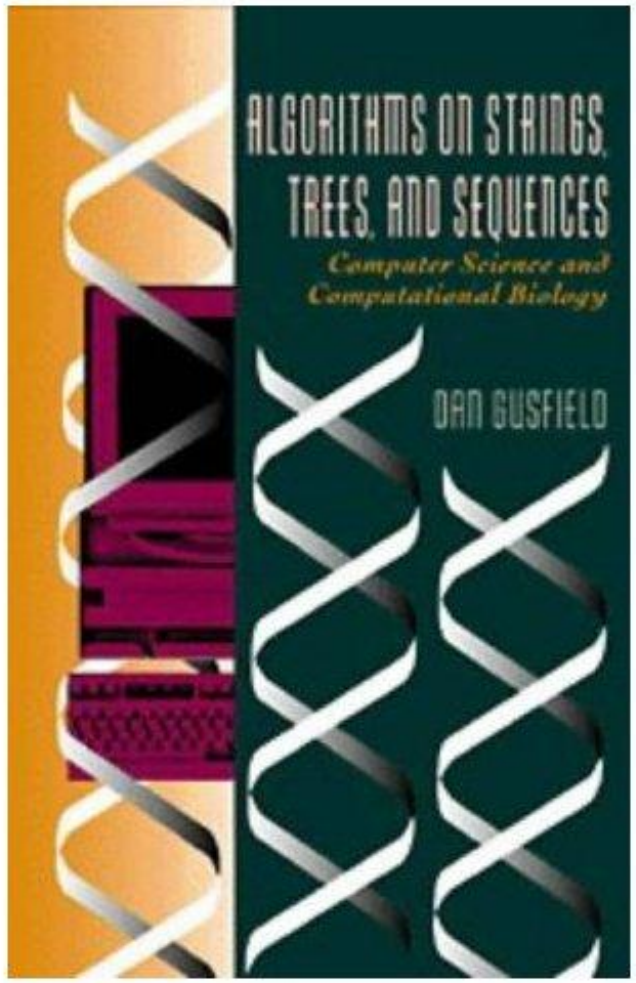
Pairwise Alignment

Main source


ALGORITHMS ON STRINGS, TREES, AND SEQUENCES
Computer Science and Computational Biology
DAN GUSFIELD

Why compare sequences?

Human hexosaminidase A vs Mouse hexosaminidase A

www.mathworks.com/.../jan04/bio_genome.html

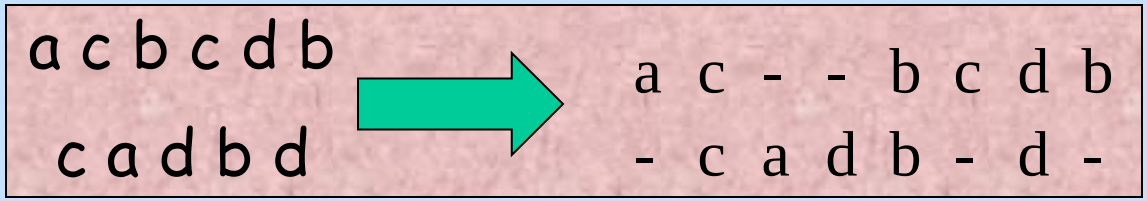
Sequence Alignment

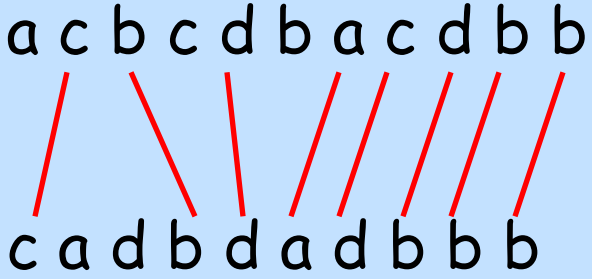
עימוד רצפים

- **The problem**: Comparing two sequences while allowing certain mismatches between them.
- **Main motivation**:
  - Comparing DNA seqs and proteins from databases,
    - Comparing two or more sequences for similarity
    - Searching databases for related sequences and subsequences
    - Finding informative elements in protein and DNA sequences

Alignment definition

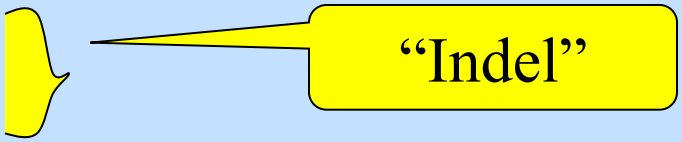
- **Input**: Two sequences of possibly different lengths
- **Goal**: Space the sequences so that they have the same length and no two spaces are matched.





Alignment scoring: Similarity vs. Difference

- **Resemblance** of DNA sequences of different organisms explained by common ancestral origin
- **Differences** are explained by mutation:
  - Insertion
  - Deletion
  - Substitution



- **Distance** between two sequences is the minimum (weighted) sum of mutations transforming one into the other.
- **Similarity** of two sequences is the maximum (weighted) sum of resemblances between them.

Nomenclature

Computer Science:	Biology:
String, word	Sequence
Substring (contiguous)	Subsequence
Subsequence	N/a
Exact matching	N/a
inexact matching	Alignment

non contiguous segment of a sequence

We shall use the biology nomenclature

Simplest model: Edit Distance

The **edit distance** between two sequences is the min no. of edit operations (single letter insertion, deletion and substitution) needed to transform one sequence into the other.

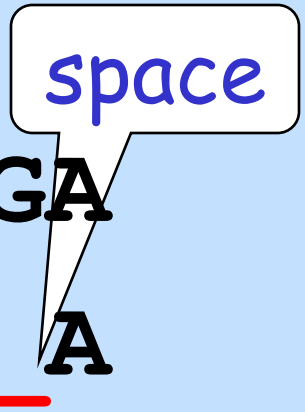
ACCTGA and AGCTTA

ACCTGA

AGCCTGA

AGCTTGA

AGCTTA



A_CCTGA

AGCTT_A

3 operations

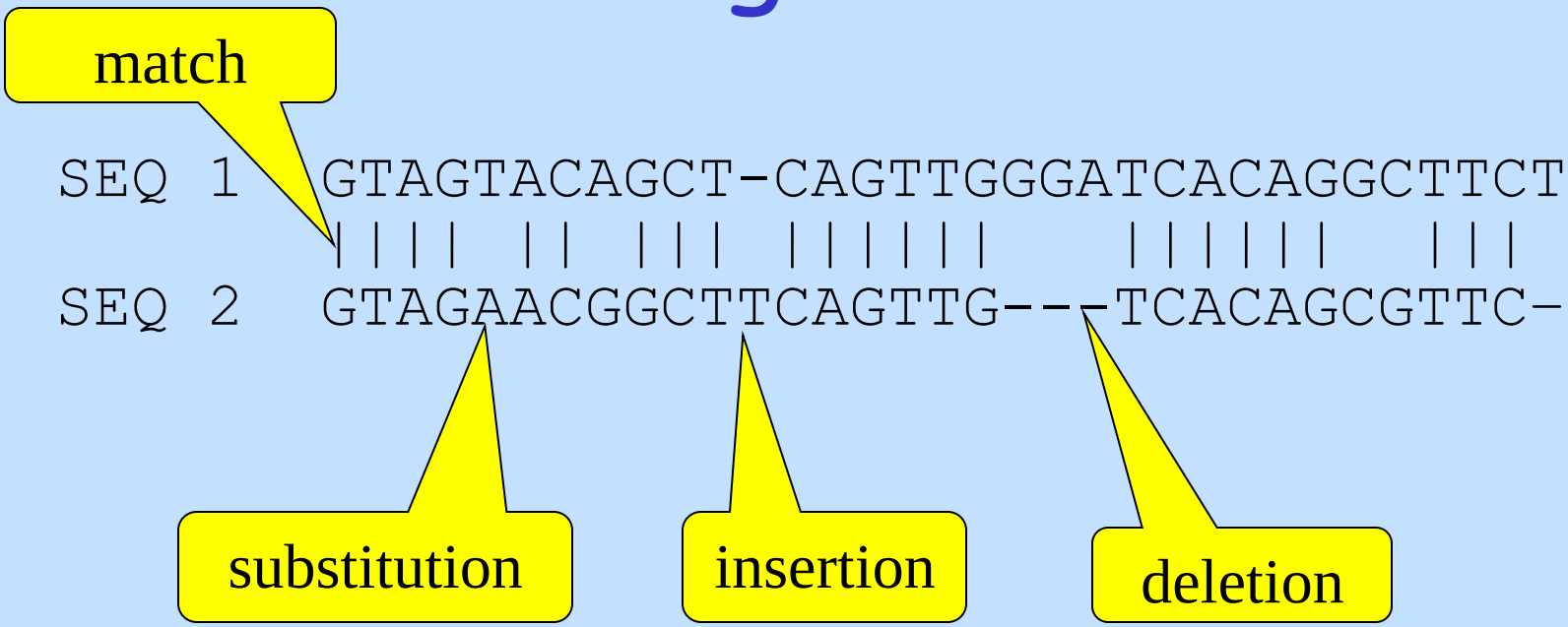
ACCTGA

AGCTTA

2 operations

dist=2

Alignment



- 24 matches,
- Subs: TA, AG, GC, CG
- Indels -T, G-, G-, A-, T-
- Distance I : match 0, subs 1, indel 2 ➔ dist = 14

Alignment

```
SEQ 1   GTAGTACAGCT-CAGTTGGGATCACAGGCTTCT
        ||||  ||  |||  ||||||    ||||||  |||
SEQ 2   GTAGAACGGCTTCAGTTG---TCACAGCGTTC-
```

- 24 matches, Subs: TA, AG, GC, CG, Indels -T, G-, G-, A-, T-

- Distance II: match 0, d(*A*,T)=d(*G*,*C*)=1, d(*A*,*G*)=1.5 indel 2

  ➔ dist=14.5

- Similarity I: match 1, subs 0, indel –1.5

  ➔ similarity =16.5

- **General setup: substitution matrix σ(i,j), indel σ(i,-)**

  Usually symmetric, Alignment – to - not allowed.

Global alignment

Human hexosaminidase A vs Mouse hexosaminidase A

Global Alignment Problem:

Input: Two sequences $S=s_1 \ldots s_n$, $T=t_1 \ldots t_m$ ($n \sim m$)

Goal: Find an optimal (max. similarity) alignment

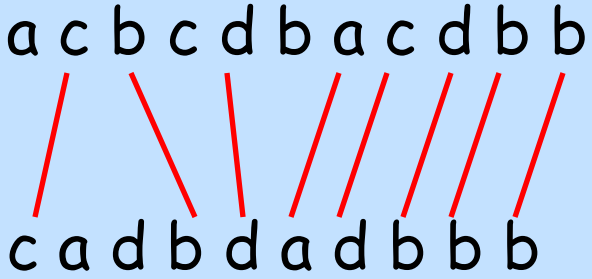
The scoring function is given.

www.mathworks.com/.../jan04/bio_genome.html

How many alignments are possible?

- Each alignment matches $0 \le k \le \min(n,m)$ pairs.
- #alignments with k matched pairs is $\binom{n}{k}\binom{m}{k}$

$$N = \sum_{k=0}^{\min(n,m)} \binom{n}{k}\binom{m}{k} = \binom{n+m}{\min(n,m)}$$



For n=m, ~4^n/sqrt(n)

Global Alignment Algorithm

- First dynamic programming solution by Needleman & Wunsch (70); improved later by Sankoff (72).

- $\sigma(a,b)$: the score (weight) of the alignment of character a with character b.

V(i,j) := optimal score of the alignment of $S'=s_1...s_i$ and $T'=t_1...t_j$ ($0 \le i \le n$, $0 \le j \le m$)

Lemma: V(i,j) has the following properties:

- Base conditions:
  - $V(i,0) = \Sigma_{k=0..i}\sigma(s_k,-)$
  - $V(0,j) = \Sigma_{k=0..j}\sigma(-,t_k)$
- Recurrence relation:

$$\forall 1\le i\le n,\ 1\le j\le m:\ V(i,j) = \max\begin{cases} V(i-1,j-1) + \sigma(s_i,t_j) \\ V(i-1,j) + \sigma(s_i,-) \\ V(i,j-1) + \sigma(-,t_j) \end{cases}$$

Alignment with 0 elements ≡ spaces

$S'=s_1...s_{i-1}$ with $T'=t_1...t_{j-1}$ s_i with t_j.

$S'=s_1...s_i$ with $T'=t_1...t_{j-1}$ and '-' with t_j.

Optimal Alignment - Tabular Computation

- Use dynamic programming to compute V(i,j) for all possible i,j values:

```
for i=1 to n do
begin
  For j=1 to m do
  begin
     Calculate V(i,j) using
     V(i-1,j-1), V(i,j-1), V(i-1,j)
  end
end
```

i \ j		0	1	2	3	4	5	
			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						

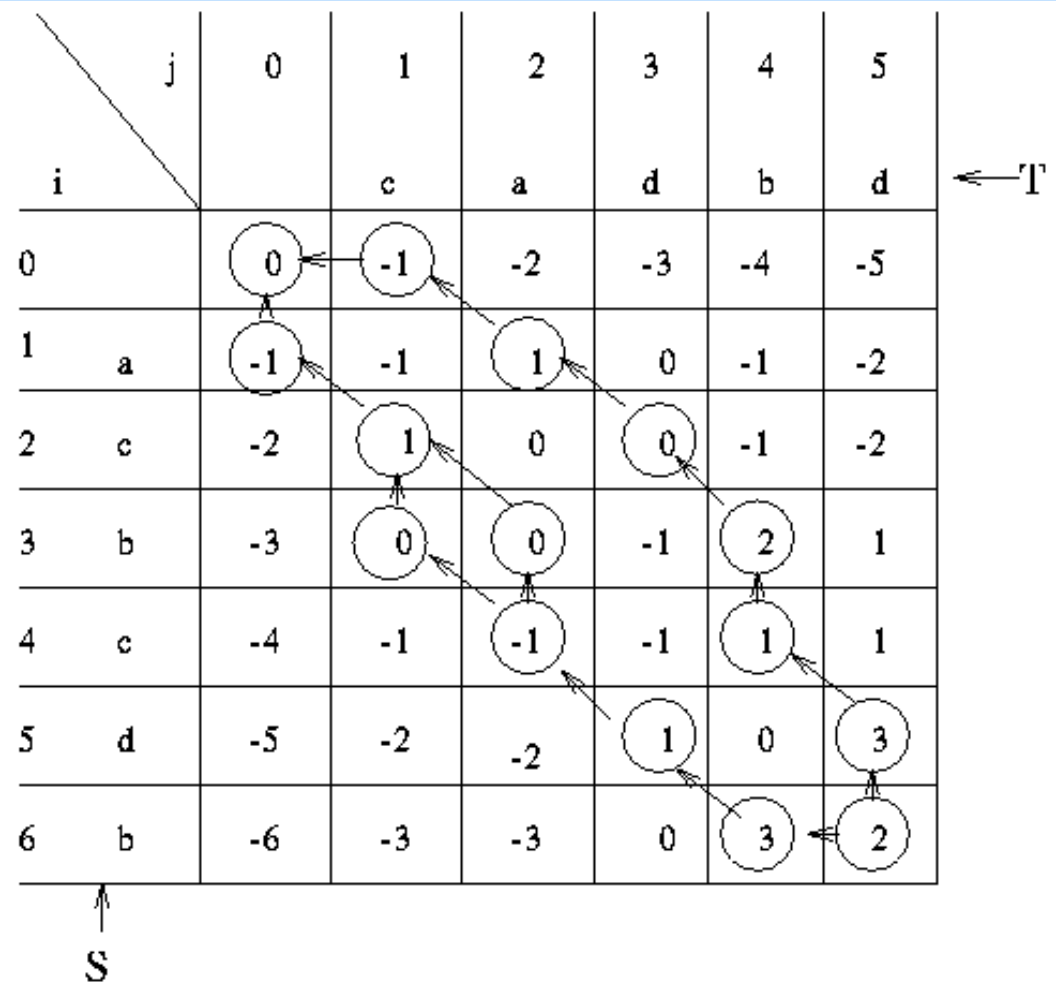
S

Costs: match 2, mismatch/indel -1

Snapshot of computing the table

Optimal Alignment - Tabular Computation

- Add back pointer(s) from cell (i,j) to father cell(s) realizing V(i,j).
- Trace back the pointers from (m,n) to (0,0)



Backtracking the alignment

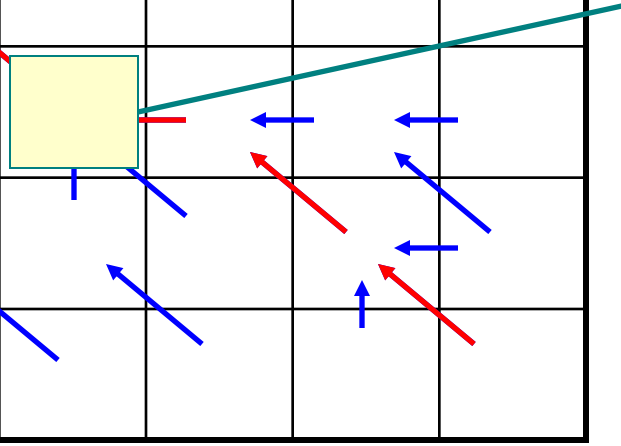
Example

x = AGTA
y = ATA

m = 1
s = -1
d = -1

F(i,j)

	i = 0	1	2	3	4
	`1`	``	A	G	T	A
j = 0						
1	A					
2	T					
3	A					



V(1, 1) =
max{V(0,0) + s(A, A),
V(0, 1) + d,
V(1, 0) + d} =

max{0 + 1,
-1 – 1,
-1 – 1} = 1

A G T A
A - T A

		λ	C	T	C	G	C	A	G	C
		0	-5	-10	-15	-20	-25	-30	-35	-40
λ	-5		10	5						
C	-10									
A	-15									
T	-20									
T	-25									
C	-30									
A	-35									
C										

+10 for match, -2 for mismatch, -5 for space

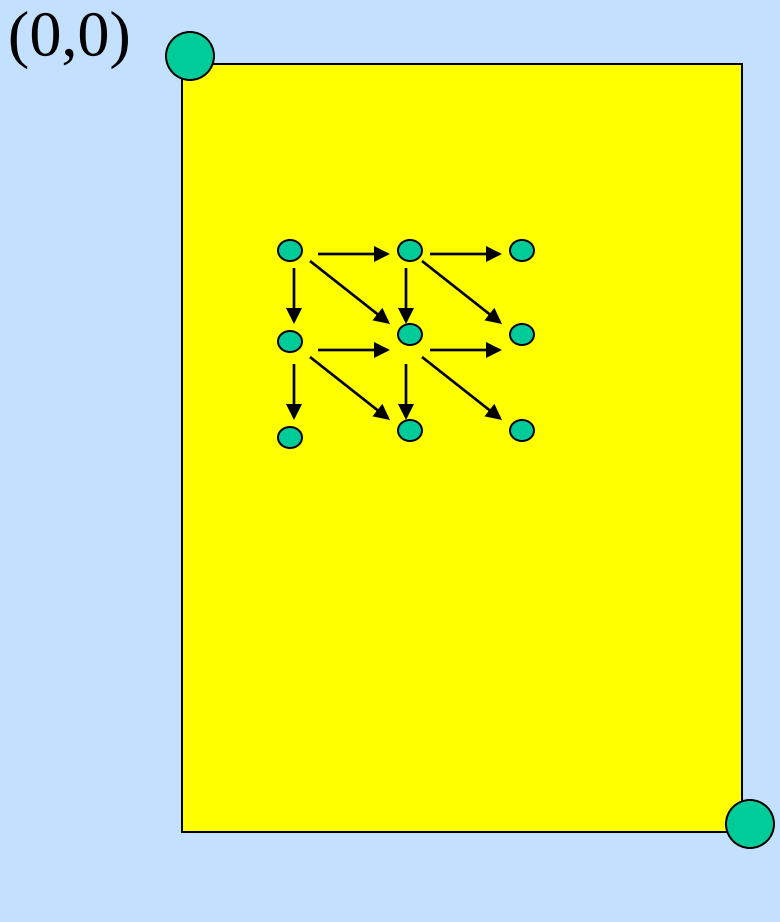
Fernandez-Baca & Dobbs http://www.cs.iastate.edu/~cs544/

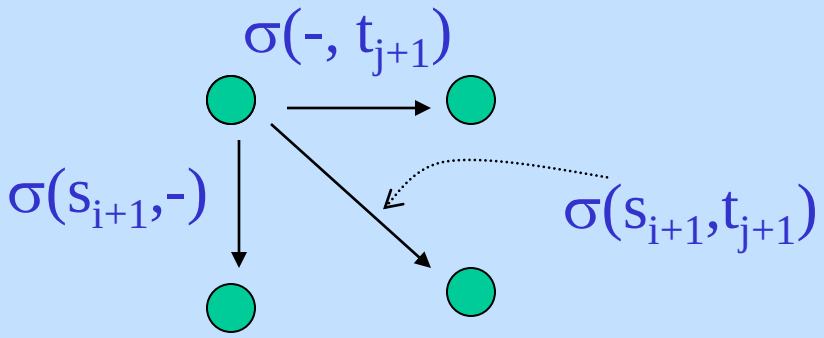
	λ	C	T	C	G	C	A	G	C
λ	0	-5	-10	-15	-20	-25	-30	-35	-40
C	-5	10	5	0	-5	-10	-15	-20	-25
A	-10	5	8	3	-2	-7	0	-5	-10
T	-15	0	15	10	5 *	0	-5	-2	-7
T	-20	-5	10 *	13	8	3	-2	-7	-4
C	-25	-10	5	20	15	18	13	8	3
A	-30	-15	0	15	18	13	28	23	18
C	-35	-20	-5	10	13	28	23	26	33

Traceback can yield both optimum alignments

Fernandez-Baca & Dobbs http://www.cs.iastate.edu/~cs544/

Alignment Graph





Alignment Graph

Definition: The **alignment graph** of sequences $S=s_1 \ldots s_n$ and $T=t_1 \ldots t_m$, is a directed graph $G=(V,E)$ on $(n+1)x(m+1)$ nodes, each labeled with a distinct pair (i,j) $(0 \le i \le n,\ 0 \le j \le m)$, with the following weighted edges:

- $((i,j), (i+1,j))$ with weight $\sigma(s_{i+1}, -)$
- $((i,j), (i,j+1))$ with weight $\sigma(-, t_{j+1})$
- $((i,j), (i+1,j+1))$ with weight $\sigma(s_{i+1}, t_{j+1})$

Note: a path from node $(0,0)$ to node (n,m) corresponds to an alignment and its total weight is the alignment score.

Goal: find an optimal path from node $(0,0)$ to node (n,m)

Complexity

- Time: $O(mn)$ (proportional to $|E|$)
- Space to find opt alignment: $O(mn)$ (proportional to $|V|$)
- Space is often the bottleneck!
- Can we improve space complexity for finding opt alignment?

Warm-up questions

How can we find opt alignment value only in O(m+n) space?

How do we efficiently compute the opt alignment scores of S to each prefix $t_1 \ldots t_k$ of T?

How do we efficiently compute the opt alignment score of the sequence suffixes $s_{i+1} \ldots s_n$ and $t_{j+1} \ldots t_m$?

Reducing Space Complexity

$V^*(n-i,m-j)$ = opt alignment value of $s_{i+1}...s_n$ and $t_{j+1}....t_m$

Lemma: $$V(n,m) = \max_{0 \le k \le m} \left\{ V(\frac{n}{2}, k) + V^*(\frac{n}{2}, m-k) \right\}$$

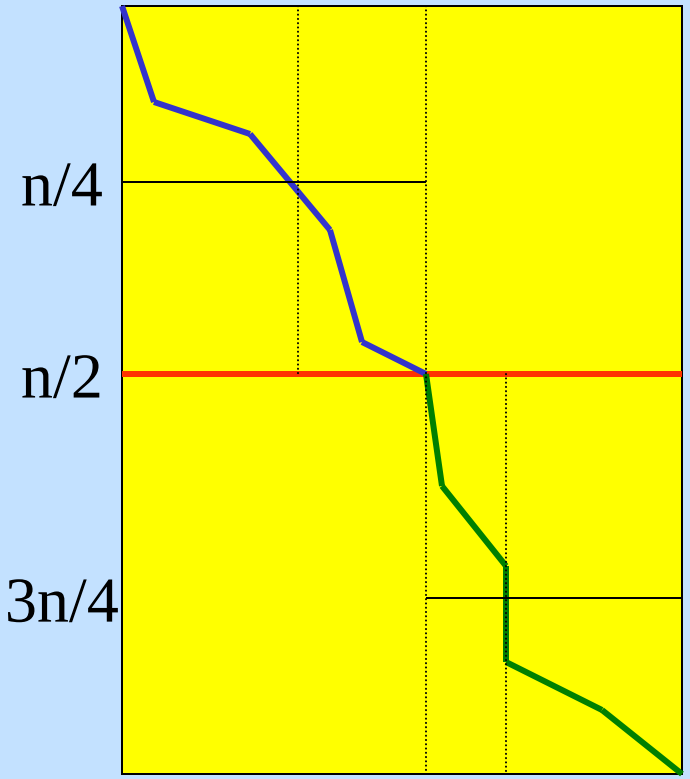
Pf: $V(n,m) \ge \max\{...\}$

- Every option of k defines a legal alignment:
- $\forall$ position k' in T, $\exists$ alignment of S and T consisting of:
  - an opt alignment of $s_1...s_{n/2}$ and $t_1...t_{k'}$ and
  - a disjoint opt alignment of $s_{n/2+1}...s_n$ and $t_{k'+1}...t_m$.

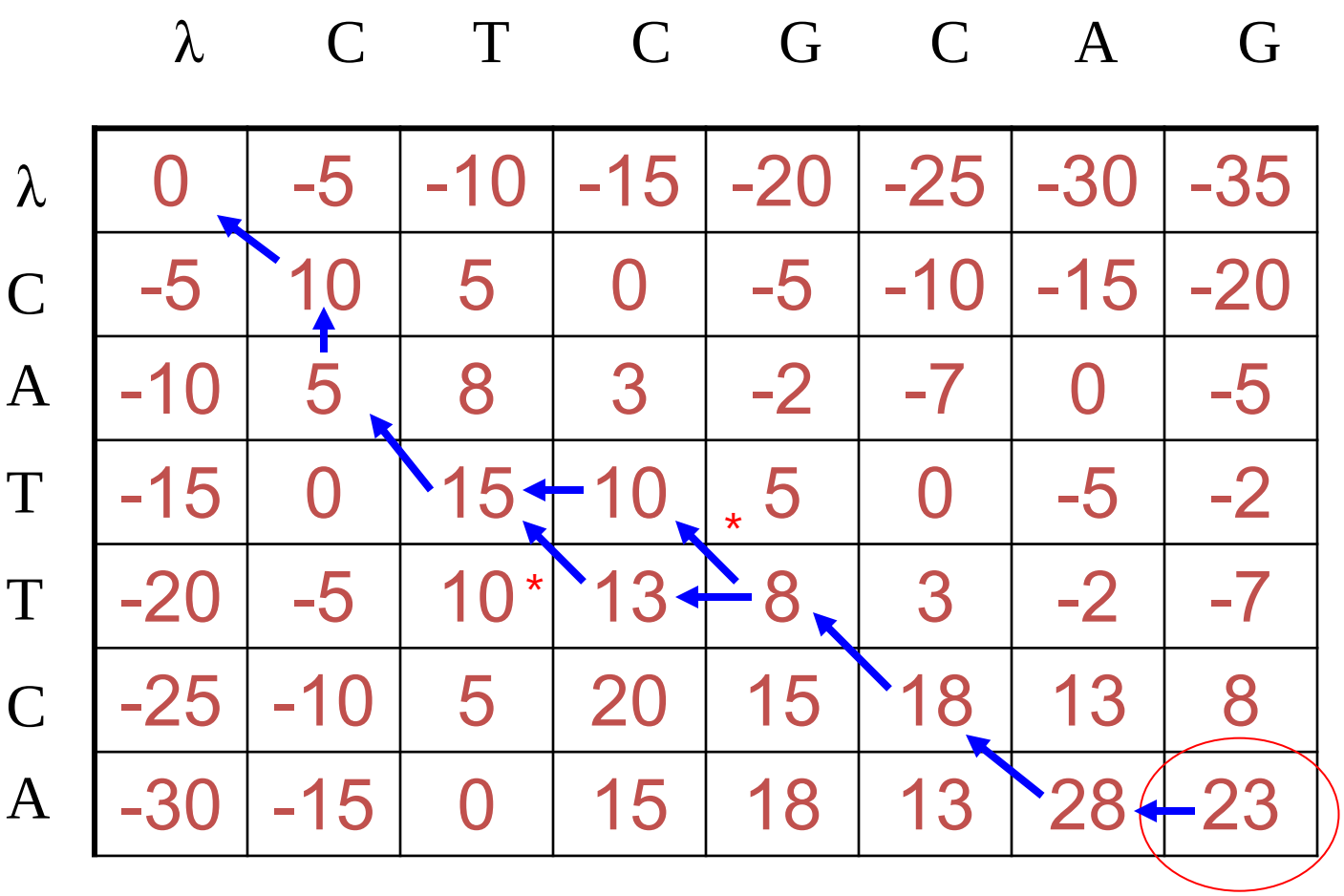
Proof (contd)

- $V(n,m) \leq max\{...\}$:
- Opt alignment corresponds to some k:
- For an opt. alignment of S and T, let k' be the rightmost position in T that is aligned with a character at or before position $n/2$ in S. Then the optimal alignment of S and T consists of:
  - an alignment of $s_1 \ldots s_{n/2}$ and $t_1 \ldots t_{k'}$ and
  - a disjoint alignment of $s_{n/2+1} \ldots s_n$ and $t_{k'+1} \ldots t_m$.

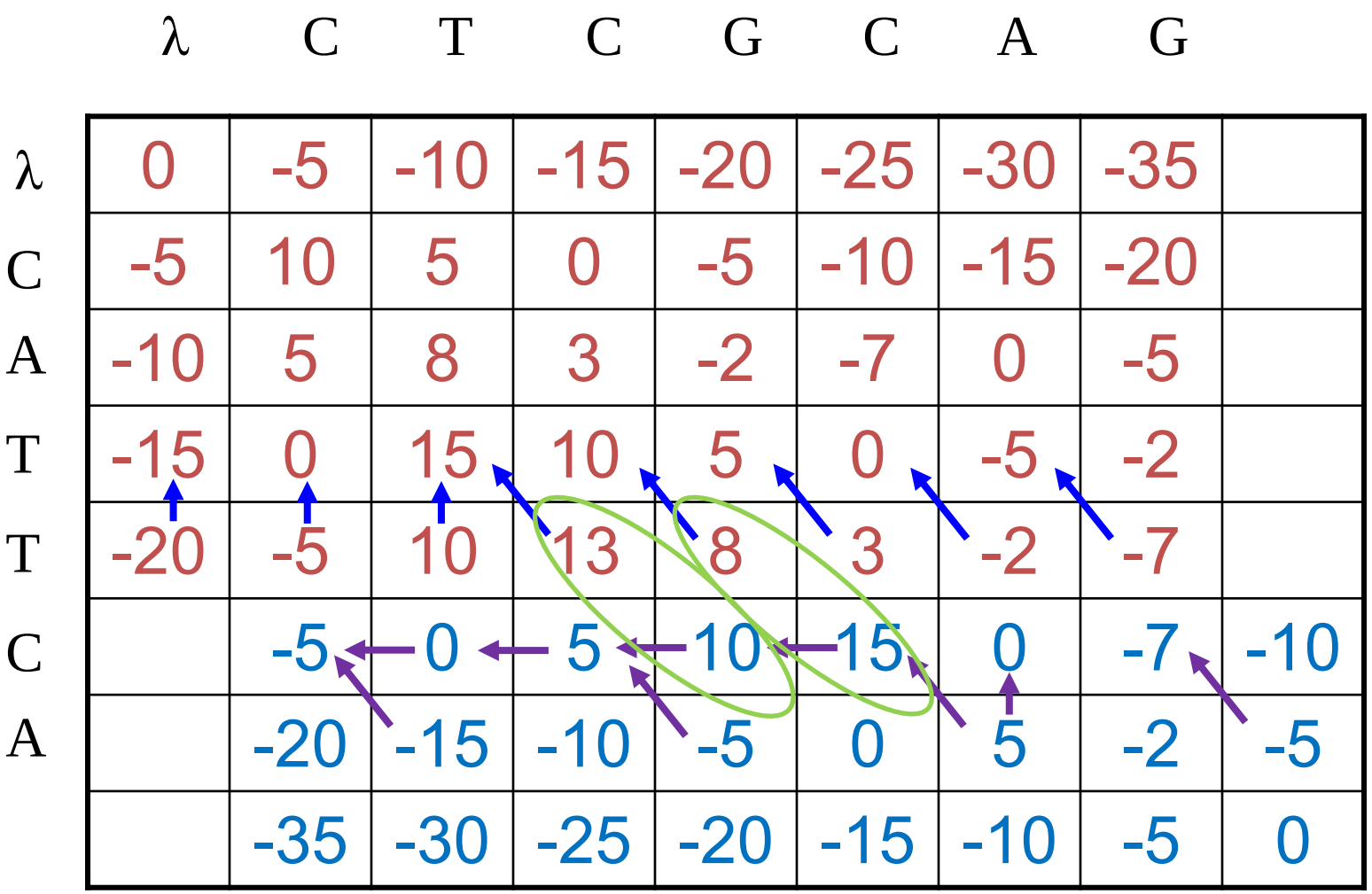
'Divide & Conquer' Alg (Hirschberg '75)



- Compute opt cost of all paths from start, to any point at centerline
- Compute opt cost of back paths from end to any pt at centerline
- Compute for every k V(n/2,k)+V*(n/2,m-k)
- Pick midpoint: k* with opt sum
- Continue recursively on the subproblems



Fernandez-Baca & Dobbs http://www.cs.iastate.edu/~cs544/



+10 for match, -2 for mismatch, -5 for space

Hirschberg Alg in more detail

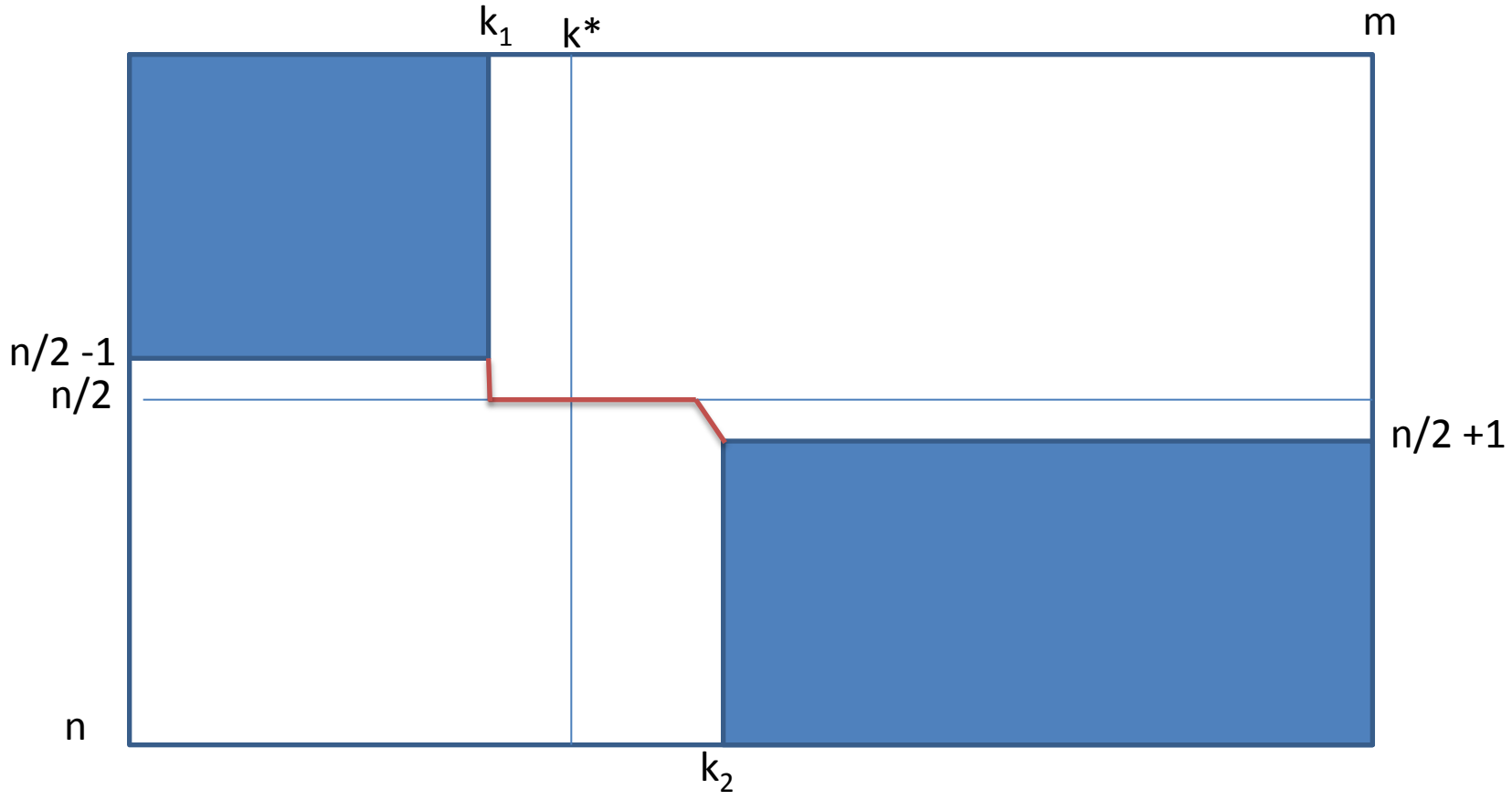
k* - position k maximizing
V(n/2,k)+V*(n/2,m-k)

Proved: ∃ opt path L through (n/2,k*)

Def: $L_{n/2}$ – subpath of L that

- starts with the last node in L in row n/2-1 and
- ends with the first node in L in row n/2+1

$L_{n/2}$



Lemma: k* can be found in O(mn) time and O(m) space. $L_{n/2}$ can be found and stored in same bounds

- Run DP up to row n/2, getting values V(n/2, i) for all i and back pointers for row n/2 — O(mn) time, O(m) space
- Run DP backwards up to row n/2, getting values V*(n/2, i) for all i and forward pointers for row n/2 — O(m) time, space
- Compute V(n/2,i)+V*(n/2,m-i) for each i, get maximizing index k* — O(m) time, space
- Use back pointers to compute subpath from (n/2,k*) to last node in row n/2-1 — O(m) time, space incl storage
- Use forward pointers to compute subpath from (n/2,k*) to first node in row n/2+1

Full Alg and Analysis

- Assume time to fill a p by q DP matrix : cpq
- ➔ time to compute rows V(n/2,.), V*(n/2,.): cmn
- ➔ time cmn, space O(m) to find k*, k_1 , k_2, $L_{n/2}$
- *Recursively* solve top subproblem of size ≤ nk*/2, bottom subproblem of size ≤ n(m-k*)/2
- Time for top level cmn, 2^{nd} level cmn/2
- Time for all i-th level computations $cmn/2^{i-1}$ (each subproblem has $n/2^i$ rows, the cols of all subprobs are disjoint)
- Total time: $\sum_{i=1 \text{ to } \log n} cmn/2^{i-1} \leq 2cmn$
- Total space: O(m+n)

Dan Hirschberg

Daniel S. Hirschberg is a full professor in Computer Science at University of California, Irvine. His research interests are in the theory of design and analysis of algorithms.

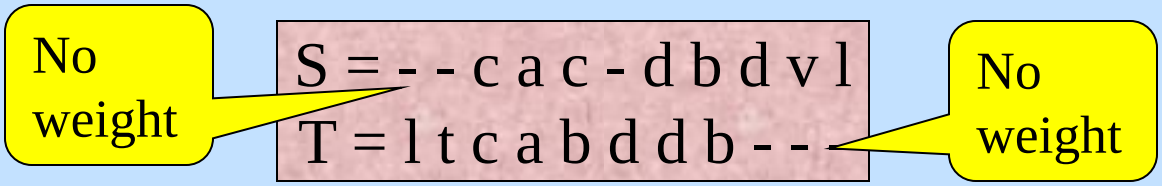
Hirschberg, D. S. (1975). "A linear space algorithm for computing maximal common subsequences". *Communications of the ACM* **18** (6): 341–343

End-Space Free Alignment

aka semi-global alignment

Input: Two sequences S, T

Goal: Find an optimal alignment between subsequences of S and T where at least one of these subsequences is a prefix of the original sequence and one (not necessarily the other) is a suffix.



- Motivation:
  - "shotgun sequence assembly" – match overlapping subsequences of an unknown target sequence.
  - Given a gene of one species, locate it in the reference sequence of another species

End-Space Free Alignment DP algorithm

- Base conditions:
  $V(i,0) = 0$
  $V(0,j) = 0$
- Recurrence relation:

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + \sigma(s_i,t_j) \\ V(i-1,j) + \sigma(s_i,-) \\ V(i,j-1) + \sigma(-,t_j) \end{cases}$$

- Search for i* and j* such that
  $V(n, i^*) = \max_i\{V(n, i)\}$
  $V(j^*, m) = \max_j\{V(j, m)\}$
- $V(S, T) = \max\{ V(n, i^*), V(j^*, m) \}$

- Time complexity: O(nm)
  - *computing the matrix: O(nm),*
  - *finding i* and j*: O(n+m).*
- Space complexity: for opt value: O(n+m)
  - *computing the matrix: O(n+m),*
  - *computing i* and j* requires the last row and column to be saved: O(n+m)*

Why compare sequences? (II)

Science

Science 15 July 1983:
Vol. 221 no. 4607 pp. 275-277
DOI: 10.1126/science.6304883

‹ Prev | Table of Contents | Next ›

REPORTS

Simian sarcoma virus onc gene, v-sis, is derived from the gene (or genes) encoding a platelet-derived growth factor

RF Doolittle, MW Hunkapiller, LE Hood, SG Devare, KC Robbins, SA Aaronson, HN Antoniades

ABSTRACT

The transforming protein of a primate sarcoma virus and a platelet-derived growth factor are derived from the same or closely related cellular genes. This conclusion is based on the demonstration of extensive sequence similarity between the transforming protein derived from the simian sarcoma virus onc gene, v-sis, and a human platelet-derived growth factor. The mechanism by which v-sis transforms cells could involve the constitutive expression of a protein with functions similar or identical to those of a factor active transiently during normal cell growth.

Local Alignment

Input: Sequences *S*, *T*

Goal: Find subsequences of *S* and β of *T*, with highest global alignment score between them.

i.e. find $Argmax\{GA(\alpha,\beta)\,|\alpha \sqsubseteq S, \beta \sqsubseteq T\}$

Motivation:

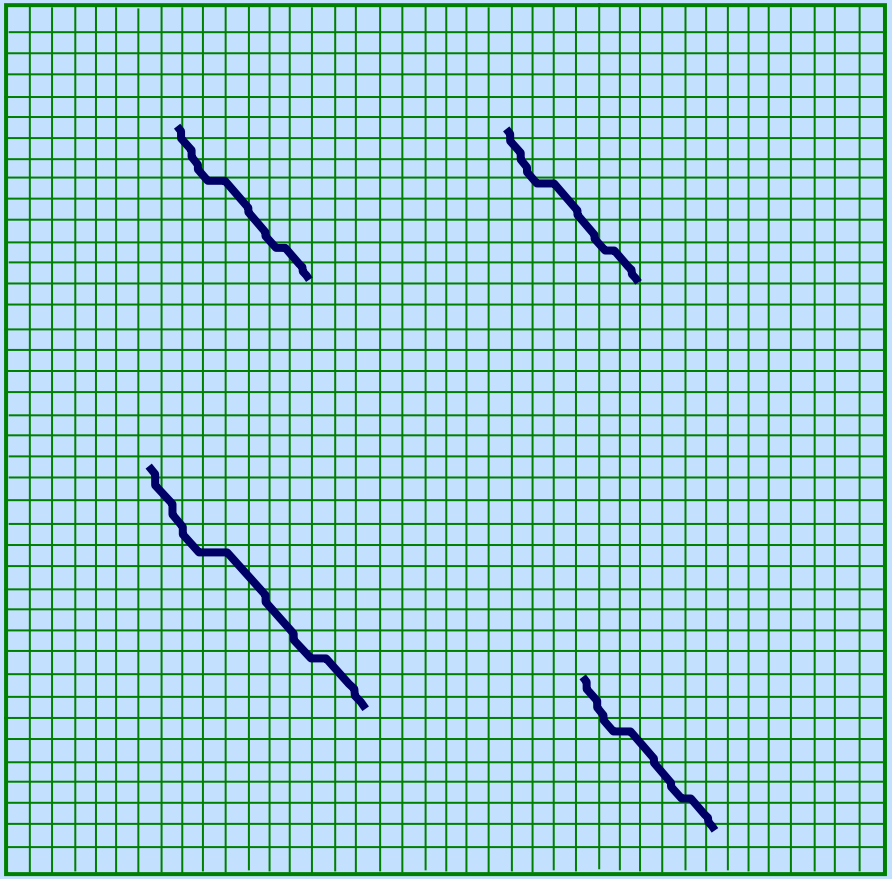
- Coding DNA segments matching the same gene
- Protein domains (functional subunits)

Example:

- *S=abcxdex*, *T=xxxcded*,
- Similarity score: *2* per match, *-1* for subs/indel,
- α*=cxde* and β*=c-de* have optimal alignment score.

	a	b	c	x	d	e	x
x	x	x	c	-	d	e	d

Local alignments in the alignment graph



Computing Local Alignment

The **local suffix alignment** problem for S', T': find a (possibly empty) suffix α of $S'=s_1 \ldots s_i$ and a (possibly empty) suffix β of $T'=t_1 \ldots t_j$ such that the value of their alignment is maximum over all values of alignments of suffixes of S' and T'.

- **V(i,j)** : the value of optimal local suffix alignment for the pair i, j of indices.
- How are the V(i,j) related to opt local alignment value?

Computing Local Alignment (2)

A scheme of the algorithm:

- Assumption: match ≥ 0, mismatch/indel ≤ 0
- Solve local suffix alignment for $S'=s_1 \ldots s_i$ and $T'=t_1 \ldots t_j$ by discarding prefixes whose similarity is ≤ 0
- Find the indices i*, j* after which the similarity only decreases.

Algorithm - Recursive Definition

Base Condition:

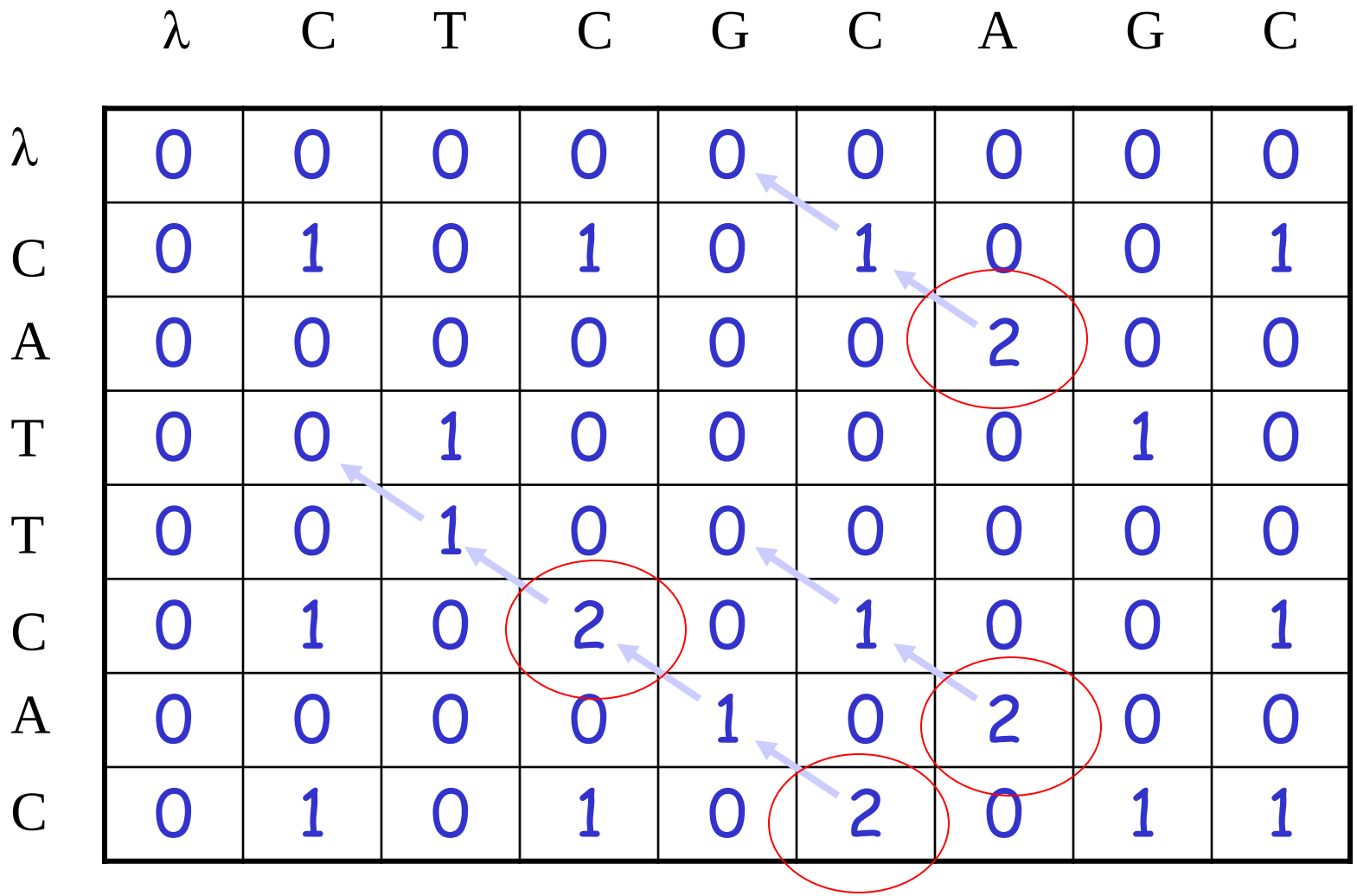
$\forall i,j \; V(i,0) = 0, \; V(0,j) = 0$

Recursion Step: $\forall \; i>0, \; j>0$

$$V(i,j) = \max \begin{cases} 0, \\ V(i-1, j-1) + \sigma(s_i, t_j), \\ V(i, j-1) + \sigma(-, t_j), \\ V(i-1, j) + \sigma(s_i, -) \end{cases}$$

Compute i*, j*

s.t. $V(i^*, j^*) = \max_{1 \leq i \leq n, \; 1 \leq j \leq m} V(i,j)$

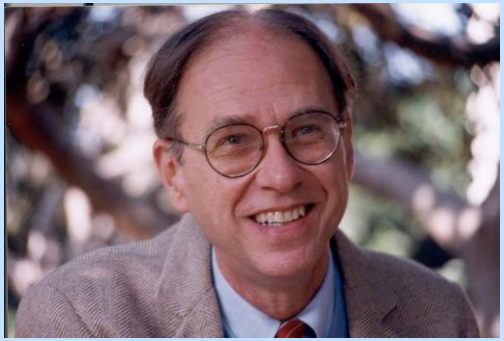


+1 for a match, -1 for a mismatch, -5 for a space

Fernandez-Baca & Dobbs http://www.cs.iastate.edu/~cs544/

Computing Local Alignment (3)

- Time O(nm)
- Space O(n+m)
  The optimum value and the ends of subsequences α and β can be found in linear space
- Finding the starting point of the two subsequences can be done in linear space (ex.)
- The actual alignment can be computed using Hirschberg's algorithm
- Smith-Waterman 81



Smith and Waterman

Gap Penalties

- Observation: spaces tend to occur in batches.
- Idea: when scoring an alignment, score contiguous spaces differently than independent spaces
- Definitions:
  - A *gap* is any maximal run of consecutive spaces in a single sequence of a given alignment.
  - The *length* of a gap is the number of spaces in it.
  - *#gaps:* No. of gaps in the alignment
- Example:

```
S= attc--ga-tggacc
T= a--cgtgatt---cc
```

  - 4 gaps, 8 spaces, 7 matches, 0 mismatches.

Gap Penalty Models

Constant Gap Penalty Model:

- Each individual space is free,
- Constant weight W_g for each gap, independent of its length (gap opening cost)

Goal: maximize $\Sigma\sigma(s'_i, t'_i) + W_g \times \#gaps$

Affine Gap Penalty Model:

- Additionally to W_g, each space has cost W_s. (gap extension cost)

Goal: max. $\Sigma\sigma(s'_i, t'_i) + W_g \times \#gaps + W_s \times \#spaces$

Alignment with Affine Gap Penalty

Three types of alignments:

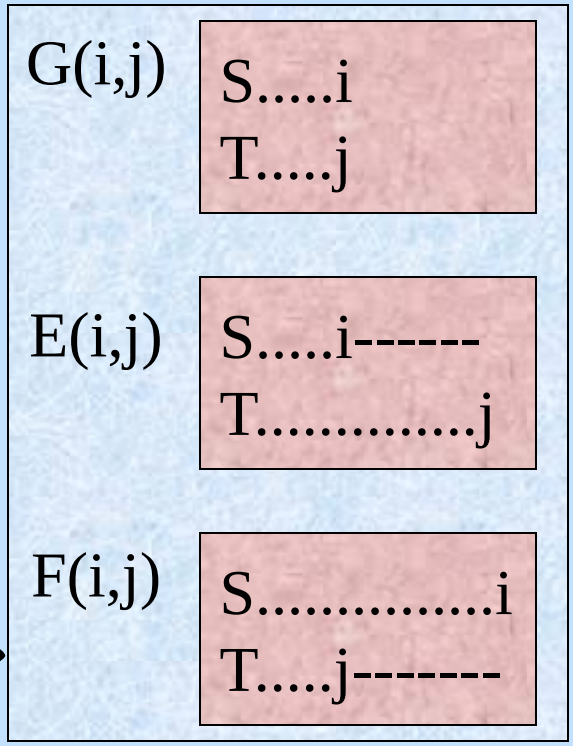
① S.....i
T.....j

② S.....i-------
T.....j

③ S.....i
T.....j-------

- G(i,j) is max value of any alignment of type 1, where s_i and t_j match
- E(i,j) is max value of any alignment of type 2, where t_j matches a space
- F(i,j) is max value of any alignment of type 3, where s_i matches a space

Alignment with Affine Gap Penalty (2)



Base Conditions:

$V(i, 0) = F(i, 0) = W_g + iW_s$

$V(0, j) = E(0, j) = W_g + jW_s$

Recursive Computation:

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

where:

- $G(i, j) = V(i-1, j-1) + \sigma(s_i, t_j)$
- $E(i, j) = \max\{ E(i, j-1) + W_s , G(i, j-1) + W_g + W_s , F(i, j-1) + W_g + W_s \}$
- $F(i, j) = \max\{ F(i-1, j) + W_s , G(i-1, j) + W_g + W_s , E(i-1, j) + W_g + W_s \}$

- Time complexity O(nm) - compute 4 matrices instead of one.
- Space complexity O(nm) - saving 4 matrices (trivial implementation).

Other Gap Penalty Models:

Convex Gap Penalty Model:

- Each additional space in a gap contributes less to the gap weight.
- Example: $W_g + \log(q)$, where q is the length of the gap.
- solvable in $O(nm \log m)$ time

Arbitrary Gap Penalty Model:

- Most general gap weight.
- Weight of a gap is an arbitrary function of its length $w(q)$.
- solvable in $O(nm^2+n^2m)$ time.