CSE 421 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

ATGTTAT vs ATCGTAC

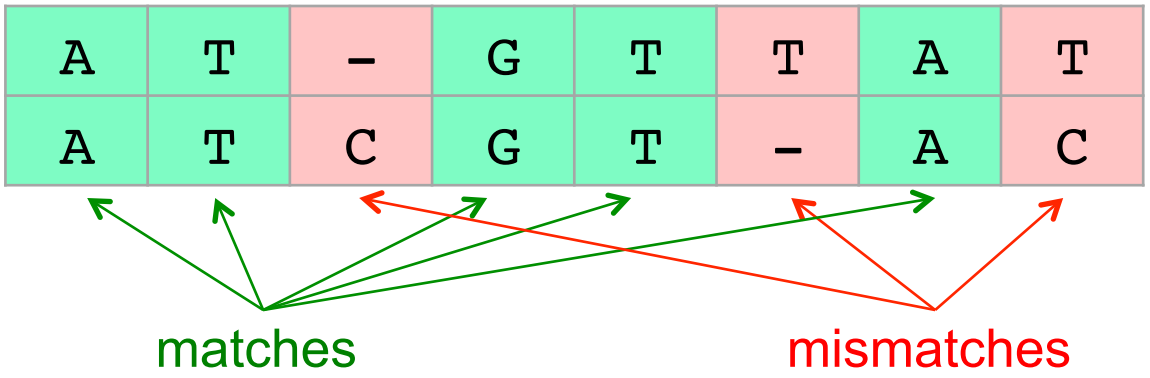
A	T	-	G	T	T	A	T
A	T	C	G	T	-	A	C

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



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, …

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.

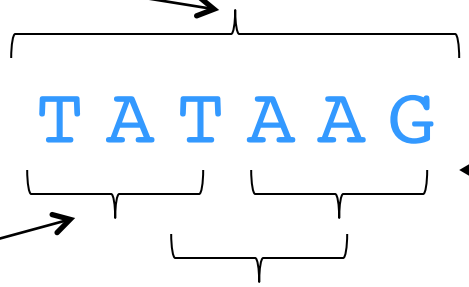
Alternate demo:

- 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



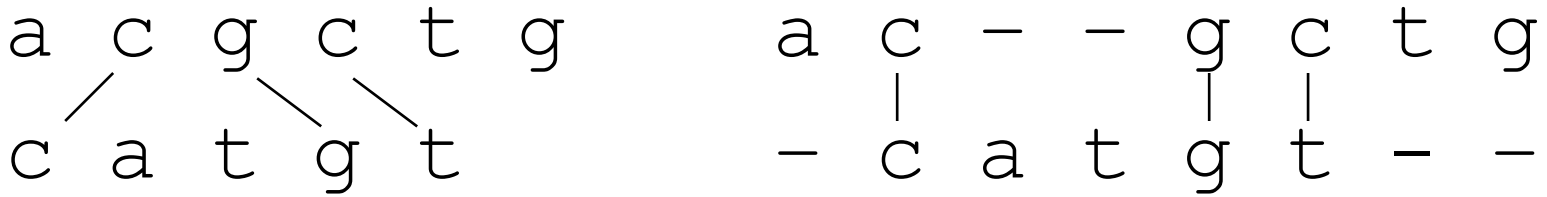
suffix
consecutive letters from back

prefix
consecutive letters from front

substring
consecutive letters from anywhere

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

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 (|S| = "length of S")
2. Removing dashes leaves S, T

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.

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

(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

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

(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]:

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

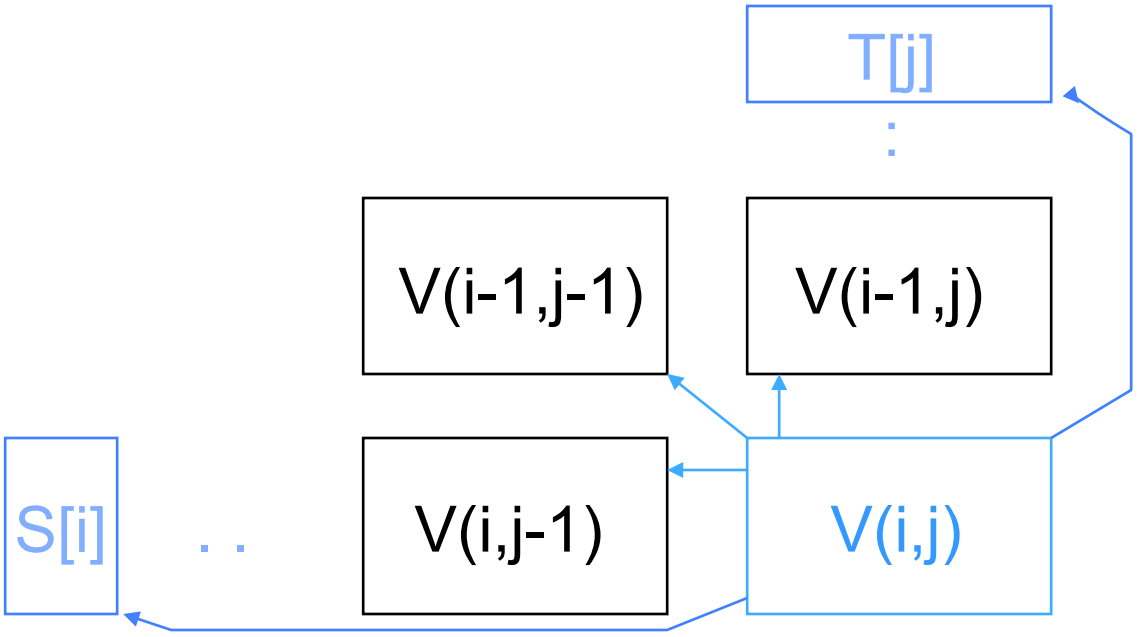
Opt align of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$

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

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	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

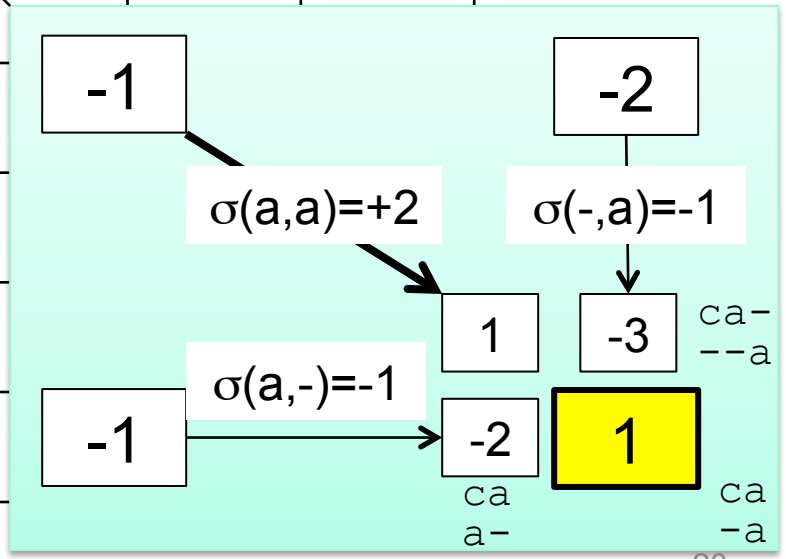
	j	0	1	2	3	4	5	
i			c	a	t	g	t	←T
0		0	-1	-2	-3	-4	-5	
1	a	-1						
2	c	-2						
3	g	-3						
4	c	-4						
5	t	-5						
6	g	-6						
	↑ S							

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				
2	c	-2						
3	g	-3						
4	c	-4						
5	t	-5						
6	g	-6						

S ↑



Mismatch = -1
Match = 2

Example

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

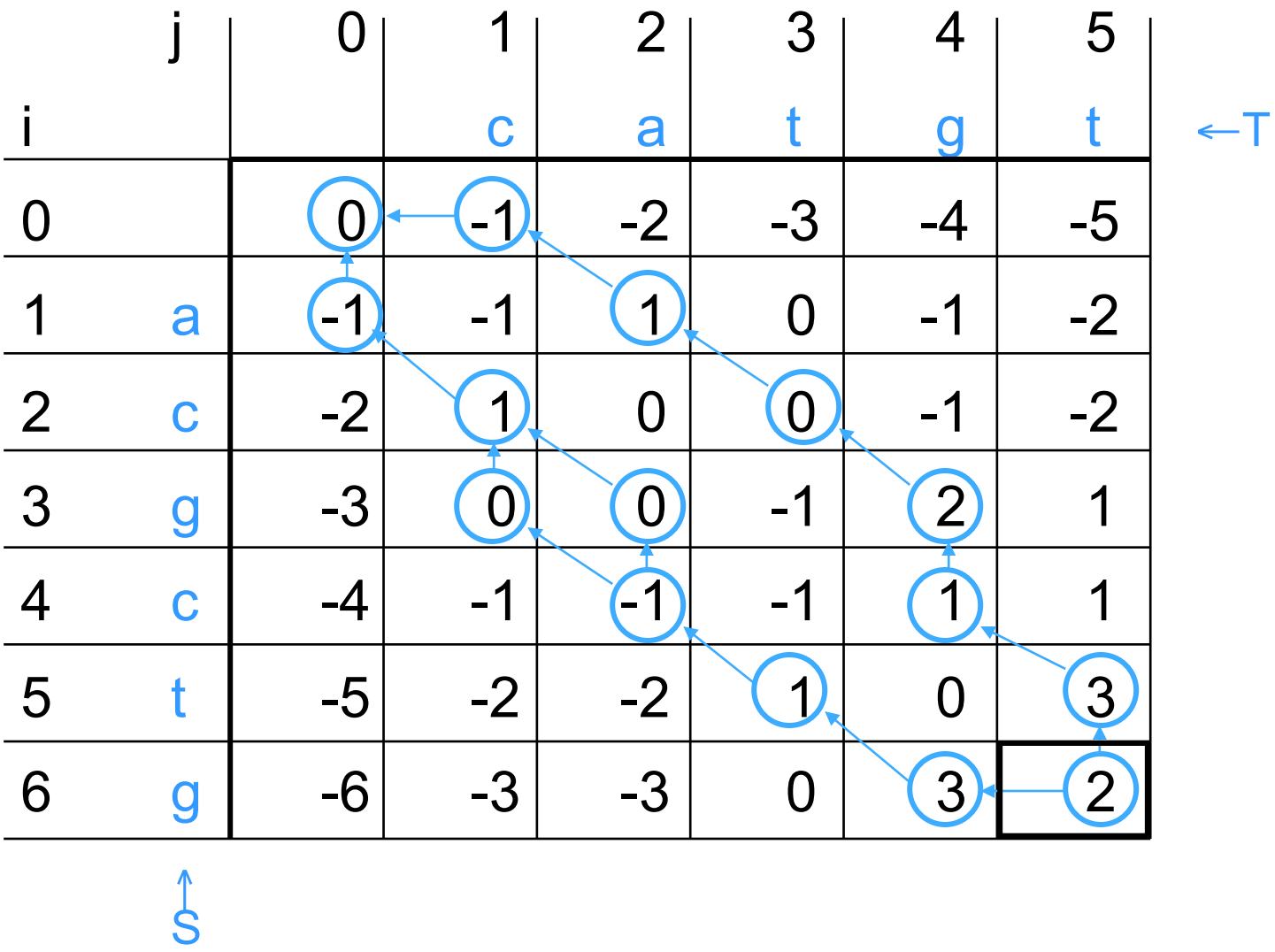
Example

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



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

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)) (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 spaces cost 10 x one space?
- 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