# CSE 521 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 G

## Sequence Similarity: What

G G A C C A

```
T A C T A A G
|   |   | | |
T C C – A A G
```

## 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

Mismatch = -1
Match = 2

## Alignment Scoring

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

## Optimal Substructure

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]:

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

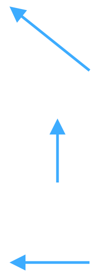
Opt align of $S_1 \ldots S_{i-1}$ & $T_1 \ldots T_{j-1}$

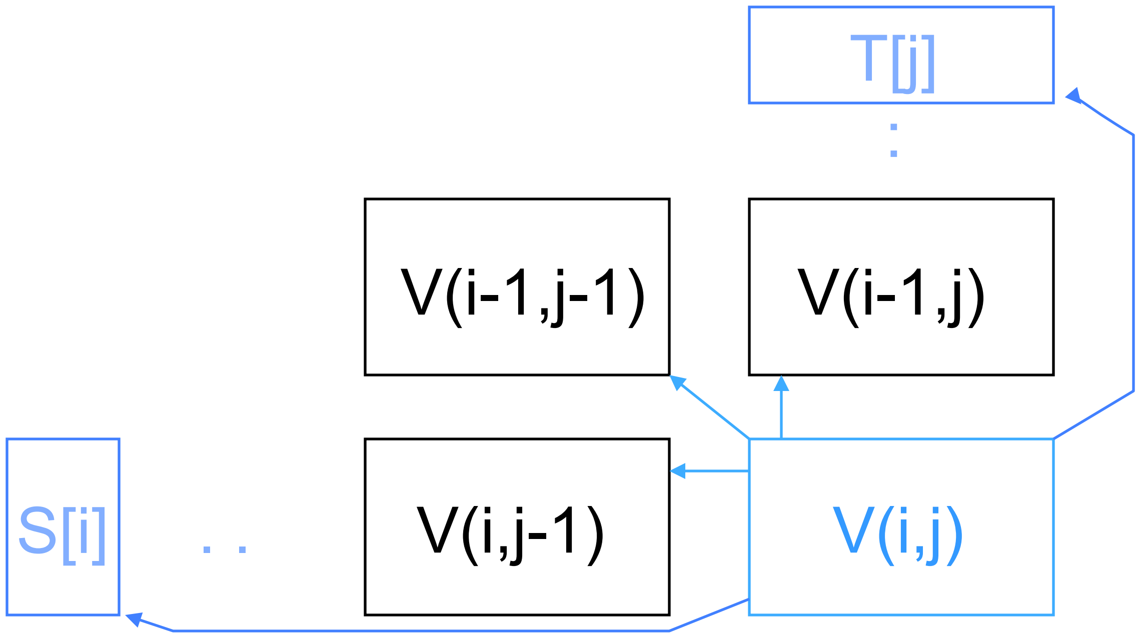
$$V(i,j) = \max \begin{Bmatrix} V(i\text{-}1,j\text{-}1) + \sigma(S[i],T[j]) \\ V(i\text{-}1,j) + \sigma(S[i], -) \\ V(i,j\text{-}1) + \sigma(-, T[j]) \end{Bmatrix}, \quad \begin{matrix} \nwarrow \\ \uparrow \\ \leftarrow \end{matrix}$$

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

## Calculating One Entry

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





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 | | | | | | |
| 2 | c | -2 | | | | | | |
| 3 | b | -3 | | | | | | |
| 4 | c | -4 | | | | | | |
| 5 | d | -5 | | | | | | |
| 6 | b | -6 | | | | | | |
| | ↑ S | | | | | | | |

$\boxed{\begin{array}{c} c \\ - \end{array}}$ 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 | $\boxed{-1}$ |   |   |   |   |   |
| 2 | c | -2 |   |   |   |   |   |
| 3 | b | -3 |   |   |   |   |   |
| 4 | c | -4 |   |   |   |   |   |
| 5 | d | -5 |   |   |   |   |   |
| 6 | b | -6 |   |   |   |   |   |

←T

↑
S

$\boxed{\begin{array}{c}-\\a\end{array}}$ Score(-,a) = -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 | | | | | |
| 2 | c | $\boxed{-2}$ | | | | | |
| 3 | b | -3 | | | | | |
| 4 | c | -4 | | | | | |
| 5 | d | -5 | | | | | |
| 6 | b | -6 | | | | | |

↑
S

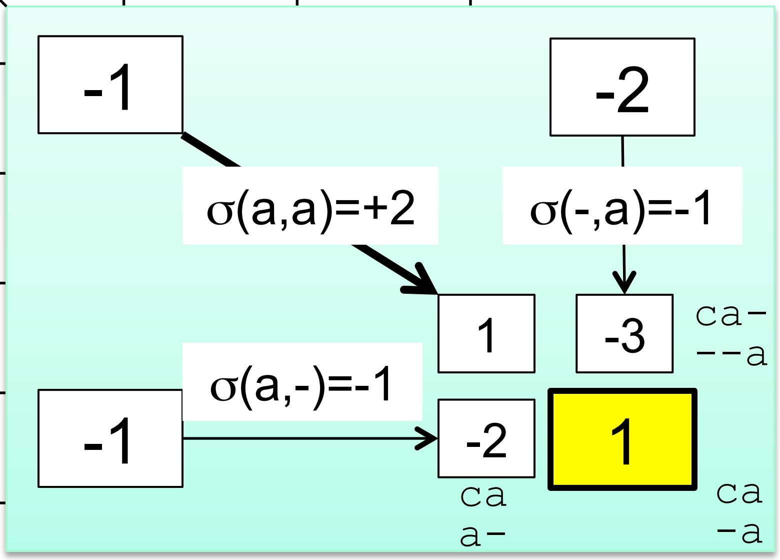
$\begin{array}{|c|c|} - & - \\ a & c \end{array}$ Score(-,c) = -1

-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 | ←T |
| 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 | | | | | | |
| | ↑ S | | | | | | | |

Time = O(mn)

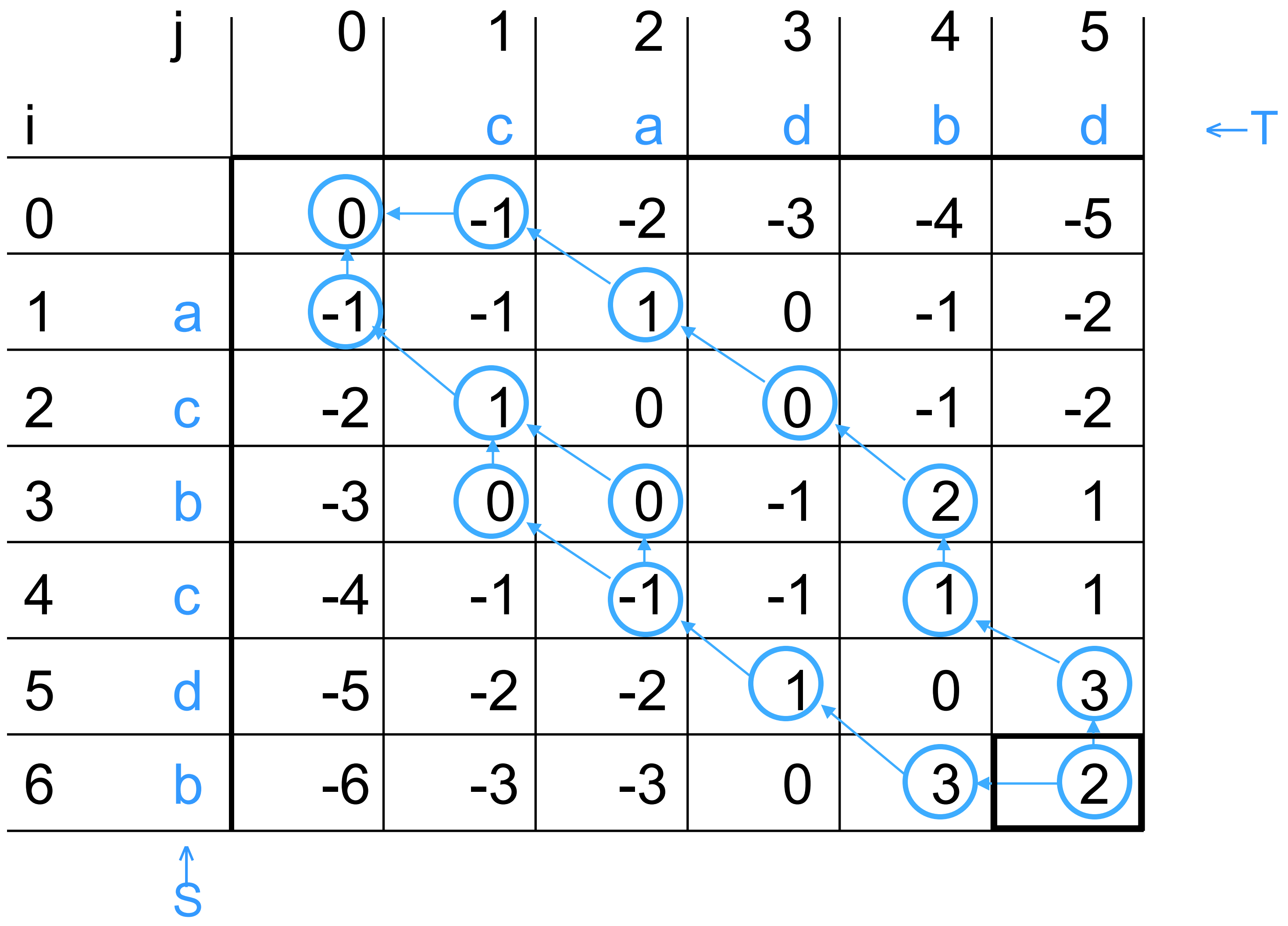
Mismatch = -1
Match = 2

# 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 | $\boxed{2}$ |
| | ↑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 |
| 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 |

←T

↑
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?*

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

## Alignment With Gap Penalties

*Gap:* maximal run of spaces in S' or T'

| | |
|---|---|
| `ab--ddc-d` | 2 gaps in S' |
| `a---ddcbd` | 1 gap in T' |

(NB: KT treats "gap" and "-" as synonyms)

Motivations, e.g.:

mutation might insert/delete several or even many residues at once

matching mRNA (no introns) to genomic DNA (exons and introns)

some parts of proteins less critical

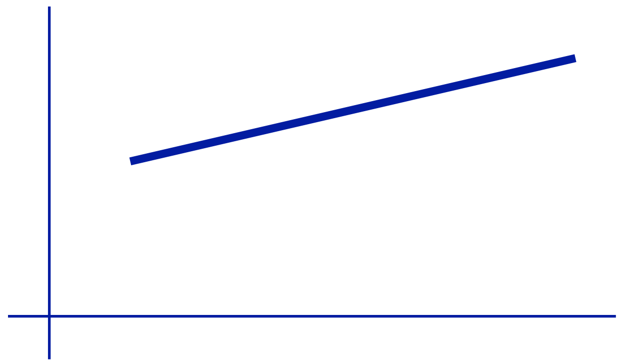
# A Protein Structure: (Dihydrofolate Reductase)

# Alignment of 5 Dihydrofolate reductase proteins

```
mouse   P00375   ----MVRPLNCIVAVSQNMGIGKNGDLPWPPLRNEFKYFQRMTTTSSVEGKQNLVIMGRK
human   P00374   ----MVGSLNCIVAVSQNMGIGKNGDLPWPPLRNEFRYFQRMTTTSSVEGKQNLVIMGKK
chicken P00378   -----VRSLNSIVAVCQNMGIGKDGNLPWPPLRNEYKYFQRMTSTSHVEGKQNAVIMGKK
fly     P17719   ----MLR-FNLIVAVCENFGIGIRGDLPWR-IKSELKYFSRTTKRTSDPTKQNAVVMGRK
yeast   P07807   MAGGKIPIVGIVACLQPEMGIGFRGGLPWR-LPSEMKYFRQVTSLTKDPNKKNALIMGRK
                      :  .. :..:  ::***  *.***  : .* :** : *. :    *:* ::**:*

        P00375   TWFSIPEKNRPLKDRINIVLSRELKEP----PRGAHFLAKSLDDALRLIEQPELASKVDM
        P00374   TWFSIPEKNRPLKGRINLVLSRELKEP----PQGAHFLSRSLDDALKLTEQPELANKVDM
        P00378   TWFSIPEKNRPLKDRINIVLSRELKEA----PKGAHYLSKSLDDALALLDSPELKSKVDM
        P17719   TYFGVPESKRPLPDRLNIVLSTTLQESDL--PKG-VLLCPNLETAMKILEE---QNEVEN
        P07807   TWESIPPKFRPLPNRMNVIISRSFKDDFVHDKERSIVQSNSLANAIMNLESN-FKEHLER
                 *: .:* . *** .*:*:::*  :::      .     . .*  *:   :.    ..::

        P00375   VWIVGGSSVYQEAMNQPGHLRLFVTRIMQEFESDTFFPEIDLGKYKLLPEYPG-------
        P00374   VWIVGGSSVYKEAMNHPGHLKLFVTRIMQDFESDTFFPEIDLEKYKLLPEYPG-------
        P00378   VWIVGGTAVYKAAMEKPINHRLFVTRILHEFESDTFFPEIDYKDFKLLTEYPG-------
        P17719   IWIVGGSGVYEEAMASPRCHRLYITKIMQKFDCDTFFPAIP-DSFREVAPDSD-------
        P07807   IYVIGGGEVYSQIFSITDHWLITKINPLDKNATPAMDTFLDAKKLEEVFSEQDPAQLKEF
                 ::::**  **.  :  .    :   . :..    :: . :   . . :    .

        P00375   VLSEVQ------------EEKGIKYKFEVYEKKD---
        P00374   VLSDVQ------------EEKGIKYKFEVYEKND---
        P00378   VPADIQ------------EEDGIQYKFEVYQKSVLAQ
        P17719   MPLGVQ------------EENGIKFEYKILEKHS---
        P07807   LPPKVELPETDCDQRYSLEEKGYCFEFTLYNRK----
                 :   ::            **.*  ::: : ::
```

*CLUSTAL W (1.82) multiple sequence alignment*
http://pir.georgetown.edu/cgi-bin/multialn.pl
2/11/2013

## Affine Gap Penalties



gap open penalty

gap extend penalty

Gap penalty = g + e*(gaplen-1), g ≥ e ≥ 0

Note: no longer suffices to know just the *score* of best subproblem(s) – *state* matters: do they end with '-' or not.

# Global Alignment with Affine Gap Penalties

V(i,j) = value of opt alignment of S[1], …, S[i] with T[1], …, T[j]

G(i,j) = …, s.t. last pair matches S[i] & T[j]

F(i,j) = …, s.t. last pair matches S[i] & –

E(i,j) = …, s.t. last pair matches – & T[j]

| S | T |
|---|---|
| x/– | x/– |
| x | x |
| x | – |
| – | x |

Time: O(mn) [calculate all, O(1) each]

# Affine Gap Algorithm

gap open penalty

gap extend penalty

Gap penalty = $g + e*(\text{gaplen}-1)$, $g \geq e \geq 0$

$V(i,0) = E(i,0) = V(0,i) = F(0,i) = -g-(i-1)*e$

$V(i,j) = \max(G(i,j), F(i,j), E(i,j))$

$G(i,j) = V(i-1,j-1) + \sigma(S[i],T[j])$

$F(i,j) = \max( \boxed{F(i-1,j)-e} , \boxed{V(i-1,j)-g} )$

$E(i,j) = \max( \boxed{E(i,j-1)-e} , \boxed{V(i,j-1)-g} )$

old gap new gap

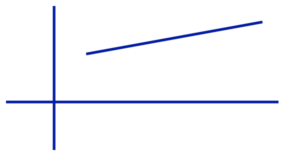
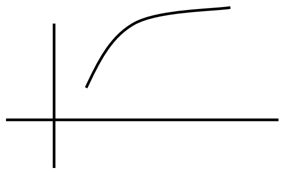
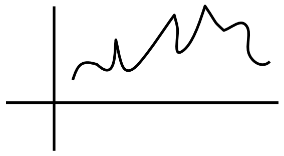
| S | T |
|---|---|
| x/– | x/– |
| x | x |
| x | – |
| – | x |

Q. Why is the “V” case a “new gap” when V includes E & F?

## Other Gap Penalties

Score = f(gap length)

Kinds, & best known alignment time

| | | |
|---|---|---|
| ☞ affine |  | $O(n^2)$ [really, $O(mn)$] |
| convex |  | $O(n^2 \log n)$ |
| general |  | $O(n^3)$ |

## 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

A *really* important algorithm design paradigm