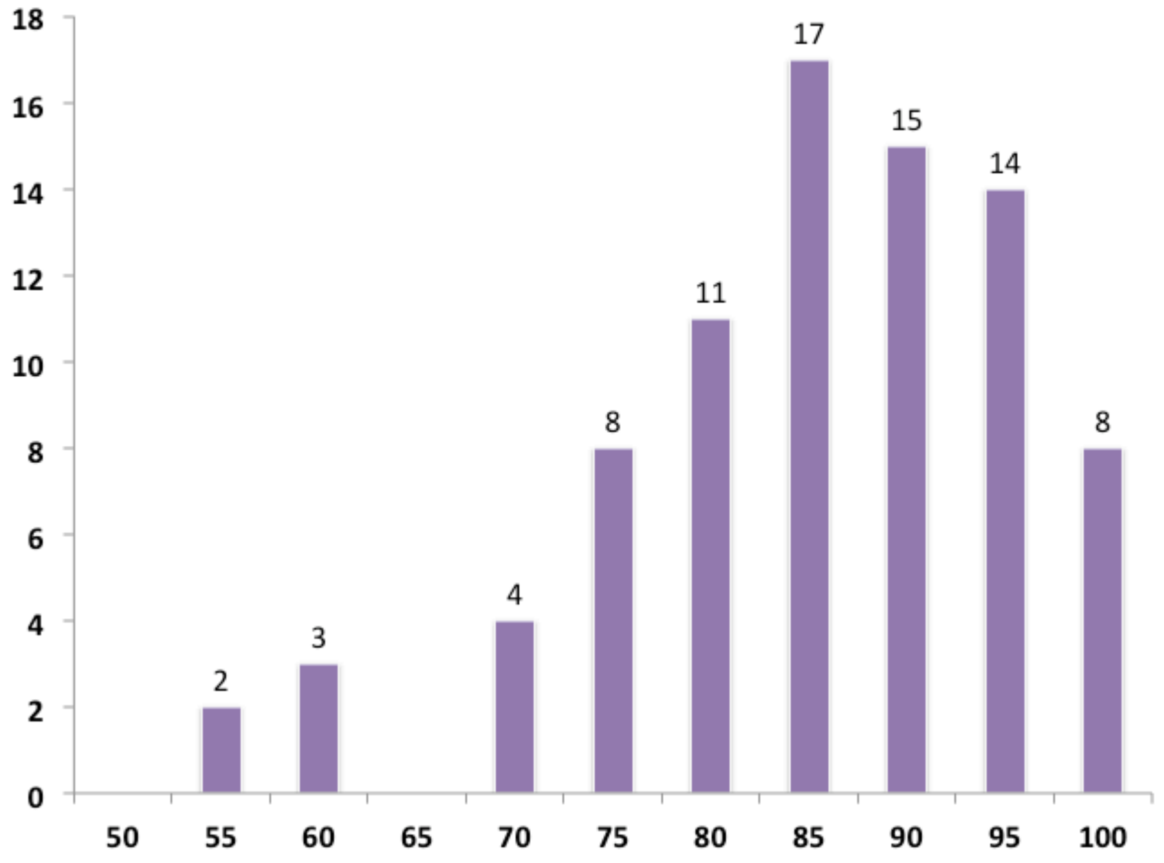
CSE 421 Midterm Scores



Mean 83
Sigma 11

CSE 421 Algorithms

Sequence Alignment

Sequence Alignment

Goal: position characters in strings so they “best” line up with one another

We can do this via Dynamic Programming

What is an alignment?

Compare two strings and see how similar they are

Maximize the # of chars in a string that line up

ATGTTAT vs ATCGTAC

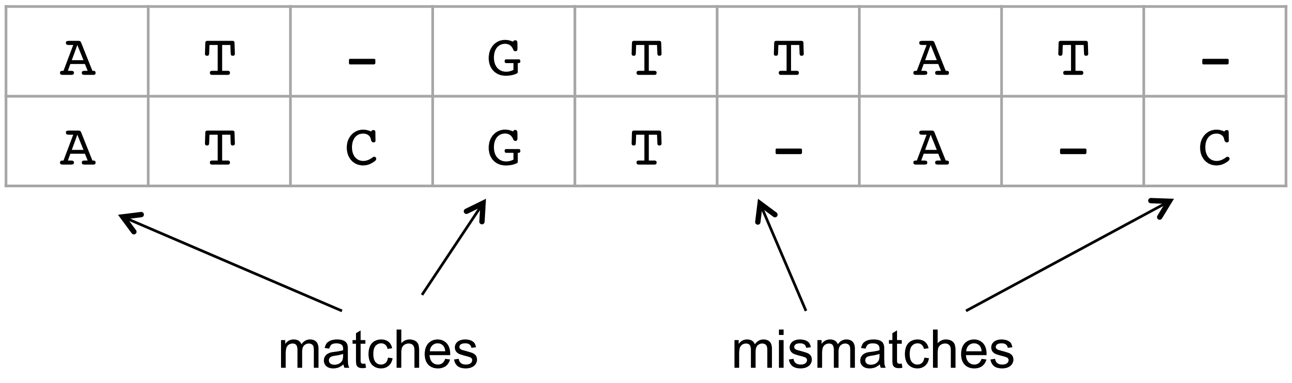
A	T	-	G	T	T	A	T	-
A	T	C	G	T	-	A	-	C

What is an alignment?

Compare two strings and see how similar they are

Maximize the # of chars in a string that line up

ATGTTAT vs ATCGTAC



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

Why do we align?

Biology

- Most widely used comp. tools in biology
- New sequences always compared to databases

Similar sequences often have similar origin and/or function

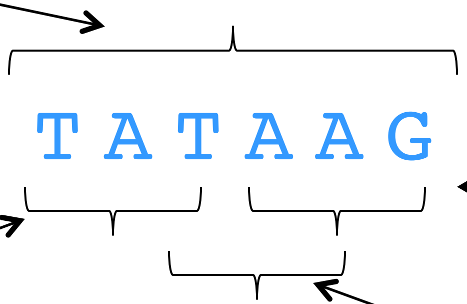
Other

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

Terminology

string
ordered list of letters

suffix
consecutive letters from back

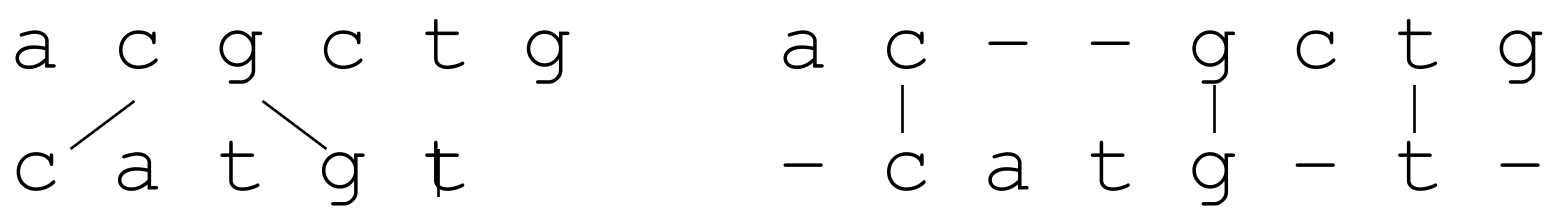


prefix
consecutive letters from front

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 represented as a pair of strings S', T' with gaps "–" s.t.

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

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

Scoring an arbitrary alignment

Want to determine whether an alignment is “good” or “bad” so we define a cost function

$$\text{score of (mis)aligning chars } x \,\&\, y \;=\; \sigma(x, y) \;=\; \begin{cases} \text{match} & 2 \\ \text{mismatch} & -1 \end{cases}$$

Total value/score of an alignment

$$\Sigma\, \sigma(S'[i], T'[i])$$

Optimal alignment

Max alignment score of all poss. alignments

Scoring an arbitrary alignment

```
 a   c   -   -   g   c   t   g
     |           |       |
 -   c   a   t   g   -   t   -

-1  +2  -1  -1  +2  -1  +2  -1
```

Score = +1

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

Can we use Dynamic Programming?

1. Identify **subproblems**

   We can reuse the solution to smaller substrings (prefixes in this case)

2. Argue that we have **optimal substructure**

   Appending two optimal alignments should also be optimally aligned (some may change at the interface)

Arguing for Optimal Substructure

Assume strings S & T are optimally aligned except for the last character

3 options for the last character:

1. match -- `S[i]` & `T[j]` aligned
2. mismatch -- `S[i]` & `"-"` aligned
3. mismatch -- `T[j]` & `"-"` aligned

* Never align `"-"` & `"-"`; i.e. $\sigma(\text{"-"}, \text{"-"}) << 0$

“Recipe” for using DP for problems like this

1. Argue for optimal substructure (☑)
2. Find a recursive relation for subproblem costs
   - Use (1), find all subproblems that might contribute to an optimal cost
3. Implement a bottom-up use of (2) to fill in a table of subproblem costs
4. Write a recursive algorithm using the table from (3) to construct actual solutions to subproblems (“traceback”)

Setting up Optimal Alignment in $O(n^2)$ via DP

Input: strings S, T

$|S| = n$, $|T| = m$

Output: optimal alignment score

→ Generate the score first and then trace backwards to recover the actual alignment

Setting up Optimal Alignment in $O(n^2)$ via DP

Compute optimal alignment of **all combinations of prefixes**, & store in a table for the future

T → S v	-	A	C	G	T	…	T
-	0	-1	-2	-3	-4		-n
A	-1	2	1	0	-1		
C	-2	1	4	3			
G	-3	0	3	★			
T	-4	-1					
…						…	
T	-n						

Start UL, nothing aligned
End LR, w/ optimal score

Move diagonally → align chars
Move vert/horiz → introduce gap

`V(i,j)` = optimal alignment score of `S[1]`…`S[i]` and `T[1]`…`T[j]`

i.e. all possible prefixes of S and T

Computing the table: Base Case

T → S v	-	A	C	G	T	…	T
-	0	-1	-2	-3	-4		-n
A	-1	2	1	0	-1		
C	-2	1	4	3			
G	-3	0	3				
T	-4	-1					
…						…	
T	-n						

Column:

S aligns with nothing in T

all mismatches

```
V(i,0) = Σσ(S[k], “-”)
       = i*σ(S[k], “-”)
```

Row:

T aligns with nothing in S

all mismatches

```
V(0,j) = Σσ(“-”, T[k])
       = j*σ(“-”, T[k])
```

Computing the table: General Case

T → S v	-	A	C	G	T	…	T
-	0	-1	-2	-3	-4		-n
A	-1	2	1	0	-1		
C	-2	1	4	3			
G	-3	0	3	★			
T	-4	-1					
…						…	
T	-n						

At any given point in computing the table, we can choose whether it's best to

Align 2 characters

Take a gap

Computing the table: General Case

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

$\uparrow$ Cost of ops so far

$\uparrow$ Cost of next op (match/mismatch)

	-	A	C	G	T	…	T
-	0	-1	-2	-3	-4		-n
A	-1	2	1	0	-1		
C	-2	1	4	3			
G	-3	0	3	★			
T	-4	-1					
…						…	
T	-n						

Need these 3 positions filled in to determine ★

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

Example: base case

T → S v			C	A	T	G	T
		i=0	1	2	3	4	5
	j=0	0	-1	-2	-3	-4	-5
A	1	-1					
C	2	-2					
G	3	-3					
C	4	-4					

```
V(i,0) = i*σ(S[k], “-”)
V(0,j) = j*σ(“-”,,T[k])
```

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

Example: general step

T → S v			C	A	T	G	T
		i=0	1	2	3	4	5
	j=0	0	-1	-2	-3	-4	-5
A	1	-1	-1	🟩			
C	2	-2					
G	3	-3					
C	4	-4					

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

Example: general step

			C	A	T	G	T
T → S v		i=0	1	2	3	4	5
	j=0	0	-1	-2	-3	-4	-5
A	1	-1	-1	🟩			
C	2	-2					
G	3	-3					
C	4	-4					

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

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

Example: general step

T →
S v

			C	A	T	G	T
		i=0	1	2	3	4	5
	j=0	0	-1	-2	-3	-4	-5
A	1	-1	-1	🟩			
C	2	-2					
G	3	-3					
C	4	-4					

$$V(i, j) = \max \begin{cases} \texttt{V(0,1) + σ(S[1], T[2])} & \nwarrow \\ \texttt{V(0,2) + σ(S[1], “-”)} & \uparrow \\ \texttt{V(1,1) + σ(“-”, T[2])} & \leftarrow \end{cases}$$

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

Example: general step

T → / S v		i=0	C 1	A 2	T 3	G 4	T 5
	j=0	0	-1	-2	-3	-4	-5
A	1	-1	-1	1			
C	2	-2					
G	3	-3					
C	4	-4					

$$V(i, j) = \max \begin{cases} -1 + 2 = \mathbf{1,\ match} \nwarrow \\ -2\ -1 = -3 \\ -1\ -1 = -2 \end{cases}$$

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

Example: completed table

T → / S v		C	A	T	G	T
	i=0	1	2	3	4	5
j=0	0	-1	-2	-3	-4	-5
A 1	-1	-1	1	0	-1	-2
C 2	-2	1	0	0	-1	-2
G 3	-3	0	0	-1	2	1
C 4	-4	-1	-1	-1	1	1

Time = O(mn) = O(|S|*|T|)

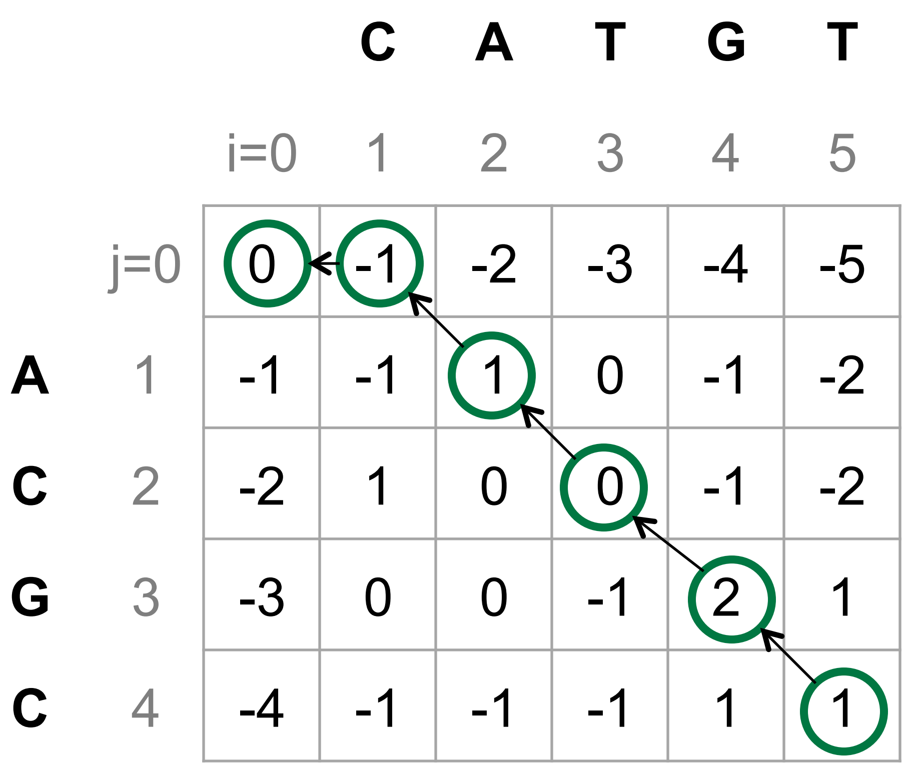
How do we find the alignment itself? Traceback

Trace LR to UL following highest score path

Can go ↖ ↑ ←

Multiple optimal alignments are possible

We can break ties arbitrarily



			C	A	T	G	T
		i=0	1	2	3	4	5
	j=0	0	-1	-2	-3	-4	-5
A	1	-1	-1	1	0	-1	-2
C	2	-2	1	0	0	-1	-2
G	3	-3	0	0	-1	2	1
C	4	-4	-1	-1	-1	1	1

Corresponding Alignment:

```
CATGT
-ACGC
```

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	$\boxed{2}$	
	↑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)) (KT section 6.7)

Significance of Alignments

Is “42” a good score? *Compared to what?*

Easier to compare when using standardized scoring functions, esp. for DNA

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, 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