CSE421 Algorithms

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 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

Bio

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

Other

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

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

```
a c b c d b          a c – – b c d b
   /   \    |           |     |    |
c a d b d            – c a d b – d –
```

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

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]:

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

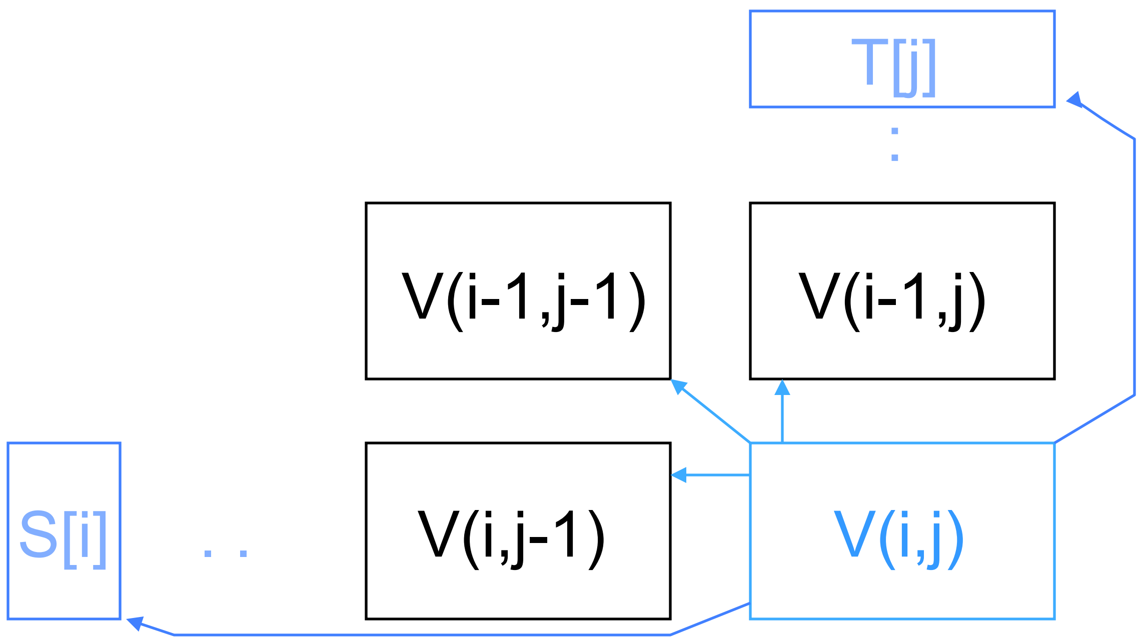
Opt align of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$

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

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

Calculating One Entry

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



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					
2	c	-2					
3	b	-3					
4	c	-4					
5	d	-5					
6	b	-6					

←T

↑
S

c
-

Score(c,-) = -1

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					
2	c	-2					
3	b	-3					
4	c	-4					
5	d	-5					
6	b	-6					

←T

↑ S

-
a

Score(-,a) = -1

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					
2	c	-2					
3	b	-3					
4	c	-4					
5	d	-5					
6	b	-6					

←T

↑
S

-	-
a	c

-1

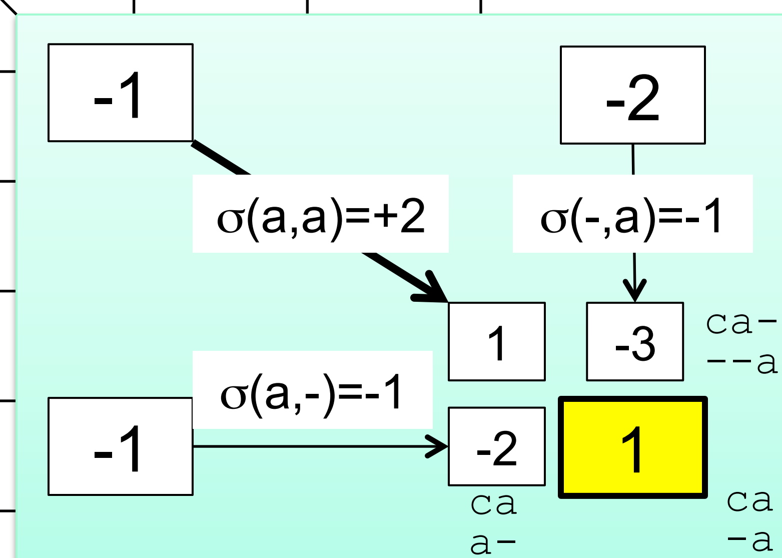
Score(-,c) = -1

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

↑ S

Time = O(mn)

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

	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

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”

Interesting stats problem; much is known

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

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

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