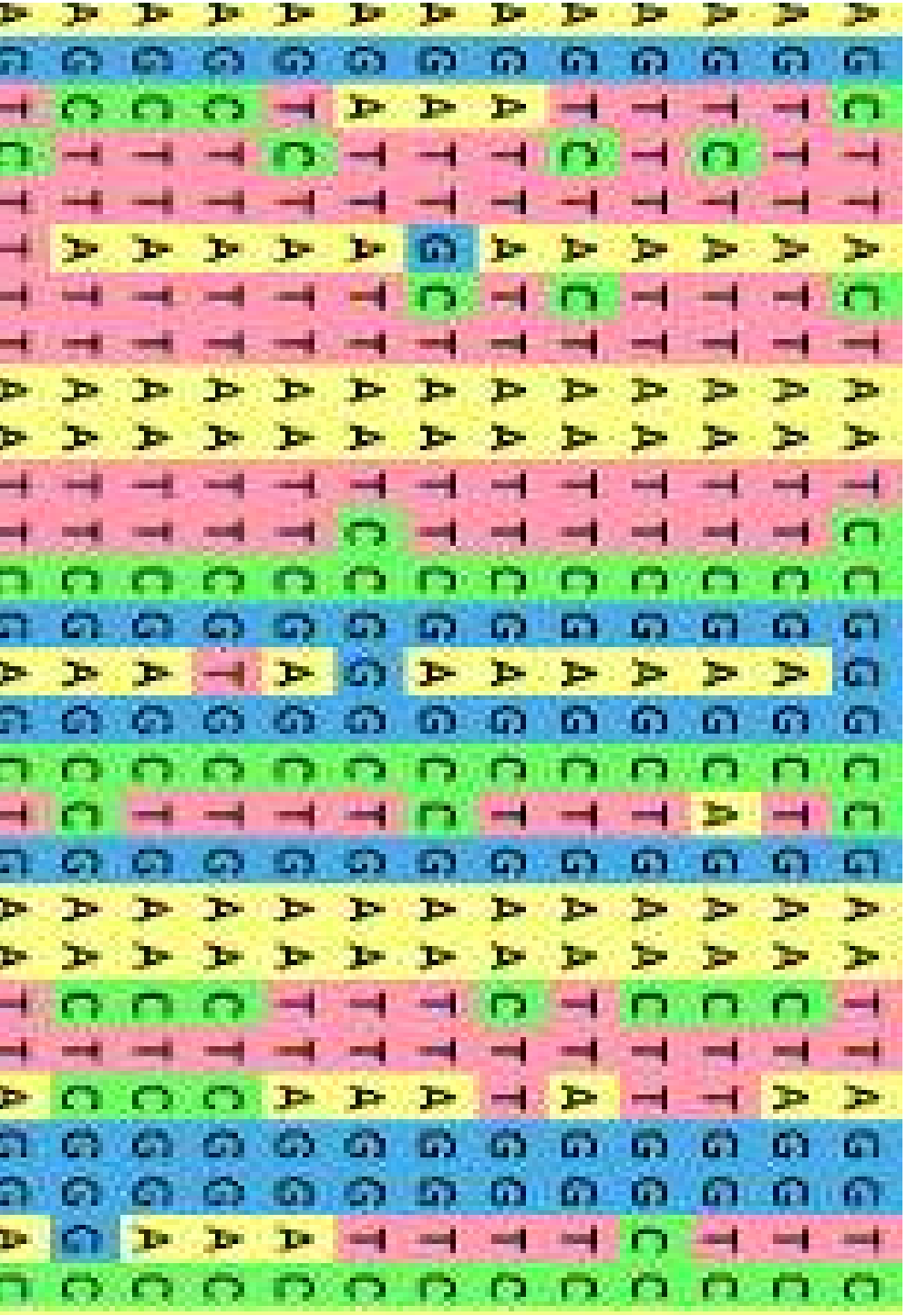


Needleman Wunsch Algorithm

DNA sequence alignmen

CS302 COURSE PROJECT

--DR APURVA MUDGAL

Team members:

2020CSB1064 ADITI DAS

2020CSB1083 DASARI LASHYANTH

2020CSB1084 DEEPIKA

2020CSB1087 GOPAGONI SREYA

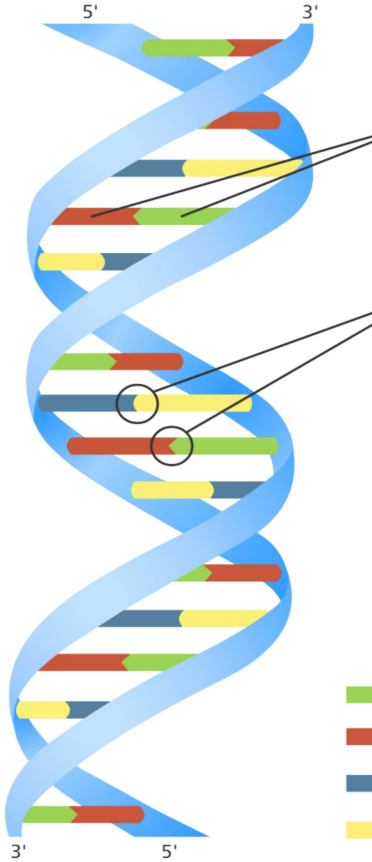
2020CSB1097 MANEPALLI VISHNU

Introduction:

DNA(deoxyribonucleic acid) is a long double helix molecule made up of four nucleotide bases (A,T,G,U). “Genome ” is a complete set of DNA of an organism and human genome is 3.2 billon bases long.

In this 3.2 billon bases long sequences it is very difficult to find similarities between two genomes or DNA sequence. so, computer adaptable method for finding similarities in the DNA sequences has been developed.

Needleman-Wunch algorithm for DNA sequence alignment interpret the similarity between two sequences using a score and dynamic programming and outputs the best possible alignment and best match score.



What is sequence Alignment

Sequence alignment is a method of comparing sequences like DNA,RNA or protein in order to find similarities between two or more sequences.

- This will provide you with an answer to the question: whether two sequences have evolved from a common ancestor or not.
- It is useful in determining evolutionary relationships between different species.

There are two types of pairwise alignment methods

- Global Alignment
- Local alignment

Global Vs Local alignment

Global Alignment : The best alignment over the entire length of two sequences.

Suitable when the two sequences are of similar length, with a significant degree
throughout.

Needleman-Wunsch algorithm (1970) is used for optimal global alignment.

Local Alignment : Involving stretches that are shorter than the entire sequences, possib
one.

Suitable when comparing substantially different sequences, which possibly differ si
length, and have only a short patches of similarity.

Smith-Waterman algorithm (1980) is used for optimal local alignment.

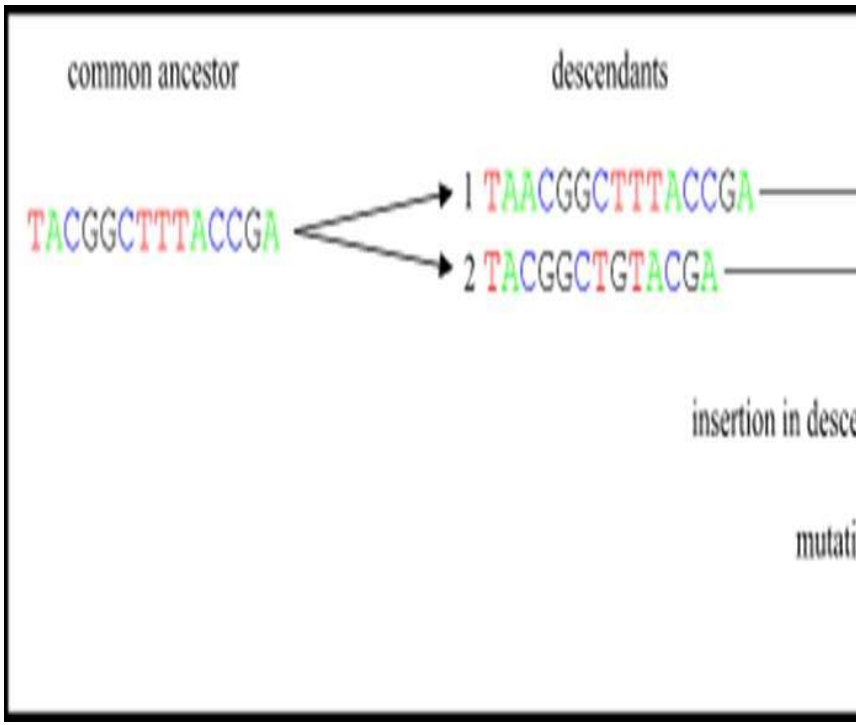
Interpretation

If two sequences in an alignment share a common ancestor

- **Mismatches** can be interpreted as point mutations or replace .

- **Gaps** are interpreted as insertion or deletion mutations introduced in one or both lineages in the time since they diverged from one another.

- The degree of similarity between amino acids or bases 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.



Input: we will be given two sequences

Output:

1. Maximum similarity or best score of the two sequences.
2. We have to find the alignment of the sequence so that it gives maximum similarity.
3. We have to find alignment with best score.

How do we find the best alignment and sco

Brute-force approach:

- Generate the list all possible alignments between two sequences.
- Score them and Select the alignment with the best score.
- The number of possible global alignments between two sequences of length
- For two sequences of 250 residues this is ~ 10^149.

The Needleman-Wunsch algorith

- A smart way to reduce the massive number of possibilities that ne considered, yet still guarantees that the best solution will be fou Needleman and Christian Wunsch, 1970).
- The basic idea is to build up the best alignment by using optimal al of smaller subsequences.
- **The Needleman-Wunsch algorithm** is an example of dynamic progra discipline invented by Richard Bellman (an American mathemat 1953!

Final Scoring Matrix:

		T	C	C	T	A
	0	-2	-4	-6	-8	-10
T	-2	5	3	1	-1	-3
C	-4	3	10	8	6	4
A	-6	1	8	9	7	11
T	-8	-1	6	7	14	12
A	-10	-3	4	5	12	19

Final Traceback Matrix:

		T	C	C	T	A
	done	left	left	left	Left	left
T	up	diag	left	left	diag	left
C	up	up	diag	Diag	left	left
A	up	up	up	diag	diag	diag
T	up	diag	up	diag	diag	left
A	up	up	up	diag	up	diag

		G	A	T
	0	0	0	0
G	0	5	1	-3
T	0	1	2	6
G	0	5	1	2
C	0	1	2	-2
C	0	-3	-2	-1
T	0	-3	-6	3

Working of Algorithm

- The maximum match is a number dependent upon the similarity of the sequences. definitions is the largest number of bases of one DNA that can be matched with those of a s allowing for all possible interruptions in either of the sequences.
- While the interruptions give rise to a very large number of comparisons, the method efficien from consideration those comparisons that cannot contribute to the maximum match.
- Comparisons are made from the smallest unit of significance, a pair of nucleotide bases each DNA. All possible pairs are represented by a two-dimensional array, and all possible c are represented by pathways through the array.
- For this maximum match only certain of the possible pathways must be evaluated. A nume one in this case, is assigned to every cell in the array representing like bases. The maximu the largest number that would result from summing the cell values of every pathway.

The walk through

- We consider all possible pairs of residue from two sequences (this gives ris
matrix representation).
- We will have two matrices: **the score matrix** and **traceback matrix.**
- The Needleman-Wunsch algorithm consists of three steps:

1. Initialization of the score matrix.
2. Calculation of scores and filling the traceback matrix.
3. Deducing the alignment from the traceback matrix or directly from score m

Scoring scheme

The scoring scheme is a set of rules which assigns the alignment score to any given
of two sequence. The alignment score is the sum of substitution scores
penalties. There are three main scoring schemas.

1. Basic score schema:

We assign values to each match, mismatch and gap as 1, -1,-2 respectively. depe
requirement we can change the value of each. For example, if we consider gaps are
then we can give gap score = -10. In basic version of Needleman-Wunsch Algorith
basic scoring schema.

- Match
- Mismatch
- Gap
  - Linear
  - Convex
  - Affine

REWARD!!

Penalise

Penalise

Algorithm with basic scoring schema:

```
for i = 0 to length(A)
   F(i,0) ← d * i
for j = 0 to length(B)
   F(0,j) ← d * j
for i = 1 to length(A)
   for j = 1 to length(B)
   {
      Match ← F(i−1, j−1) + M(a[i],b[j])
      Delete ← F(i−1, j) + d
      Insert ← F(i, j−1) + d
      F(i,j) ← max(Match, Insert, Delete)
   }
```

- d ← Gap penalty score, M ← match mismatch score
- F(i,j) ← Score at position (i,j)
- M(a[i],b[j]) ← {if a[i]=b[j] then score, otherwise mismatch score}

2. The substitution matrix

Similarity matrix: Assign values not only for the type of alteration, but also for t
that are involved. For example, a match between A and A may be given 10, bu
between T and T may be given 8. Here more importance is given to the As matchin
Ts, i.e. the As matching is assumed to be more significant to the alignment. This
based on letters also applies to mismatches.

In the code, we took below shown substitution matrix for implementation.

	A	G	C	T
A	10	−1	−3	−4
G	−1	7	−5	−3
C	−3	−5	9	0
T	−4	−3	0	8

Algorithm using substitution matrix for finding b

score in sequence alignment

```
for i = 0 to length(A)
    F(i,0) ← d * i
for j = 0 to length(B)
    F(0,j) ← d * j
for i = 1 to length(A)
    for j = 1 to length(B)
    {
        Match ← F(i–1, j–1) + S(A[i-1], B[j-1])
        Delete ← F(i–1, j) + d
        Insert ← F(i, j–1) + d
        F(i,j) ← max(Match, Insert, Delete)
    }
```

- d = Gap penalty score
- F(i,j) ← Score at position (i,j
- S(A[i],B[j]) = Substitution mat

The best alignment or path

The alignment is deduced from the values of cells along the traceba
by taking into account the values of the cell in the traceback matrix:

- **diag** – the letters from two sequences are aligned
- **left** – a gap is introduced in the left sequence
- **up** – a gap is introduced in the top sequence. Sequences are

backwards.

Algorithm to compute an alignment that actually gives the best score:

```
while (i > 0 or j > 0)
{
    if (i > 0 and j > 0 and F(i, j) == F(i-1, j-1) + S(Ai, Bj))
    {
        AlignmentA ← Ai + AlignmentA
        AlignmentB ← Bj + AlignmentB
        i ← i - 1
        j ← j - 1
    }
    else if (i > 0 and F(i, j) == F(i-1, j) + d)
    {
        AlignmentA ← Ai + AlignmentA
        AlignmentB ← "-" + AlignmentB
        i ← i - 1
    }
    else
    {
        AlignmentA ← "-" + AlignmentA
        AlignmentB ← Bj + AlignmentB
        j ← j - 1
    }
}
```

- Alignmen
- Alignmen
- i ← leng
- j ← leng

```
#include<bits/stdc++.h>
using namespace std;
#define MAX 1001

int dp[MAX][MAX]={0};

int s[4][4]=
{
    10, -1, -3, -4,
   -1,  7, -5, -3,
   -3, -5,  9,  0,
   -4, -3,  0,  8
};
```

(1)

```
int c(char a)
{
    if(a == 'A')
        return 0;
    else if(a == 'G')
        return 1;
    else if(a == 'C')
        return 2;
    else if(a == 'T')
        return 3;
    else
    {
        cout<<"Wrong Format
"<<a<<endl;
        exit(0);
    }
}
```

(2)

```
void print(int s1,int s2)
{
    for(int i=0; i<=s1; i++
    {
        for(int j=0; j<=s2; j
        {
            cout<<dp[i][j]<<
        }
        cout<<endl;
    }
}
```

(3)

```
int max_score(string s1, string s2, int g)
{
    int s1l = s1.length();
    int s2l = s2.length();
    int x, y, z;

    for(int i=1; i<=s1l; i++)
    {
        dp[i][0] = dp[i-1][0]-g;

    }
    for(int i=1; i<=s2l; i++)
    {
        dp[0][i] = dp[0][i-1]-g;
    }
```

(4)

```
for(int i=1; i<=s1l; i++)
    {
        for(int j=1; j<=s2l; j++)
        {

            x = dp[i-1][j]-g;
            y = dp[i-1][j-1]+ s[c(s1[i-1])][c(s2[j-1]
            z = dp[i][j-1]-g;

            dp[i][j]=max(x,max(y,z));
        }
    }
    return dp[s1l][s2l];
}
```

(5)

```
pair<string,string> path(string s1, string s2)
{
    int i = s1.length();
    int j = s2.length();
    string f = "", l = "";
    while(i>0 && j>0)
    {
        if(i > 0 and j > 0 and dp[i][j]==dp[i-1][j-1] + s[c(s1[i-1])][c(s2[j-1])])
        {
            f = s1[i-1] + f;
            l = s2[j-1] + l;
            i = i-1;
            j =j-1;
        }
        else if(i > 0 and dp[i][j] == dp[i-1][j]-5)
        {
            f = s1[i-1] + f;
            l = '_'+l;
            i=i-1;
        }
```

(6)

```
    else
        {
            f = '_'+f;
            l = s2[j-1] + l;
            j=j-1;
        }

    }
    while(j > 0)
    {
        l= s2[j-1]+l;
        f= '_'+f;
        j -= 1;
}
    while( i > 0){
        l = '_'+l;
        f = s1[i-1]+f;
        i -= 1;}
    return make_pair(f, l
}
```

(7)

```
int main()
{
    string a, b;
    cin>>a>>b;
    string c, d;

    int max_s = max_score(a, b, 5);
    c = path(a, b).first;
    d = path(a, b).second;

    cout<<"\nMaximum/best score obtained: "<<max_s<<endl;
    cout<<"\nSequence Alignment with best score: "<<endl;
    cout<<"1)\t"<<c<<endl;
    cout<<"2)\t"<<d<<endl;

    return 0;
}
```

Output from code

Input1:

ATGTAGTGTATAGTACATGCA

ATGTATATAGTACATATGGCA

Output:

Maximum/best score attained: 127

Sequence Alignment with best score:

1) ATGTAGTGTATAGTAC__AT_GCA

2) ATGTA___TATAGTACATATGGCA

Input2:

AGGTAT

TAGGTA

Output:

Maximum score attained: 32

Sequence Alignment with best sco

1) _AGGTAT

2) TAGGTA_

For example:

Seq 1 : TGGTG

Seq 2 : ATCGT

Step 1 : Initialise Matrix

		m	T	G	G	T	G
	n	0	-5	-10	-15	-20	-25
1	A	-5	x?				
2	T	-10					
3	C	-15					
4	G	-20					
5	T	-25					

Using substitution Matrix

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

For value of x :

$$\Rightarrow \max \begin{cases} 0 + s[A,T] \\ -5 + 8 \\ -5 + 8 \end{cases}$$

$\Rightarrow$ Mismatch penalty from substitution matrix $\Rightarrow -4$

$\Rightarrow$ gap penalty $\Rightarrow -5$

So,

$$x = \max \begin{cases} -4 \\ -10 \\ -10 \end{cases}$$

$x = -4$

Similarly:

		m	T	G	G	T	G
	n	0	-5	-10	-15	-20	-25
	A	-5	-4	-6	-11	-16	-21
	T	-10	3	-2	-7	-3	-8
	C	-15	-2	-2	-7	-7	-8
	G	-20	-7	5	5	0	0
	T	-25	-12	0	2	13	$\boxed{8}$

$dp[i][j][0]$

So, the max score possible for similarity among "TGGTG" & "ATCGT" is 8

To find best score alignment need to find which amon dp[i][i-1], dp[i-1][j-1]} is rel contributing to the value of

for eg for dp[5][5]

$$dp[5][5] = 8 = \max \begin{cases} 0 \\ 13 \\ 0 \end{cases}$$

Similarly we can mark the a shown in dp[i][i] and get using:

"↑" "←"

gap: Arrow is directed towards string having gap

Alignment with best score: A

with best score"

Claims and Observations

1. Penalty for gap should be higher than mismatch score.
2. A gap, which indicates a residue-to-nothing match, may be introduced in either sequ
   gap-to-gap match is meaningless and is not allowed.
3. Claim 1: the below recursion is true.

- f(i,0)=g*i
- f(0,j)=g*j
- f(i,j)=max{ f(i-1, j-1) + S(A[i], B[j])
  f(i-1, j) + d
  f(i, j-1) + d }

Given two strings or sequences

$\underset{A}{\underset{\parallel}{Seq1}}[1 \cdots m]$ and $\underset{B}{\underset{\parallel}{Seq2}}[1 \cdots n]$

Subproblem:

$F(i,j)$, $0 \leq i \leq m$, $0 \leq j \leq n$

$F(i,j)$ is equal to the best score possible among all alignment possible for $\underset{A}{\underset{\parallel}{seq1}}[1 \cdots i]$ and $\underset{B}{\underset{\parallel}{seq2}}[1 \cdots j]$

When $i=0$ or $j=0$, the corresponding sequence is empty.

So, $(m+1)\cdot(n+1)$ subproblems.

Proof of Claim1:

$f(0,0) = 0$

$f(i,0) = g*i$

$f(0,j) = g*j$

$$f(i,j) = \max \begin{cases} f(i-1,j-1) + S(A[i], B[j]) \\ f(i-1,j) + d \\ f(i,j-1) + d \end{cases}$$

Base cases:

$\Rightarrow f(0,0)$

both sequences are emp
for their optimal alignmen

$\Rightarrow f(i,0)$

One sequence has length i
sequence is empty, so th
gap penalty for every el
in sequence A due to it
sequence B

so,

$f(i,0) = \underset{\text{length of seq A}}{\underline{i}} * \underset{\text{gap}}{\underline{g}}$

$\Rightarrow$ Similarly, $f(0,j) =$
leng
of seq B

Recurrance:

Lets consider following operation/alterations possible on given sequences:

(a) match
(b) mismatch
(c) gap in first sequence
(d) gap in second sequence.

We can keep a sequence of alteration
$Alt = a_1 a_2 \cdots a_\ell$ be alterations corresponding to best score.

Now look at the rightmost alteration a_ℓ
There are 4 cases possible.

(i) a_ℓ is match

In this case $a_1 a_2 \cdots a_{\ell-1}$ is best score alteration for $A[1 \cdots (i-1)]$ & $B[1 \cdots (j-1)]$ & we assign a score from substitution matrix for the match.
So,

$$F(i,j) = F(i-1, j-1) + \sigma(S[\ ])$$

From substitution matrix

(ii) a_ℓ is mismatch
Similar to match, we calculate $F(i,j)$ as

$$F(i,j) = F(i-1, j-1) + \sigma(S[\ ])$$

From substitution matrix

(iii) a_ℓ is gap is first sequence

In this case $a_1 a_2 \cdots a_{\ell-1}$ alteration for $A[1 \cdots i]$ &
So,

$$F(i,j) = F(i, j-1) +$$

(iv) a_ℓ is gap in second sequence

In this case $a_1 a_2 \cdots a_{\ell-}$ alteration for $A[1 \cdots (i-1)]$
So,

$$F(i,j) = F(i-1, j) +$$

To get the maximum s
result from possible 4 case

$$F(i,j) = \max \begin{cases} F(i- \\ F(i, \\ F(i-1, \end{cases}$$

3.Gap penalty

Gap penalty: When aligning sequences there are often gaps, sometimes large one
in an alignment is introduced by deletion or insertion of a base.

A conventional wisdom dictates that the penalty for a gap must be several times gr
the penalty for a mutation. That is because

- A gap/extra residue Interrupts the entire polymer chain.
- In DNA shifts the reading frame

There are several gap penalty functions such as

- **Constant gap penalty,**
- **Linear gap penalty,**
- **Affine gap penalty**
- **Convex gap penalty etc.**

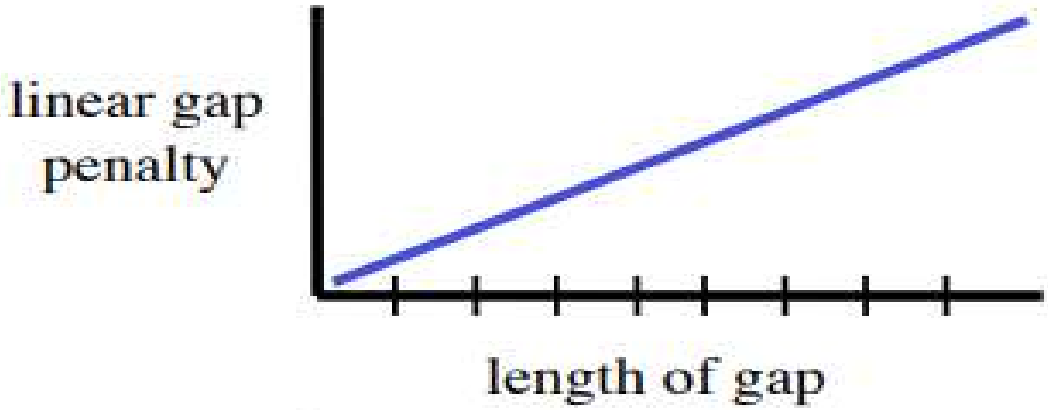
Constant gap penalty:

We just assign constant gap score to each and every gap in th
alignment. The code we implemented using basic schema use
constant gap penalty.

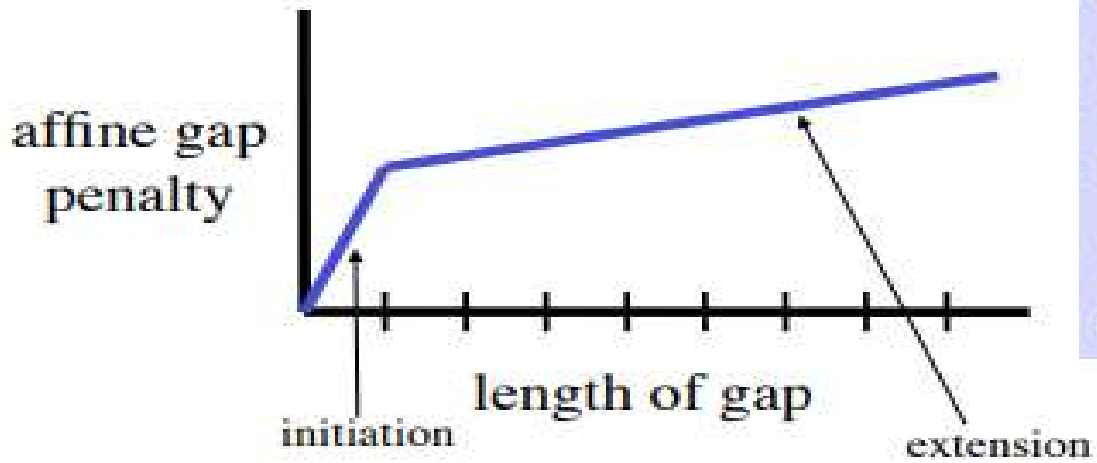
Linear gap penalty:

Compared to the constant gap penalty, the linear gap penalty
into account the length (L) of each insertion/deletion in the
Therefore, if the penalty for each inserted/deleted element
and the length of the gap L; the total gap penalty would b
product of the two BL.

Linear versus Affine gap penalty



Gap penalty for the whole sequence is just the total number of gap characters times a constant.



Gap penalty for the whole sequence is the function.
N*(gap initiation penalty) +
E*(gap extension penalty)

where N is the number of gap initiation characters, E is the numbe of gap extension characters

Three Matrices

We now keep 3 different matrices:

M[i,j] = score of best alignment of x[1..i] and y[1..j] ending with a character-character **match or mismatch**.

X[i,j] = score of best alignment of x[1..i] and y[1..j] ending with a **space in X**.

Y[i,j] = score of best alignment of x[1..i] and y[1..j] ending with a **space in Y**.

$$M[i,j] = \text{match}(i,j) + \max \begin{cases} M[i-1,j-1] \\ X[i-1,j-1] \\ Y[i-1,j-1] \end{cases}$$

$$X[i,j] = \max \begin{cases} M[i,j-k] - \text{gap}(k) & \text{for } 1 \le k \le j \\ Y[i,j-k] - \text{gap}(k) & \text{for } 1 \le k \le j \end{cases}$$

$$Y[i,j] = \max \begin{cases} M[i-k,j] - \text{gap}(k) & \text{for } 1 \le k \le i \\ X[i-k,j] - \text{gap}(k) & \text{for } 1 \le k \le i \end{cases}$$

1

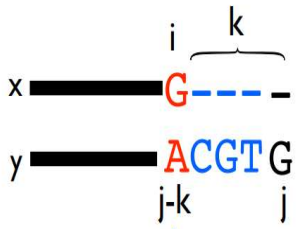
By definition, alignment ends in a match.

$$M[i,j] = \text{match}(i,j) + \max \begin{cases} M[i-1,j-1] \\ X[i-1,j-1] \\ Y[i-1,j-1] \end{cases}$$

Any kind of alignment is allowed before the match.

2

The X (and Y) matrices



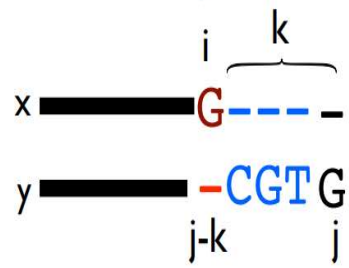
k decides how long to make the gap.

We have to make the whole gap at once in order to know how to score it.

$$X[i,j] = \max \begin{cases} M[i,j-k] - \text{gap}(k) & \text{for } 1 \le k \le j \\ Y[i,j-k] - \text{gap}(k) & \text{for } 1 \le k \le j \end{cases}$$

3

$$X[i,j] = \max \begin{cases} M[i,j-k] - \text{gap}(k) & \text{f} \\ Y[i,j-k] - \text{gap}(k) & \text{f} \end{cases}$$



```
#include<bits/stdc++.h>
using namespace std;
#define MAX 1001

float dp[MAX][MAX]={0};
float up_g[MAX][MAX]={0};
float lw_g[MAX][MAX]={0};

int s[4][4]=
{
    10, -1, -3, -4,
    -1,  7, -5, -3,
    -3, -5,  9,  0,
    -4, -3,  0,  8
};
```

```
int c(char a)
{
    if(a == 'A')
        return 0;
    else if(a == 'G')
        return 1;
    else if(a == 'C')
        return 2;
    else if(a == 'T')
        return 3;
    else
    {
        cout<<"Wrong Format "<<a<<endl;
        exit(10);
    }
}
```

```
float gap(int
{
    return 5*k
}
```

```
float max_score(string s1, string
s2)
{
    int s1l = s1.length();
    int s2l = s2.length();
    for(int i=1; i<=s1l; i++)
    {
        dp[i][0] = 0-gap(i);
        up_g[i][0] = 0-gap(i);
        lw_g[i][0] = 0-gap(i);
    }
    for(int i=1; i<=s2l; i++)
    {
        dp[0][i] = 0-gap(i);
        lw_g[0][i] = 0-gap(i);
        up_g[0][i] = 0-gap(i);
    }
```

```
for(int i=1; i<=s1l; i++)
    {
        for(int j=1; j<=s2l; j++)
        {
            dp[i][j] = s[c(s1[i-1])][c(s2[j-1])]+max(dp[i-1][j-1],max(u
1],lw_g[i-1][j-1]));
            up_g[i][j]=-1*INT_MAX;
            lw_g[i][j]=-1*INT_MAX;
            for(int k=1;k<=i;k++)
            {
                up_g[i][j]=max(up_g[i][j],max(dp[i-k][j]-gap(k),lw_g[i-
            }
            for(int k=1;k<=j;k++)
            {
                lw_g[i][j]=max(lw_g[i][j],max(dp[i][j-k]-gap(k),up_g[i][
            }
        }
    }
    return max(dp[s1l][s2l],max(lw_g[s1l][s2l],up_g[s1l][s2l]));
}
```

```
pair<string,string> path(string s1, string s2,float val)
{
    int i = s1.length();
    int j = s2.length();
    string f = "", l = "";
    while((i!=0) || (j!=0))
    {
        if(val==dp[i][j] )
        {
            f = s1[i-1] + f;
            l = s2[j-1] + l;
            val-=s[c(s1[i-1])][c(s2[j-1])];
            i = i-1;
            j =j-1;
        }
       for(int k=1;k<=i;k++)
        {
            if((val == dp[i-k][j]-gap(k)) or (val == lw_g[i-k][j]-gap(k)))
            {
                int e = k;
                while(e–)

        {
                    f = s1[i-1] + f;
                    l = '_'+l;
                    i–;
                }
                val+=gap(k);
                break;
            }
        }
```

```
For(i nt k=1;k<=j;k++)
            {
              if ((val == dp[i][j-k]-gap(k))
(val == up_g[i][j-k]-gap(k)))
              {
                  int e = k;
                  while(e–)
                  {
                      f = '_' + f;
                      l = s2[j-1] + l;
                      j–;
                  }
                  val+=gap(k);
                  break;
              }
          }
          //cout<<val<<endl;
      }
      return make_pair(f, l);
}
```

```
int main()
{
    string a, b;
    cin>>a>>b;
    string c, d;

    float max_s = max_score(a, b);
    cout<<"\nMaximum score attained - "<<max_s<<endl;
    //print(a.length(),b.length());
    c = path(a, b, max_s).first;
    d = path(a, b, max_s).second;


    cout<<"\nAttained Matching of strings"<<endl;
    cout<<"1)\t"<<c<<endl;
    cout<<"2)\t"<<d<<endl;

    return 0;
}
```

Output

ATGTAGTGTATAGTACATGCA

ATGTATATAGTACATATGGCA

Maximum score attained - 126

Attained Matching of strings

1) ATGTAGTGTATAGTACAT___GCA

2) ATGTA___TATAGTACATATGGCA

Input2:
AAATAAGCATGCGCTA
AAATGCTA

OUTPUT:
Maximum score attained - 31.5

Attained Matching of strings
1) AAATAAGCATGCGCTA
2) AAAT_____GCTA

ADVANCEMENT

DOMS Page No.
Date / /

Recurrance for Linear gap penalty:

→ We now keep 3 different matrices

$M[i,j]$ ⇒ Score of best alignment of $A[i \cdots j]$ & $B[1 \cdots j]$ ending with ch. match or mismatch

$X[i,j]$ → Score of best alignment ending with space in "A"

$Y[i,j]$ = Score of best alignment ending with space in "B"

→ Recurrance 1

$$M[i,j] = S[\ ] + \max \begin{cases} M[i-1, j-1] \\ X[i-1, j-1] \\ Y[i-1, j-1] \end{cases}$$

corresponding value from substitution matrix

$$X[i,j] = \max \begin{cases} M[i, j-k] - gap(k) \ ; \ 1 \le k \le j \\ Y[i, j-k] - gap(k) \ ; \ 1 \le k \le j \end{cases}$$

$$Y[i,j] = \max \begin{cases} M[i-k, j] - gap(k) \ ; \ 1 \le k \le i \\ X[i-k, j] - gap(k) \ ; \ 1 \le k \le i \end{cases}$$

→ Input constraint:
Gap penalty per gap decreases with increase in consecutive no. of gaps.

→ Let $a_1 a_2 \cdots a_L$ be the best score alteration sequence for $A[1 \cdots i]$ & $B[1 \cdots j]$

→ a_L can be
(i) Match or mismatch
(ii) Having 'k' consecutive in A seq
(iii) " " " B seq.

→ Claim: If $f: a_1 a_2 \cdots a_L$ is o
sequence than $f': a_1 a_2 \cdots a_{L-1}$ is
alteration

Proof: For sake of contradiction
f is not optimal

Let f^* be optimal seq for b_1

score(f^*) > score(f

score f' = score of f + a_L
< score of f^* + a_L
< score of seq (b_1 -

score f' < score (some seq

So, f' is not optimal
Hence assumption is wrong.

→ Observation: Optimal sequence
maximum of $M[i,j]$, $X[i,$

Proof:
optimal seq ⇒ ~~max of~~ Possibilities at

So, considering
optimal score = max

→ Case I

a_L is comparison fun (match/mismatch)

$M[i,j] = M[i-1, j-1] + \cancel{\text{mate}}\ s[i,j]$

→ Case II

Having K gaps in 1st sequence.

⇒ a_L is operation placing ('K') max consecutive gaps in 1st seq.

Claim: If A_L is operation placing 'K' max consecutive gaps in 1st seq. then A_{L-1} is not same operation.

Proof: For sake of controdiction:

Take $f = a_1 \cdots a_L$ with a_L & a_{L-1} are placing K_1, K_2 gaps in 1st seq.

score of (a_L) = -gap(K) (input)

score of f = score of $[a_1 \cdots a_{L-2}]$ + score (a_{L-1}) + score $[a_L]$

Let "b" be operation of placing K_1+K_2 gap in 1st seq.

From input, score$(a_{L-1} + a_1)$ > score(b).

⇒ score of f < score $(a_1 \cdots a_{L-2})$ + score(b).

Here f is not optimal because $(a_1 \cdots a_{L-2}, b)$ has higher score.

⇒ f is not optimal, our assumption is wrong.

Proved

Then $a_1 \cdots a_{L-1}$ is an operation seq which has 2 possibilities. 1) from M

2) from Y

We take max from these fu

$X[i,j] = \max(M[i,j], Y[i,$

$X[i,j] = \max(M[i,j], Y[i,$

→ Case III

Similarly for operation for ga sequence B. is same as ~~pr~~ by exchanging sequences

⇒ $X \rightleftarrows Y$ $\quad$ $X \circlearrowleft Y$

$Y[i,j] = \max[M[i,j], Y[i,$

To get optimal answer w max of these 3 cases

$$M[i,j] = \max \begin{cases} M[i-1, j- \\ X[i-1, j-1 \\ Y[i-1, j-1] \end{cases}$$

Required score = Max [M[a,

where a & b are

Thm: Linear sequence Alignment Matching always optimal score

Proof:- Let $f = (a_1 \cdots a_L)$ be output and
$f^* = (b_1 \cdots b_L)$ be optimal. ~~for~~
Both f and f^* are on i, j elements respectively.
score of $f \leq$ score of f^* → (I)
Score of f = score $(a_1 \cdots a_{L-1})$ + score(a_L).

→ if b_L is comparision.
score (a_L) := score(b_L)
Since $(a_1 \cdots a_{L-1})$ and $(b_1 \cdots b_{L-1})$ are optimal on (i-1, j-1) elements score$(a_1 \cdots a_{L-1})$ = score$(b_1 \cdots b_{L-1})$ → (1)
score of f = score of f^* → (a)

→ if b_L is gaps in 1st sequence. b_L complacing k_1 gaps

From the algorithm.

$$\text{score of } f = \max\Big(\text{score of } \overset{f_r(i,j-k)}{(a_1 \cdots a_{i-1})} + \text{score of } (k \text{ gaps})\Big)_{1 \leq k \leq j}$$

$$\geq \text{score of } (b_1 \cdots b_{L-1}) + \text{score of } (k_1 \text{ gaps}) \quad 1 \leq k_1 \leq j$$

score of $f \geq$ score of f^* (1)
→ (b)

→ if ~~gaps~~ b_L is placing 2nd seq. lly by taking k_2, and same comparision we get
score of $f \geq$ score of f^* → (c)

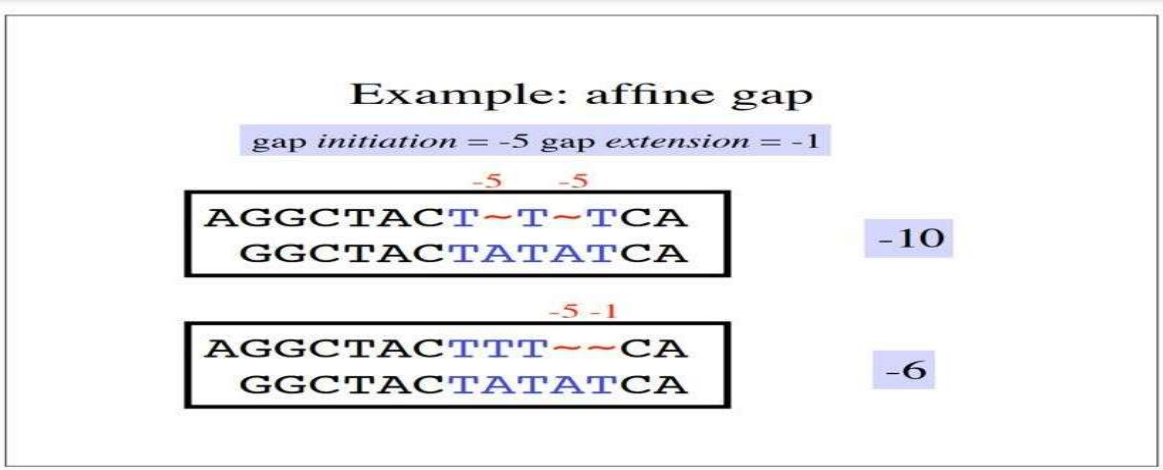
From (a)(b)(c)
score of $f \geq$ score of $f^* \Rightarrow$ score of f = score of f^* (2)
∴ Linear Sequence Alignment gives correct output

Affine Gap Penalty

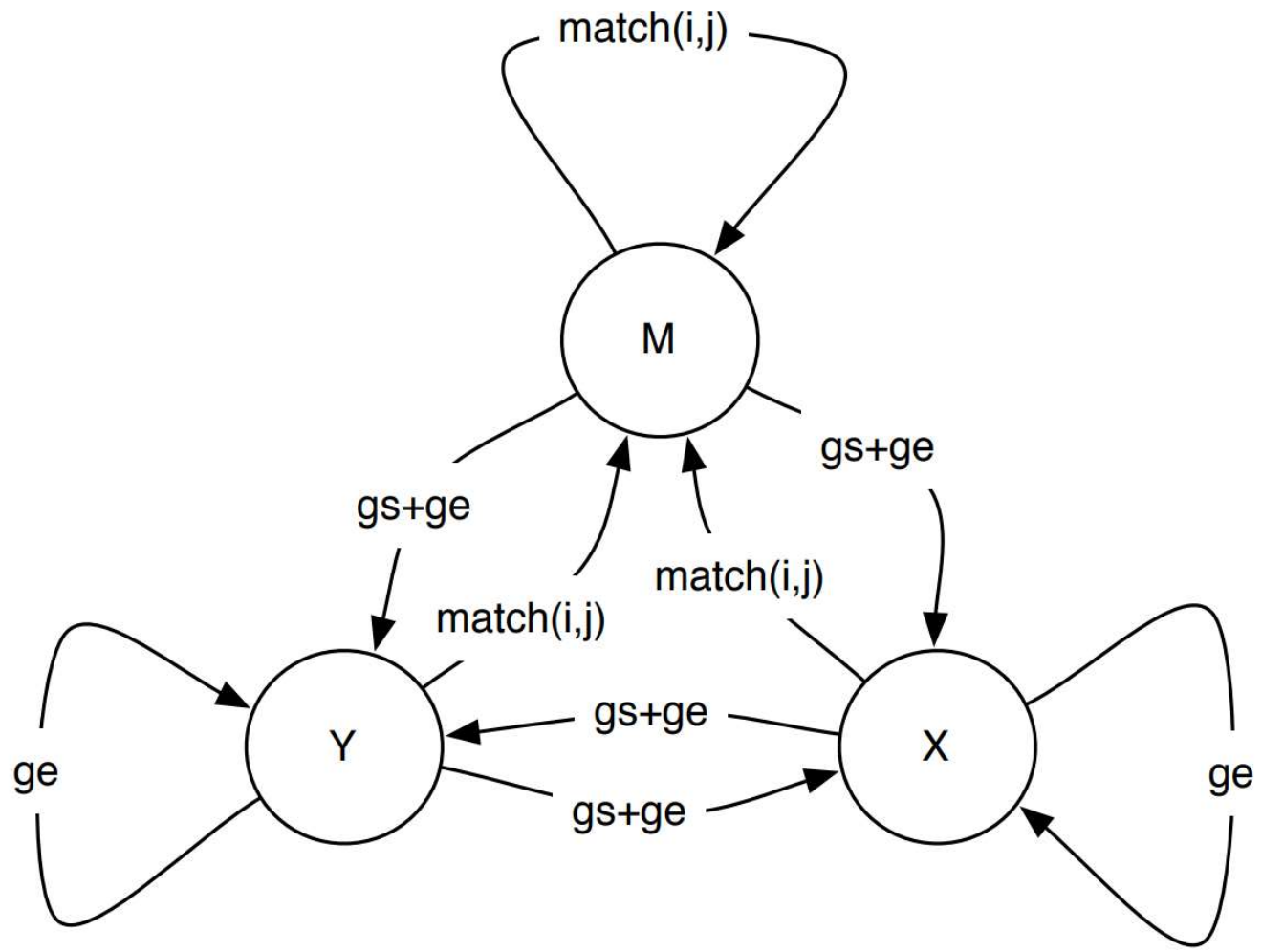
Biologically, a large gap is more likely to occur as one large deletion as opposed single deletions. Hence two small indels should have a worse score than one large one.

Affine gap penalty = gap opening penalty+ gap extension penalty * length of the gap

A large gap-start score for a new indel and a smaller gap-extension every letter which extends the indel. For example, new-indel may cost -5 and exten cost –1.



Affine Gap as Finite State Machine



Affine Gap Penalties

$$M[i,j] = \text{match}(i,j) + \max \begin{cases} M[i-1,j-1] \\ X[i-1,j-1] \\ Y[i-1,j-1] \end{cases}$$

match between x and y

If previous alignment ends in match, this is a new gap

$$X[i,j] = \max \begin{cases} \text{gap\_start} + \text{gap\_extend} + M[i,j-1] \\ \text{gap\_extend} + X[i,j-1] \\ \text{gap\_start} + \text{gap\_extend} + Y[i,j-1] \end{cases}$$

gap in x

$$Y[i,j] = \max \begin{cases} \text{gap\_start} + \text{gap\_extend} + M[i-1,j] \\ \text{gap\_start} + \text{gap\_extend} + X[i-1,j] \\ \text{gap\_extend} + Y[i-1,j] \end{cases}$$

gap in y

Time complexity

Constant gap penalty: mn subproblems

- Each one takes constant time
- Total runtime O(m*n)

Linear gap penalty: 3mn subproblems

- Each one takes o(n) time complexity
- Total runtime complexity 0(n^3) if m=n

Running time complexity

Time complexities for various gap penalty models

Type	Time
Constant gap penalty	O(mn)
Affine gap penalty	O(mn)
Convex gap penalty	O(mn lg(m+n))

Improvements

This improved algorithm effectively brings the information avai
BLOCKs+ database into the Needleman-Wunsch global sequence ali
This makes it very convenient for researchers to pay more attentio
biological characters on sequence alignment than common string s
alignment. As an increase of protein-coding sequences, the block
tools become more important for interpreting the large volume of s
data. We anticipate blocks-based tools play a very important role a
sequence alignment.

In biology more than just aligning the sequences, we also care for it properties and biological characters. Hence we find some blocks in the whose biological characteristics are identified and are important. So, oth normal score schema we also give reward to that block and compute the bes

Algorithm:

$$S_{ij} = \begin{cases} S_{i-1,j-1} + S(a_i b_j) \\ max(S_{i-x,j} - w_x) \\ max(S_{i,j-y} - w_y) \\ max(\text{blocks ending at(i,j)}) S_{i-len,j-len} \\ \qquad + |score(blockp)| \times reward + score(blockp) \end{cases}$$

- s(aibj) is the score for aligning the characters at positions i and j,

References

https://drive.google.com/drive/folders/1-x1ZGc-PMiPpvTXvatEoQM45oSa0-woo?usp=s

THANK YOU