

CSE 417 Algorithms

Sequence Alignment

Sequence Alignment

What

Why

A Dynamic Programming Algorithm

Sequence Alignment

Goal: position characters in two strings to “best” line up identical/similar ones with one another

i.e. think back to finding similarities in RNA / DNA structure across different proteins in different organisms: similar sequences often have similar structures and helps us understand functionality of

We can do this via Dynamic Programming

What is an alignment?

Compare two strings to see how “similar” they are
E.g., maximize the # of identical chars that line up

ATGTTAT vs
ATCGTAC

A	T	-	G	T	T	A	T
A	T	C	G	T	-	A	C

What is an alignment?

Compare two strings to see how “similar” they are
E.g., maximize the # of identical chars that line up

ATGTTAT vs
ATCGTAC

A	T	-	G	T	T	A	T
A	T	C	G	T	-	A	C

matches

mismatches

can even consider “missing” characters

Sequence Alignment: Why

Biology

Among most widely used comp. tools in biology

DNA sequencing & assembly

New sequence always compared to data bases

Similar sequences often have similar origin and/or function

Recognizable similarity after $10^8 - 10^9$ yr

Other


keep track of differences rather
than storing each copy of the
new file.. i.e. version tree

spell check/correct, diff, svn/git/..., plagiarism, ...

Accession	Entry name	Status	Protein names	Organism	Length
Q7T109	Q7T109_XENTR	★	MyoD protein	Xenopus tropicalis (Western clawed frog) (<i>Xenopus tropicalis</i>)	288

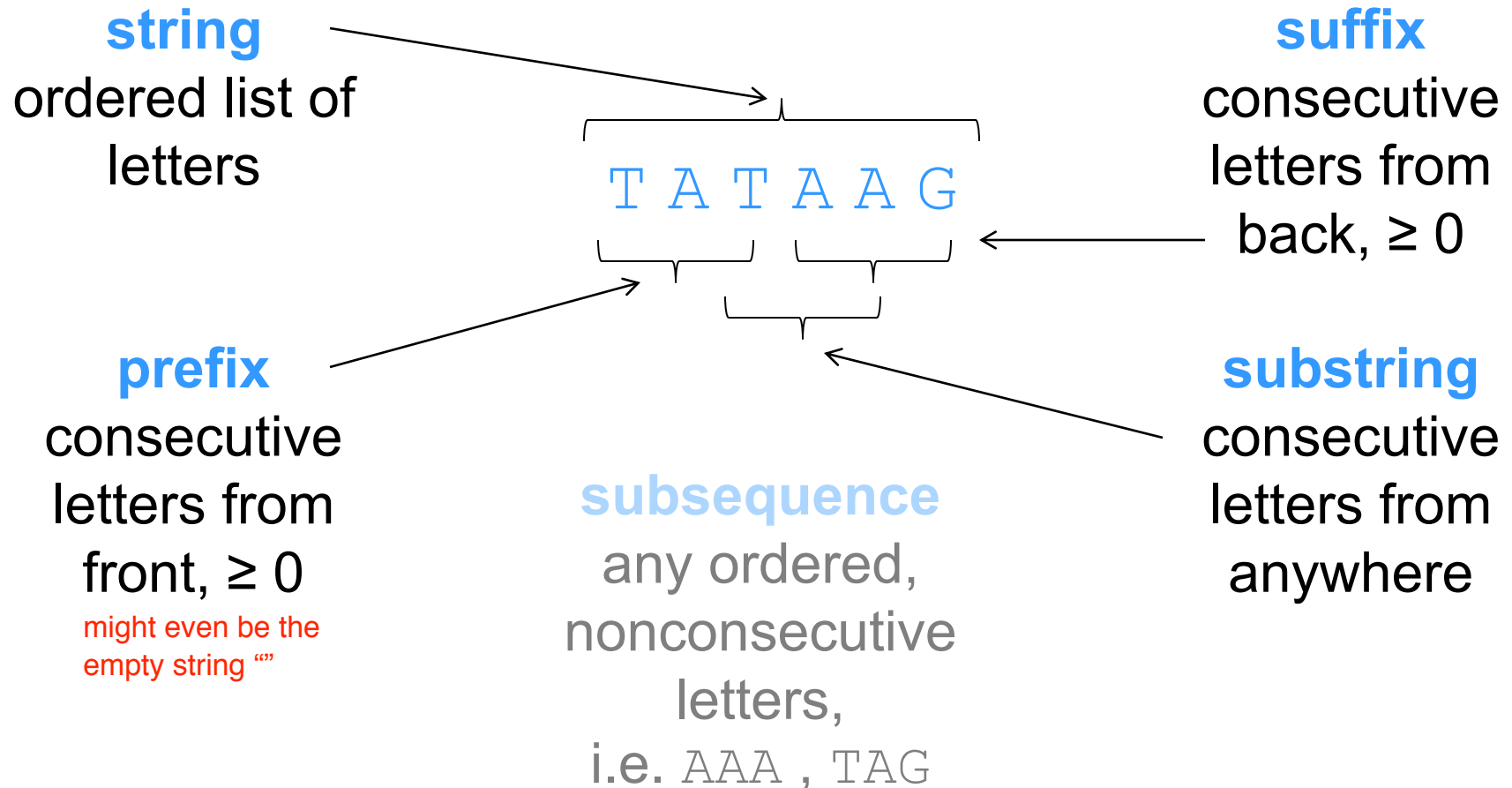
Some Details from #25

Alignment 1 against Q7T109

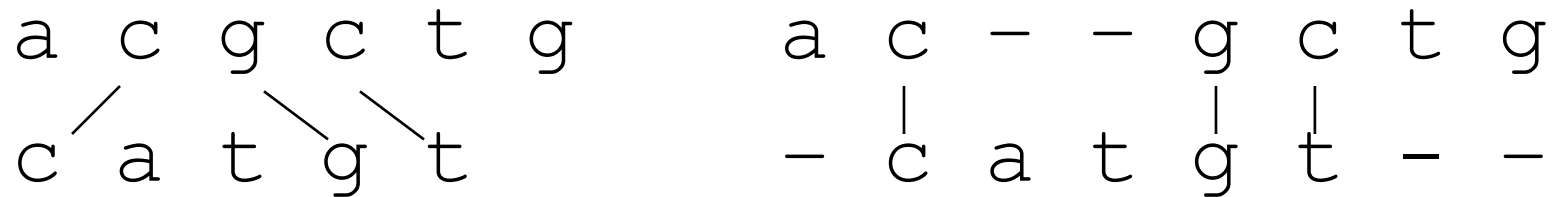
Score	964	E-value	1.0 ×10 ⁻¹⁰²
Identity	64.0%	Positives	74.0%
Query length	320	Match length	288
Position	Q7T109 matches from 1 to 288 (288AA), in the query sequence from 1 to 320 (320AA)		
Graphical			

1	MELLSPPLRDVDLTAPDGS LCS FAT TDDFYDDPCFDSPDLRFFEDLDPRLMHVGALLKPE	60	P15172
	MELL PPLRD+++T +GSLCSF T DDFYDDPCF++ D+ FFEDLDPRL+HV ALLKPE		
1	MELLPPPLRDMEVT--EGSLCSFPTPDDFYDDPCFNTSDMSFFEDLDPRLVHV-ALLKPE	57	Q7T109
61	EHS HFPA AVHPAPGAREDEHVRAPSGHHQAGRCLLWACKACKRKT TNADRRKAATMRERR	120	P15172
	+ H EDEHVRAPSGHHQAGRCLLWACKACKRKT TNADRRKAATMRERR		
58	DPHH-----NEDEHVRAPSGHHQAGRCLLWACKACKRKT TNADRRKAATMRERR	106	Q7T109
121	RLSKVNEAFETLKRCTSSNP NQRLPKVEILRNAIRYIEGLQALLRDQDAAPPGAAAAFYA	180	P15172
	RLSKVNEAFETLKRCTS+NP NQRLPKVEILRNAIRYIE LQ+LLR Q+ +FY		
107	RLSKVNEAFETLKRCTSTNP NQRLPKVEILRNAIRYIESLQSLLRGQE-----ESFY-	158	Q7T109
181	PGPLPPGRGGEHYSGDSDASSPRSNCS DGMMDYSGPPSGARRRNCYEGAYYNEAPSEPRP	240	P15172
	P+ EHYS GDSDASSPRSNCS DGM DYS PP G+RRRN Y+ ++Y+++P+ R		
159	--PVL-----EHYS GDSDASSPRSNCS DGMTDYS-PPCGSRRRNSYDSSFYS DSPNGLRL	210	Q7T109
241	GKSA AVSSLDCLSSIVERISTESPAAPALLLADV PSESPPRRQEAAAPSEGES---SGDP	297	P15172
	GKS+ +SSLDCLSSIVERISTES P + AD SE P +P +GE+ SG	7	
211	GKSSVISSLDCLSSIVERISTESPVCPVIPAADSGSEGSP-----CSPLQGETLSESGII	265	Q7T109

Terminology



Formal definition of an alignment



An **alignment** of strings S , T is a pair of strings S' , T' with dash characters “-” inserted, so that

1. $|S'| = |T'|$, and **strings are same length** ($|S|$ = “length of S ”)
2. Removing dashes leaves S , T **dont delete any letters from strings nor change the order of any characters**

Consecutive dashes are called “**a gap**.”

(Note that this is a definition for a general alignment, not optimal.)

Scoring an arbitrary alignment

Define a score for *pairs* of aligned chars, e.g.

scoring system can be more complicated than this. Does not influence algorithm as long as we are only comparing single columns

$$\sigma(x, y) = \begin{cases} \text{match} & 2 \\ \text{mismatch} & -1 \end{cases}$$

(Toy scores for examples in slides)

Apply that *per column*, then *add*.

a	c	-	-	g	c	t	g
-	c	a	t	g	t	-	-
-1	+2	-1	-1	+2	-1	-1	-1

Total Score = -2

NB: my slides: maximize similarity; KT minimizes diffs

textbook minimizes differences in their algorithm

Can we use Dynamic Programming?

1. Can we decompose into **subproblems**?

E.g., can we align smaller substrings (say, prefix/suffix in this case), then combine them somehow?

2. Do we have **optimal substructure**?

I.e., is optimal solution to a subproblem *independent of context*? E.g., is appending two optimal alignments also be optimal? Perhaps, but some changes at the interface might be needed?

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

(assume $\sigma(-, -) < 0$, so never align dash with dash)

for any such alignment where a dash is aligned with a dash, we could just remove these pairing(s) to generate an alignment without any dashes matched. If $\sigma(-, -) < 0$ doing so will always improve the overall score.

*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
 $S[1], \dots, S[i]$ with $T[1], \dots, T[j]$
for **all** $0 \leq i \leq n, 0 \leq j \leq m$.

Base Cases

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

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

our scoring function from earlier.

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

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

General Case

Opt align of $S[1], \dots, S[i]$ vs $T[1], \dots, T[j]$:

The three cases mentioned earlier: on slide 12

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

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

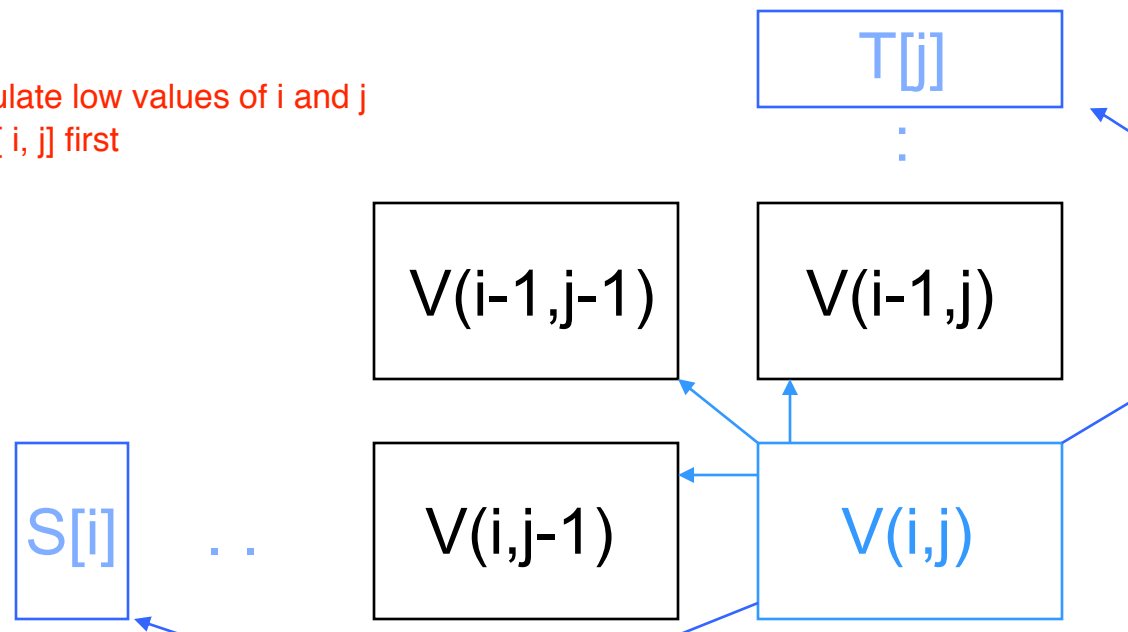
$$V(i,j) = \max \left\{ \begin{array}{l} V(i-1,j-1) + \sigma(S[i], T[j]) \\ V(i-1,j) + \sigma(S[i], -) \\ V(i,j-1) + \sigma(-, T[j]) \end{array} \right\},$$

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

Calculating One Entry

$$V(i,j) = \max \left\{ \begin{array}{l} V(i-1,j-1) + \sigma(S[i], T[j]) \\ V(i-1,j) + \sigma(S[i], -) \\ V(i,j-1) + \sigma(-, T[j]) \end{array} \right\}$$

Calculate low values of i and j
for V[i, j] first



Example

scoring function { Mismatch = -1
Match = 2

		j	0	1	2	3	4	5	←T
i				c	a	t	g	t	
0			0	-1	-2	-3	-4	-5	
1	a	-1							
2	c	-2							
3	g	-3							
4	c	-4							
5	t	-5							
6	g	-6							

↑S

c
-

Score(c,-) = -1

Example

Mismatch = -1

Match = 2

		j	0	1	2	3	4	5	←T
i				c	a	t	g	t	
0			0	-1	-2	-3	-4	-5	
1	a	-1							
2	c	-2							
3	g	-3							
4	c	-4							
5	t	-5							
6	g	-6							

↑S

-
a

Score(-,a) = -1

Example

Mismatch = -1

Match = 2

		j	0	1	2	3	4	5	
i				c	a	t	g	t	←T
	0		0	-1	-2	-3	-4	-5	
	1	a	-1						
	2	c	-2						
	3	g	-3						
	4	c	-4						
	5	t	-5						
	6	g	-6						

↑S

-	-
a	c
-1	

Score(-,c) = -1

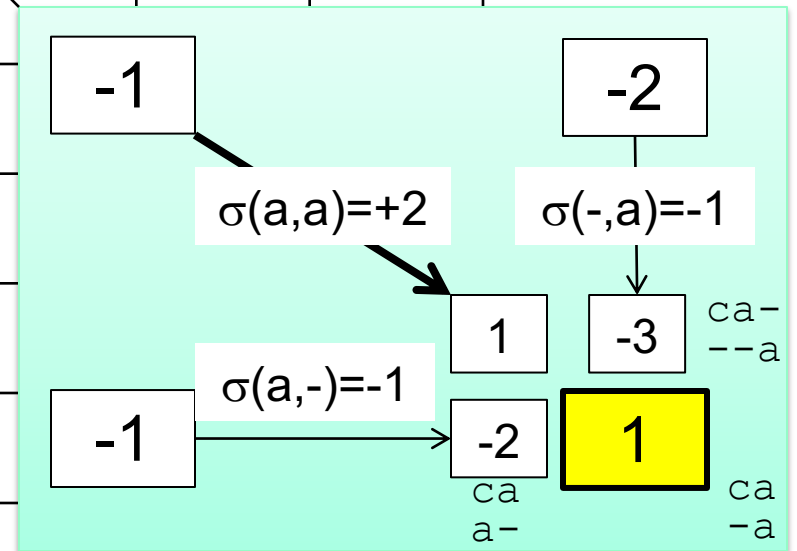
Mismatch = -1

Match = 2

Example

		j	0	1	2	3	4	5	←T
i				c	a	t	g	t	
0			0	-1	-2	-3	-4	-5	
1	a	-1	-1	1					
2	c	-2							
3	g	-3							
4	c	-4							
5	t	-5							
6	g	-6							

↑S



Mismatch = -1

Match = 2

Example

		j	0	1	2	3	4	5
i				c	a	t	g	t
0		0	-1	-2	-3	-4	-5	
1	a	-1	-1	1				
2	c	-2	1					
3	g	-3						
4	c	-4						
5	t	-5						
6	g	-6						

←T

Time =
 $O(mn)$

↑
S

Mismatch = -1

Match = 2

Example

		j	0	1	2	3	4	5	←T
i				c	a	t	g	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	g		-3	0	0	-1	2	1	
4	c		-4	-1	-1	-1	1	1	
5	t		-5	-2	-2	1	0	3	
6	g		-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

		j	0	1	2	3	4	5	←T
i				c	a	t	g	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	g		-3	0	0	-1	2	1	
4	c		-4	-1	-1	-1	1	1	
5	t		-5	-2	-2	1	0	3	
6	g		-6	-3	-3	0	3	2	

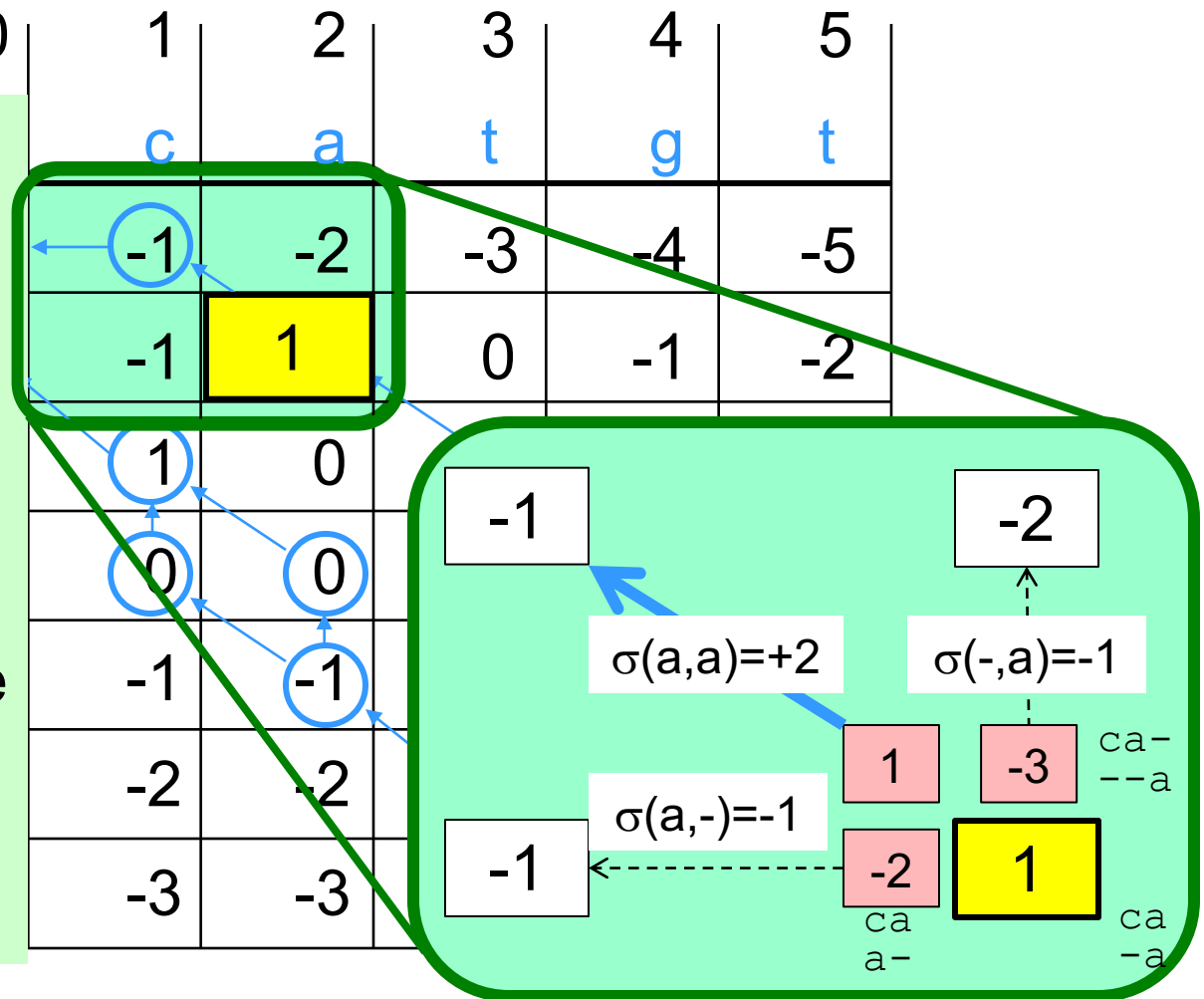
↑S

Ex: what are the 3 alignments? C.f. slide 12.

Finding Alignments: Trace Back

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

NB: trace back follows max *terms* (pink boxes; $ngbr + \sigma$), not max neighbors (white boxes). E.g., TB from yellow cell is only *diagonal* ($ngbr = -1$, $term = 1$), not to the equally-good horizontal neighbor ($term = -2$)



Complexity Notes

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

Space = $O(mn)$

Easy to get ^{NOT alignment though} value in Time = $O(mn)$ and
Space = $O(\min(m,n))$

Only need to store the previous row... but makes traceback harder.

Possible to get value *and alignment* in
Time = $O(mn)$ and Space = $O(\min(m,n))$,
but tricky. (KT section 6.7)

Variations

Local Alignment best substring of S and substrings of T, find the ones with the highest similarity

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 dashes cost 10 x one dash?

Many others

Similarly fast DP algs often possible

Significance of Alignments

Is “42” a good score?

Compared to what?

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

Interesting stats problem; much is known

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 affine gap model

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