Wagner and Phyllis Gardner, *Annual Review of Medicine* **48**, 203-216 (1997)
8. Rouillard J. M. Herbert C. J. and Zukar M. Oligoarrays, Bioinformatics (Application Note), 18:486-487, 2002.
9. Blohm Dietmar H and Guiseppi. New development in microarray technology12:4147, 2001.
10. Beheshti. B. Braude . 1 Park P.C and squire JA Microarray cgh Methods Mol Biol 204:191-207, 2002.
11. Rahmann S. Rapid large-scale oligonucleotide selection for microarrays. In Proc of the second workshop on algorithms in Bioinformatics, 2002.
12. Bailey W.F. and Monahan A.S. In J.Chem Ed.1978.