CSE 527
Computational Biology
Autumn 2009

2. Sequence Alignment

Seminars

CSE 590C

"Reading and Research in Computational Biology"

Mondays, 3:30-4:30ish, EEB 026

http://www.cs.washington.edu/590c

GENOME 521

"COMBI"

Wednesdays,1:30-2:50 Foege S060

http://www.gs.washington.edu/news/combi.htm

This Week

Sequence alignment

Weekly “bio” interlude - DNA replication

More sequence alignment

Sequence Alignment

Part I

Motivation, dynamic programming, global alignment

Sequence Alignment

What

Why

A Simple Algorithm

Complexity Analysis

A better Algorithm:

"Dynamic Programming"

Sequence Similarity: What

G G A C C A

T A C T A A G

T C C A A T

Sequence Similarity: What

G G A C C A

T A C T A A G

| : | : | | :

T C C – A A T

Sequence Similarity: Why

Most widely used comp. tools in biology

New sequence always compared to sequence data bases

Similar sequences often have similar origin or function

Recognizable similarity after 10^8 –10^9 yr

BLAST Demo

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

Try it!

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

Taxonomy Report

root	64 hits	16 orgs	
. Eukaryota	62 hits	14 orgs	[cellular organisms]
. . Fungi/Metazoa group	57 hits	11 orgs	
. . . Bilateria	38 hits	7 orgs	[Metazoa; Eumetazoa]
. . . . Coelomata	36 hits	6 orgs	
. . . . . Tetrapoda	26 hits	5 orgs	[;;; Vertebrata;;;; Sarcopterygii]
. . . . . Eutheria	24 hits	4 orgs	[Amniota; Mammalia; Theria]
. . . . . Homo sapiens	20 hits	1 orgs	[Primates;; Hominidae; Homo]
. . . . . Murinae	3 hits	2 orgs	[Rodentia; Sciurognathi; Muridae]
. . . . . Rattus norvegicus	2 hits	1 orgs	[Rattus]
. . . . . Mus musculus	1 hits	1 orgs	[Mus]
. . . . . Sus scrofa	1 hits	1 orgs	[Cetartiodactyla; Suina; Suidae; Sus]
. . . . . Xenopus laevis	2 hits	1 orgs	[Amphibia;;;;;; Xenopodinae; Xenopus]
. . . . . Drosophila melanogaster	10 hits	1 orgs	[Protostomia;;;; Drosophila;;;]
. . . . Caenorhabditis elegans	2 hits	1 orgs	[; Nematoda;;;;;; Caenorhabditis]
. . . Ascomycota	19 hits	4 orgs	[Fungi]
. . . . Schizosaccharomyces pombe	10 hits	1 orgs	[;;;; Schizosaccharomyces]
. . . . Saccharomycetales	9 hits	3 orgs	[Saccharomycotina; Saccharomycetes]
. . . . . Saccharomyces	8 hits	2 orgs	[Saccharomycetaceae]
. . . . . Saccharomyces cerevisiae .	7 hits	1 orgs	
. . . . . Saccharomyces kluyveri ...	1 hits	1 orgs	
. . . . . Candida albicans	1 hits	1 orgs	[mitosporic Saccharomycetales;]
. . Arabidopsis thaliana	2 hits	1 orgs	[Viridiplantae; …Brassicaceae;]
. . Apicomplexa	3 hits	2 orgs	[Alveolata]
. . . Plasmodium falciparum	2 hits	1 orgs	[Haemosporida; Plasmodium]
. . . Toxoplasma gondii	1 hits	1 orgs	[Coccidia; Eimeriida; Sarcocystidae;]
. synthetic construct	1 hits	1 orgs	[other; artificial sequence]
` mphocystis disease virus	1 hits	1 orgs	[Viruses; dsDNA viruses, no RNA …]

Terminology
(CS, not necessarily Bio)

String: ordered list of letters TATAAG

Prefix: consecutive letters from front

empty, T, TA, TAT, ...

Suffix: … from end

empty, G, AG, AAG, ...

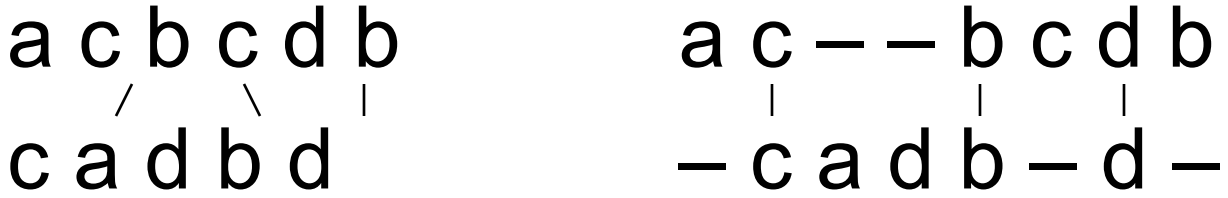
Substring: … from ends or middle

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 spaces) s.t.

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

(2) removing all spaces 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 spaces) 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

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

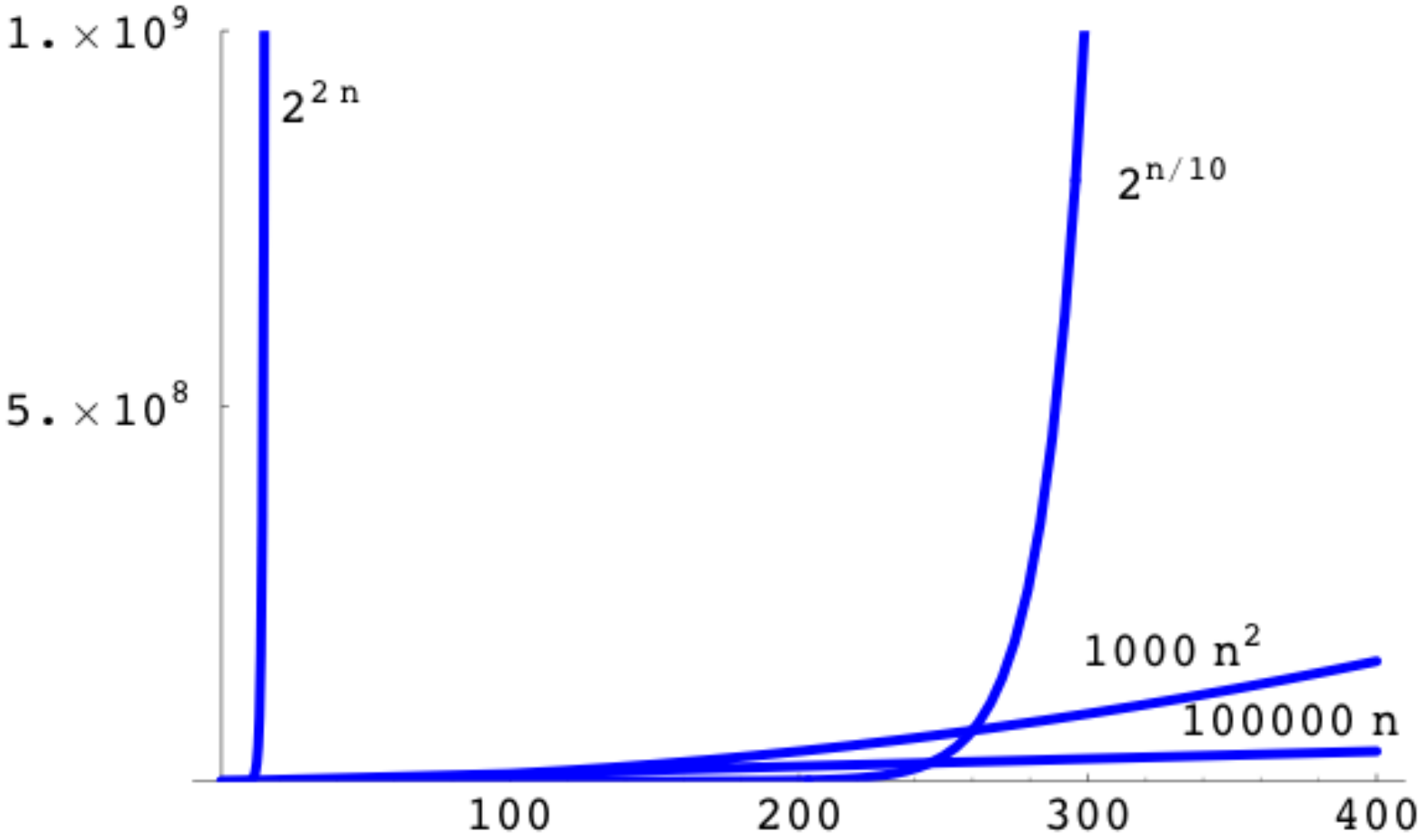
say k of them are in S

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



Asymptotic Analysis

How does run time grow as a function of problem size?

n^2 or $100\ n^2 + 100\ n + 100$ vs 2^{2n}

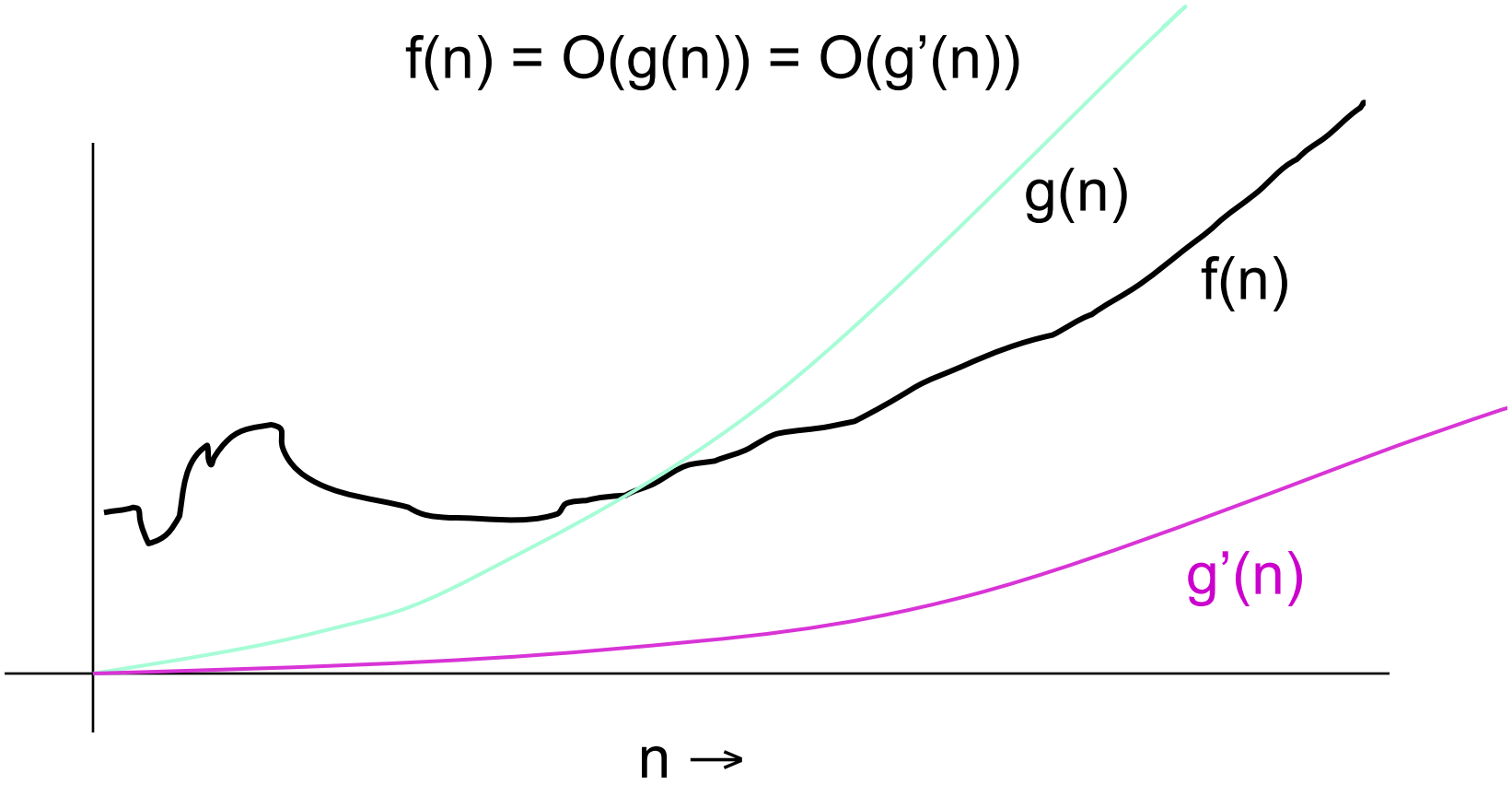
Defn: f(n) = O(g(n)) iff there is a constant c s.t. $|f(n)| \leq cg(n)$ for all sufficiently large n.

$100\ n^2 + 100\ n + 100 = O(n^2)$ [e.g. c = 101]

$n^2 = O(2^{2n})$

2^{2n} is *not* $O(n^2)$

Big-O Example



Utility of Asymptotics

"All things being equal," smaller asymptotic growth rate is better

All things are never equal

Even so, big-O bounds often let you quickly pick most promising candidates among competing algorithms

Poly time algs often practical; non-poly algs seldom are.

(Yes, there are exceptions.)

Fibonacci Numbers (recursion)

```
fib(n) {
  if (n <= 1) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
```

Simple recursion, but many repeated subproblems!!

$\Rightarrow$

Time = $\Omega(1.61^n)$

Fibonacci, II
(dynamic programming)

```
int fib[n];
fib[0] = 1;
fib[1] = 1;
for(i=2; i<=n; i++) {
  fib[i] = fib[i-1] + fib[i-2];
}
return fib[n];
```

Avoid repeated subproblems by tabulating their solutions

⇒

Time = O(n)

(in this case)

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 space in T
- last char of T aligned with space in S
- (never align space with space; $\sigma(-, -) < 0$)

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 spaces

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

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

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

General Case

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

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

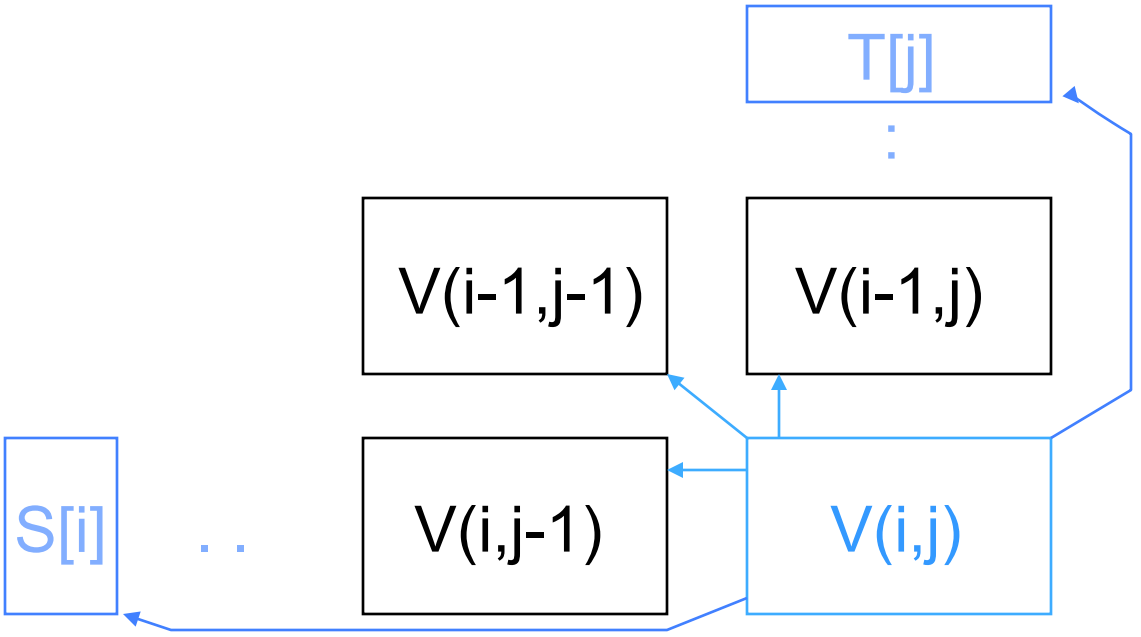
Opt align of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$

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

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						

c
-

Score(c,-) = -1

↑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						
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

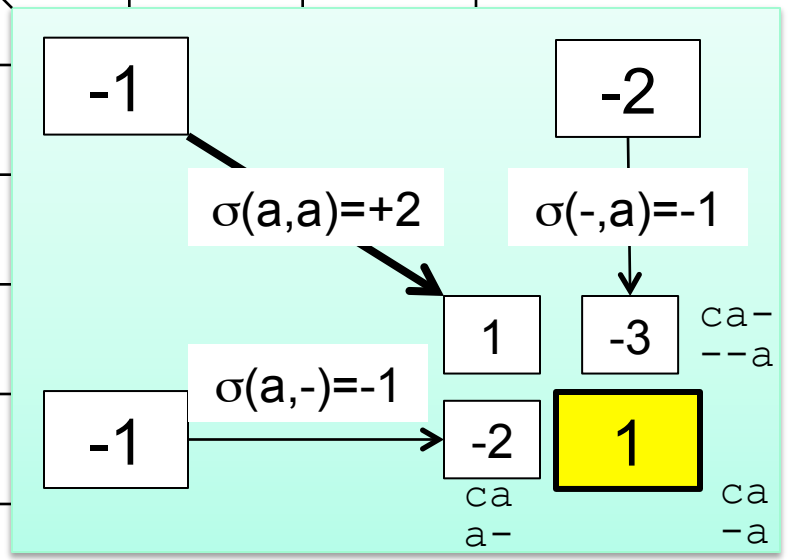
Mismatch = -1
Match = 2

Example

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

←T

↑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

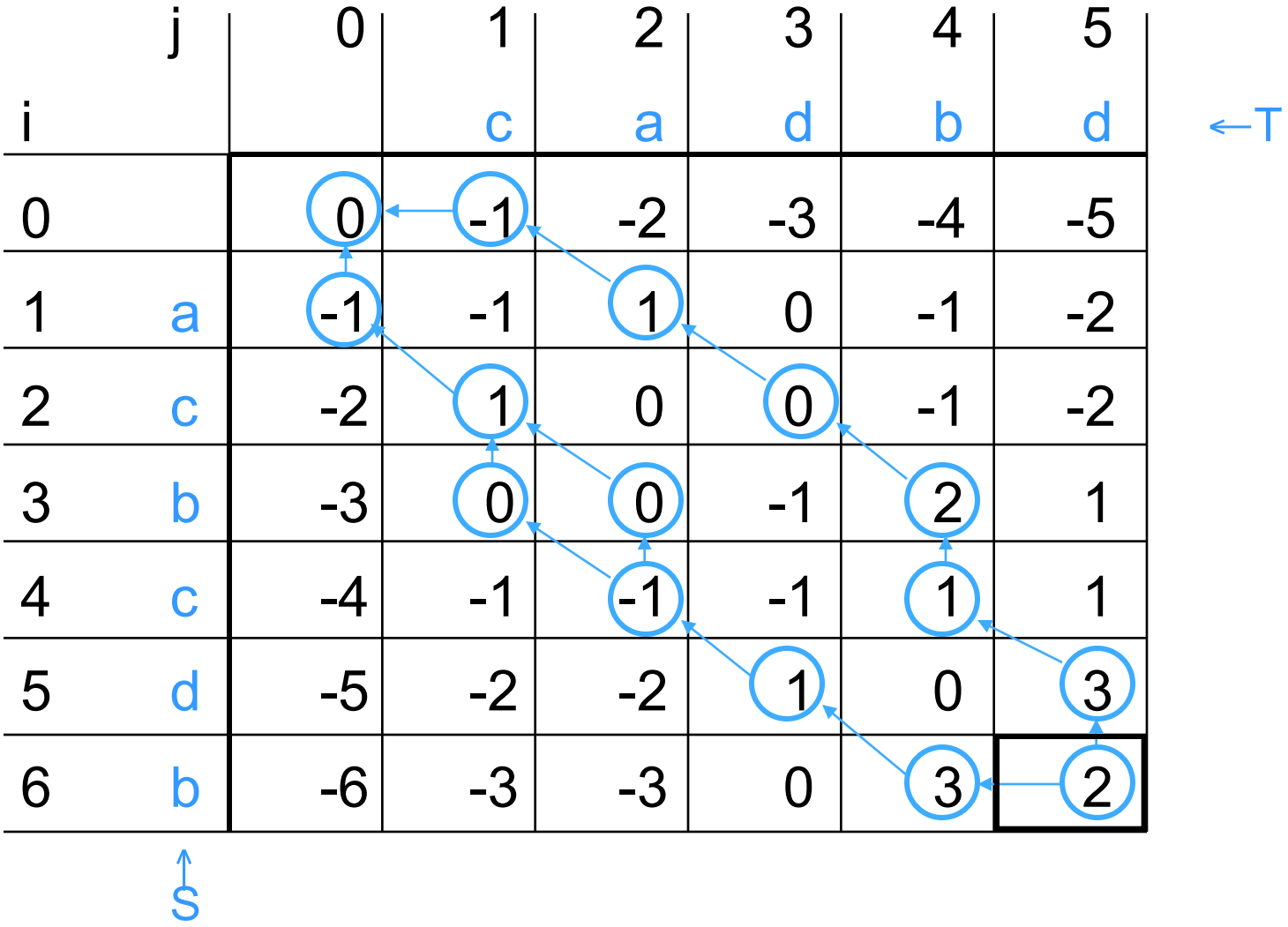
Mismatch = -1
Match = 2

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)) but tricky.

Significance of Alignments

Is “42” a good score?

Compared to what?

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

Overall Alignment Significance, II
Empirical (via randomization)

Generate N random 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" sequences?

Scores are often sensitive to sequence composition

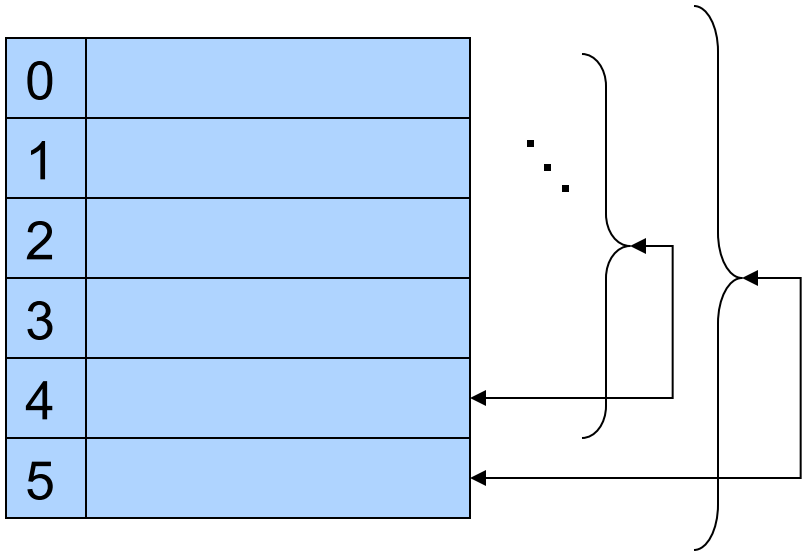
So uniform 1/20 or 1/4 is a bad idea

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];
}
```

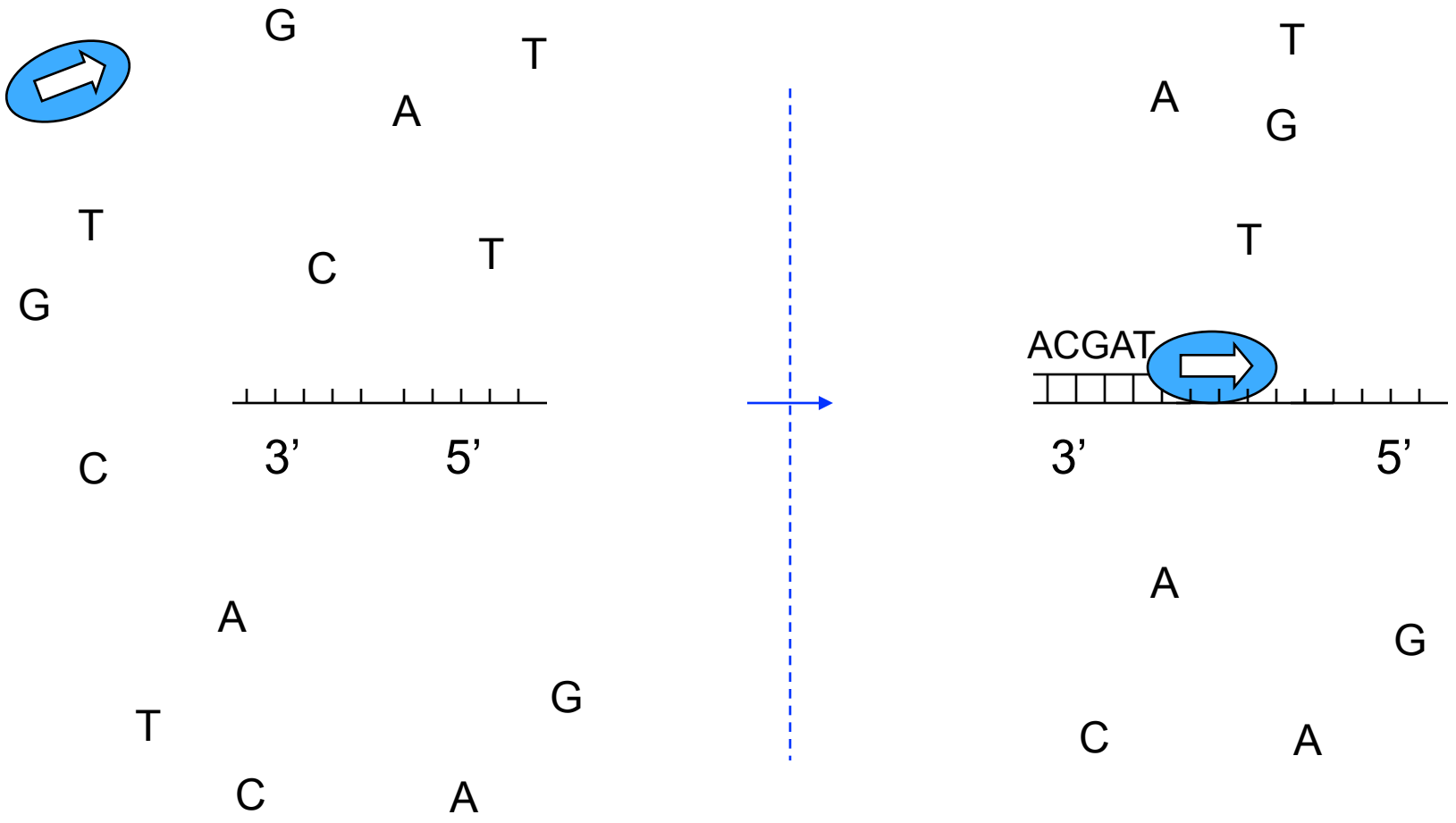


All n! permutations of the original data equally likely: A specific element will be last with prob 1/n; given that, a specific other element will be next-to-last with prob 1/(n-1), …; overall: 1/(n!)

Weekly Bio Interlude

DNA Replication

DNA Replication: Basics



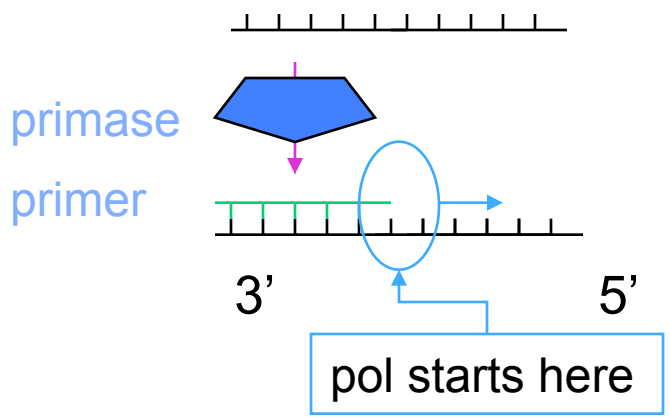
Issues & Complications, I

1st ~10 nt's added are called the *primer*

In simple model, DNA pol has 2 jobs: prime & extend

Priming is error-prone

So, specialized *primase* does the priming; pol specialized for fast, accurate extension



Still doesn't solve the accuracy problem (hint: primase makes an *RNA* primer)

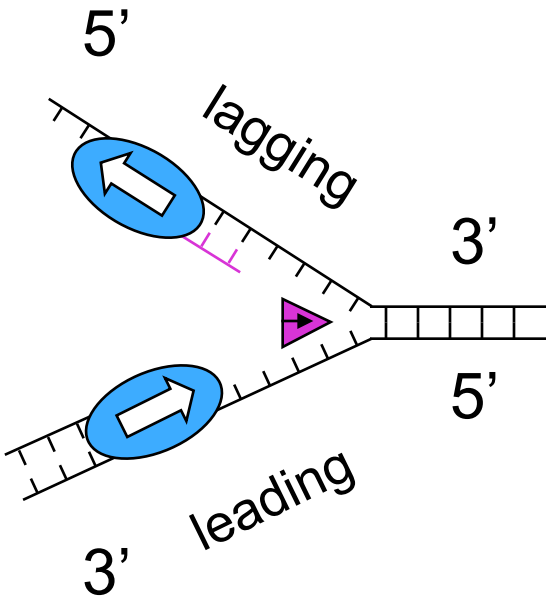
Issue 2: Rep Forks & Helices

"Replication Fork": DNA double helix is progressively unwound by a DNA helicase, and both resulting single strands are duplicated

DNA polymerase synthesizes new strand 5' -> 3'(reading its template strand 3' -> 5')

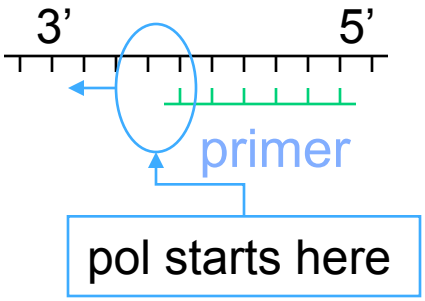
That means on one (the "leading") strand, DNA pol is chasing/pushing the replication fork

But on the other "lagging" strand, DNA pol is running away from it.

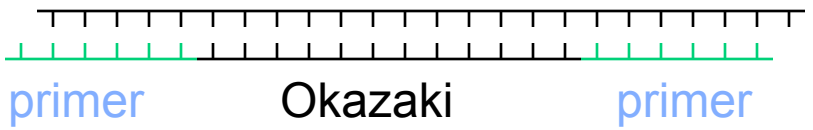


Issue 3: Fragments

Lagging strand gets a series of “Okazaki fragments” of DNA (~200nt in eukaryotes) following each primer

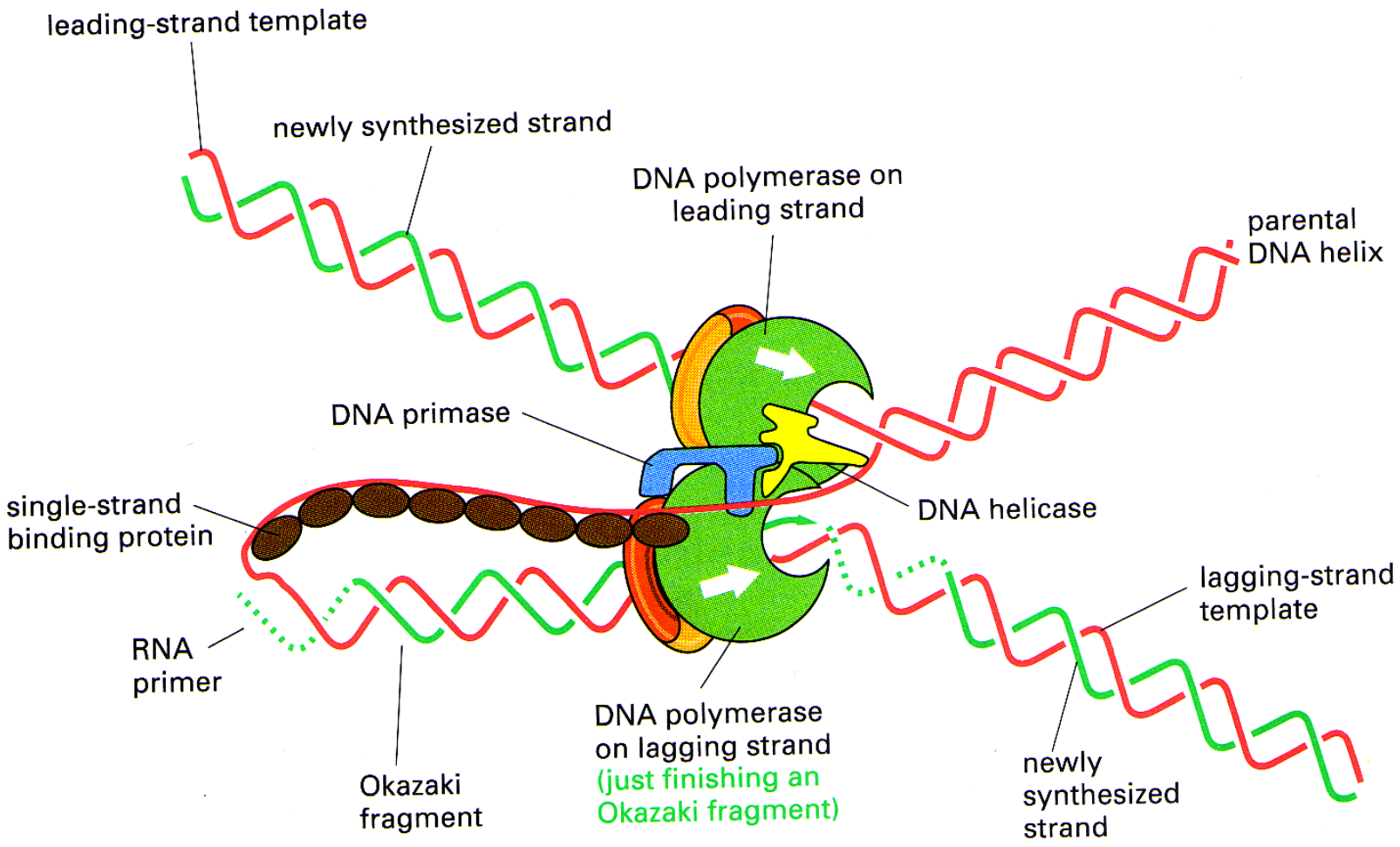


The RNA primers are later removed by a *nuclease* and *DNA* pol fills gaps (more accurate than primase; primed by *DNA* from adjacent Okazaki frag

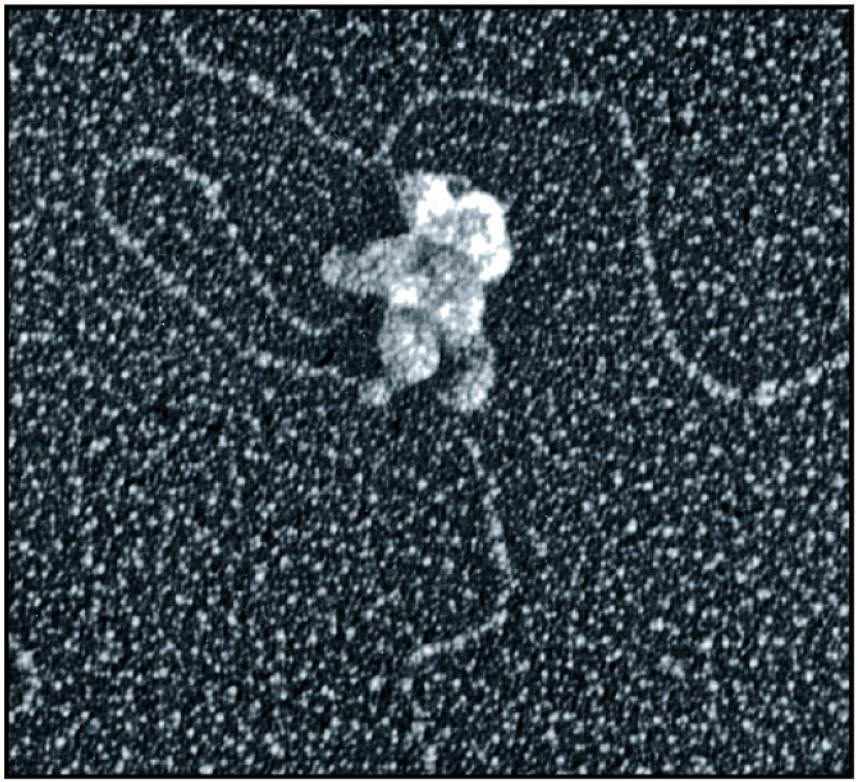


Fragments joined by *ligase*

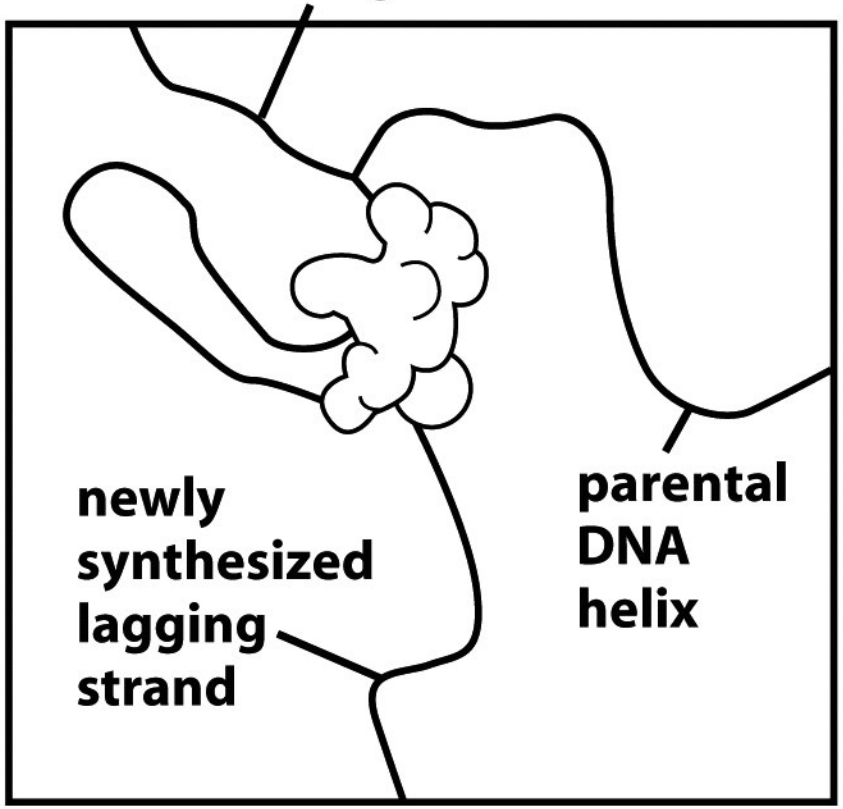
Issue 4: Coord Lead/Lag



Alberts et al., Mol. Biol. of the Cell, 3rd ed, p258



(B)

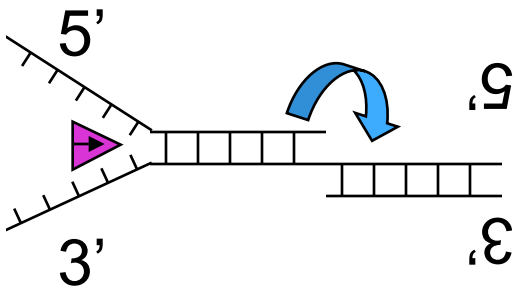


(C)

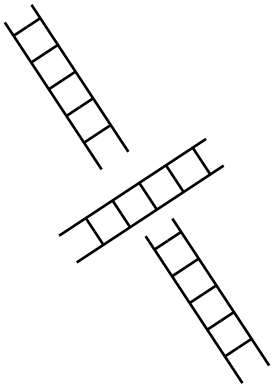
Figure 5-19bc Molecular Biology of the Cell 5/e (© Garland Science 2008)

Issue 5: Twirls & Tangles

Unwinding helix (~10 nucleotides per turn) would cause stress. *Topoisomerase I* cuts DNA backbone on *one* strand, allowing it to spin about the remaining bond, relieving stress



Topoisomerase II can cut & rejoin *both* strands, after allowing another double strand to pass through the gap, de-tangling it.



Issue 6: Proofreading

Error rate of pol itself is ~10^{-4}, but overall rate is 10^{-9}, due to proofreading & repair, e.g.

- pol itself can back up & cut off a mismatched base if one happens to be inserted
- priming the new strand is hard to do accurately, hence RNA primers, later removed & replaced
- other enzymes scan helix for “bulges” caused by base mismatch, figure out which strand is original, cut away new (faulty) copy; DNA pol fills gap
- which strand is original? Bacteria: “methylate” some A’s, eventually. Euks: strand nicking

Replication Summary

Speed: 50 (eukaryotes) to
500 (prokaryotes) bp/sec

Accuracy: 1 error per 10^9 bp

Complex & highly optimized

Highly similar across all living cells

More info:
Alberts et al., *Mol. Biol. of the Cell*

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

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

	j	0	1	2	3	4	5	6	
i			x	x	x	c	d	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
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

←T

↑
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'

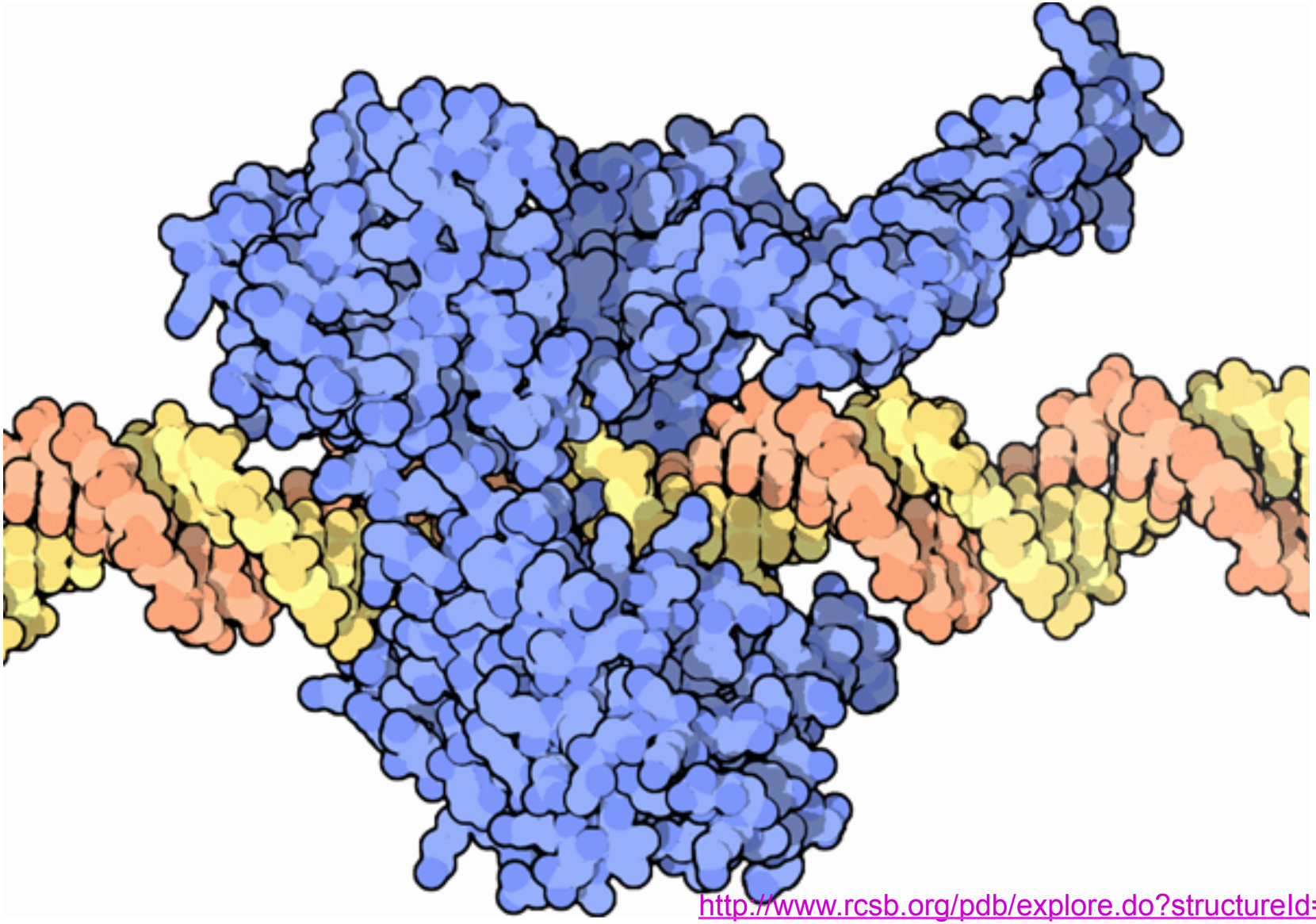
```
ab--ddc-d        2 gaps in S'
a---ddcbd        1 gap in T'
```

Motivations, e.g.:

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

A Protein Structure: (Dihydrofolate Reductase)

Topoisomerase I



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

Sequence Evolution

"Nothing in Biology Makes Sense Except in the Light of Evolution" – Theodosius Dobzhansky, 1973

Changes happen at random

Deleterious/neutral/advantageous changes unlikely/ possibly/likely spread widely in a population

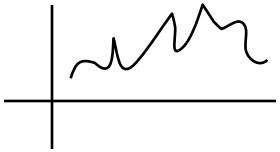
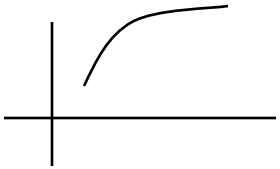
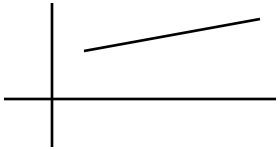
Changes are less likely to be tolerated in positions involved in many/close interactions, e.g.

- enzyme binding pocket
- protein/protein interaction surface
- …

Gap Penalties

Score = f(gap length)

Kinds, & best known alignment time

general		$O(n^3)$
convex		$O(n^2 \log n)$
affine		$O(n^2)$ [really, $O(mn)$]

Global Alignment with Affine Gap Penalties

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

G(i,j) = …, s.t. last pair matches S[i] & T[j]

F(i,j) = …, s.t. last pair matches S[i] & –

E(i,j) = …, s.t. last pair matches – & T[j]

Time: O(mn) [calculate all, O(1) each]

Affine Gap Algorithm

Gap penalty = g + s*(gap length), g,s ≥ 0

V(i,0) = E(i,0) = V(0,i) = F(0,i) = -g-i*s

V(i,j) = max(G(i,j), F(i,j), E(i,j))

G(i,j) = V(i-1,j-1) + σ(S[i],T[j])

F(i,j) = max(F(i-1,j)-s , V(i-1,j)-g-s)

E(i,j) = max(E(i,j-1)-s , V(i,j-1)-g-s)

old gap new gap

Summary

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, possibly 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