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

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

But we’ll see more subtle measures

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

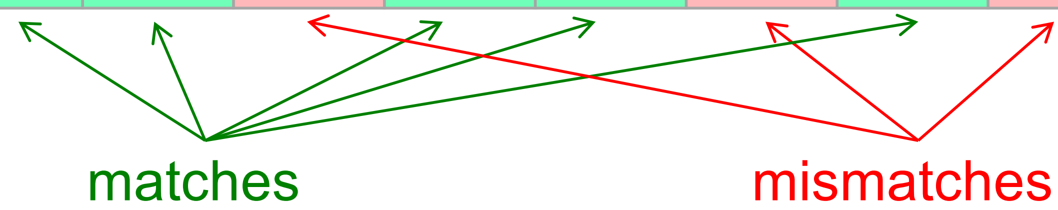
E.g., maximize the # of identical chars that line up

But we’ll see more subtle measures

ATGTTAT vs

ATCGTAC

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



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

- 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) (Silurana tropicalis)	288

Some Details from #25

Alignment 1 against Q7T109

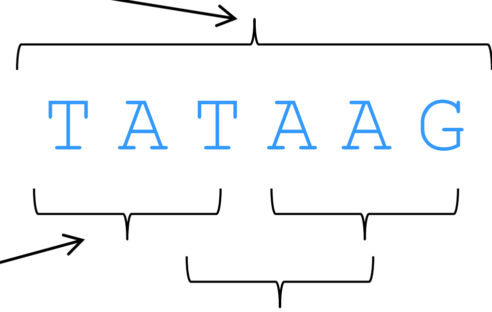
Score	964	**E-value**	1.0×10^{-102}
Identity	64.0%	**Positives**	74.0%
Query length	320	**Match length**	288
Position	**Q7T109** matches from 1 to 288 (288AA), in the query sequence from 1 to 320 (320AA)		
Graphical			

```
1     MELLSPPLRDVDLTAPDGSLCSFATTDDFYDDPCFDSPDLRFFEDLDPRLMHVGALLKPE   60 P15172
      MELL PPLRD+++T  +GSLCSF T DDFYDDPCF++ D+ FFEDLDPRL+HV ALLKPE
1     MELLPPPLRDMEVT--EGSLCSFPTPDDFYDDPCFNTSDMSFFEDLDPRLVHV-ALLKPE   57 Q7T109

61    EHSHFPAAVHPAPGAREDEHVRAPSGHHQAGRCLLWACKACKRKTTNADRRKAATMRERR  120 P15172
      +  H            EDEHVRAPSGHHQAGRCLLWACKACKRKTTNADRRKAATMRERR
58    DPHH-----------NEDEHVRAPSGHHQAGRCLLWACKACKRKTTNADRRKAATMRERR  106 Q7T109

121   RLSKVNEAFETLKRCTSSNPNQRLPKVEILRNAIRYIEGLQALLRDQDAAPPGAAAAFYA  180 P15172
      RLSKVNEAFETLKRCTS+NPNQRLPKVEILRNAIRYIE LQ+LLR Q+         +FY
107   RLSKVNEAFETLKRCTSTNPNQRLPKVEILRNAIRYIESLQSLLRGQE-------ESFY-  158 Q7T109

181   PGPLPPGRGGEHYSGDSDASSPRSNCSDGMMDYSGPPSGARRRNCYEGAYYNEAPSEPRP  240 P15172
        P+      EHYSGDSDASSPRSNCSDGM DYS PP G+RRRN Y+ ++Y+++P+  R
159   --PVL-----EHYSGDSDASSPRSNCSDGMTDYS-PPCGSRRRNSYDSSFYSDSPNGLRL  210 Q7T109

241   GKSAAVSSLDCLSSIVERISTESPAAPALLLADVPSESPPRRQEAAAPSEGES---SGDP  297 P15172
      GKS+ +SSLDCLSSIVERISTESP  P +  AD  SE  P      +P +GE+   SG
211   GKSSVISSLDCLSSIVERISTESPVCPVIPAADSGSEGSP-----CSPLQGETLSESGII  265 Q7T109
```

Terminology

string
ordered list of letters

suffix
consecutive letters from back, ≥ 0



prefix
consecutive letters from front, ≥ 0

substring
consecutive letters from anywhere

subsequence
any ordered, nonconsecutive letters,
i.e. AAA , TAG

Formal definition of an alignment

```
a  c  g  c  t  g          a  c  -  -  g  c  t  g
  /    \  \                  |           |  |
c  a  t  g  t             -  c  a  t  g  t  -  -
```

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

1. |S'| = |T'|, and (|S| = "length of S")
2. Removing dashes leaves S, T

Consecutive dashes are called "a gap."

(NB: this is a defn for a general alignment, not necessarily optimal.)

Scoring an arbitrary alignment

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

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

(Toy scores for examples in slides)

NB: I maximize similarity; KT minimizes difference

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

More Realistic Scores: BLOSUM 62

(the “σ” scores)

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**4**	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
R	-1	**5**	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
N	-2	0	**6**	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
D	-2	-2	1	**6**	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
C	0	-3	-3	-3	**9**	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Q	-1	1	0	0	-3	**5**	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
E	-1	0	0	2	-4	2	**5**	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
G	0	-2	0	-1	-3	-2	-2	**6**	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
H	-2	0	1	-1	-3	0	0	-2	**8**	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	**4**	2	-3	1	0	-3	-2	-1	-3	-1	3
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	**4**	-2	2	0	-3	-2	-1	-2	-1	1
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	**5**	-1	-3	-1	0	-1	-3	-2	-2
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	**5**	0	-2	-1	-1	-1	-1	1
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	**6**	-4	-2	-2	1	3	-1
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	**7**	-1	-1	-4	-3	-2
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	**4**	1	-3	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	**5**	-2	-2	0
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	**11**	2	-3
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	**7**	-1
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	**4**

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 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)

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

$S[1], \ldots, S[i]$ with $T[1], \ldots, T[j]$

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

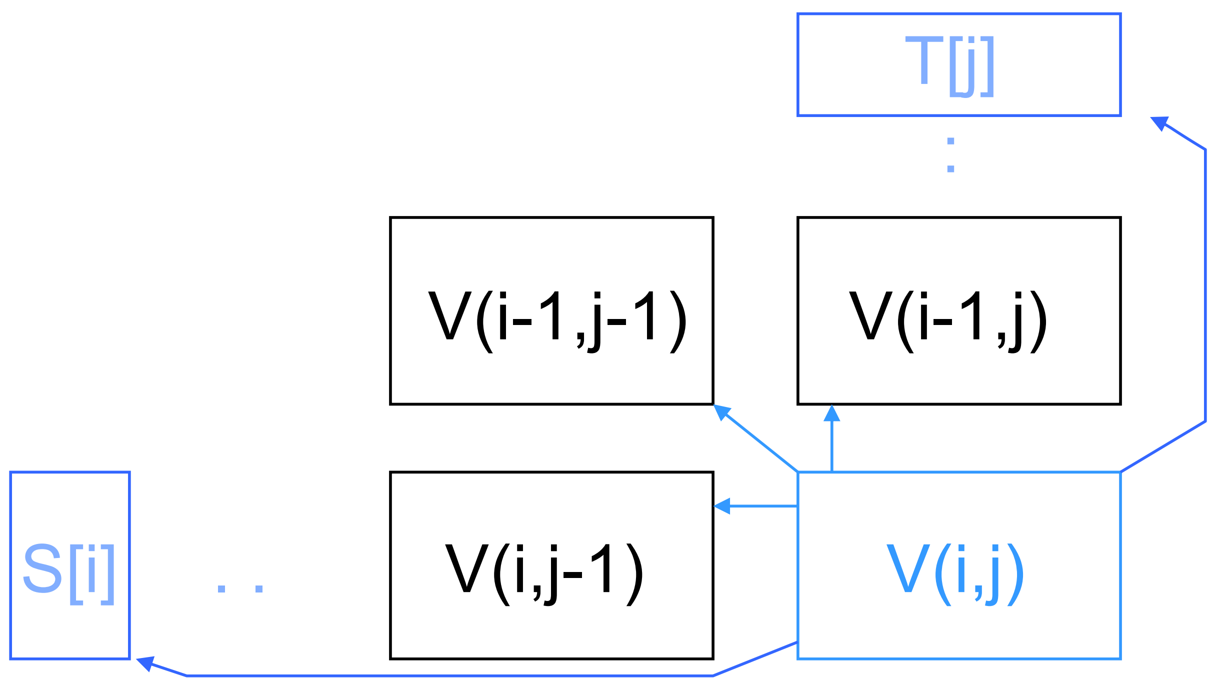
$$V(i,j) = \max \begin{Bmatrix} V(i\text{-}1, j\text{-}1) + \sigma(S[i], T[j]) \\ V(i\text{-}1, j) + \sigma(S[i], -) \\ V(i, j\text{-}1) + \sigma(-, T[j]) \end{Bmatrix},$$

↖ ↑ ←

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

Calculating One Entry

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



Mismatch = -1
Match = 2

Example

i	j	0	1	2	3	4	5
			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

c
-

Score(c,-) = -1

Mismatch = -1
Match = 2

Example

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

Score(-,a) = -1

Mismatch = -1
Match = 2

Example

i	j	0	1	2	3	4	5	
			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

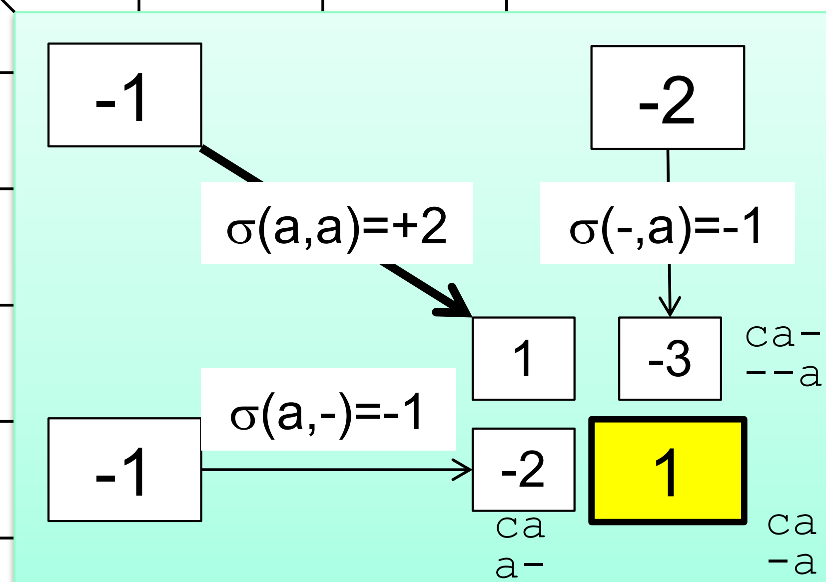
Mismatch = -1
Match = 2

Example

i		j 0	1	2	3	4	5
			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					

←T

↑
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

↑ S

Time = O(mn)

Mismatch = -1
Match = 2

Example

j		0	1	2	3	4	5
i			c	a	t	g	t	←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

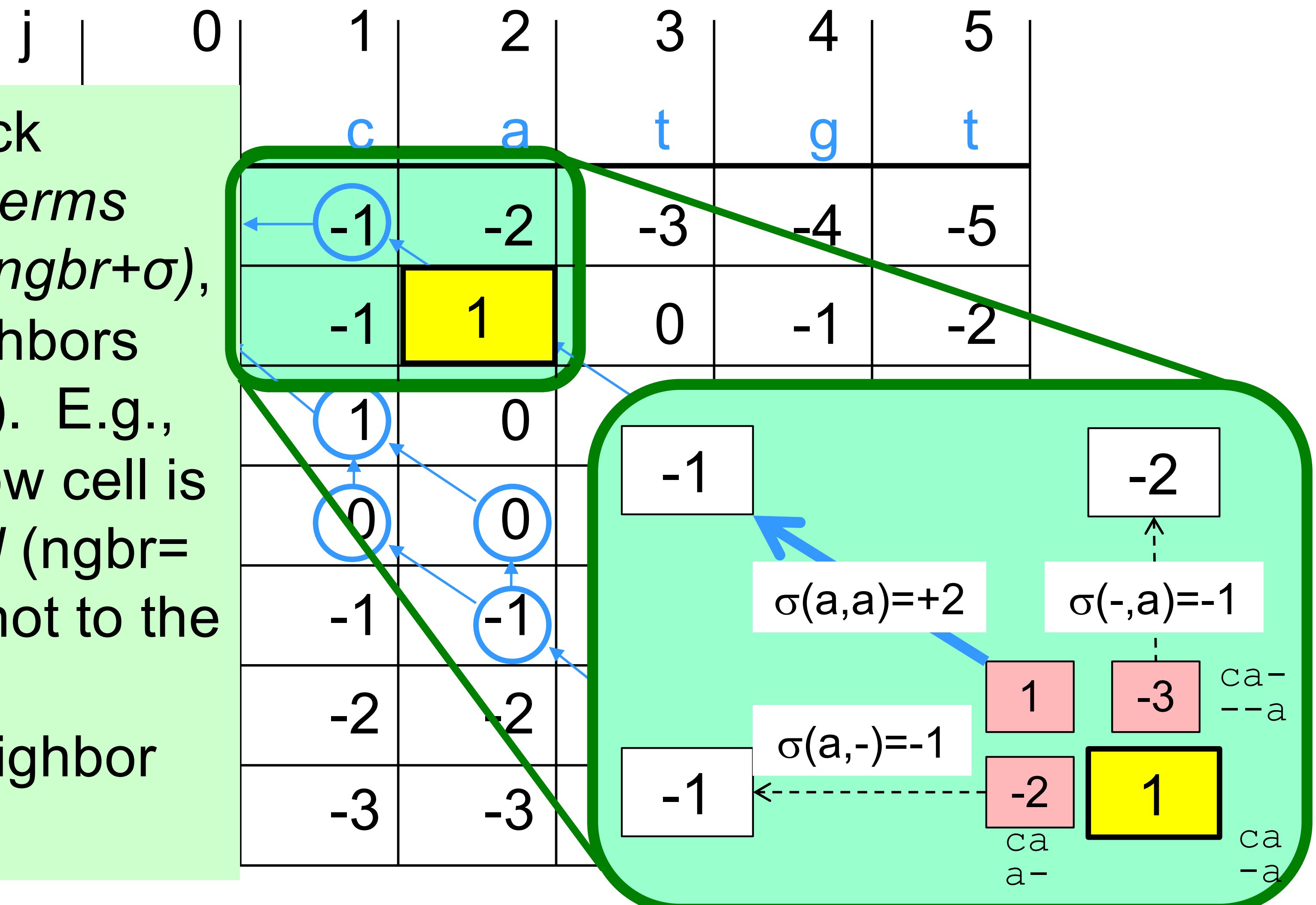
	j	0	1	2	3	4	5	
i			c	a	t	g	t	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1	-1	1	0	-1	-2	
2	c	-2	1	0	0	-1	-2	
3	g	-3	0	0	-1	2	1	
4	c	-4	-1	-1	-1	1	1	
5	t	-5	-2	-2	1	0	3	
6	g	-6	-3	-3	0	3	2	
	↑ S							

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+σ)*, 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 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. (KT section 6.7)

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 dashes cost 10 x one dash?
- 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 (toy σ)

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 $0 \leq i \leq n$, $0 \leq j \leq m$:

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

V(i,0): some suffix of first i chars of S; all match dashes 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]:

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

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

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

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 *term* (not max neighbor)

	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	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

One align-ment is:

```
c-de
cxde
```

What's the other?

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"

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 gaps

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

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