Welcome to
CS262: Computational Genomics

Instructor:

Serafim Batzoglou

TA:

Paul Chen

email: cs262-win2015-staff@lists.stanford.edu

Tuesdays & Thursdays 12:50-2:05pm
Clark S361

http://cs262.stanford.edu

Goals of this course

- Introduction to Computational Biology & Genomics
  - Basic concepts and scientific questions
  - Why does it matter?
  - Basic biology for computer scientists
  - In-depth coverage of algorithmic techniques
  - Current active areas of research
- Useful algorithms
  - Dynamic programming
  - String algorithms
  - HMMs and other graphical models for sequence analysis

Topics in CS262

Part 1: Basic Algorithms

- Dynamic Programming & sequence alignment
- HMMs, CRFs & sequence modeling
- Sequence indexing; Burrows-Wheeler transform, De Brujin graphs

Part 2: Topics in computational genomics and areas of active research

- DNA sequencing and assembly
- Comparative genomics
- Human genome resequencing
  - Alignment
  - Compression
  - Human genome variation
- Cancer genomics
- Functional genomics
- Population genomics

Course responsibilities

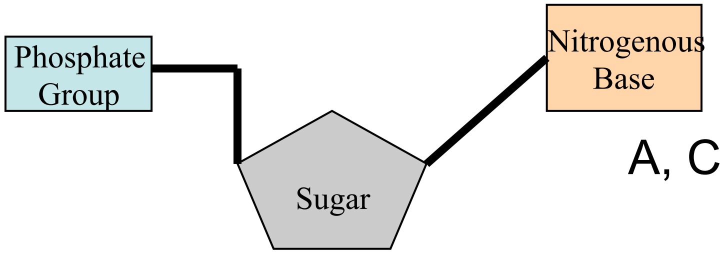
- Homeworks
  - 4 challenging problem sets, 4-5 problems/pset
    - Due at beginning of class
    - Up to 3 late days (24-hr periods) for the quarter
  - Collaboration allowed – please give credit
    - Teams of 2 or 3 students
    - Individual writeups
    - If individual (no team) then drop score of worst problem per problem set
- (Optional) Scribing
  - Due one week after the lecture, except special permission
  - Scribing grade replaces 2 lowest problems from all problem sets
    - First-come first-serve, email staff list to sign up

Reading material

- Main Reading:
  - Lecture notes
  - Papers

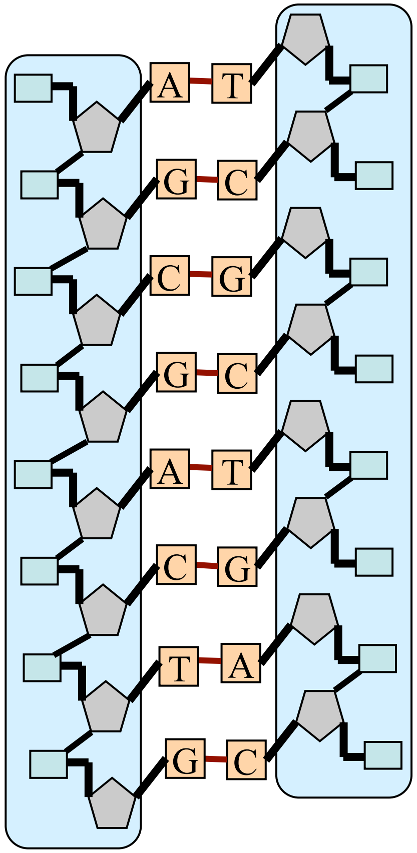
- Optional:
  - "Biological sequence analysis" by Durbin, Eddy, Krogh, Mitchison
    - Chapters 1-4, 6, 7-8, 9-10

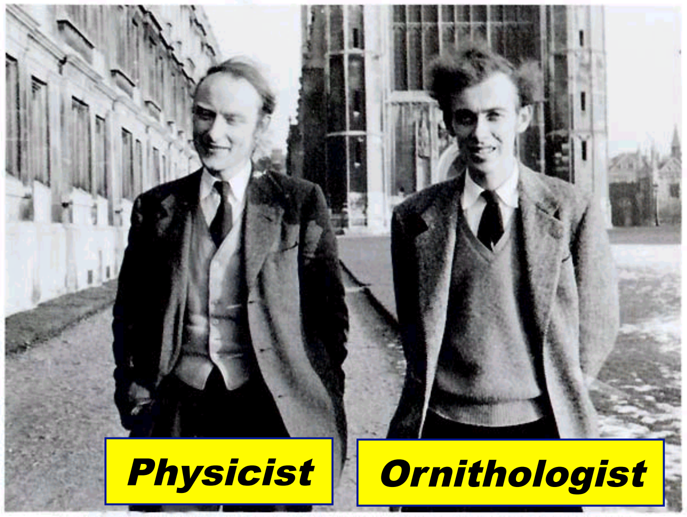
Birth of Molecular Biology



A, C, G, T

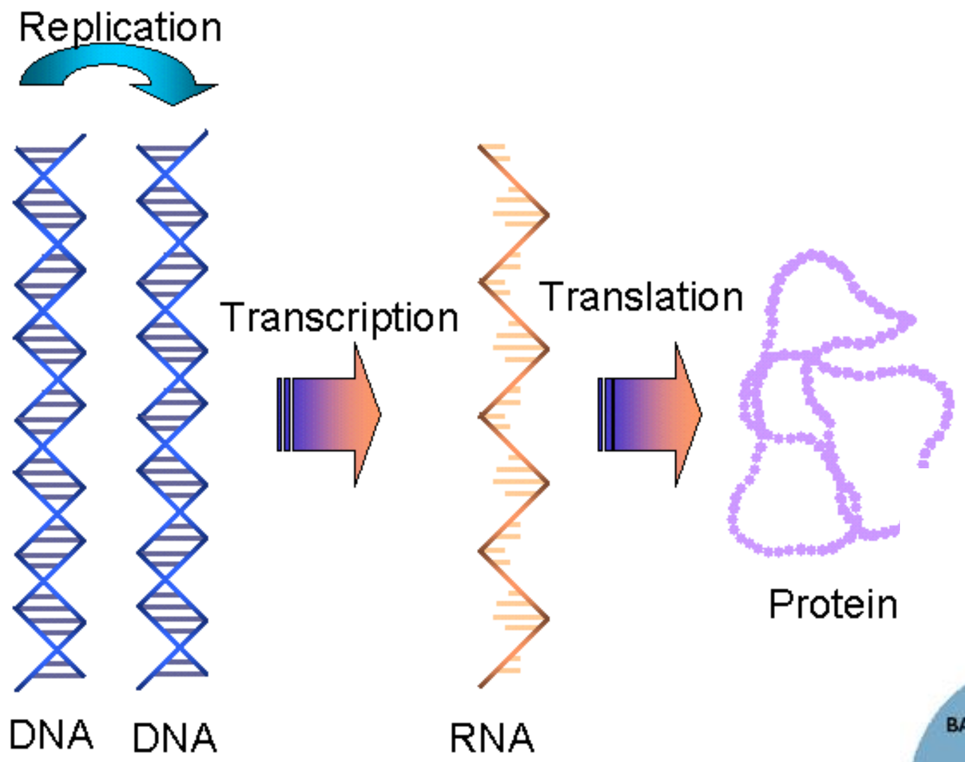
DNA

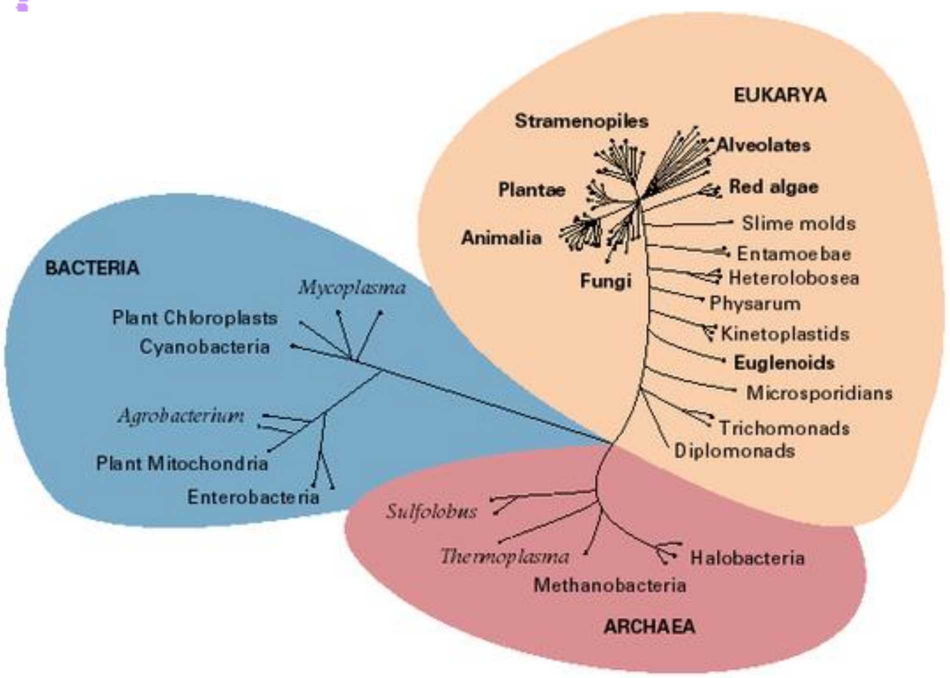




Courtesy of Cold Spring Harbor Laboratory Archives. Noncommercial, educational use only.

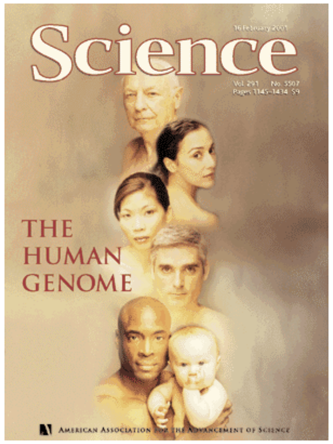
Genetics in the 20th Century





Human Genome Project

nature
the human genome
Nuclear fission
Five-dimensional energy landscapes
Seafloor spreading
The view from under the Arctic ice
Career prospects
Sequence creates new opportunities
naturejobs
genomics special



1990: Start

2000: Bill Clinton: *"most important scientific discovery in the 20th century"*

2001: Draft

2003: Finished

3 billion basepairs

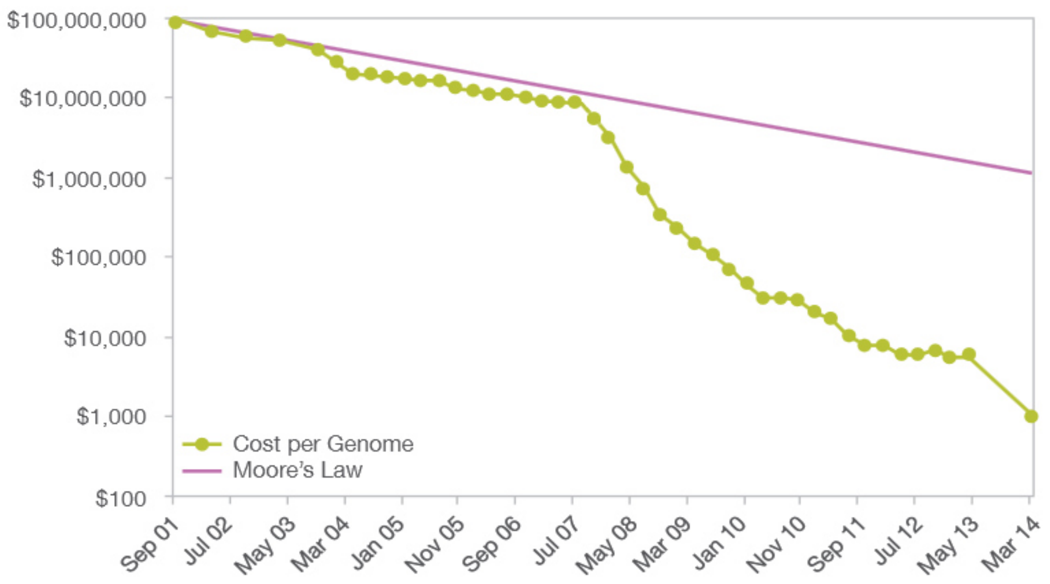
$3 billion

now what?

Sequencing Growth

<u>Cost of one human genome</u>

• 2004:	$30,000,000
• 2008:	$100,000
• 2010:	$10,000
• **2014:**	**“$1,000”**
• ???:	$300



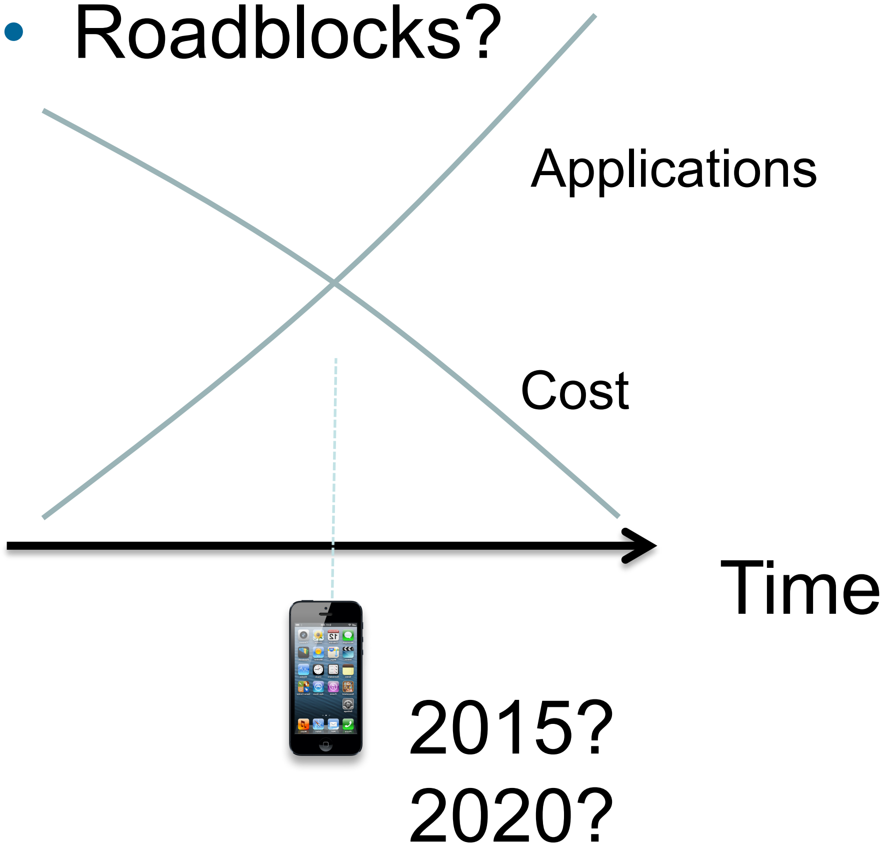
How much would you pay for a smartphone?

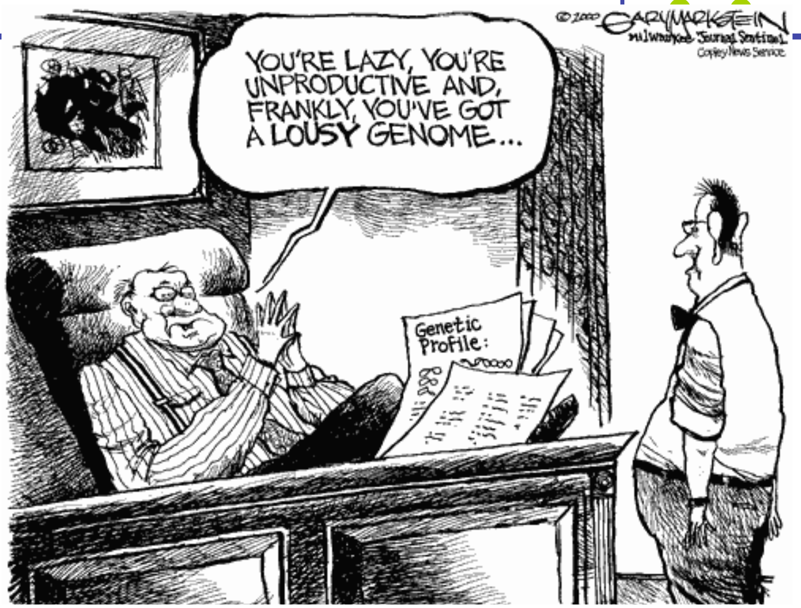
Uses of Genomes

- Medicine
  - Prenatal/Mendelian diseases
  - Drug dosage (eg. Warfarin)
  - Disease risk
  - Diagnosis of infections
  - …
- Ancestry
- Genealogy
- Nutrition?
- Psychology?
- Baby Engineering???...
- **Ethical Issues**

How soon will we all be sequenced?

- Cost
- Killer apps
- Roadblocks?

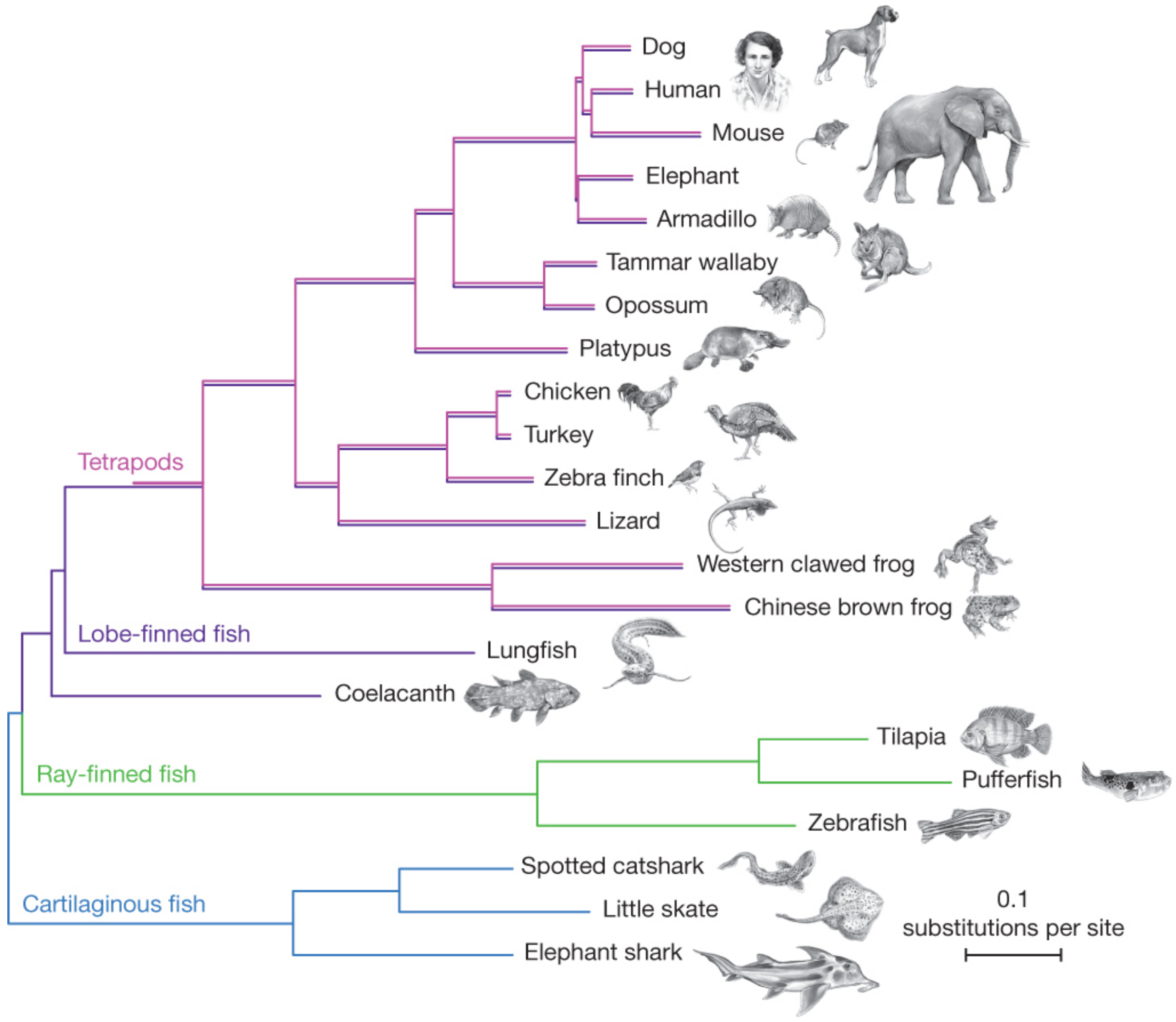



© 2000 GARY MARKSTEIN Milwaukee Journal Sentinel Copley News Service
YOU'RE LAZY, YOU'RE UNPRODUCTIVE AND, FRANKLY, YOU'VE GOT A LOUSY GENOME...
Genetic Profile:

Intro to Biology

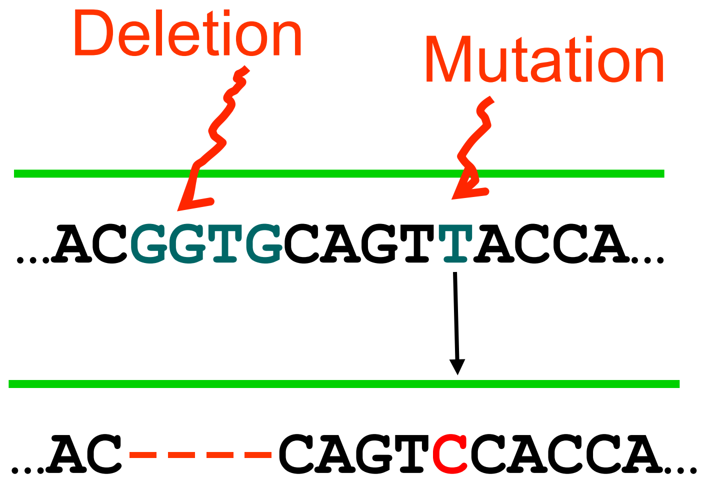
Sequence Alignment

Evolution



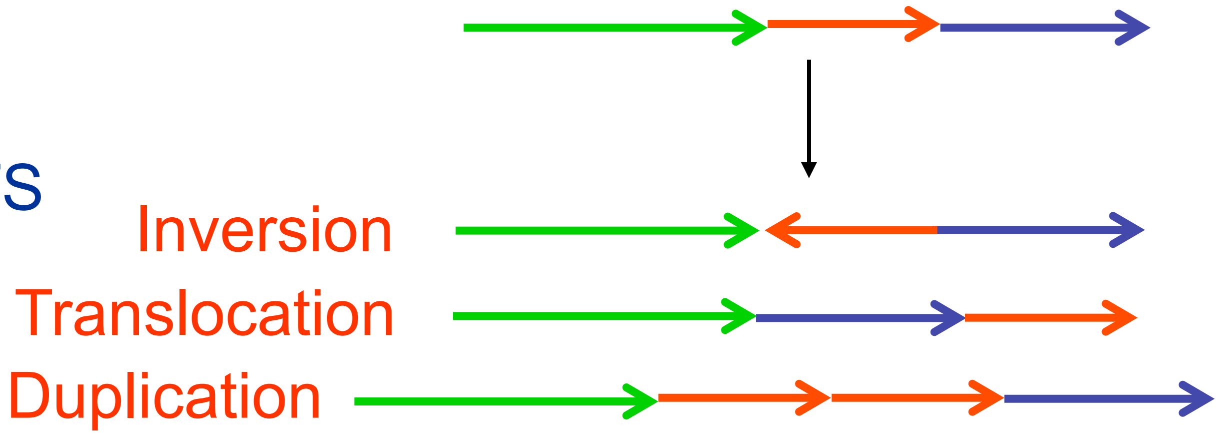
CT Amemiya *et al. Nature* **496**, 311-316 (2013)
doi:10.1038/nature12027

Evolution at the DNA level

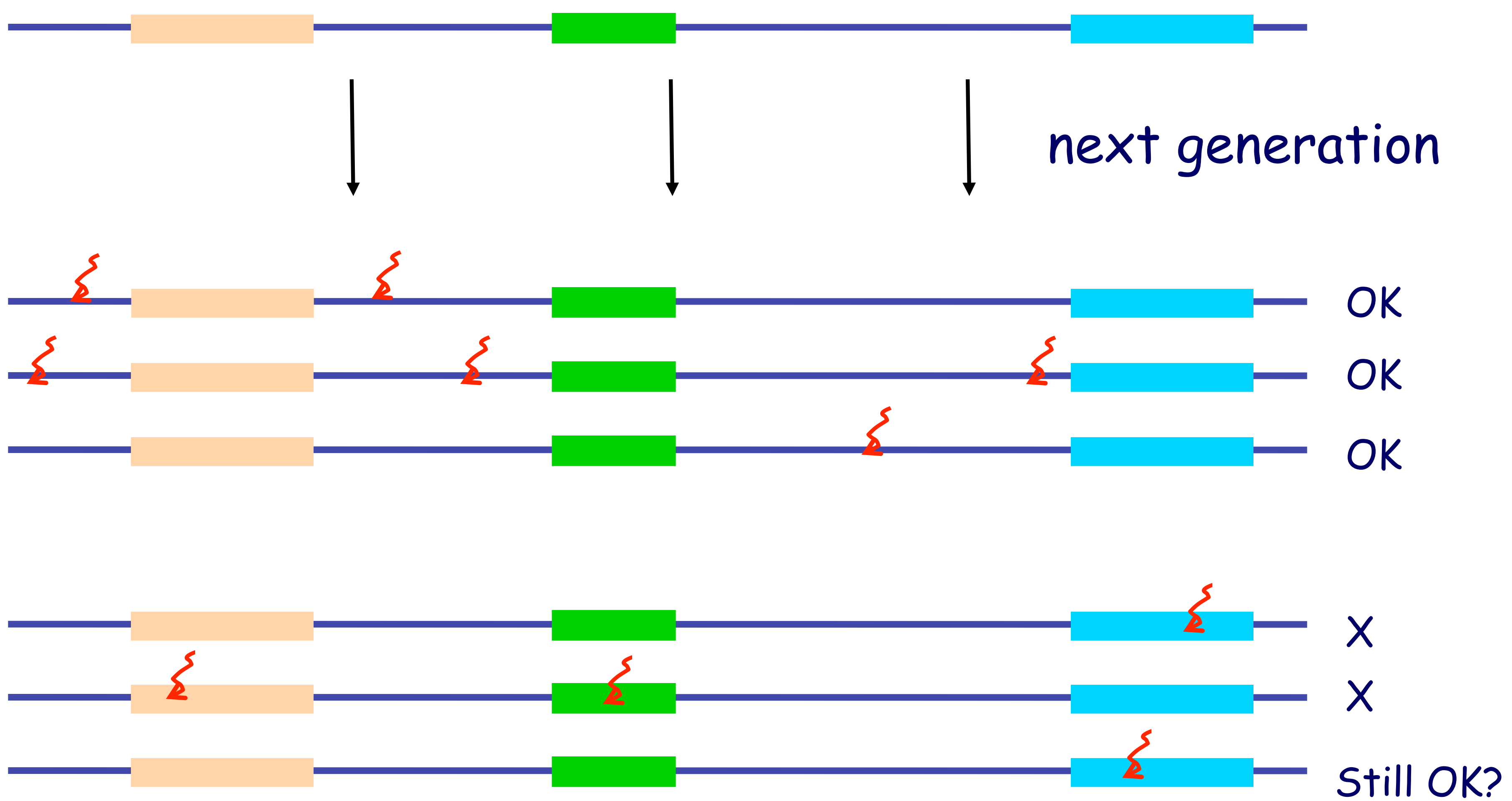


SEQUENCE EDITS

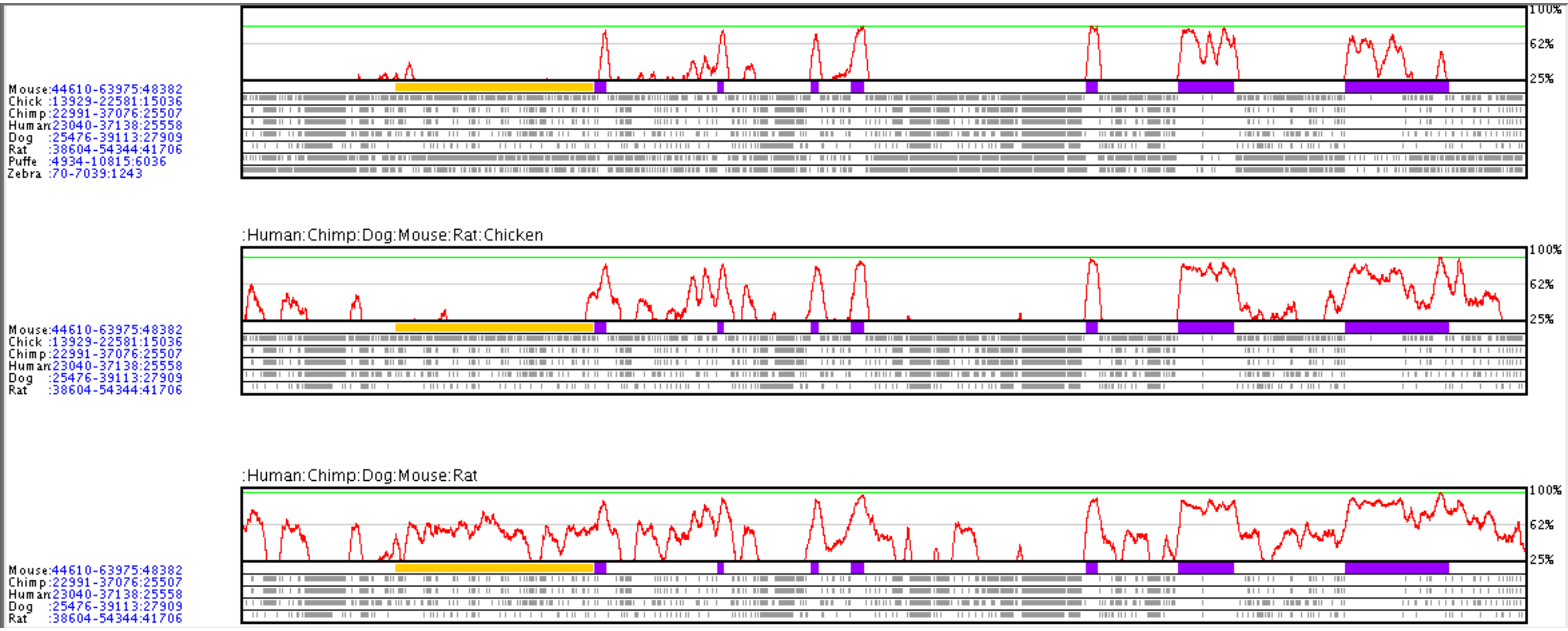
REARRANGEMENTS



Evolutionary Rates

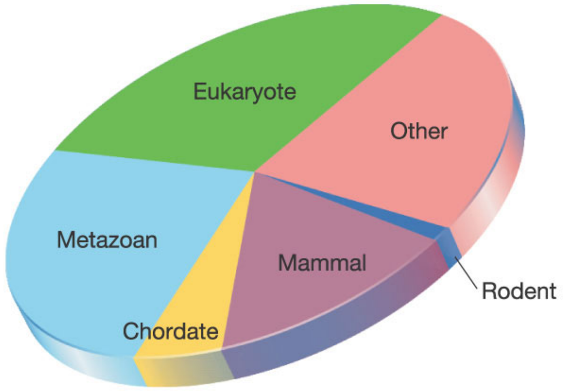


Sequence conservation implies function



Alignment is the key to

- Finding important regions
- Determining function
- Uncovering evolutionary events



Sequence Alignment

```
AGGCTATCACCTGACCTCCAGGCCGATGCCC
TAGCTATCACGACCGCGGTCGATTTGCCCGAC
```

```
-AGGCTATCACCTGACCTCCAGGCCGA--TGCCC---
TAG-CTATCAC--GACCGC--GGTCGATTTGCCCGAC
```

Definition

Given two strings $x = x_1x_2...x_M$, $y = y_1y_2…y_N$,

an alignment is an assignment of gaps to positions 0,…, N in x, and 0,…, N in y, so as to line up each letter in one sequence with either a letter, or a gap in the other sequence

What is a good alignment?

```
AGGCTAGTT,
AGCGAAGTTT
```

```
AGGCTAGTT-
AGCGAAGTTT
```

6 matches, 3 mismatches, 1 gap

```
AGGCTA-GTT-
AG-CGAAGTTT
```

7 matches, 1 mismatch, 3 gaps

```
AGGC-TA-GTT-
AG-CG-AAGTTT
```

7 matches, 0 mismatches, 5 gaps

Scoring Function

- Sequence edits:

	AGGCCTC
▪ Mutations	AGGACTC
▪ Insertions	AGGGCCTC
▪ Deletions	AGG . CTC

Scoring Function:

Match:	+m
Mismatch:	-s
Gap:	-d

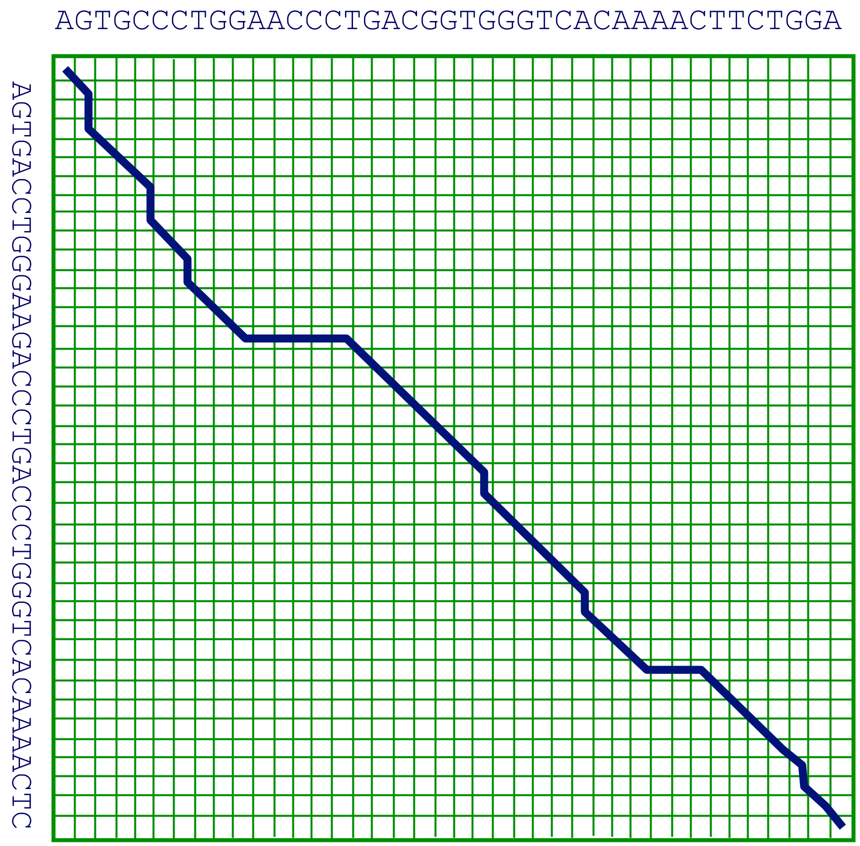
Alternative definition:

minimal edit distance

"Given two strings x, y, find minimum # of edits (insertions, deletions, mutations) to transform one string to the other"

Score F = (# matches) × m - (# mismatches) × s – (#gaps) × d

How do we compute the best alignment?


AGTGCCCTGGAACCCTGACGGTGGGTCACAAAACTTCTGGA
AGTGACCTGGGAAGACCCTGACCCTGGGTCACAAAACTC

Too many possible alignments:

$$>> 2^N$$

(exercise)

Alignment is additive

Observation:

The score of aligning $x_1 \ldots\ldots x_M$

$y_1 \ldots\ldots y_N$

is additive

Say that $x_1 \ldots x_i$ $x_{i+1} \ldots x_M$

aligns to $y_1 \ldots y_j$ $y_{j+1} \ldots y_N$

The two scores add up:

$$F(x[1:M], y[1:N]) = F(x[1:i], y[1:j]) + F(x[i+1:M], y[j+1:N])$$

Dynamic Programming

- There are only a polynomial number of subproblems
  - Align $x_1 \ldots x_i$ to $y_1 \ldots y_j$
- Original problem is one of the subproblems
  - Align $x_1 \ldots x_M$ to $y_1 \ldots y_N$
- Each subproblem is easily solved from smaller subproblems
  - We will show next
- Then, we can apply *Dynamic Programming!!!*

Let

$F(i, j)$ = optimal score of aligning

$x_1 \ldots\ldots x_i$

$y_1 \ldots\ldots y_j$

F is the DP "Matrix" or "Table"

"Memoization"

Dynamic Programming (cont'd)

Notice three possible cases:

1. x_i aligns to y_j

   $x_1 \ldots\ldots x_{i-1} \quad x_i$

   $y_1 \ldots\ldots y_{j-1} \quad y_j$

$$F(i, j) = F(i-1, j-1) + \begin{cases} m, \text{ if } x_i = y_j \\ -s, \text{ if not} \end{cases}$$

2. x_i aligns to a gap

   $x_1 \ldots\ldots x_{i-1} \quad x_i$

   $y_1 \ldots\ldots y_j \quad -$

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

3. y_j aligns to a gap

   $x_1 \ldots\ldots x_i \quad -$

   $y_1 \ldots\ldots y_{j-1} \quad y_j$

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

Dynamic Programming (cont'd)

How do we know which case is correct?

Inductive assumption:

$F(i, j - 1), F(i - 1, j), F(i - 1, j - 1)$ are optimal

Then,

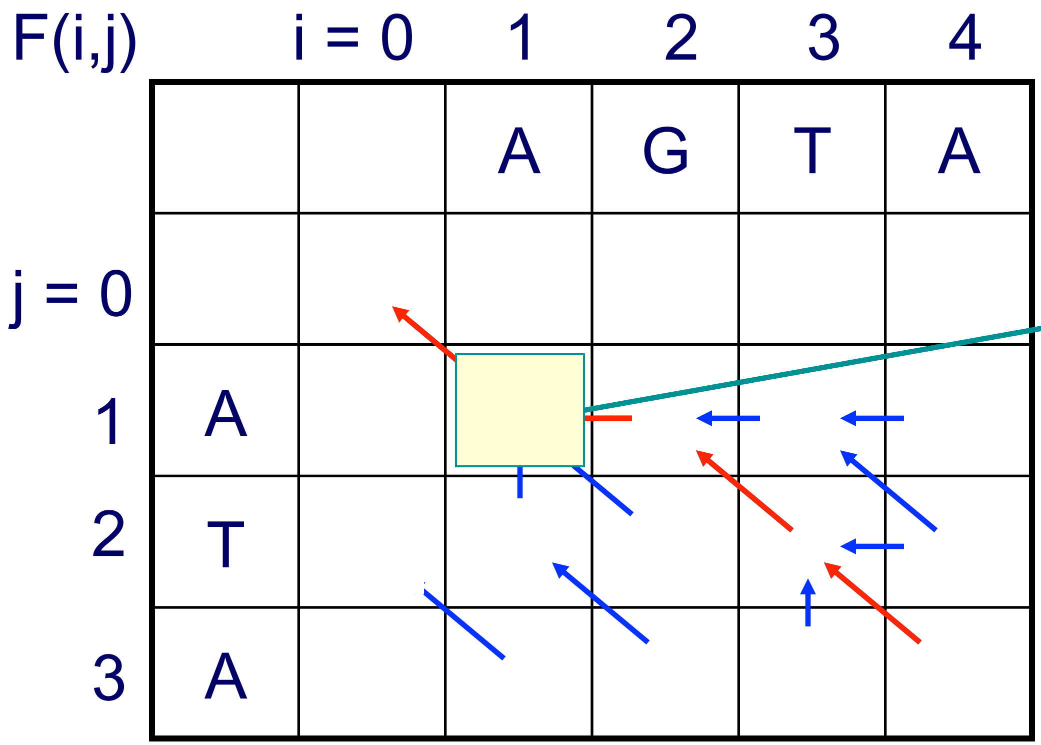
$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i-1, j) - d \\ F(i, j-1) - d \end{cases}$$

Where $s(x_i, y_j) = m$, if $x_i = y_j$; $-s$, if not

Example

x = AGTA

y = ATA

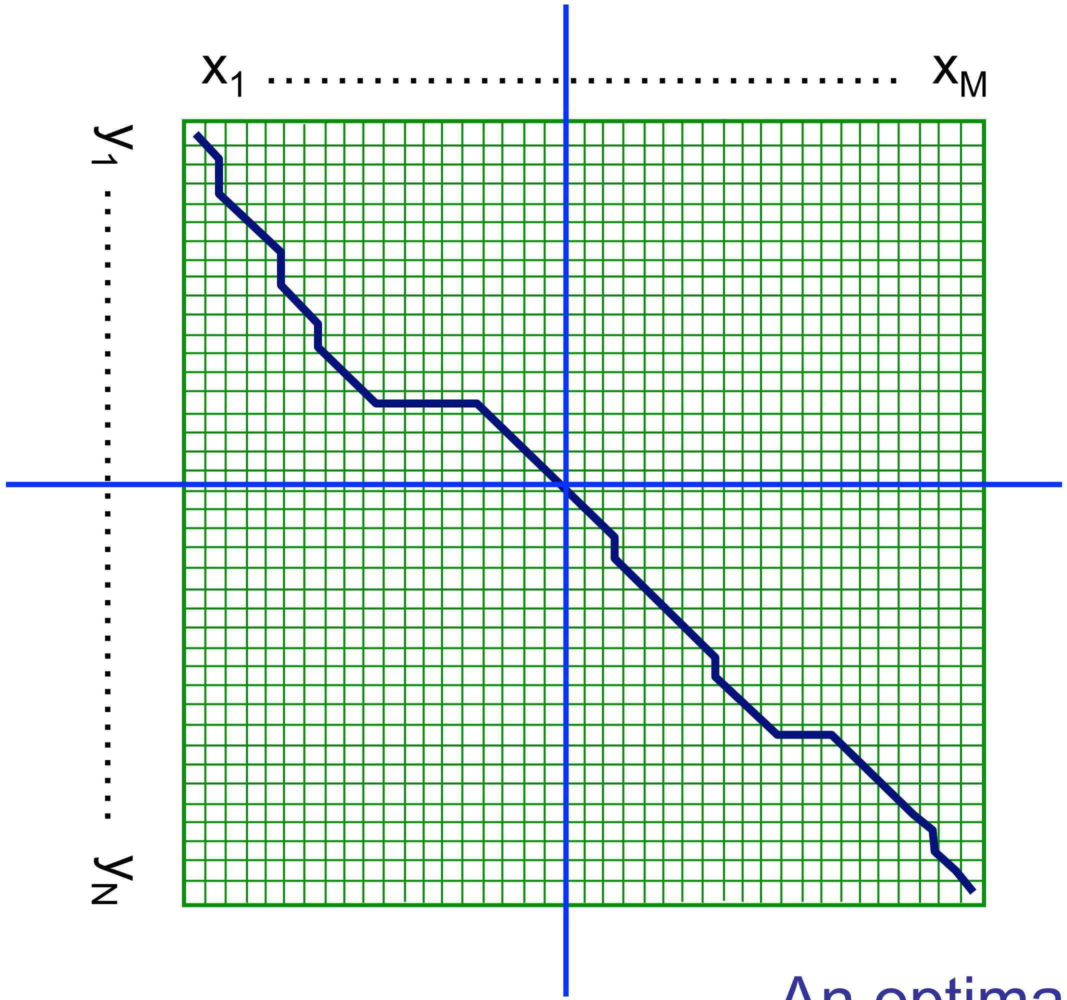


Procedure to output Alignment

- Follow the backpointers
- When diagonal,
  OUTPUT x_i, y_j
- When up,
  OUTPUT y_j
- When left,
  OUTPUT x_i

```
A G T A
A - T A
```

The Needleman-Wunsch Matrix



Every nondecreasing path

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

corresponds to
an alignment
of the two sequences

An optimal alignment is composed of optimal subalignments

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……M
    For each   j = 1……N
```

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

$$Ptr(i, j) = \begin{cases} \text{DIAG,} & \text{if [case 1]} \\ \text{LEFT,} & \text{if [case 2]} \\ \text{UP,} & \text{if [case 3]} \end{cases}$$

3. Termination. F(M, N) is the optimal score, and from Ptr(M, N) can trace back optimal alignment

Performance

- Time:

  O(NM)

- Space:

  O(NM)

- Later we will cover more efficient methods

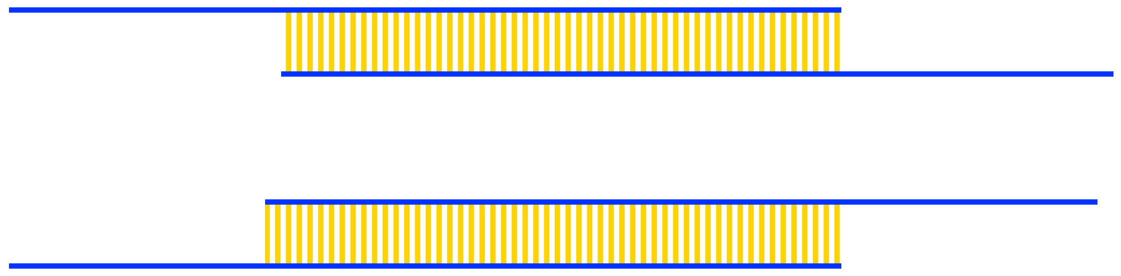
A variant of the basic algorithm:

- Maybe it is OK to have an unlimited # of gaps in the beginning and end:

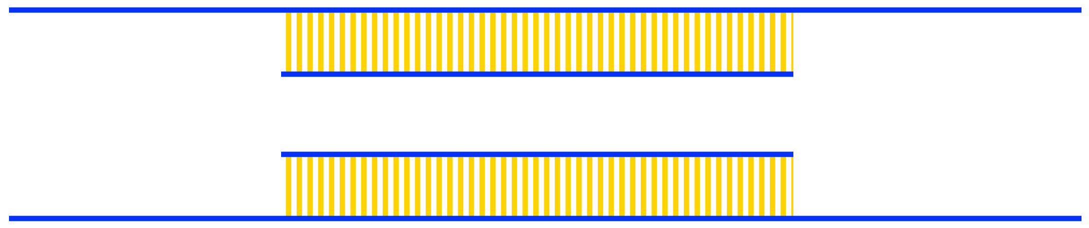
```
----------CTATCACCTGACCTCCAGGCCGATGCCCCTTCCGGC
          |||||||  |||| |  || ||
GCGAGTTCATCTATCAC--GACCGC--GGTCG--------------
```

- Then, we don’t want to penalize gaps in the ends

Different types of overlaps

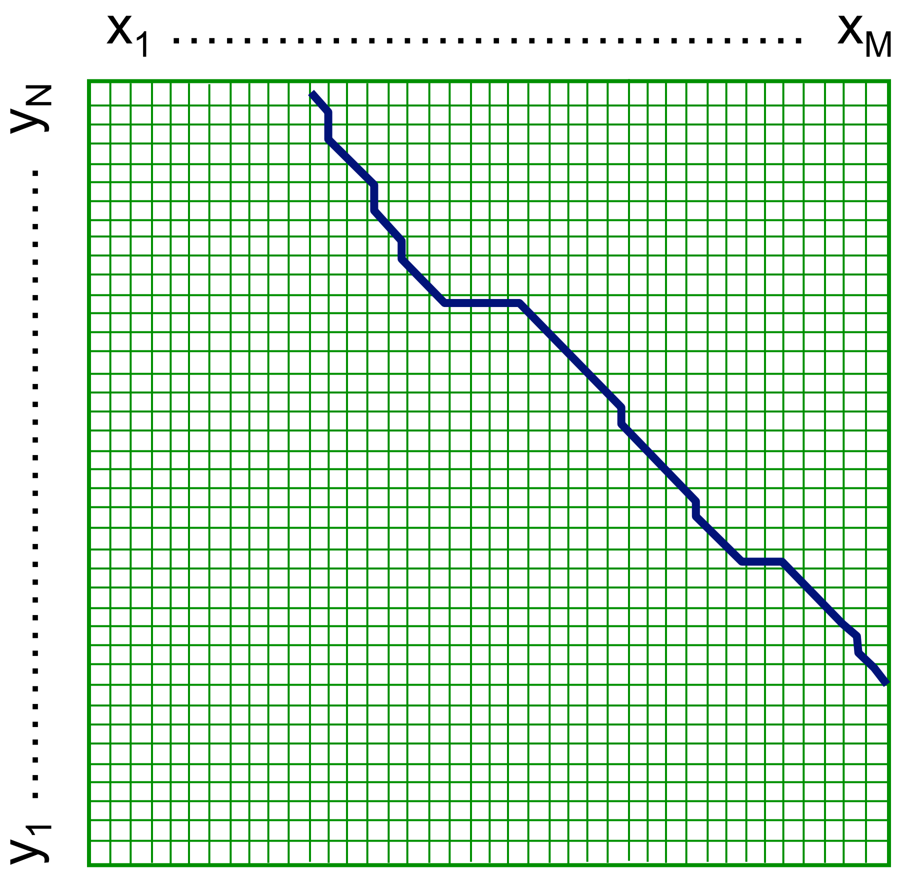


Example:
2 overlapping"*reads*" from a sequencing project



Example:
Search for a mouse gene within a human chromosome

The Overlap Detection variant



Changes:

1. Initialization

   For all i, j,

   $F(i, 0) = 0$

   $F(0, j) = 0$

2. Termination

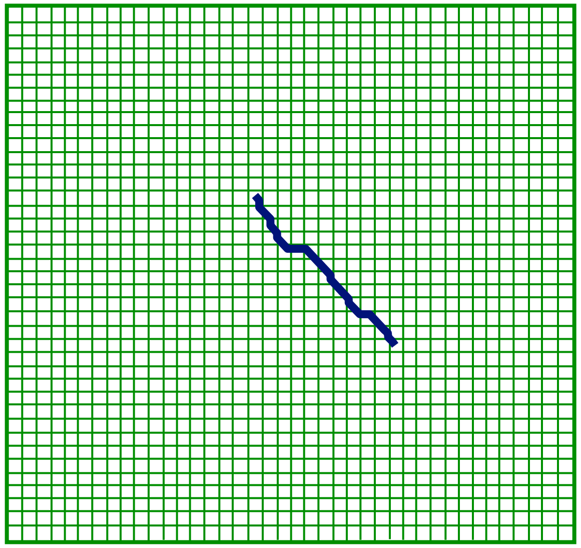
$$F_{OPT} = \max \begin{cases} \max_i F(i, N) \\ \max_j F(M, j) \end{cases}$$

The local alignment problem

Given two strings $x = x_1 \ldots\ldots x_M$,
$y = y_1 \ldots\ldots y_N$

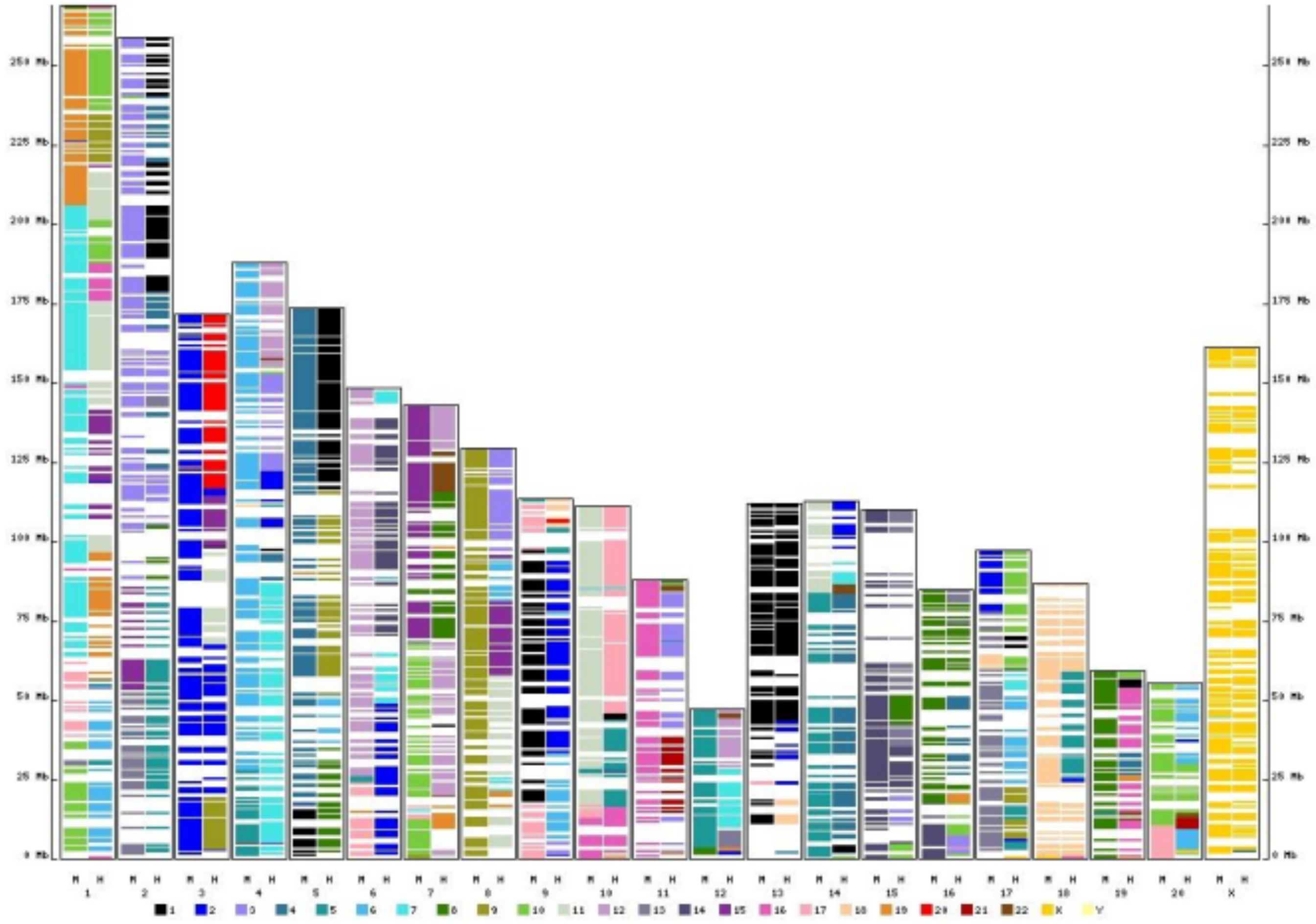
Find substrings x', y' whose similarity
(optimal global alignment value)
is maximum

x = aaaacccccgggggtta
y = ttcccgggaaccaacc



Why local alignment

- Genes are shuffled between genomes



Cross-species genome similarity

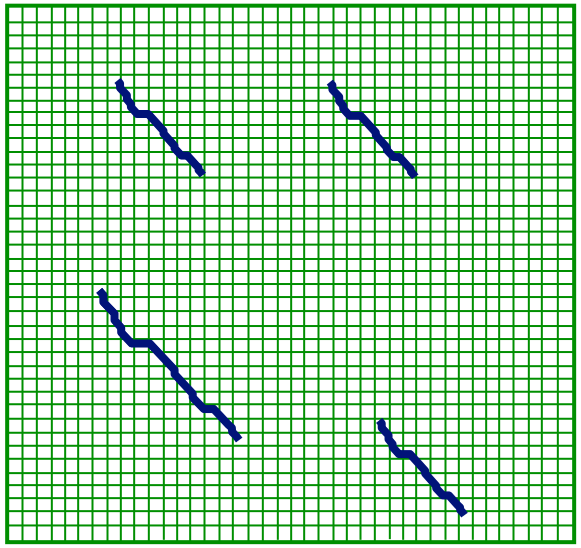
- 98% of genes are conserved between any two mammals
- >70% average similarity in protein sequence

```
hum_a : GTTGACAATAGAGGGTCTGGCAGAGGCTC--------------------- @ 57331/400001
mus_a : GCTGACAATAGAGGGGCTGGCAGAGGCTC--------------------- @ 78560/400001
rat_a : GCTGACAATAGAGGGGCTGGCAGAGACTC--------------------- @ 112658/369938
fug_a : TTTGTTGATGGGGAGCGTGCATTAATTTCAGGCTATTGTTAACAGGCTCG @ 36008/68174

hum_a : CTGGCCGCGGTGCGGAGCGTCTGGAGCGGAGCACGCGCTGTCAGCTGGTG @ 57381/400001
mus_a : CTGGCCCCGGTGCGGAGCGTCTGGAGCGGAGCACGCGCTGTCAGCTGGTG @ 78610/400001
rat_a : CTGGCCCCGGTGCGGAGCGTCTGGAGCGGAGCACGCGCTGTCAGCTGGTG @ 112708/369938
fug_a : TGGGCCGAGGTGTTGGATGGCCTGAGTGAAGCACGCGCTGTCAGCTGGCG @ 36058/68174

hum_a : AGCGCACTCTCCTTTCAGGCAGCTCCCCGGGGAGCTGTGCGGCCACATTT @ 57431/400001
mus_a : AGCGCACTCG-CTTTCAGGCCGCTCCCCGGGGAGCTGAGCGGCCACATTT @ 78659/400001
rat_a : AGCGCACTCG-CTTTCAGGCCGCTCCCCGGGGAGCTGCGCGGCCACATTT @ 112757/369938
fug_a : AGCGCTCGCG------------------------AGTCCCTGCCGTGTCC @ 36084/68174

hum_a : AACACCATCATCACCCCTCCCCGGCCTCCTCAACCTCGGCCTCCTCCTCG @ 57481/400001
mus_a : AACACCGTCGTCA-CCCTCCCCGGCCTCCTCAACCTCGGCCTCCTCCTCG @ 78708/400001
rat_a : AACACCGTCGTCA-CCCTCCCCGGCCTCCTCAACCTCGGCCTCCTCCTCG @ 112806/369938
fug_a : CCGAGGACCCTGA------------------------------------- @ 36097/68174
```

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

The Smith-Waterman algorithm

Idea: Ignore badly aligning regions



Modifications to Needleman-Wunsch:

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

Iteration:

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

The Smith-Waterman algorithm

Termination:

1. If we want the best local alignment…

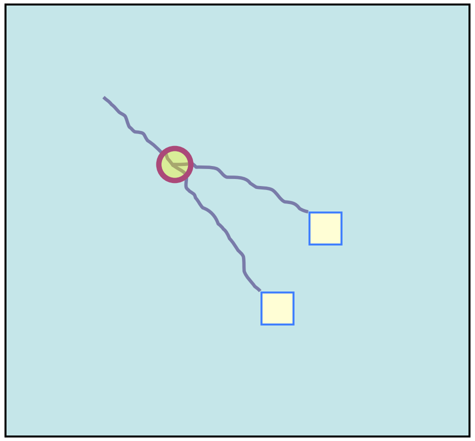
$$F_{OPT} = \max_{i,j} F(i, j)$$

Find F_{OPT} and trace back

2. If we want all local alignments scoring > t

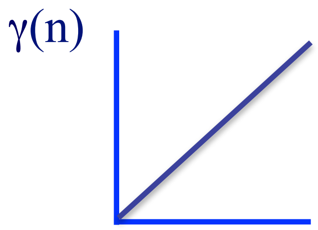
?? For all i, j find F(i, j) > t, and trace back?

Complicated by overlapping local alignments



Waterman–Eggert '87: ***find all non-overlapping local alignments with minimal recalculation of the DP matrix***

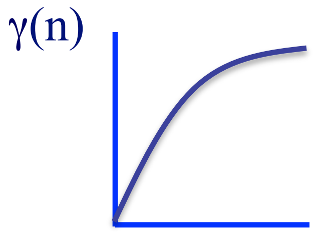
Scoring the gaps more accurately



Current model:

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

However, gaps usually occur in bunches



Concave gap penalty function $\gamma(n)$
(aka Convex $-\gamma(n)$):

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

Convex gap dynamic programming

Initialization: same

Iteration:

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

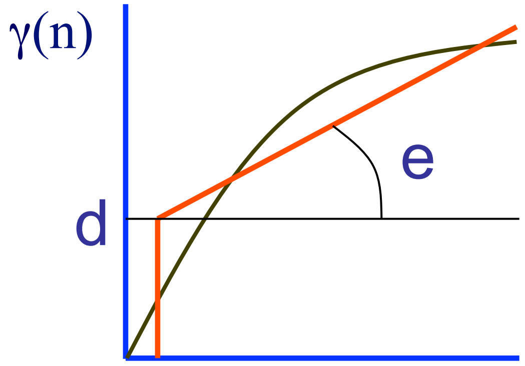
Termination: same

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

Space: $O(NM)$

Compromise: affine gaps

$\gamma(n) = \underbrace{d}_{\text{gap open}} + (n - 1)\times \underbrace{e}_{\text{gap extend}}$



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 $x_1\ldots x_i$ to $y_1\ldots y_j$
if x_i aligns to y_j

G(i, j): score **if** x_i aligns to a gap after y_j
H(i, j): score **if** y_j aligns to a gap after x_i

V(i, j) = best score of alignment $x_1\ldots x_i$ to $y_1\ldots y_j$

Needleman-Wunsch with affine gaps

Why do we need matrices F, G, H?

Because, perhaps

G(i, j) < **V**(i, j)

(it is best to align x_i to y_j if we were aligning only $x_1 \ldots x_i$ to $y_1 \ldots y_j$ and not the rest of x, y),

but on the contrary

G(i, j) – e > **V**(i, j) – d

(i.e., had we "fixed" our decision that x_i aligns to y_j, we could regret it at the next step when aligning $x_1 \ldots x_{i+1}$ to $y_1 \ldots y_j$)

Add -d

G(i+1, j) = **F**(i, j) – d

Add -e

G(i+1, j) = **G**(i, j) – e

Needleman-Wunsch with affine gaps

Initialization:

$V(i, 0) = d + (i - 1) \times e$
$V(0, j) = d + (j - 1) \times e$

Iteration:

$V(i, j) = \max\{ F(i, j), G(i, j), H(i, j) \}$

$F(i, j) = V(i - 1, j - 1) + s(x_i, y_j)$

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

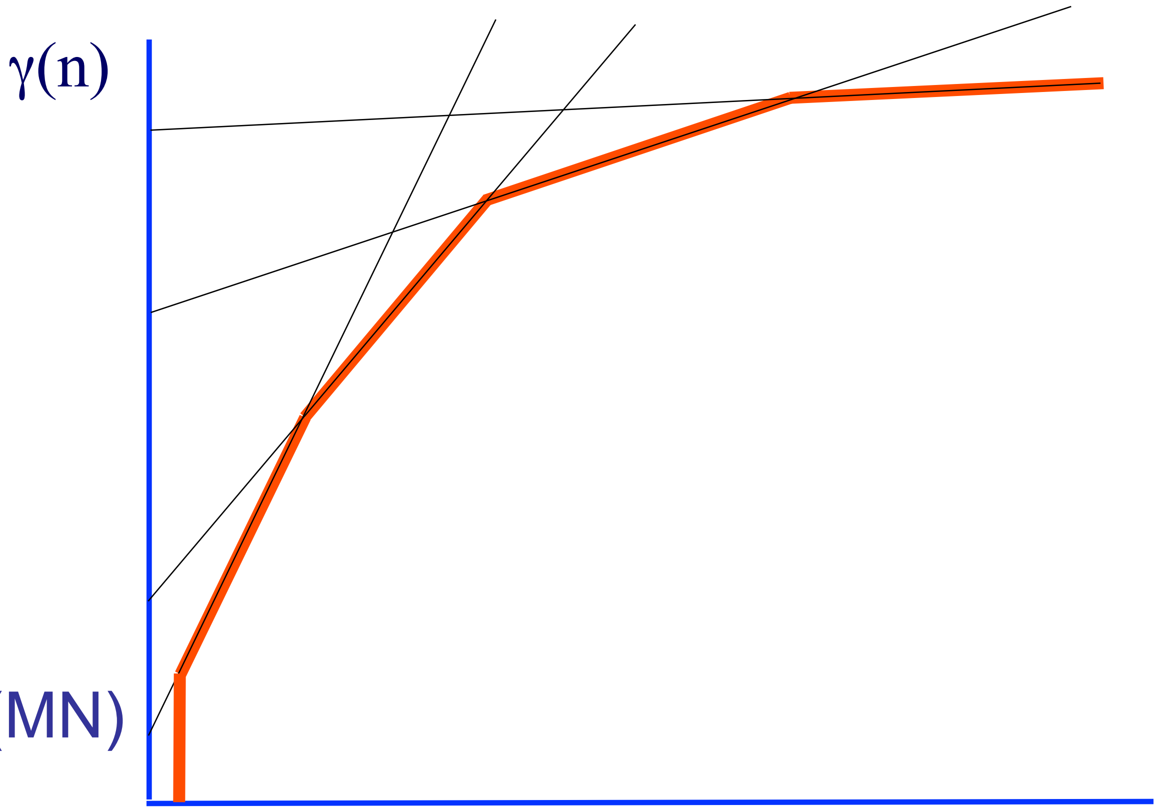
$$H(i, j) = \max \begin{cases} V(i, j - 1) - d \\ H(i, j - 1) - e \end{cases}$$

Termination: $V(i, j)$ has the best alignment

Time?
Space?

To generalize a bit…

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



….in time O(MN)

Bounded Dynamic Programming

Assume we know that x and y are very similar

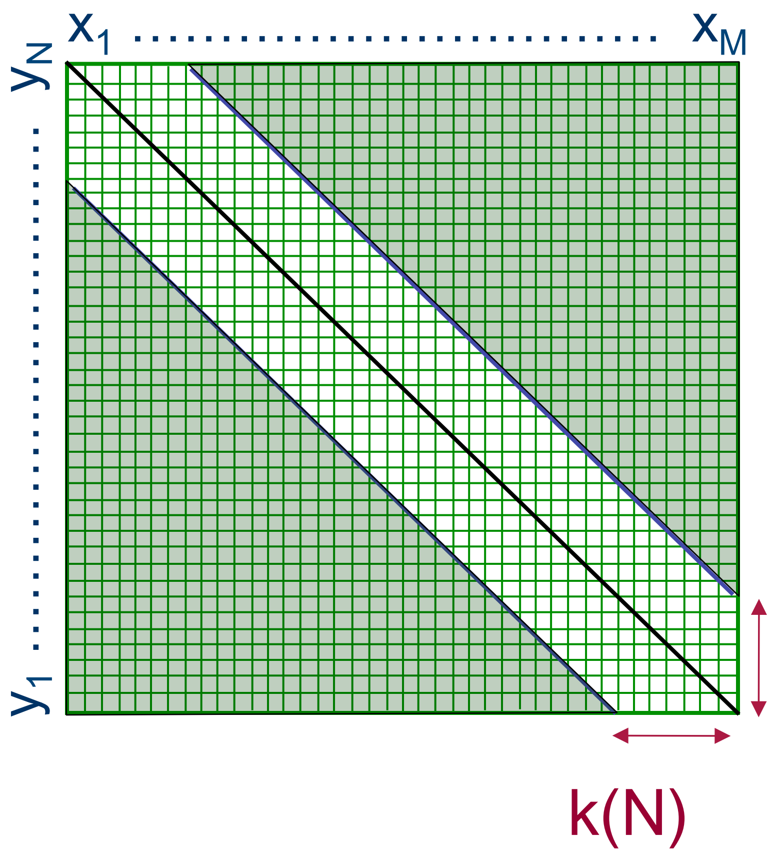
Assumption: # gaps(x, y) $< k(N)$

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

We can align x and y more efficiently:

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

Bounded Dynamic Programming



Initialization:

F(i,0), F(0,j) undefined for i, j > k

Iteration:

```
For i = 1…M
  For j = max(1, i – k)…min(N, i+k)
```

$$F(i, j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j) \\ F(i, j-1) - d, \text{ if } j > i - k(N) \\ F(i-1, j) - d, \text{ if } j < i + k(N) \end{cases}$$

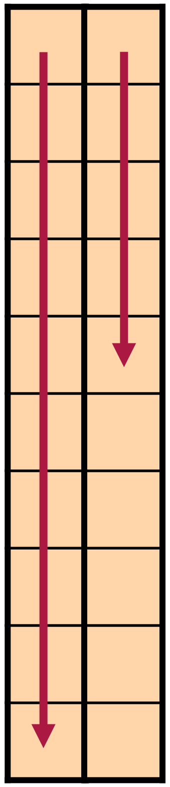
Termination: same

Easy to extend to the affine gap case

Outline

- Linear-Space Alignment
- BLAST – local alignment search
- Ultra-fast alignment for (human) genome resequencing

Linear-Space Alignment



Subsequences and Substrings

Definition A string x’ is a ***substring*** of a string x,

if x = ux’v for some prefix string u and suffix string v

(similarly, $x' = x_i \ldots x_j$, for some $1 \le i \le j \le |x|$)

A string x’ is a ***subsequence*** of a string x

if x’ can be obtained from x by deleting 0 or more letters

($x' = x_{i1} \ldots x_{ik}$, for some $1 \le i_1 \le \ldots \le i_k \le |x|$)

Note: a substring is always a subsequence

Example:	**x = abracadabra**	
	y = cadabr;	***substring***
	z = brcdbr;	***subseqence, not substring***

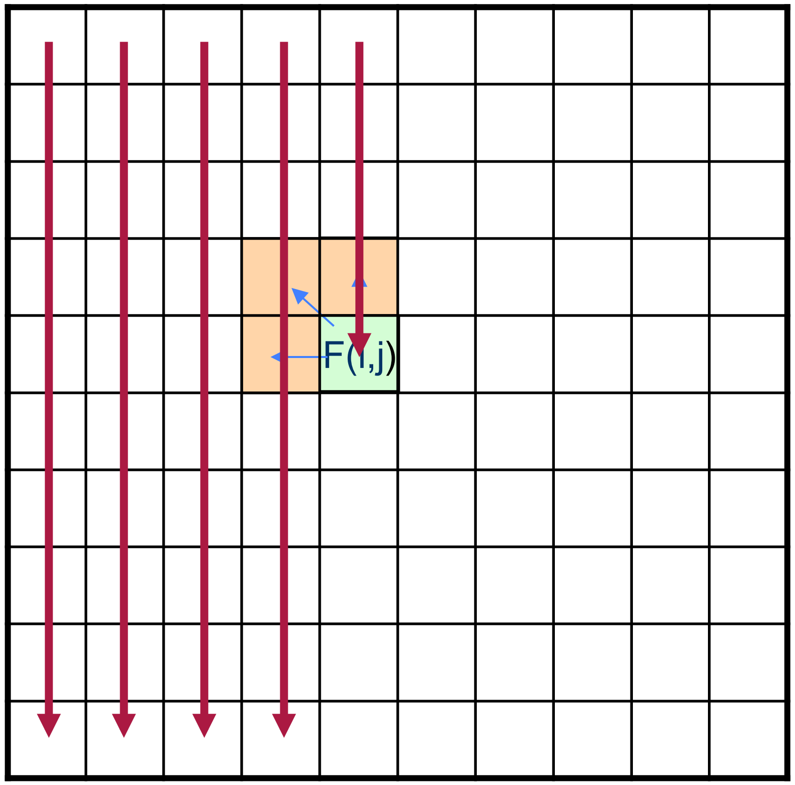
Hirschberg's algortihm

Given a set of strings x, y,…, a ***common subsequence*** is a string u that is a subsequence of all strings x, y, …

- Longest common subsequence
  - Given strings $x = x_1 x_2 \ldots x_M$, $y = y_1 y_2 \ldots y_N$,
  - Find longest common subsequence $u = u_1 \ldots u_k$
- Algorithm:
  - $$F(i, j) = \max \begin{cases} F(i-1, j) \\ F(i, j-1) \\ F(i-1, j-1) + [1, \text{ if } x_i = y_j;\ 0 \text{ otherwise}] \end{cases}$$
  - Ptr(i, j) = *(same as in N-W)*
  - Termination: trace back from Ptr(M, N), and prepend a letter to u whenever
    - Ptr(i, j) = DIAG **and** $F(i-1, j-1) < F(i, j)$
- Hirschberg's original algorithm solves this in linear space

Introduction: Compute 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) = …
```

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 = accgg;

x^r = ggcca

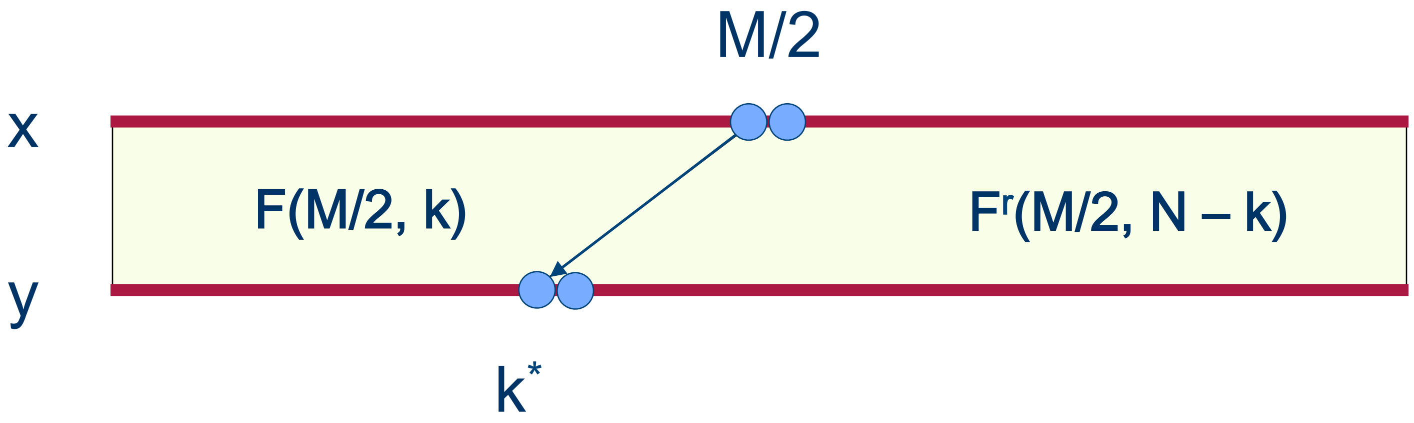
$F^r(i, j)$: optimal score of aligning $x^r_1 \ldots x^r_i$ & $y^r_1 \ldots y^r_j$

same as aligning $x_{M-i+1} \ldots x_M$ & $y_{N-j+1} \ldots y_N$

Linear-space alignment

Lemma: (assume M is even)

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



Example:

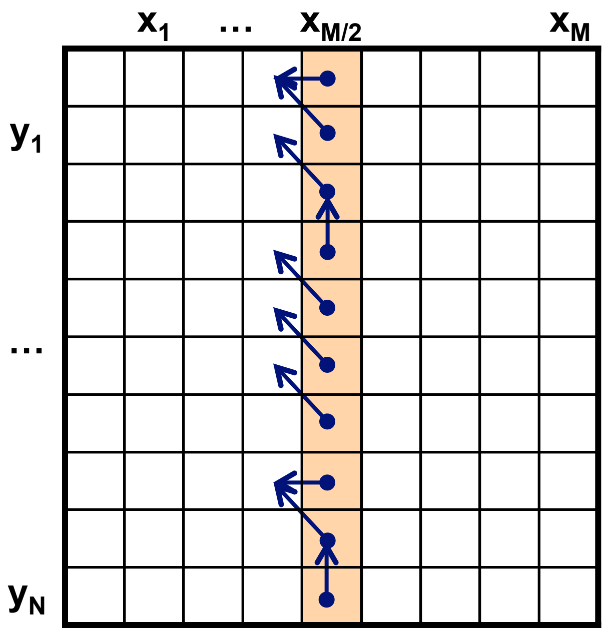
```
ACC-GGTGCCCAGGACTG--CAT
ACCAGGTG----GGACTGGGCAG
```

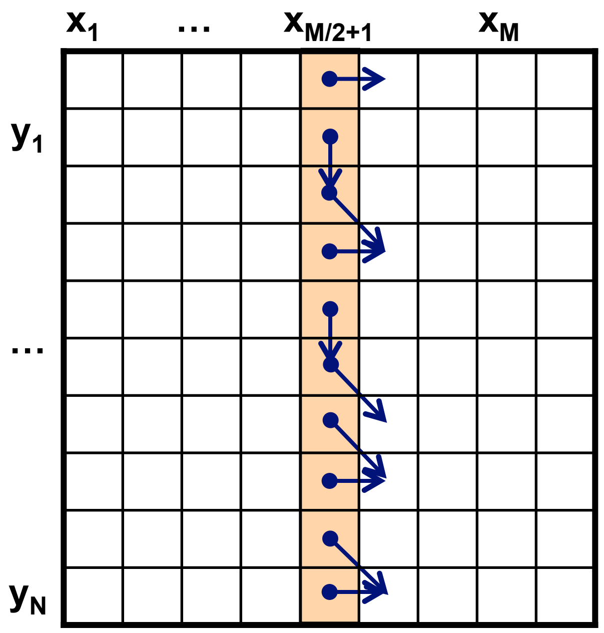
$k^* = 8$

Linear-space alignment

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

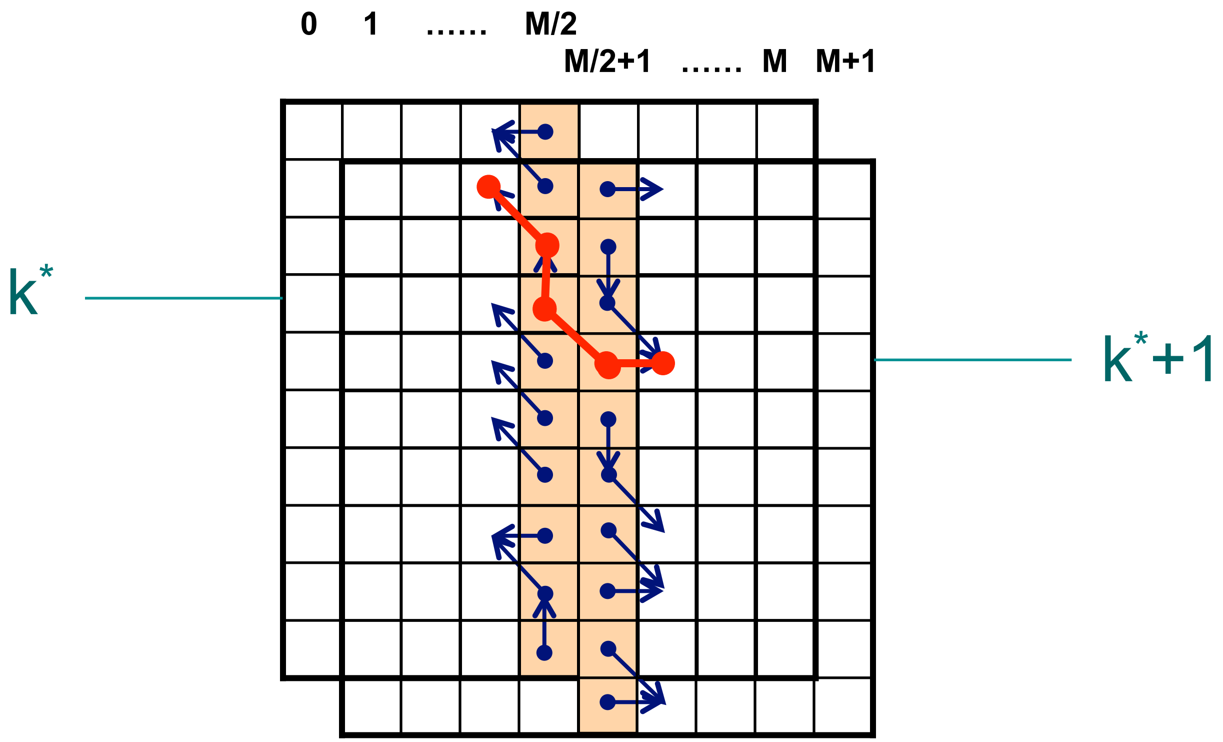
PLUS the backpointers





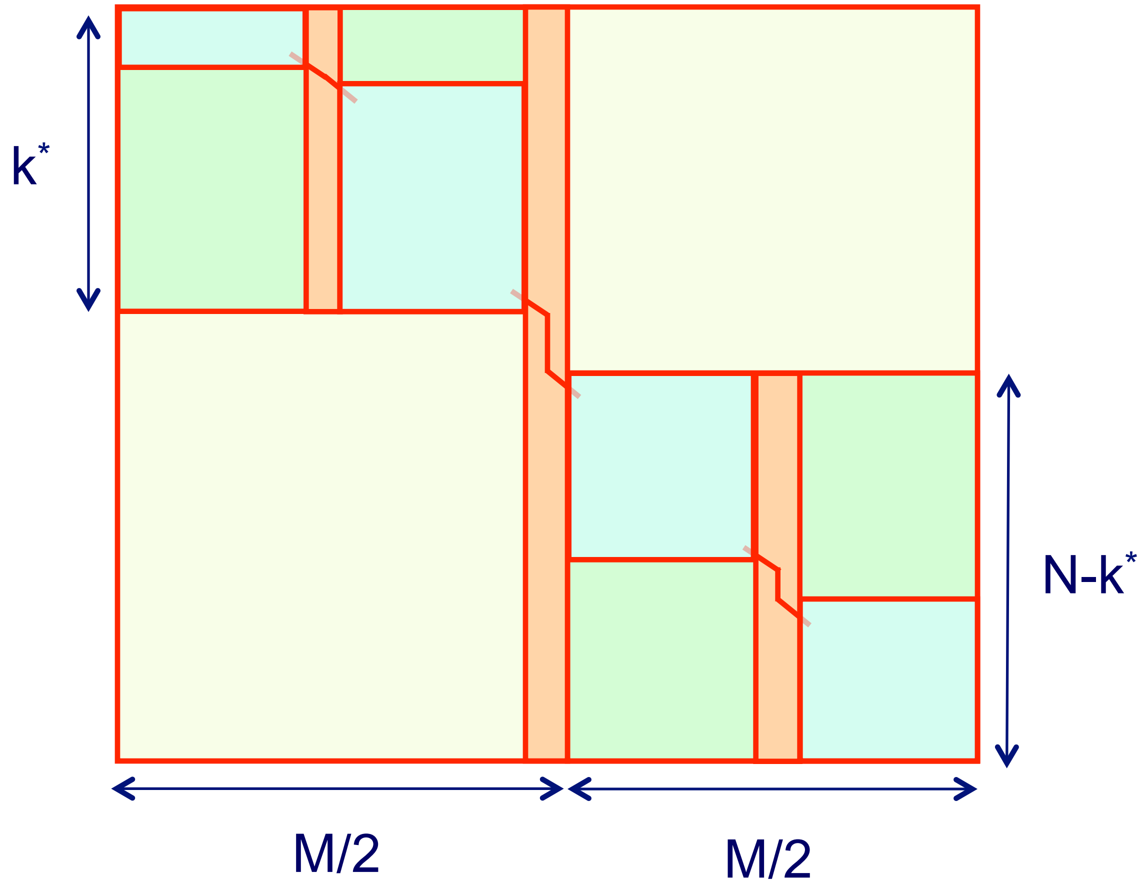
Linear-space alignment

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



Linear-space alignment

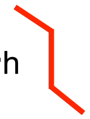
- Iterate this procedure to the left and right!



Linear-space alignment

Hirschberg's Linear-space algorithm:

```
MEMALIGN($l$, $l'$, $r$, $r'$):                    (aligns $x_l \ldots x_{l'}$ with $y_r \ldots y_{r'}$)

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,      $L_h$, entering column $h - 1$, exiting column $h$
     Let $k_1$ = pos'n at column $h - 2$ where $L_h$ enters
         $k_2$ = pos'n at column $h + 1$ where $L_h$ exits

3.   MEMALIGN($l$, $h - 2$, $r$, $k_1$)

4.   Output $L_h$

5.   MEMALIGN($h + 1$, $l'$, $k_2$, $r'$)
```



Top level call: MEMALIGN(1, M, 1, N)

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: $cMN + cMN/2 + cMN/4 + \ldots.. = 2cMN = O(MN)$

Total Space: $O(N)$ for computation,

$O(N + M)$ to store the optimal alignment