CSE 427 Comp Bio

Sequence Alignment

Sequence Alignment

What

Why

A Dynamic Programming Algorithm

Sequence Similarity: What

G G A C C A

T A C T A A G

T C C A A G

Sequence Similarity: What

G G A C C A

T A C T A A G

| | | | |

T C C – A A G

Sequence Similarity: Why

Bio

- Most widely used comp. tools in biology
- New sequence always compared to data bases
- **Similar sequences often have similar origin and/or function**
- Recognizable similarity after 10^8 –10^9 yr
- DNA sequencing & assembly

Other

- spell check/correct, diff, svn/git/…, plagiarism, …

BLAST Demo

http://www.ncbi.nlm.nih.gov/blast/

Taxonomy Report

```
root ..............................    64 hits   16 orgs
. Eukaryota .......................    62 hits   14 orgs [cellular organisms]
```

Try it!

pick any protein, e.g. hemoglobin, insulin, exportin,… BLAST to find distant relatives.

Alternate demo:

- go to http://www.uniprot.org/uniprot/O14980 "**Exportin-1"**
- find "BLAST" button about ½ way down page, under "Sequences", just above big grey box with the amino sequence of this protein
- click "go" button
- after a minute or 2 you should see the 1st of 10 pages of "hits" – matches to similar proteins in other species
- you might find it interesting to look at the species descriptions and the "identity" column (generally above 50%, even in species as distant from us as fungus -- extremely unlikely by chance on a 1071 letter sequence over a 20 letter alphabet)
- Also click any of the colored "alignment" bars to see the actual alignment of the human XPO1 protein to its relative in the other species – in 3-row groups (query 1st, the match 3rd, with identical letters highlighted in between)

```
` mphocystis disease virus .........     1 hits    1 orgs [Viruses; dsDNA viruses, no RNA …]
```

Terminology

String: ordered list of letters TATAAG

Prefix: consecutive letters from front

empty, T, TA, TAT, ...

Suffix: … from end

empty, G, AG, AAG, ...

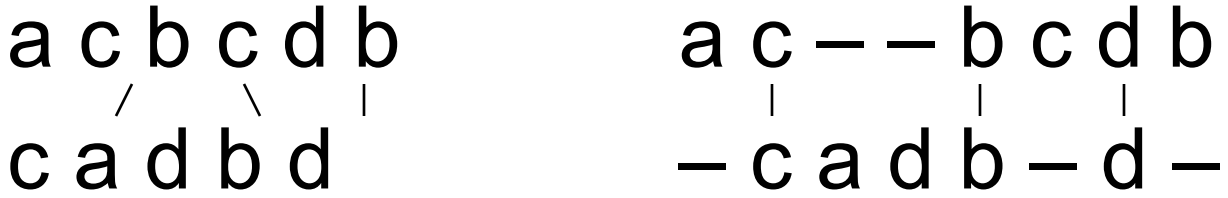
Substring: … from ends or middle

empty, TAT, AA, ...

Subsequence: ordered, nonconsecutive

TT, AAA, TAG, ...

Sequence Alignment



Defn: An *alignment* of strings S, T is a pair of strings S', T' (with dashes) s.t.

(1) |S'| = |T'|, and (|S| = "length of S")

(2) removing all dashes leaves S, T

Alignment Scoring

Mismatch = -1
Match = 2

```
a c b c d b        a c - - b c d b
c a d b d          - c a d b - d -
                   -1 2 -1 -1 2 -1 2 -1
                   Value = 3*2 + 5*(-1) = +1
```

The *score* of aligning (characters or dashes) x & y is $\sigma(x,y)$.

Value of an alignment $\sum_{i=1}^{|S'|} \sigma(S'[i],T'[i])$

An *optimal alignment:* one of max value

(Assume $\sigma(-,-) < 0$)

Optimal Alignment: A Simple Algorithm

for all subseqs A of S, B of T s.t. |A| = |B| **do**
  align A[i] with B[i], 1 ≤ i ≤ |A|
  align all other chars to spaces
  compute its value
  retain the max
end
output the retained alignment

S = abcd A = cd
T = wxyz B = xz

```
-abc-d   a-bc-d
w--xyz   -w-xyz
```

Analysis

Assume |S| = |T| = n

Cost of evaluating one alignment: ≥ n

How many alignments are there: $\geq \binom{2n}{n}$

pick n chars of S,T together

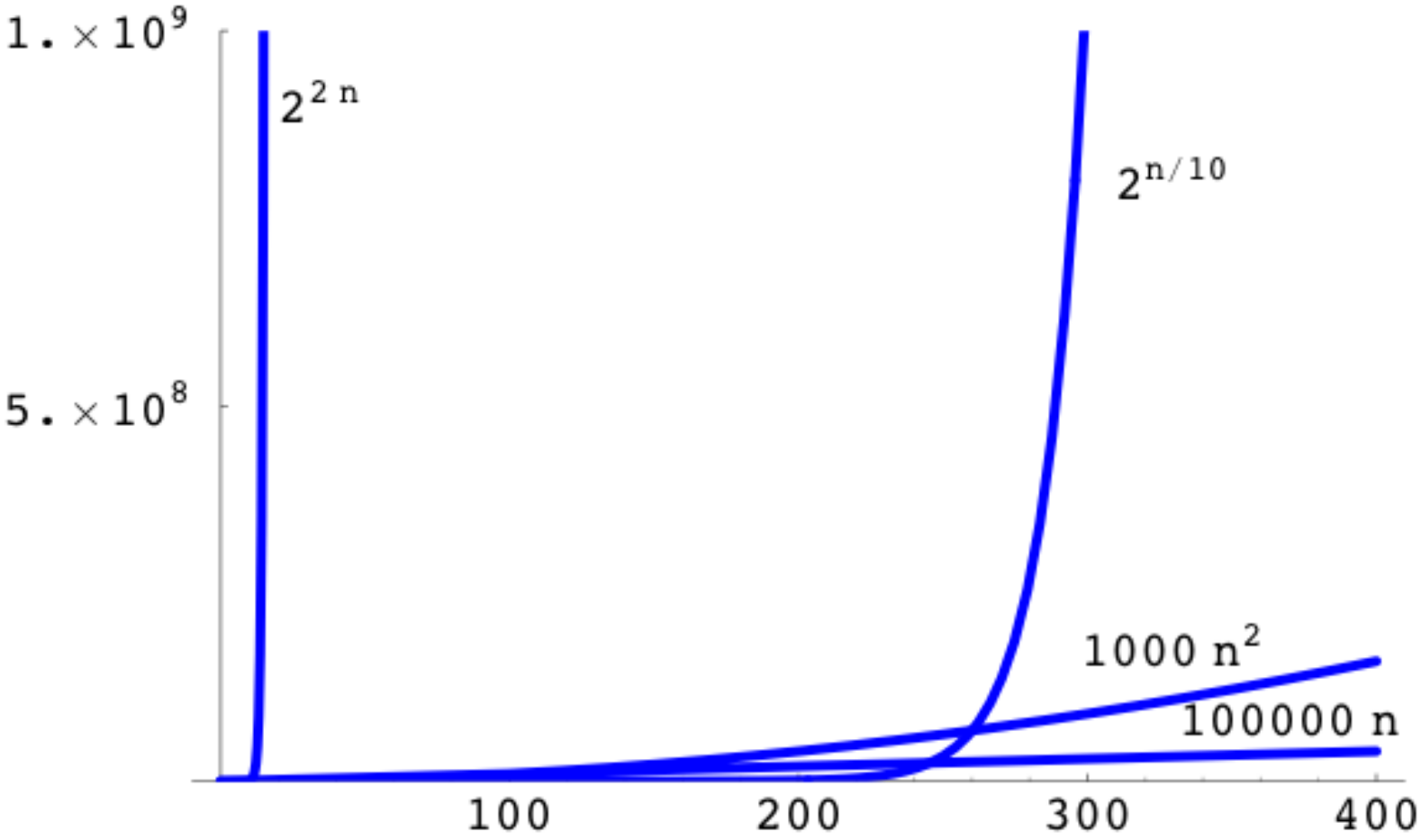
say k of them are in S

match these k to the k *un*picked chars of T

Total time: $\geq n\binom{2n}{n} > 2^{2n}$, for $n > 3$

E.g., for n = 20, time is > 2^{40} operations

Polynomial vs Exponential Growth



Alignment by Dynamic Programming?

Common Subproblems?

Plausible: probably re-considering alignments of various small substrings unless we're careful.

Optimal Substructure?

Plausible: left and right "halves" of an optimal alignment probably should be optimally aligned (though they obviously interact a bit at the interface).

(Both made rigorous below.)

Optimal Substructure
(In More Detail)

Optimal alignment *ends* in 1 of 3 ways:

- last chars of S & T aligned with each other
- last char of S aligned with dash in T
- last char of T aligned with dash in S
- (never align dash with dash; $\sigma(-, -) < 0$)

In each case, the *rest* of S & T should be *optimally* aligned to each other

Optimal Alignment in $O(n^2)$ via "Dynamic Programming"

Input: S, T, |S| = n, |T| = m

Output: value of optimal alignment

Easier to solve a "harder" problem:

V(i,j) = value of optimal alignment of

S[1], …, S[i] with T[1], …, T[j]

for all $0 \le i \le n$, $0 \le j \le m$.

Base Cases

V(i,0): first i chars of S all match dashes

$$V(i,0) = \sum_{k=1}^{i} \sigma(S[k], -)$$

V(0,j): first j chars of T all match dashes

$$V(0,j) = \sum_{k=1}^{j} \sigma(-, T[k])$$

General Case

Opt align of S[1], …, S[i] vs T[1], …, T[j]:

$$\begin{bmatrix} \sim\sim\sim\sim & S[i] \\ \sim\sim\sim\sim & T[j] \end{bmatrix}, \quad \begin{bmatrix} \sim\sim\sim\sim & S[i] \\ \sim\sim\sim\sim & - \end{bmatrix}, \text{ or } \begin{bmatrix} \sim\sim\sim\sim & - \\ \sim\sim\sim\sim & T[j] \end{bmatrix}$$

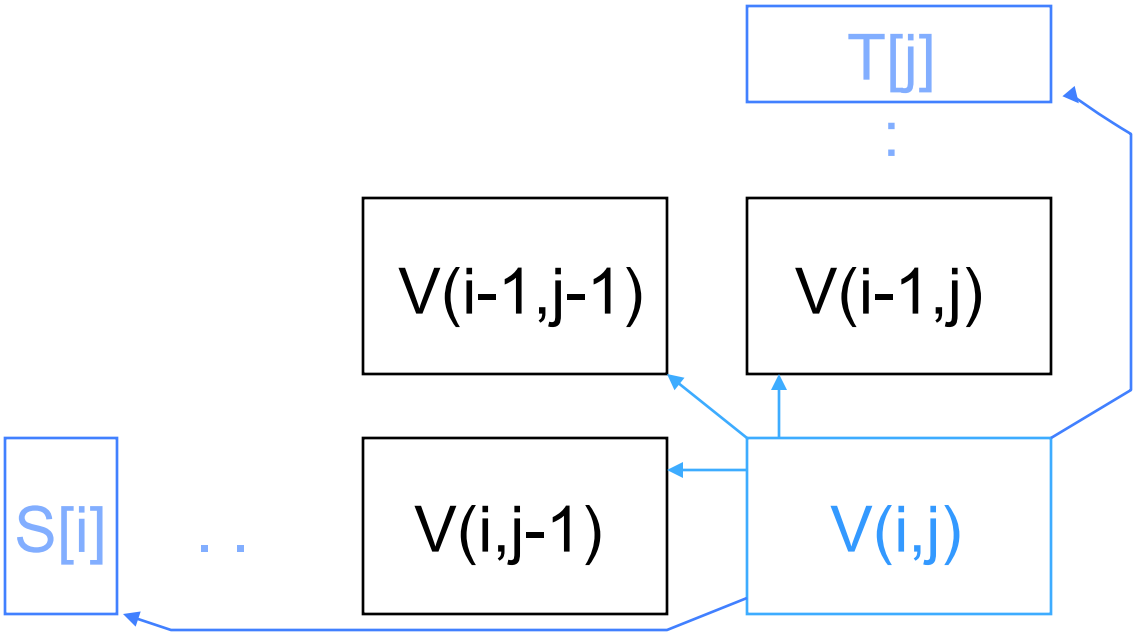
Opt align of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$

$$V(i,j) = \max \begin{cases} V(i\text{-}1,j\text{-}1) + \sigma(S[i],T[j]) \\ V(i\text{-}1,j) + \sigma(S[i], \text{ - }) \\ V(i,j\text{-}1) + \sigma(\text{ - }, T[j]) \end{cases},$$

for all $1 \le i \le n, \ 1 \le j \le m.$

Calculating One Entry

$$V(i,j) = \max \begin{cases} V(i\text{-}1,j\text{-}1) + \sigma(S[i],T[j]) \\ V(i\text{-}1,j) + \sigma(S[i], \text{ - }) \\ V(i,j\text{-}1) + \sigma(\text{ - }, T[j]) \end{cases}$$



Mismatch = -1
Match = 2

Example

	j	0	1	2	3	4	5	
i			c	a	d	b	d	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1						
2	c	-2						
3	b	-3						
4	c	-4						
5	d	-5						
6	b	-6						
	↑ S							

c
-

Score(c,-) = -1

Mismatch = -1
Match = 2

Example

	j	0	1	2	3	4	5	
i			c	a	d	b	d	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1						
2	c	-2						
3	b	-3						
4	c	-4						
5	d	-5						
6	b	-6						
	↑ S							

-
a

Score(-,a) = -1

Mismatch = -1
Match = 2

Example

	j	0	1	2	3	4	5	
i			c	a	d	b	d	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1						
2	c	-2						
3	b	-3						
4	c	-4						
5	d	-5						
6	b	-6						
	↑S							

-	-
a	c

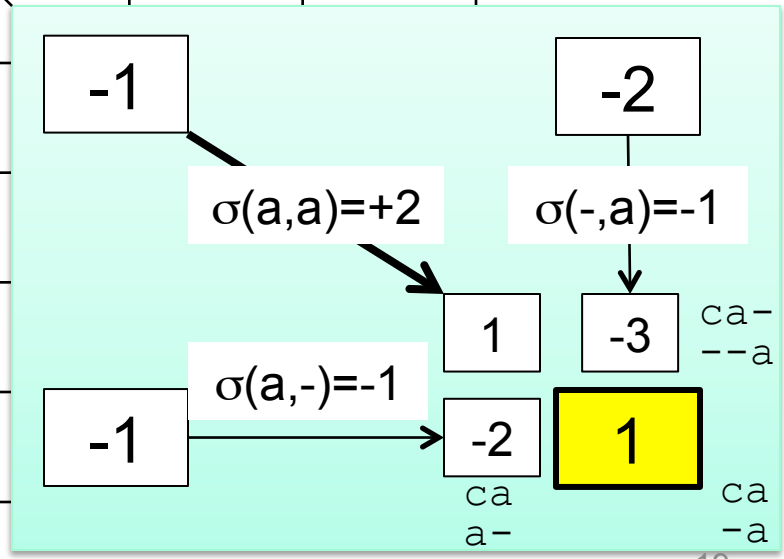
-1

Score(-,c) = -1

Mismatch = -1
Match = 2

Example

	j	0	1	2	3	4	5	
i			c	a	d	b	d	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1	-1	1				
2	c	-2						
3	b	-3						
4	c	-4						
5	d	-5						
6	b	-6						
		S						



Mismatch = -1
Match = 2

Example

	j	0	1	2	3	4	5
i			c	a	d	b	d
0		0	-1	-2	-3	-4	-5
1	a	-1	-1	1			
2	c	-2	1				
3	b	-3					
4	c	-4					
5	d	-5					
6	b	-6					

←T

Time = O(mn)

↑S

Mismatch = -1
Match = 2

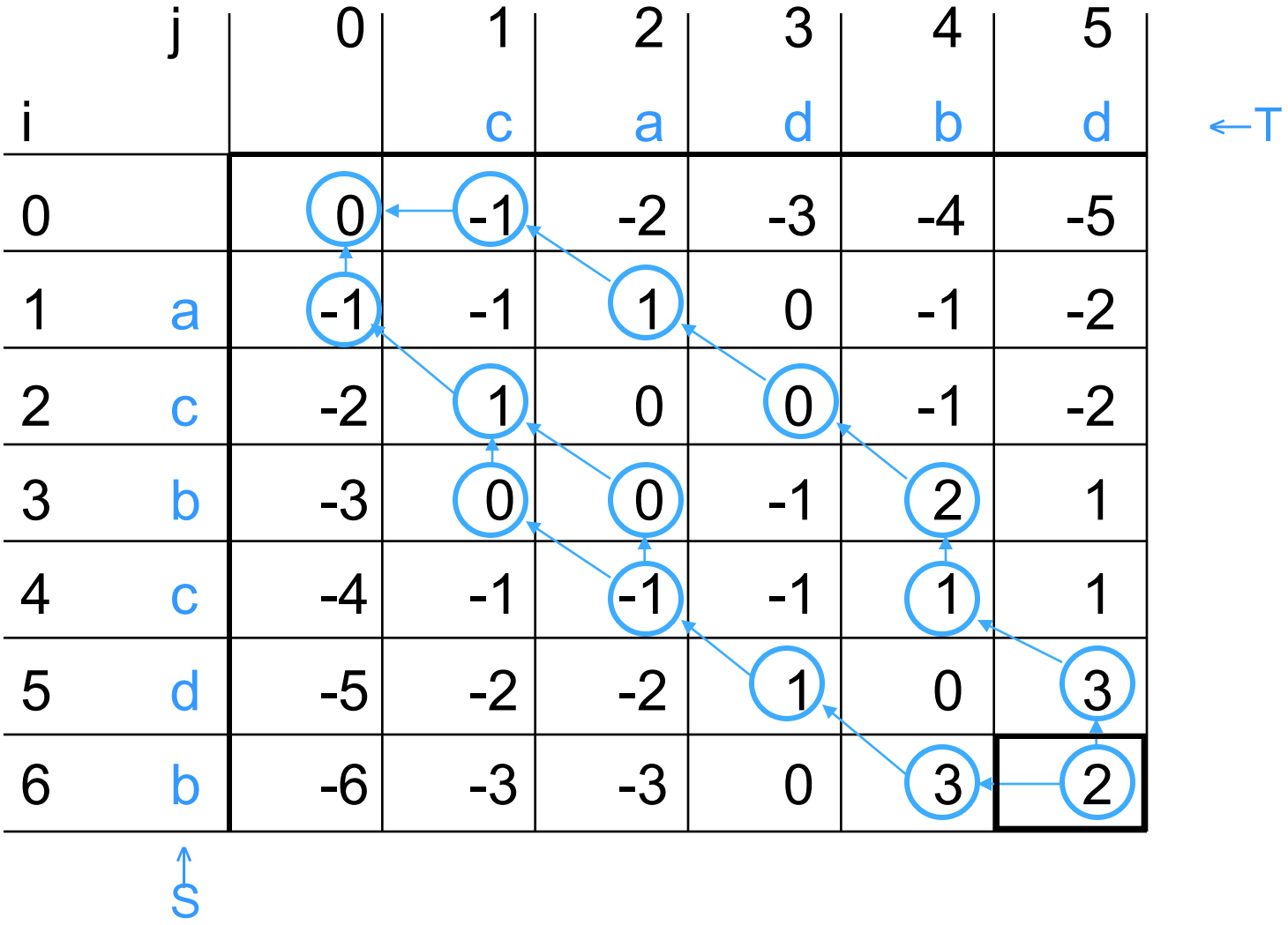
Example

	j	0	1	2	3	4	5	
i			c	a	d	b	d	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1	-1	1	0	-1	-2	
2	c	-2	1	0	0	-1	-2	
3	b	-3	0	0	-1	2	1	
4	c	-4	-1	-1	-1	1	1	
5	d	-5	-2	-2	1	0	3	
6	b	-6	-3	-3	0	3	2	

↑S

Finding Alignments: Trace Back

Arrows = (ties for) max in V(i,j); 3 LR-to-UL paths = 3 optimal alignments



Complexity Notes

Time = O(mn), (value and alignment)

Space = O(mn)

Easy to get value in Time = O(mn) and Space = O(min(m,n))

Possible to get value *and alignment* in Time = O(mn) and Space =O(min(m,n))

Significance of Alignments

Is “42” a good score?

Compared to what?

Usual approach: compared to a specific “null model”, such as “random sequences”

More on this later; a taste today, for use in next HW

Overall Alignment Significance, II
Empirical (via randomization)

You just searched with x, found "good" score for x:y

Generate N random "y-like" sequences (say N = 10^3 - 10^6)

Align x to each & score

If k of them have better score than alignment of x to y, then the (empirical) probability of a chance alignment as good as observed x:y alignment is (k+1)/(N+1)

e.g., if 0 of 99 are better, you can say "estimated p < .01"

How to generate "random y-like" seqs? Scores depend on:

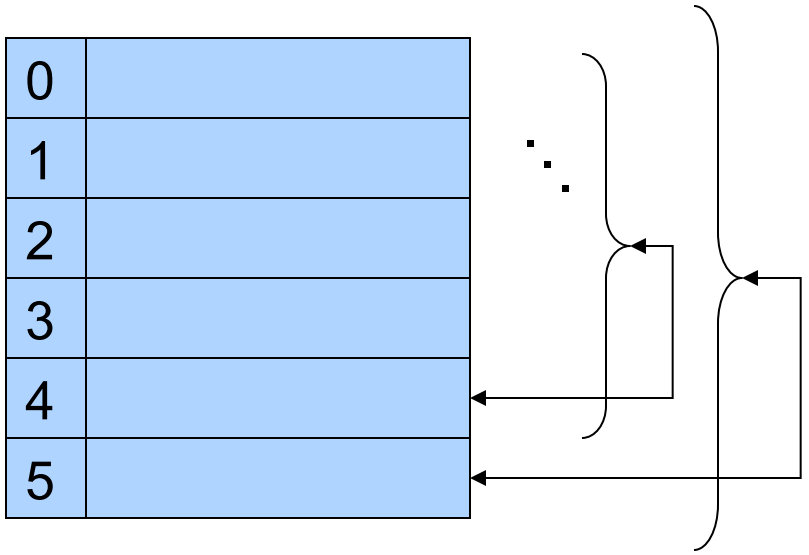
Length, so use same length as y

Sequence composition, so uniform 1/20 or 1/4 is a bad idea; even background p_i can be dangerous

Better idea: *permute* y N times

Generating Random Permutations

```
for (i = n-1; i > 0; i--){
    j = random(0..i);
    swap X[i] <-> X[j];
}
```



All n! permutations of the original data equally likely: A specific element will be last with prob 1/n; given that, a specific other element will be next-to-last with prob 1/(n-1), …; overall: 1/(n!)

C.f. http://en.wikipedia.org/wiki/Fisher–Yates_shuffle and (for subtle way to go wrong) http://www.codinghorror.com/blog/2007/12/the-danger-of-naivete.html

Sequence Alignment

Part II

Local alignments & gaps

Variations

- Local Alignment
  - Preceding gives *global* alignment, i.e. full length of both strings;
  - Might well miss strong similarity of part of strings amidst dissimilar flanks
- Gap Penalties
  - 10 adjacent spaces cost 10 x one space?
- Many others
- Similarly fast DP algs often possible

Local Alignment: Motivations

"Interesting" (evolutionarily conserved, functionally related) segments may be a small part of the whole

- "Active site" of a protein
- Scattered genes or exons amidst "junk", e.g. retroviral insertions, large deletions
- Don't have whole sequence

Global alignment might miss them if flanking junk outweighs similar regions

Local Alignment

Optimal *local alignment* of strings S & T: Find substrings A of S and B of T having max value global alignment

S = abcxdex A = c x d e

T = xxxcde B = c - d e value = 5

Local Alignment: "Obvious" Algorithm

for all substrings A of S and B of T:
    Align A & B via dynamic programming
    Retain pair with max value
end ;

Output the retained pair

Time: $O(n^2)$ choices for A, $O(m^2)$ for B, $O(nm)$ for DP, so $O(n^3m^3)$ total.

[Best possible? Lots of redundant work…]

Local Alignment in O(nm) via Dynamic Programming

Input: S, T, |S| = n, |T| = m

Output: value of optimal local alignment

Better to solve a "harder" problem
for all $0 \le i \le n$, $0 \le j \le m$:

V(i,j) = max value of opt (global)
alignment of a suffix of S[1], …, S[i]
with a suffix of T[1], …, T[j]

Report best i,j

Base Cases

Assume $\sigma(x,-) \leq 0$, $\sigma(-,x) \leq 0$

V(i,0): some suffix of first i chars of S; all match spaces in T; best suffix is empty

$V(i,0) = 0$

V(0,j): similar

$V(0,j) = 0$

General Case Recurrences

Opt suffix align S[1], …, S[i] vs T[1], …, T[j]:

$$\left[\begin{matrix}\sim\sim\sim\sim & S[i]\\ \sim\sim\sim\sim & T[j]\end{matrix}\right],\ \left[\begin{matrix}\sim\sim\sim\sim & S[i]\\ \sim\sim\sim\sim & -\end{matrix}\right],\ \left[\begin{matrix}\sim\sim\sim\sim & -\\ \sim\sim\sim\sim & T[j]\end{matrix}\right],\ \text{or}\ \left[\begin{matrix} & \\ & \end{matrix}\right]$$

Opt align of suffix of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$

$$V(i,j) \;=\; \max\begin{cases} V(i\text{-}1,j\text{-}1) + \sigma(S[i],T[j]) \\ V(i\text{-}1,j) + \sigma(S[i],\ \text{-}\ ) \\ V(i,j\text{-}1) + \sigma(\ \text{-}\ ,\ T[j]) \\ 0 \end{cases},$$

$$\text{for all } 1 \le i \le n,\ 1 \le j \le m.$$

opt suffix alignment has: 2, 1, 1, 0 chars of S/T

Scoring Local Alignments

i	j	0	1 x	2 x	3 x	4 c	5 d	6 e	←T
0		0	0	0	0	0	0	0	
1	a	0							
2	b	0							
3	c	0							
4	x	0							
5	d	0							
6	e	0							
7	x	0							

↑
S

Finding Local Alignments

Again, arrows follow max

i	j	0	1 x	2 x	3 x	4 c	5 d	6 e ←T
0		0	0	0	0	0	0	0
1	a	0	0	0	0	0	0	0
2	b	0	0	0	0	0	0	0
3	c	0	0	0	0	2	1	0
4	x	0	2	2	2	1	1	0
5	d	0	1	1	1	1	3	2
6	e	0	0	0	0	0	2	5
7	x	0	2	2	2	1	1	4

↑
s

Notes

Time and Space = O(mn)

Space O(min(m,n)) possible with time O(mn), but finding alignment is trickier

Local alignment: “Smith-Waterman”

Global alignment: “Needleman-Wunsch”

Alignment With Gap Penalties

Gap: maximal run of spaces in S' or T'

```
ab--ddc-d        2 gaps in S'
a---ddcbd        1 gap in T'
```

Motivations, e.g.:

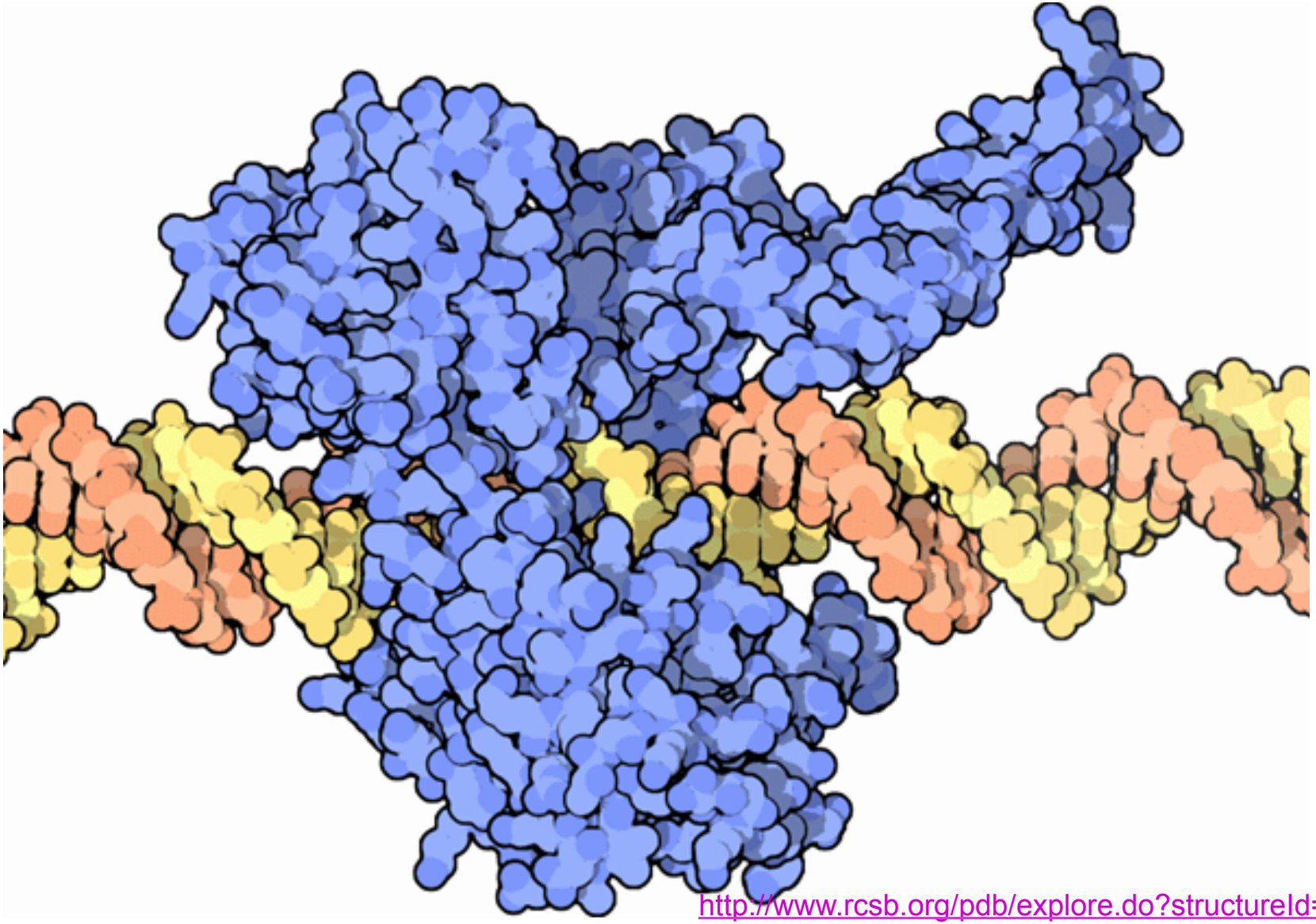
- mutation might insert/delete several or even many residues at once
- matching mRNA (no introns) to genomic DNA (exons and introns)
- some parts of proteins less critical

A Protein Structure: (Dihydrofolate Reductase)

Alignment of 5 Dihydrofolate reductase proteins

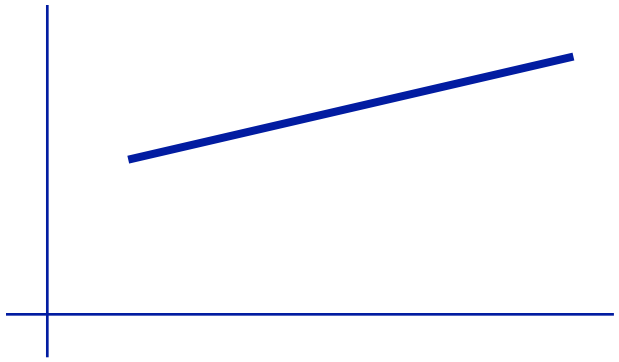
CLUSTAL W (1.82) multiple sequence alignment
http://pir.georgetown.edu/cgi-bin/multialn.pl
2/11/2013

Topoisomerase I



http://www.rcsb.org/pdb/explore.do?structureId=1a36

Affine Gap Penalties



Gap penalty = g + e*(gaplen-1), g ≥ e ≥ 0

Note: no longer suffices to know just the *score* of best subproblem(s) – *state* matters: do they end with '-' or not.

Summary: Alignment

Functionally similar proteins/DNA often have recognizably similar sequences even after eons of divergent evolution

Ability to find/compare/experiment with “same” sequence in other organisms is a huge win

Surprisingly simple scoring works well in practice: score positions separately & add, usually w/ fancier gap model like affine

Simple dynamic programming algorithms can find *optimal* alignments under these assumptions in poly time (product of sequence lengths)

This, and heuristic approximations to it like BLAST, are workhorse tools in molecular biology, and elsewhere.

Summary: Dynamic Programming

Keys to D.P. are to

a) identify the subproblems (usually repeated/overlapping)

b) solve them in a careful order so all small ones solved before they are needed by the bigger ones, and

c) build table with solutions to the smaller ones so bigger ones just need to do table lookups (*no* recursion, despite recursive formulation implicit in (a))

d) Implicitly, optimal solution to whole problem devolves to optimal solutions to subproblems

A *really* important algorithm design paradigm

The $1000 Genome arrives?

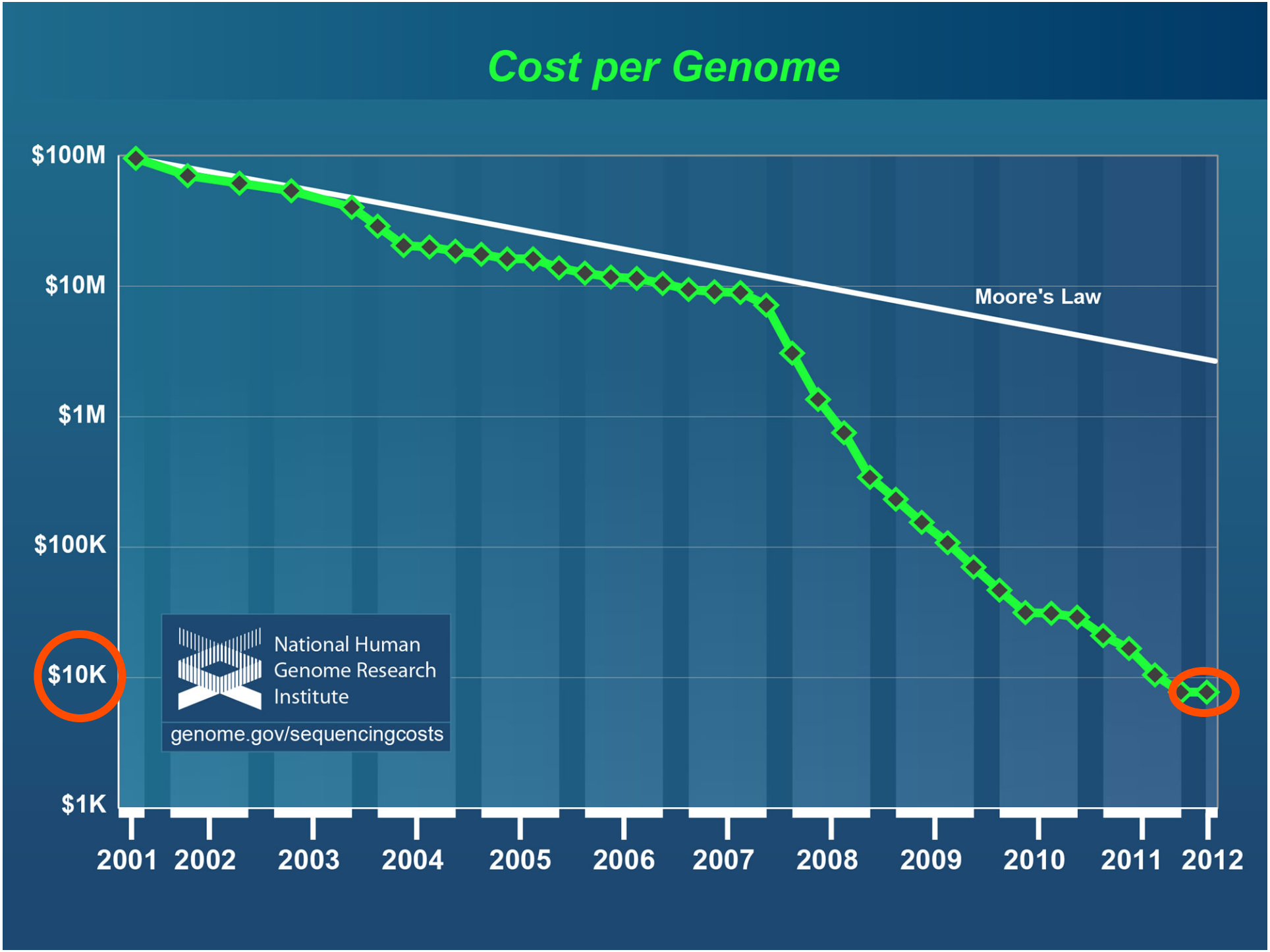
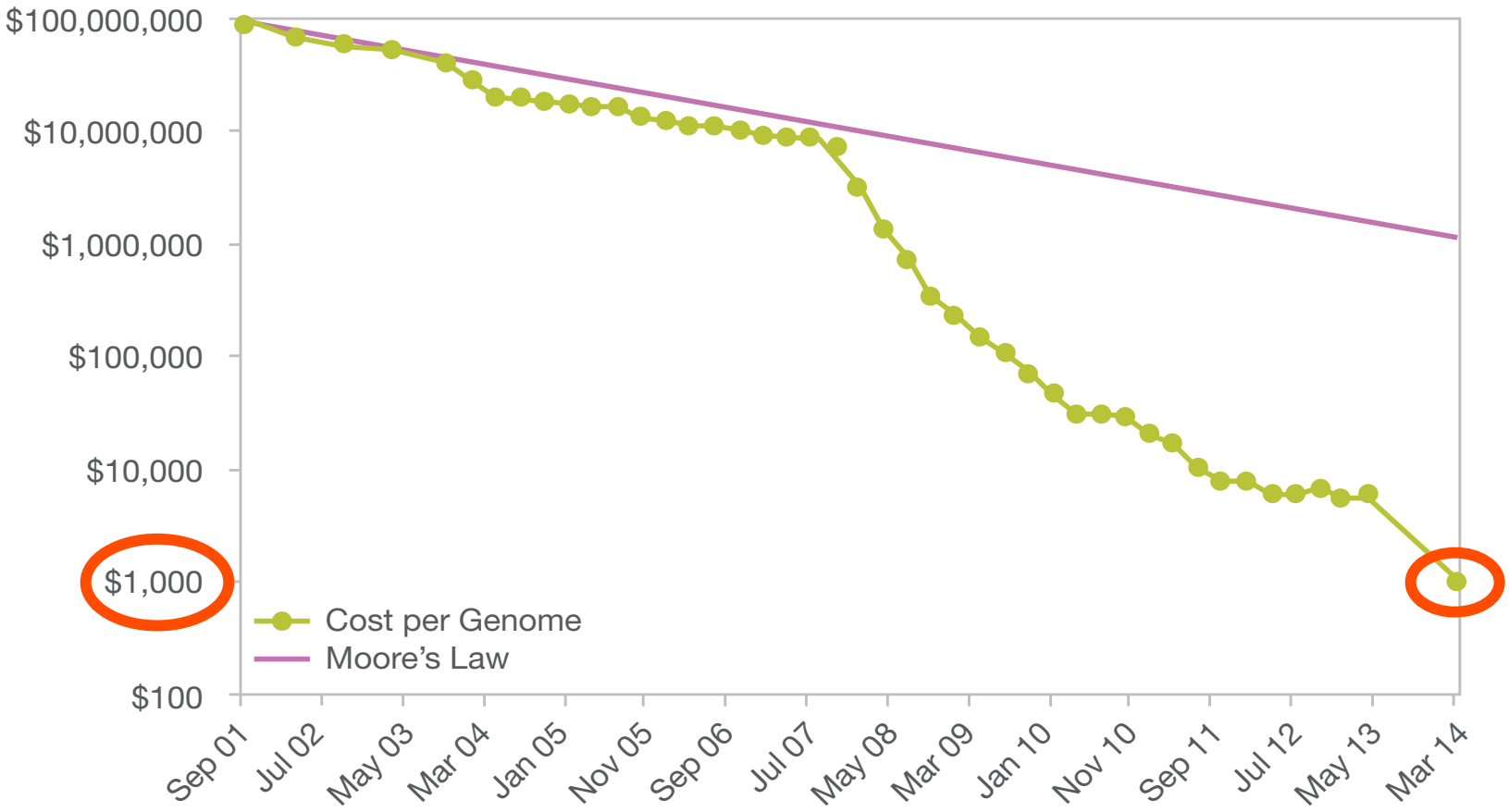

Cost per Genome
$100M
$10M
$1M
$100K
$10K
$1K
Moore's Law
National Human Genome Research Institute
genome.gov/sequencingcosts
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012

Figure 3: Illumina Sequencing Technology Outpaces Moore's Law for the Price of Whole Human Genome Sequencing



Announced
1/14/2014

Table 1: HiSeq X Ten Preliminary Performance Parameters*

	Dual Flow Cell	Single Flow Cell
Output/Run	1.6–1.8 Tb	800–900 Gb
Reads Passing Filter†	≤ 6 billion	≤ 3 billion
Supported Read Length	2 × 150	
Run Time	< 3 days	
Quality	≥ 75% of bases above Q30 at 2 × 150 bp	

*Specifications based on Illumina PhiX control library at supported cluster densities (between 1,255–1,412 K clusters/mm^2). Supported library preparation kit includes TruSeq Nano DNA HT kit with 350 bp target insert size and HiSeq X HD reagents. HiSeq X was designed and optimized for human whole-genome sequencing; other applications and species are not supported.

†Single-end reads.