

Sequence Alignment



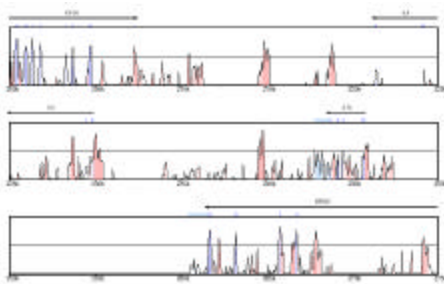
Lecture 2, Thursday April 3, 2003

Review of Last Lecture



Lecture 2, Thursday April 3, 2003

Sequence conservation implies function



Interleukin region in human and mouse

Lecture 2, Thursday April 3, 2003

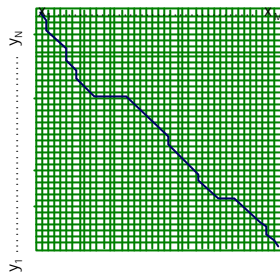
Sequence Alignment

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTGCCCCGAC
```

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC--
||| ||| ||| ||| ||| ||| ||| ||| |||
TAG-CTATCAC--GACCGC--GGTCGATTGCCCCGAC
```

Lecture 2, Thursday April 3, 2003

The Needleman-Wunsch Matrix



Every nondecreasing path

from (0,0) to (M, N)

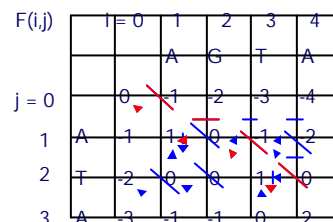
corresponds to an alignment of the two sequences

Lecture 2, Thursday April 3, 2003

The Needleman-Wunsch Algorithm

x = AGTA
y = ATA

m = 1
s = -1
d = -1



Optimal Alignment:

F(4,3) = 2

AGTA
A - TA

Lecture 2, Thursday April 3, 2003

The Needleman-Wunsch Algorithm

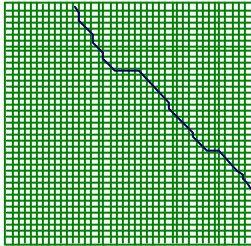
- Initialization
 - $F(0, 0) = 0$
 - $F(0, j) = -j \times d$
 - $F(i, 0) = -i \times d$
- Main Iteration, Filling-in partial alignments
 - For each $i = 1, \dots, M$
 For each $j = 1, \dots, N$

$$F(i, j) = \max \begin{cases} F(i-1, j) - d & \text{[case 1]} \\ F(i, j-1) - d & \text{[case 2]} \\ F(i-1, j-1) + s(x_i, y_j) & \text{[case 3]} \end{cases}$$

$$\text{Ptr}(i, j) = \begin{cases} \text{UP,} & \text{if [case 1]} \\ \text{LEFT} & \text{if [case 2]} \\ \text{DIAG} & \text{if [case 3]} \end{cases}$$
- Termination, $F(M, N)$ is the optimal score, and from $\text{Ptr}(M, N)$ can trace back optimal alignment

Lecture 2, Thursday April 3, 2003

The Overlap Detection variant



Changes:

1. Initialization
For all i, j ,
 $F(i, 0) = 0$
 $F(0, j) = 0$
2. Termination
$$F_{OPT} = \max \begin{cases} \max_i F(i, N) \\ \max_j F(M, j) \end{cases}$$

Lecture 2, Thursday April 3, 2003

Today

- Structure of a genome, and cross-species similarity
- Local alignment
- More elaborate scoring function
- Linear-Space Alignment
- The Four-Russian Speedup

Lecture 2, Thursday April 3, 2003

Structure of a genome

Human Genome:

- 3×10^9 bp
- ~30,000 genes
- ~200,000 exons
- ~23 Mb coding
- ~15 Mb noncoding

Flow of Genetic Information:

- a gene** (DNA double helix)
- transcription** (produces **pre-mRNA**)
- splicing** (removes introns, joins exons)
- mature mRNA**
- translation** (produces **protein**)

Structure of a genome

The diagram illustrates the structure of a genome, showing a sequence of genes (A, B, C, D) and regulatory elements. The genes are represented by colored boxes: A (green), B (red), C (yellow), and D (green). The regulatory elements are represented by blue boxes: A (green), B (red), C (yellow), and D (green). The diagram shows how the expression of one gene can regulate the expression of another, forming a regulatory network. The text "short sequences regulate expression of genes" and "lots of 'junk' sequence" are present. The text "e.g. ~50% repeats selfish DNA" is also present. The diagram shows a regulatory network where gene A regulates gene B, gene B regulates gene C, and gene C regulates gene D. The text "If A and B then D" is present. The text "If D then B" is present. The text "Make D" is present. The text "Make B" is present. The text "If B then NOT D" is present. The text "If A and B then D" is present. The text "If D then B" is present. The text "Make D" is present. The text "Make B" is present. The text "If B then NOT D" is present. The text "short sequences regulate expression of genes" is present. The text "lots of 'junk' sequence" is present. The text "e.g. ~50% repeats selfish DNA" is present.

Structure of a genome

short sequences regulate expression of genes

lots of "junk" sequence

e.g. ~50% repeats selfish DNA

If A and B then D

If D then B

Make D

Make B

If B then NOT D

Cross-species genome similarity

- 98% of genes are conserved between any two mammals
- ~75% average similarity in protein sequence

```

hum.h  : GTTGACACATGAGGGTCTTCGACAGAGCTC----- @ 57331/400001
mus.h  : GTTGACAAATGAGGGGCTTCGACAGAGCTC----- @ 78560/400001
rat.h  : GTTGACAAATGAGGGGCTTCGACAGAGCTC----- @ 112558/369398
Eug.h  : TTCTTGATGAGGAGAGCTCTCAATTTCTTGGAGCTATCTTAAAGAGAGCTC @ 36508/68674

hum.h  : CTGCGCCGCGGCTTCGAGAGCTCTTGACAGGACACACGCTCTTCAGCTGGTG @ 57361/400001
mus.h  : CTGCGCCCGGCTTCGAGAGCTCTTGACAGGACACCGCTCTTCAGCTGGTG @ 78610/400001
rat.h  : CTGCGCCCGGCTTCGAGAGCTCTTGACAGGACACCGCTCTTCAGCTGGTG @ 112708/369398
Eug.h  : TGCGGCGAGTGTCTTGAGTGCGCTGACAGAGAGAGAGAGCGCTCTTCAGCTGGTG @ 36558/68674

```

"atoh" enhancer in human, mouse, rat, fugu fish

```

hum.h  : AGGCGACCTCTCTTGGACGAGCTCTCCCGGGAGAGCTGGAGCGGCACATT @ 57433/400001
mus.h  : AGGCGACCTCTCTTGGAGCGCTCTCCCGGGAGAGCTGGAGCGGCACATT @ 78659/400001
rat.h  : AGGCGACCTCTCTTGGAGCGCTCTCCCGGGAGAGCTGGAGCGGCACATT @ 112757/369398
Eug.h  : AGGCGTGTGCTCTTGGAGCGCTCTCCCGGGAGAGCTGGAGCGGCACATT @ 36584/68674

hum.h  : AAGACAGCTATATACAGCTCTCCCGGGAGCTCTGACATCTGGAGCTCTCTCTCG @ 57482/400001
mus.h  : AAGACAGCTGTGCTCTCTCCCGGGAGCTCTGACATCTGGAGCTCTCTCTCTCG @ 78708/400001
rat.h  : AAGACAGCTGTGCTCTCTCCCGGGAGCTCTGACATCTGGAGCTCTCTCTCTCG @ 112804/369398
Eug.h  : CGAGAGAGACTCTGCTCTCCCGGGAGCTCTGACATCTGGAGCTCTCTCTCTCG @ 36597/68674

```

Lecture 2, Thursday April 3, 2003

The local alignment problem

Given two strings

$$x = x_1 \dots x_M,$$
$$y = y_1 \dots y_N$$

Find substrings x' , y' whose similarity
(optimal global alignment value)
is maximum

e.g. $x = \text{aaaaccccgagg}$
 $y = \text{cccgaggaaaccaacc}$

Lecture 2, Thursday April 3, 2003

Why local alignment

- Genes are shuffled between genomes
- Portions of proteins (domains) are often conserved

(A)

Helix 1 Helix 2 Helix 3

Pfx1
G s g
Qia1
R924
Rut

(B)

Pfx1
SQTALKEQFL
ADLRADA
ENRRERKIQ

G s g
SQAVLVTRD
GLARAVE
RVRLRGEG

Pfx1
SQTALKEQFL
ADLRADA
ENRRERKIQ

G s g
SQAVLVTRD
GLARAVE
RVRLRGEG

Lecture 2, Thursday April 3, 2003

Lecture 2, Thursday April 3, 2003

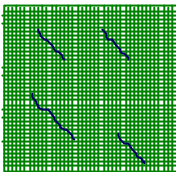
The Smith-Waterman algorithm

Idea: Ignore badly aligning regions

Modifications to Needleman-Wunsch:

Initialization: $F(0, j) = F(i, 0) = 0$

Iteration:
$$F(i, j) = \max \begin{cases} 0 \\ F(i-1, j) - d \\ F(i, j-1) - d \\ F(i-1, j-1) + s(x_i, y_j) \end{cases}$$



Lecture 2, Thursday April 3, 2003

Lecture 2, Thursday April 3, 2003

The Smith-Waterman algorithm

- Termination:
 - If we want the **best** local alignment...
$$F_{OPT} = \max_{i,j} F(i, j)$$
 - If we want **all** local alignments **scoring** $> t$

For all i, j find $F(i, j) > t$, and trace back


Lecture 2, Thursday April 3, 2003

Lecture 2, Thursday April 3, 2003

Scoring the gaps more accurately


Current model:

Gap of length	n
incurs penalty	$n \times d$



However, gaps usually occur in bunches

Convex gap penalty function:



$\gamma(n)$:

for all n , $\gamma(n+1) - \gamma(n) \leq \gamma(n) - \gamma(n-1)$

Lecture 2, Thursday April 3, 2003

Lecture 2, Thursday April 3, 2003

General gap dynamic programming

Initialization: same

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ \max_{k=0, \dots, i-1} F(k, j) - \gamma(i-k) \\ \max_{k=0, \dots, j-1} F(i, k) - \gamma(j-k) \end{cases}$$

Termination: same

Running Time: $O(N^2M)$ (assume $N > M$)

Space: $O(NM)$

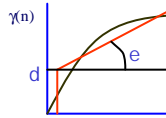
Lecture 2, Thursday April 3, 2003

Space: $O(NM)$

Compromise: affine gaps

$$\gamma(n) = d + (n-1) \times e$$

|
|
 gap open gap extend



To compute optimal alignment,

At position i, j , need to "remember" best score if gap is open
best score if gap is not open

$F(i, j)$: score of alignment $x_1 \dots x_i$ to $y_1 \dots y_j$
if x_i aligns to y_j

$G(i, j)$: score if x_i or y_j aligns to a gap

Lecture 2, Thursday April 3, 2003

Needleman-Wunsch with affine gaps

Initialization: $F(i, 0) = d + (i-1) \times e$
 $F(0, j) = d + (j-1) \times e$

Iteration:

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ G(i-1, j-1) + s(x_i, y_j) \end{cases}$$

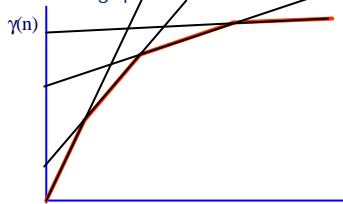
$$G(i, j) = \max \begin{cases} F(i-1, j) - d \\ F(i, j-1) - d \\ G(i-1, j) - e \\ G(i, j-1) - e \end{cases}$$

Termination: same

Lecture 2, Thursday April 3, 2003

To generalize a little...

... think of how you would compute optimal alignment with this gap function



....in time $O(MN)$

Lecture 2, Thursday April 3, 2003

Bounded Dynamic Programming

Assume we know that x and y are very similar

Assumption: $\# \text{gaps}(x, y) < k(N)$ (say $N > M$)

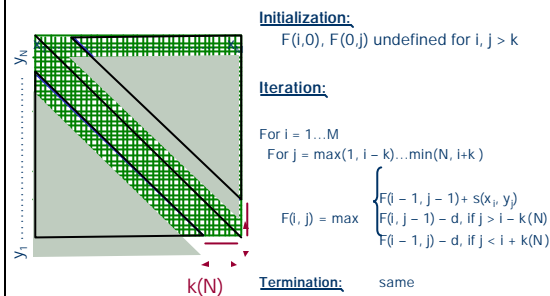
Then, $\begin{matrix} x_i \\ | \\ y_j \end{matrix}$ implies $|i - j| < k(N)$

We can align x and y more efficiently:

Time, Space: $O(N \times k(N)) \ll O(N^2)$

Lecture 2, Thursday April 3, 2003

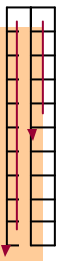
Bounded Dynamic Programming



Easy to extend to the affine gap case

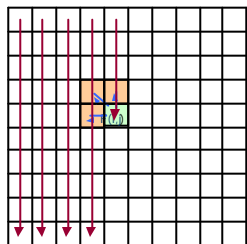
Lecture 2, Thursday April 3, 2003

Linear-Space Alignment



Introduction: Compute the optimal score

It is easy to compute $F(M, N)$ in linear space



```

Allocate ( column[1] )
Allocate ( column[2] )

For i = 1...M
  If i > 1, then:
    Free( column[i - 2] )
    Allocate( column[ i ] )
  For j = 1...N
    F(i, j) = ...
    
```

Lecture 2, Thursday April 3, 2003

Linear-space alignment

To compute both the optimal score and the optimal alignment:

Divide & Conquer approach:

Notation:

x^r, y^r : reverse of x, y

E.g. $x = \text{accgg}$

$x^r = \text{ggcca}$

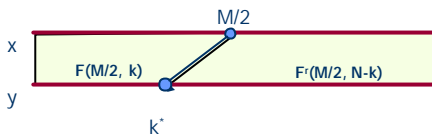
$F^r(i, j)$: optimal score of aligning $x^r_1 \dots x^r_i$ & $y^r_1 \dots y^r_j$
same as $F(M-i+1, N-j+1)$

Lecture 2, Thursday April 3, 2003

Linear-space alignment

Lemma:

$$F(M, N) = \max_{k=0 \dots N} (F(M/2, k) + F^r(M/2, N-k))$$

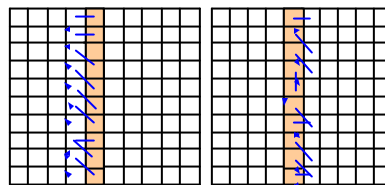


Lecture 2, Thursday April 3, 2003

Linear-space alignment

Now, using 2 columns of space, we can compute
for $k = 1 \dots M$, $F(M/2, k)$, $F^r(M/2, k)$

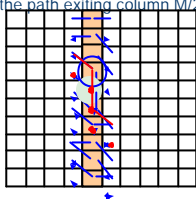
PLUS the backpointers



Lecture 2, Thursday April 3, 2003

Linear-space alignment

Now, we can find k^* maximizing $F(M/2, k) + F^r(M/2, k)$
Also, we can trace the path exiting column $M/2$ from k^*

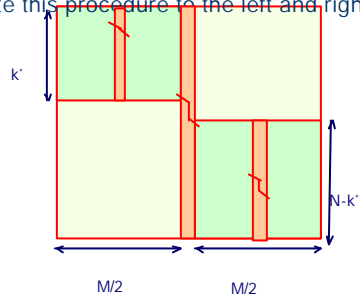


Conclusion: In $O(NM)$ time, $O(N)$ space,
we found optimal alignment path at column $M/2$

Lecture 2, Thursday April 3, 2003

Linear-space alignment

• Iterate this procedure to the left and right!



Lecture 2, Thursday April 3, 2003

Linear-space alignment

Hirschberg's Linear-space algorithm:

MEMALIGN(l, l', r, r'): (aligns $x_l \dots x_r$ with $y_{l'} \dots y_{r'}$)

1. Let $h = \lceil (l' - l) / 2 \rceil$
2. Find in Time $O((l' - l) \times (r' - r))$, Space $O(r' - r)$ the optimal path, L_h , at column h
Let $k_1 = \text{pos}'n$ at column $h - 1$ where L_h enters
 $k_2 = \text{pos}'n$ at column $h + 1$ where L_h exits
1. MEMALIGN($l, h - 1, r, k_1$)
2. Output L_h
3. MEMALIGN($h + 1, l', k_2, r'$)

Lecture 2, Thursday April 3, 2003

Linear-space Alignment

Time, Space analysis of Hirschberg's algorithm:

To compute optimal path at middle column,

For box of size $M \times N$,

Space: $2N$

Time: cMN , for some constant c

Then, left, right calls cost $c(M/2 \times k' + M/2 \times (N - k')) = cMN/2$

All recursive calls cost

Total Time: $cMN + cMN/2 + cMN/4 + \dots = 2cMN = O(MN)$

Total Space: $O(N)$ for computation,
 $O(N+M)$ to store the optimal alignment

Lecture 2, Thursday April 3, 2003

The Four-Russian Algorithm

A useful speedup of Dynamic Programming



Main Observation

Within a rectangle of the DP matrix, values of D depend only on the values of A, B, C , and substrings $x_{l..r}, y_{l'..r'}$

Definition:

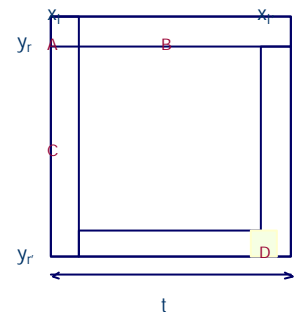
A t -block is a $t \times t$ square of the DP matrix

Idea:

Divide matrix in t -blocks,
Precompute t -blocks

Speedup: $O(t)$

Lecture 2, Thursday April 3, 2003

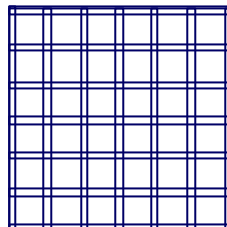


The Four-Russian Algorithm

Main structure of the algorithm:

- Divide $N \times N$ DP matrix into $K \times K$ $\log_2 N$ -blocks that overlap by 1 column & 1 row
- For $i = 1 \dots K$
- For $j = 1 \dots K$
- Compute D_{ij} as a function of $A_{ij}, B_{ij}, C_{ij}, x[l_i \dots l'_j], y[r_j \dots r'_j]$

Time: $O(N^2 / \log^2 N)$
times the cost of step 4



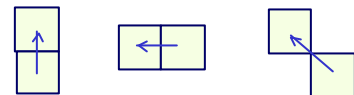
Lecture 2, Thursday April 3, 2003

The Four-Russian Algorithm

Another observation:

(Assume $m = 1, s = 1, d = 1$)

Two adjacent cells of $F(\dots)$ differ by at most 1.



Lecture 2, Thursday April 3, 2003

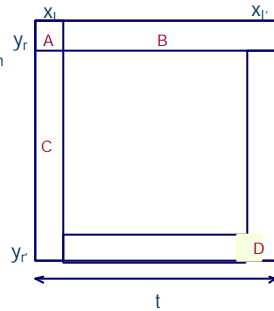
The Four-Russian Algorithm

Definition:

The offset vector is a t -long vector of values from $\{-1, 0, 1\}$, where the first entry is 0

If we know the value at A, and the top row, left column offset vectors, and $x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}$,

Then we can find D



Lecture 2, Thursday April 3, 2003

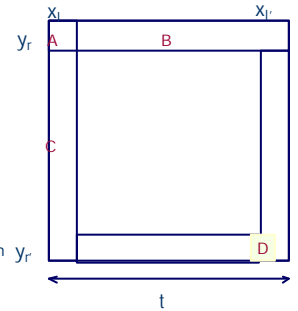
The Four-Russian Algorithm

Definition:

The offset function of a t -block is a function that for any

given offset vectors of top row, left column, and $x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}$,

produces offset vectors of bottom row, right column



Lecture 2, Thursday April 3, 2003

The Four-Russian Algorithm

We can pre-compute the offset function:

$3^{2(t-1)}$ possible input offset vectors

4^{2t} possible strings $x_1, \dots, x_{t-1}, y_1, \dots, y_{t-1}$

Therefore $3^{2(t-1)} \times 4^{2t}$ values to pre-compute

We can keep all these values in a table, and look up in linear time, or in $O(1)$ time if we assume constant-lookup RAM for log-sized inputs

Lecture 2, Thursday April 3, 2003