Semi-global and Local Alignment, and Gap Penalties

CMSC 423 · Fall 2026

Which parts of two strings should align — and what should a gap cost?



UNIVERSITY OF MARYLAND

Follows the treatments of Carl Kingsford and Ben Langmead (langmead-lab.org/teaching-materials). Figures redrawn.

From global alignment to its variants

Lecture 8

How far apart are two whole strings?

global alignment · edit distance

→

Today

Which parts should align? What does a gap cost?

semi-global · local · general & affine gaps

- Global alignment makes *every* character of both strings take part. Often only **part** of one string should: a read inside a genome, two reads that overlap, one shared domain in two proteins.
- Global alignment also charges a run of k gaps k times. One long gap is usually *one* event, so it should cost less than k separate ones.
- *Both fixes are small changes to the Lecture 8 table.*

From costs to scores: min becomes max

Lecture 8: cost – smaller is better

$$\mathrm{OPT}(i, j) = \min \begin{cases} \mathrm{cost}(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ \mathrm{gap} + \mathrm{OPT}(i-1, j) \\ \mathrm{gap} + \mathrm{OPT}(i, j-1) \end{cases}$$

Today: score – larger is better

$$\mathrm{OPT}(i, j) = \max \begin{cases} s(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ g + \mathrm{OPT}(i-1, j) \\ g + \mathrm{OPT}(i, j-1) \end{cases}$$

match **+10** mismatch **−5** gap **−7**

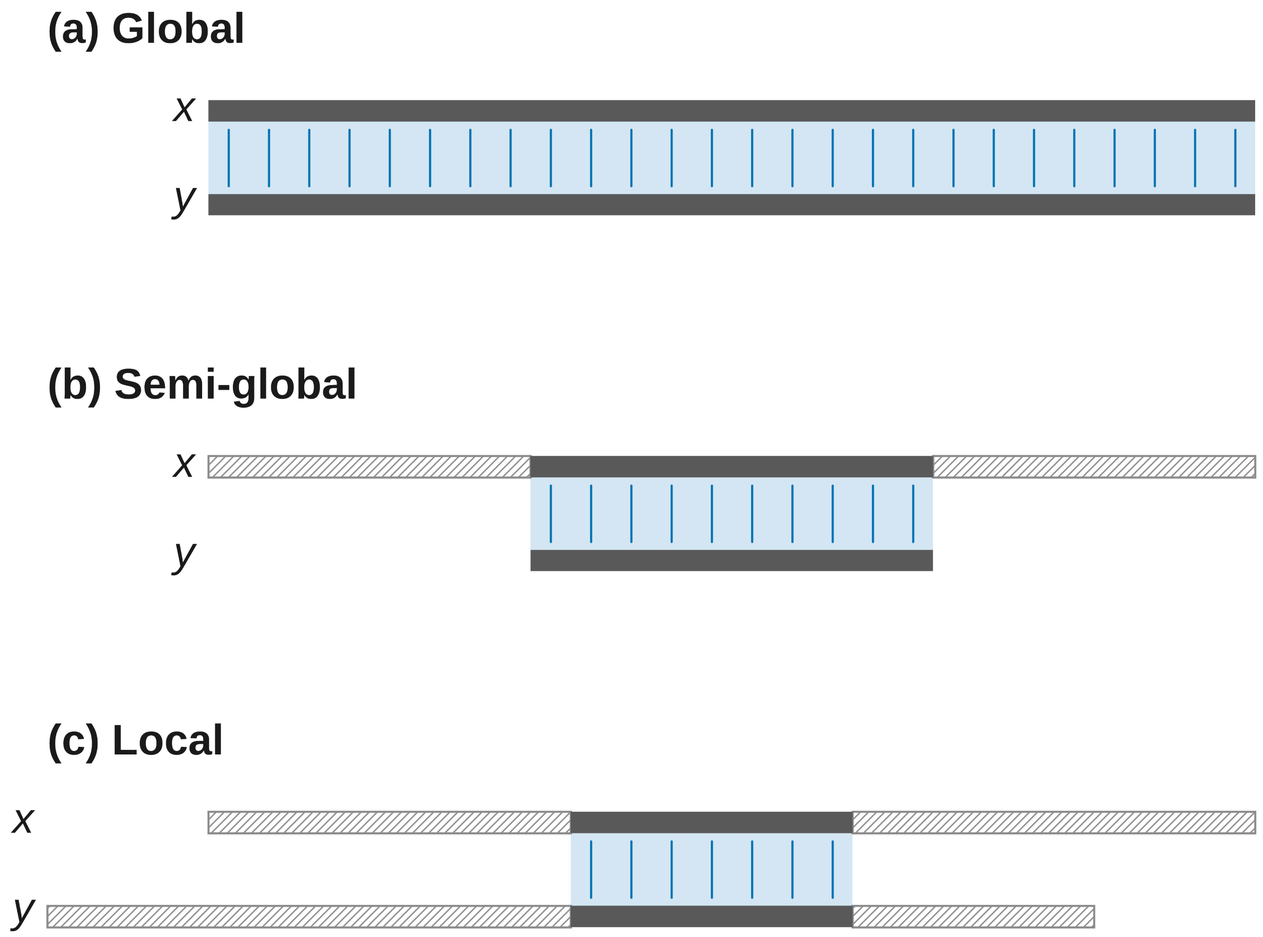
- Same table, same fill order, same traceback – only **min → max** and the signs change. A match earns points; mismatches and gaps lose them.
- We use the scores in the grey box for every linear-gap example today.
- Same geometry as Lecture 8: *x* (length m) runs along the bottom as column i, *y* (length n) up the side as row j; (0, 0) is bottom-left.

Three kinds

Which parts of x and y must take part?

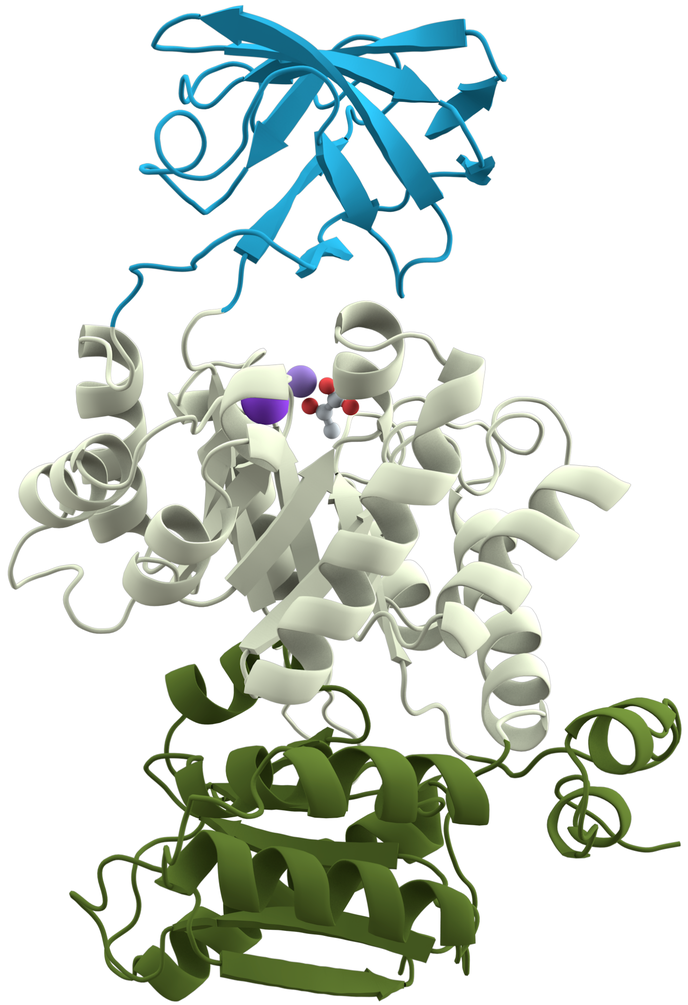
Three kinds – Semi-global – Local – General gaps – Affine gaps

Global, semi-global, local



- **Global**: every character of *x* and of *y* is in the alignment.
- **Semi-global**: gaps at the ends are free – e.g. all of *y* against a stretch of *x* (shown), or the end of one string against the start of the other.
- **Local**: some substring of *x* against some substring of *y* – the best-scoring pair.
- *Dark = aligned and scored; hatched = left out at no cost.*

Which one when?



- **Global** – two versions of the same gene or protein from closely related species: similar from end to end.
- **Semi-global** – a sequencing read against the stretch of genome it came from (*fitting*); two reads whose ends overlap (*overlap*, for assembly).
- **Local** – proteins from distant species. The whole chain need not be conserved, but a *domain* – a unit like one of the coloured lobes – often is.

Pyruvate kinase, three domains. Image: T. Shafee, Wikimedia Commons, CC BY 4.0

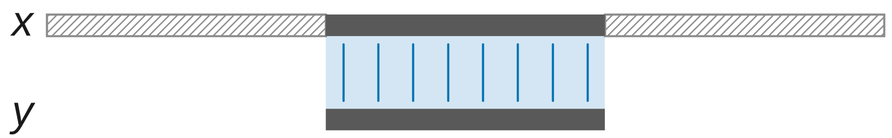
Semi-global

Free gaps at the ends

Three kinds – **Semi-global** – Local – General gaps – Affine gaps

Two semi-global shapes

(a) Fitting: y inside x



(b) Overlap: end of x, start of y



- **Fitting** (also called *cost-free ends*): align all of the short *y* to the best-matching window of *x*. Read mapping, once an index (Lectures 5–7) has found the neighbourhood, is a fitting problem.
- **Overlap**: a suffix of *x* against a prefix of *y* — how assemblers decide two reads came from overlapping places.
- *We work through fitting first; overlap changes the same two things.*

Recall: the global table

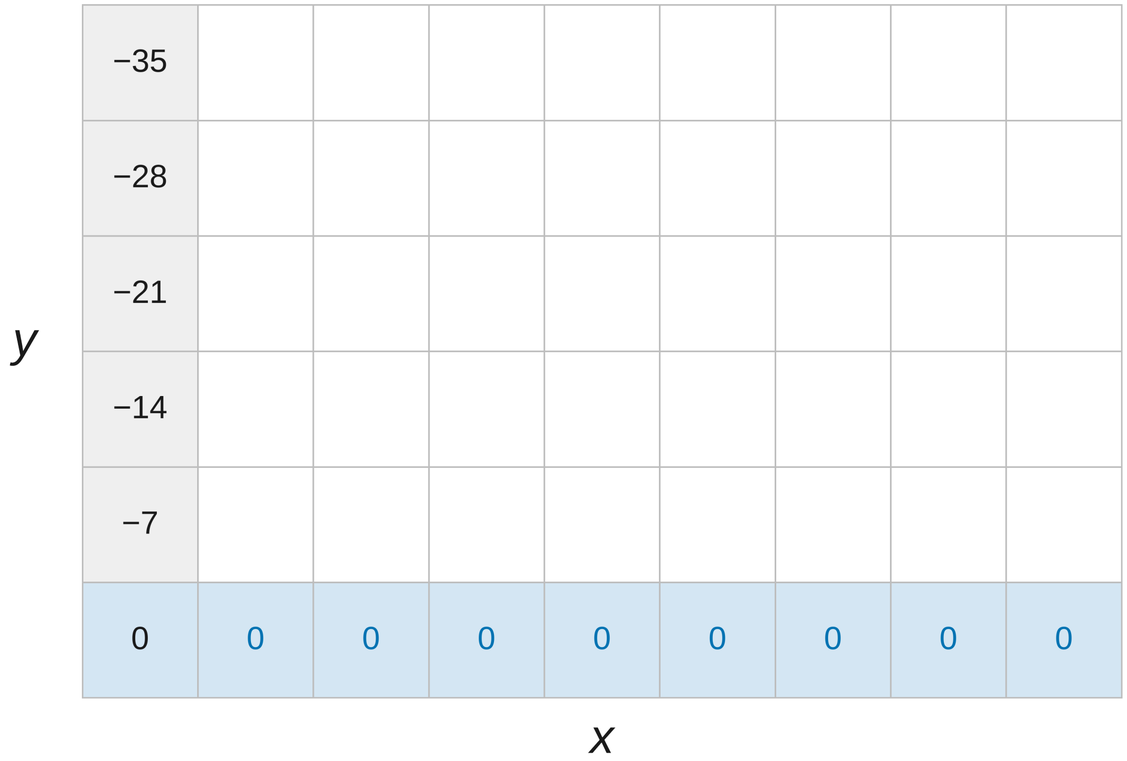
−35								m, n
−28								
−21								
−14								
−7								
0	−7	−14	−21	−28	−35	−42	−49	−56

y

x

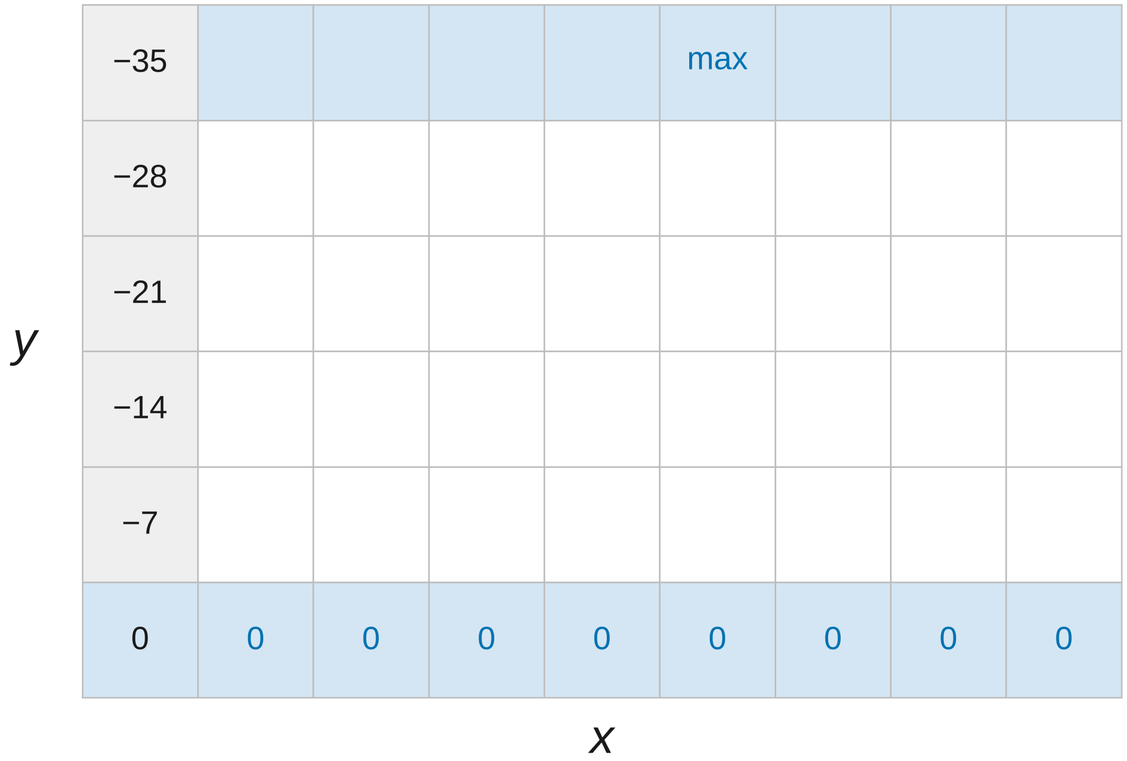
- **OPT(i, j)** = best score of $x_1 \ldots x_i$ against $y_1 \ldots y_j$.
- Base cases charge for gaps: OPT(i, 0) = i · g and OPT(0, j) = j · g (here g = −7).
- The answer is the top-right corner, OPT(m, n): all of *x* against all of *y*.
- Semi-global alignment keeps the recurrence and changes only these two things: the **base cases** and **where the answer is read**.

Change 1: y may start anywhere in x (switch 3)



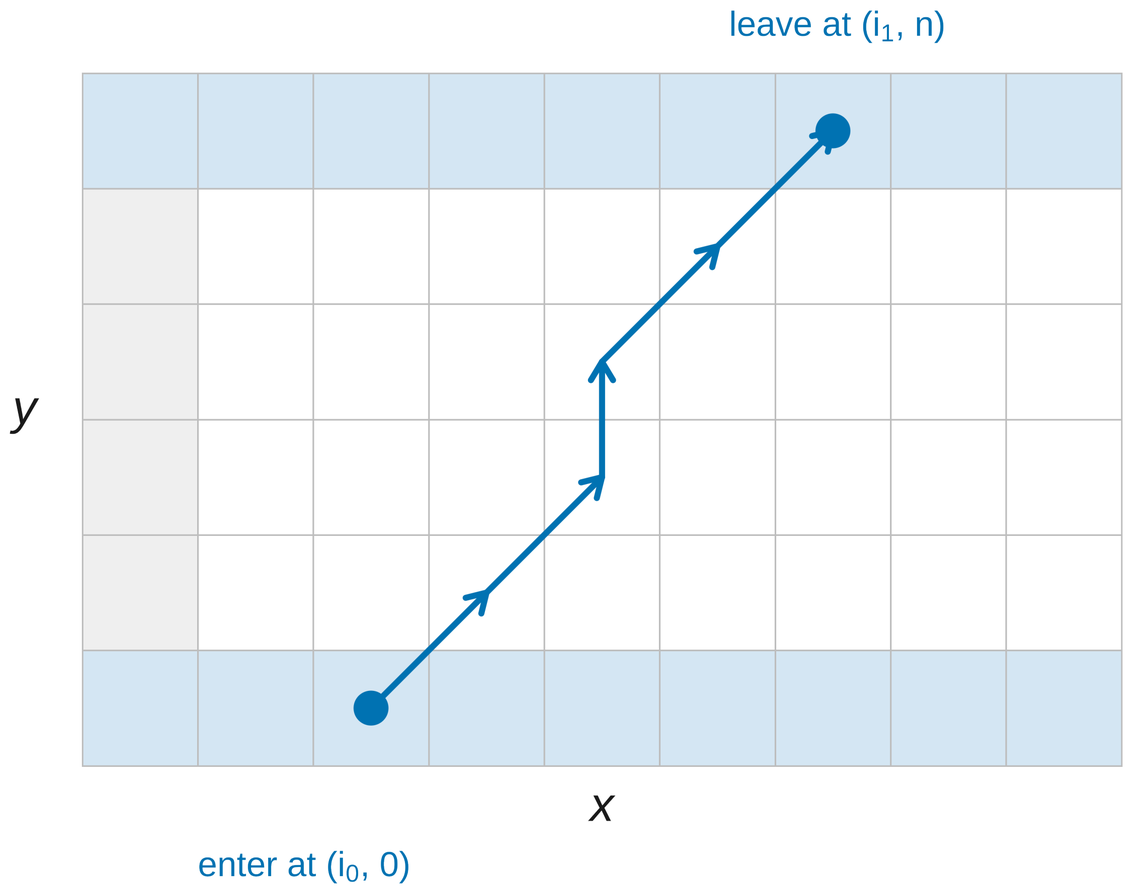
- **OPT(i, 0) = 0** for every i: leaving $x_1 \ldots x_i$ unaligned before *y* starts costs nothing.
- Column 0 is unchanged, OPT(0, j) = j · g: every character of *y* must still be aligned, and a gap opposite one still costs.
- A path may now enter the table anywhere along the bottom row.

Change 2: y may end anywhere in x (switch 4)



- Every cell (i, n) on the top row holds a finished alignment of **all of y** with $x_1 \ldots x_i$.
- Leaving $x_{i+1} \ldots x_m$ unaligned afterwards is free, so the answer is
- **max**$_{0 \le i \le m}$ **OPT(i, n)** – and the traceback starts there.

Why these two changes are exactly right



- A path from (i_0, 0) to (i_1, n) is a **global** alignment of *y* with the substring $x_{i_0+1} \ldots x_{i_1}$.
- Free bottom row = it may start at any i_0. Max over the top row = it may stop at any i_1.
- So one O(mn) table finds the best global alignment of *y* against *every* substring of *x* – not O(m²) separate tables.

Example: fitting ATCTA into GGATCGTACT

		–	G	G	A	T	C	G	T	A	C	T
		0	1	2	3	4	5	6	7	8	9	10
A	5	−35	−33	−31	−14	−1	16	18	26	**43**	36	29
T	4	−28	−26	−24	−11	6	23	25	**33**	26	19	23
C	3	−21	−19	−17	−4	13	**30**	**23**	16	9	13	6
T	2	−14	−12	−10	3	**20**	13	6	5	3	5	13
A	1	−7	−5	−5	**10**	3	−4	−5	−5	10	3	−4
–	0	0	0	**0**	0	0	0	0	0	0	0	0

```
G G A T C G T A C T
    | | |   | |
– – A T C – T A – –
  10 10 10 −7 10 10
```

- Bottom row all 0; best value on the top row: **OPT(8, 5) = 43**.
- Trace back until the path reaches row 0, at i = 2. The x characters outside the path face free gaps (light).
- 5 matches, 1 gap: 5 × 10 − 7 = 43.

Same pair, aligned globally

(a) Global: score 15

```
G G A T C G T A C T
    |   |   |     |   |
- - A T C - T A - -
−7 −7 10 10 10 −7 10 10 −7 −7
```

(b) Fitting: score 43

```
G G A T C G T A C T
    |   |   |     |   |
- - A T C - T A - -
   10 10 10 −7 10 10
```

- The same core, `ATCGTA / ATC–TA` – but global alignment must also pay for the four end gaps: 4 × 7 = **28** points (43 – 28 = 15).
- Those gaps say nothing about similarity: *y* was always going to be shorter.

Global vs fitting: what changes

	global	**fitting (y into x)**
recurrence	max of three	same
bottom row OPT(i, 0)	i · g	**0**
left column OPT(0, j)	j · g	j · g (all of y must align)
answer	OPT(m, n)	**$\max_i$ OPT(i, n)**
traceback stops at	(0, 0)	the bottom row
time / space	O(mn)	O(mn)

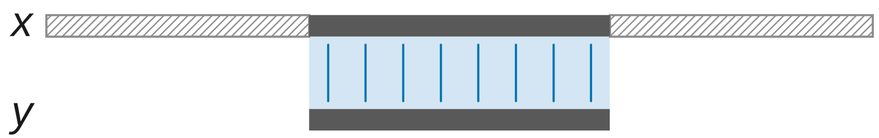
Four independent switches

	what becomes free	change
1	gaps before x (y may start first)	OPT(0, j) = 0
2	gaps after x (y may end last)	answer: max over column m
3	gaps before y (x may start first)	OPT(i, 0) = 0
4	gaps after y (x may end last)	answer: max over row n

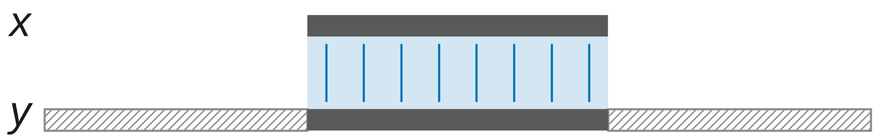
- Switches 1 and 3 change a **base case**; switches 2 and 4 change **where the answer is read**. With both 2 and 4, take the max over the top row and the right column.
- Fitting *y* into *x* is switches 3 + 4. Global is none; local is a different change altogether.

Four kinds of semi-global alignment

(a) y inside x — switches 3 + 4



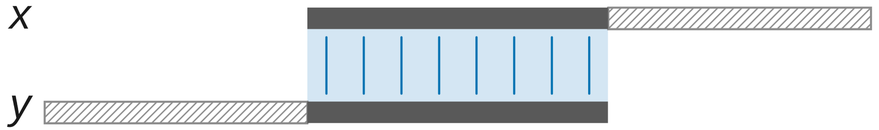
(b) x inside y — switches 1 + 2



(c) end of x, start of y — 2 + 3



(d) end of y, start of x — 1 + 4



Hatched = the end gaps that are free. Which switches you flip depends only on which ends may hang over.

Example: an overlap (switches 2 + 3)

		–	G	T	C	A	G	T	T	A	C	G	A
		0	1	2	3	4	5	6	7	8	9	10	11
C	9	−63	−46	−44	−27	−12	5	7	9	2	19	12	7
C	8	−56	−39	−37	−20	−5	12	14	7	9	17	12	14
G	7	−49	−32	−30	−15	2	19	12	14	7	2	19	21
T	6	−42	−40	−23	−8	9	11	19	12	5	9	11	28
A	5	−35	−33	−18	−1	16	9	2	−2	6	16	18	**35**
C	4	−28	−26	−11	6	8	1	3	−4	6	23	**25**	18
C	3	−21	−19	−4	13	6	8	1	−2	13	**30**	23	16
A	2	−14	−12	3	5	13	6	3	5	**20**	13	6	5
T	1	−7	−5	10	3	−4	−5	10	**10**	3	−4	−5	−5
–	0	0	0	0	0	0	0	**0**	0	0	0	0	0

```
G T C A G T T A C G A – – – –
          | | | × |
– – – – – – T A C C A T G C C
          10 10 10 −5 10
```

- Bottom row 0 (switch 3); answer = max over the right column (switch 2): **OPT(11, 5) = 35**.
- The last five characters of *x* overlap the first five of *y* with one mismatch: 4 × 10 − 5 = 35.
- *Aligned globally, the same pair scores only −2.*

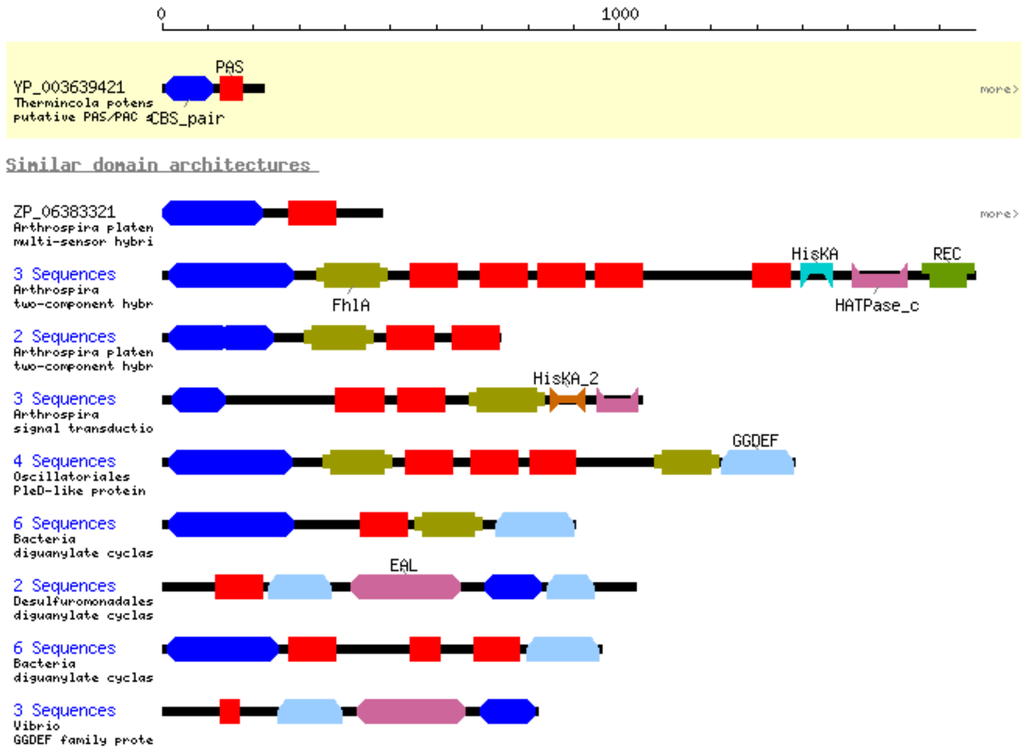
Local

The best-matching pair of substrings

Three kinds – Semi-global – **Local** – General gaps – Affine gaps

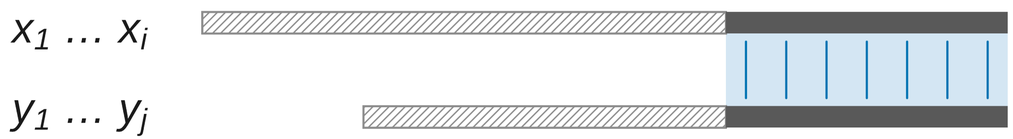
Local alignment

- **Local alignment** of *x* and *y*: the best alignment of a **substring** *x′* of *x* with a substring *y′* of *y*.
- Substring = contiguous. (Over *subsequences* you could skip any letters for free – a different problem.)
- The empty alignment is allowed and scores 0, so the answer is never negative.
- **Why**: proteins are built from *domains* – units that fold and work on their own and are reused in new combinations. Related proteins can share one domain and nothing else.



Domain architectures of related proteins (NCBI CDD).
Shared blocks, new arrangements: only local alignment can match them.

A new meaning for each cell



ends exactly at i and j

- **OPT(i, j)** = best score of an alignment of **a suffix** of $x_1 \ldots x_i$ with **a suffix** of $y_1 \ldots y_j$ (either may be empty).
- Every local alignment ends somewhere — at some (i, j) — so the answer is the **largest** OPT(i, j) anywhere in the table.
- Row 0 and column 0 are 0: the same base-case trick as semi-global, now on both strings.
- **What else has to change?** Hint: the empty alignment is always a candidate.

The local recurrence

$$\mathrm{OPT}(i, j) = \max \begin{cases} s(x_i, y_j) + \mathrm{OPT}(i-1, j-1) & x_i \text{ over } y_j \\ g + \mathrm{OPT}(i-1, j) & x_i \text{ over a gap} \\ g + \mathrm{OPT}(i, j-1) & y_j \text{ over a gap} \\ \mathbf{0} & \text{start over: the empty alignment} \end{cases}$$

- Without the 0, a bad stretch drags every later cell down. With it, a cell that would go negative **resets**: the best suffix alignment ending here is the empty one.
- Base cases: $\mathrm{OPT}(i, 0) = \mathrm{OPT}(0, j) = 0$.

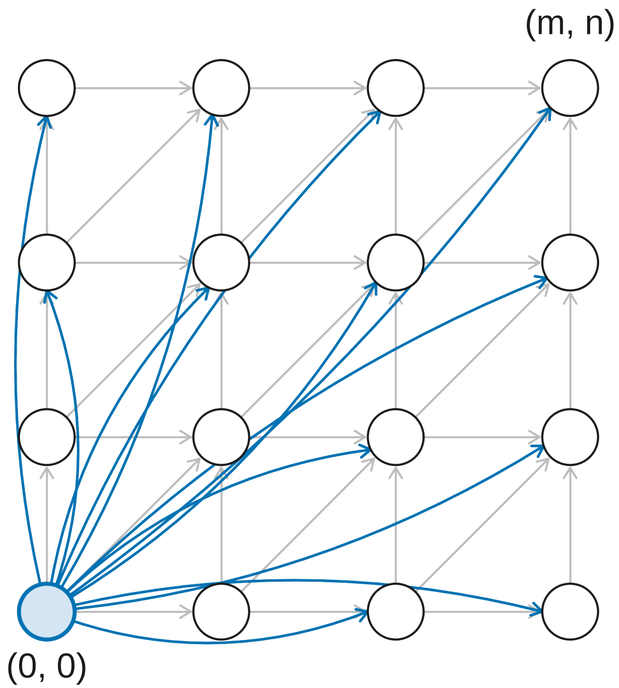
The local algorithm (Smith–Waterman)

```
row 0 and column 0 start at 0
for i = 1 … m:
    for j = 1 … n:
        A[i][j] = max( A[i–1][j–1] + s(xᵢ, yⱼ),
                       A[i–1][j] + g,  A[i][j–1] + g,
                       0 )
best = largest A[i][j] anywhere
```

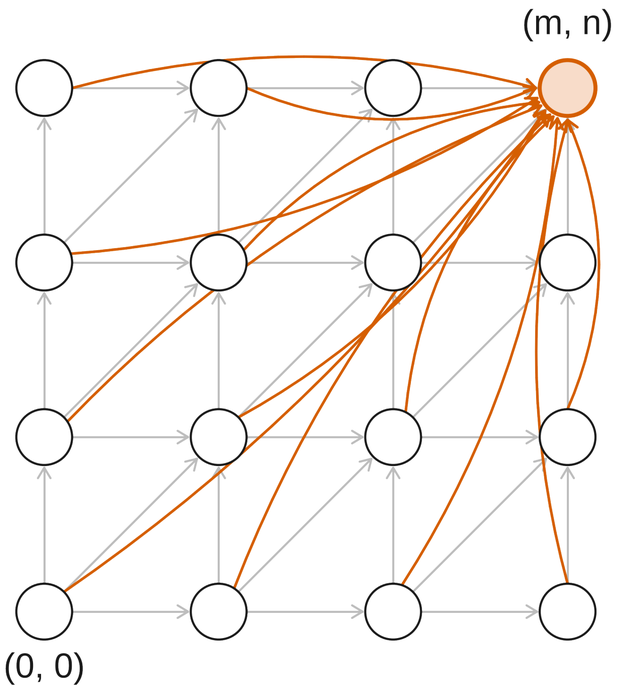
- Answer: the **largest entry** in the table, not the corner.
- Alignment: trace back from that entry as usual, and **stop at the first 0** – that is where the local alignment starts.
- O(mn) time and space, exactly as before.

Local alignment as a path in the edit graph

(a) Start anywhere



(b) Stop anywhere



- Global alignment = best path (0, 0) → (m, n). Add score-0 edges **from (0, 0) to every node** (start anywhere) and **from every node to (m, n)** (stop anywhere).
- The best path in this graph is the best local alignment – the 0 case is the edge from the source, the max over the table is the edge to the sink.

Example 1: AGCGTAG vs CTCGTC

		–	A	G	C	G	T	A	G
		0	1	2	3	4	5	6	7
C	6	0	0	0	13	6	23	25	18
T	5	0	0	3	5	13	**30**	23	16
G	4	0	0	10	3	**20**	13	6	18
C	3	0	0	0	**10**	3	6	8	1
T	2	0	0	**0**	3	5	13	6	0
C	1	0	0	0	10	3	0	0	0
–	0	0	0	0	0	0	0	0	0

```
A G C G T A G
    | | |
C T C G T C
  10 10 10
```

- Scores +10 / −5 / −7. Negative values never appear: they reset to 0.
- Largest entry: **30** at (5, 5). Trace back to the 0 at (2, 2).
- Local alignment `CGT` / `CGT`: three matches, 30.

Example 2: bestoftimes vs soften

		–	b	e	s	t	o	f	t	i	m	e	s
		0	1	2	3	4	5	6	7	8	9	10	11
n	6	0	0	3	5	0	0	2	19	21	23	22	24
e	5	0	0	10	3	3	5	9	26	28	21	29	22
t	4	0	0	0	0	10	3	16	**33**	26	19	12	5
f	3	0	0	0	0	0	6	**23**	16	9	2	0	0
o	2	0	0	0	3	5	**13**	6	0	0	0	0	3
s	1	0	0	0	**10**	**3**	0	0	0	0	0	0	10
–	0	0	0	**0**	0	0	0	0	0	0	0	0	0

```
b e s t o f t i m e s
    |   | | |
  s – o f t e n
10 –7 10 10 10
```

- Largest entry **33** at (7, 4); the traceback reaches 0 at (2, 0).
- `stoft / s–oft`: four matches and one gap, 4 × 10 – 7 = 33.
- *A gap inside a local alignment is fine, as long as what it connects pays for it.*

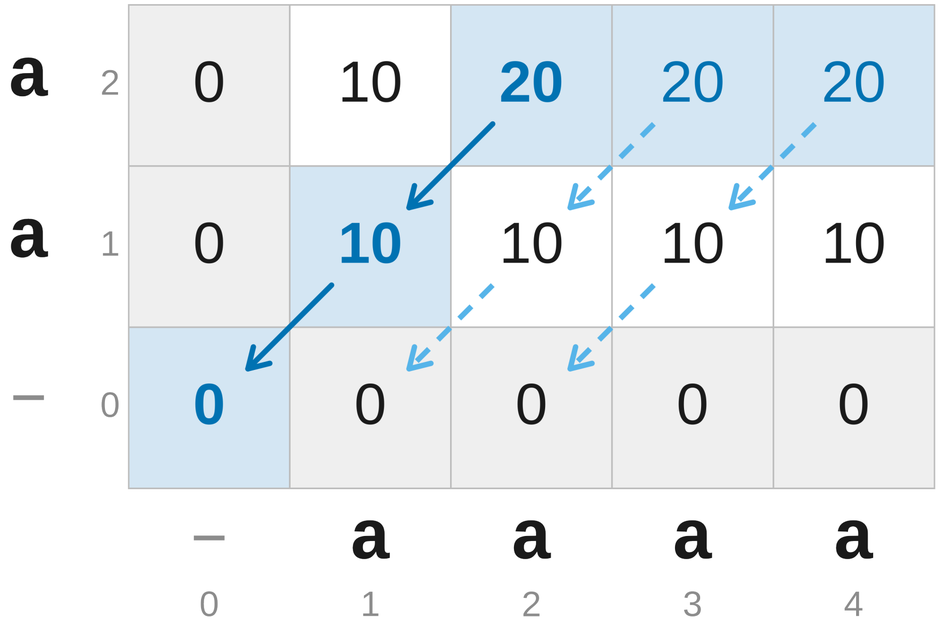
Example 3: mississippi vs issp

		–	m	i	s	s	i	s	s	i	p	p	i
		0	1	2	3	4	5	6	7	8	9	10	11
p	4	0	0	0	0	6	23	25	18	23	25	**33**	26	19
s	3	0	0	0	0	13	30	23	**30**	**23**	16	9	2
s	2	0	0	0	3	20	13	6	**20**	13	6	5	0	3
i	1	0	0	10	3	0	**10**	3	0	10	3	0	10
–	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0

```
m i s s i s s i p p i
      | | |   |
      i s s – p
      10 10 10 −7 10
```

- Best: **33** at (9, 4): `issip / iss–p`: 40 – 7 = 33 beats 30.
- The exact match `iss` alone scores only 30; paying the gap to reach the `p` adds 3.

Ties: aaaa vs aa



- The largest value, **20**, appears **three times** – at (2, 2), (3, 2) and (4, 2).
- Each is a different optimal local alignment: aa against positions 1–2, 2–3 or 3–4 of aaaa.
- Repeats make ties common. A tool must choose – report one, report all, or report every non-overlapping hit.

One pair, three answers

(a) Global: 10

```
b e s t o f t i m e s
    |   | | |     | ×
– – s – o f t – – e n
−7 −7 10 −7 10 10 10 −7 −7 10 −5
```

(b) Fitting soften into bestoftimes: 24

```
b e s t o f t i m e s
    |   | | |     | ×
– – s – o f t – – e n
   10 −7 10 10 10 −7 −7 10 −5
```

(c) Local: 33

```
b e s t o f t i m e s
    |   | | |
    s – o f t e n
   10 −7 10 10 10
```

- Global pays for `––` before `s`; fitting drops that, but must still align all of `soften`.
- Local keeps only the part that pays: `stoft / s–oft`, and ignores `en`.
- 10 ≤ 24 ≤ 33: each variant can do anything the one before it could.

Names you will see

- **Needleman–Wunsch** (1970): global alignment by dynamic programming.
- **Smith–Waterman** (1981): local alignment — today's 0 case.
- **Levenshtein / edit distance**: the unit-cost, *minimising* version from Lecture 8. A similarity score is a different number: it is maximised, under any scores.
- **BLAST**: finds short high-scoring word hits (exact words for DNA), then extends them with an X-drop local DP. Fast, but a heuristic — it can miss what full Smith–Waterman finds.

General gaps

A run of k gaps is one event, not k

Three kinds – Semi-global – Local – **General gaps** – Affine gaps

Gaps come in runs

(a) Four separate gaps: 32

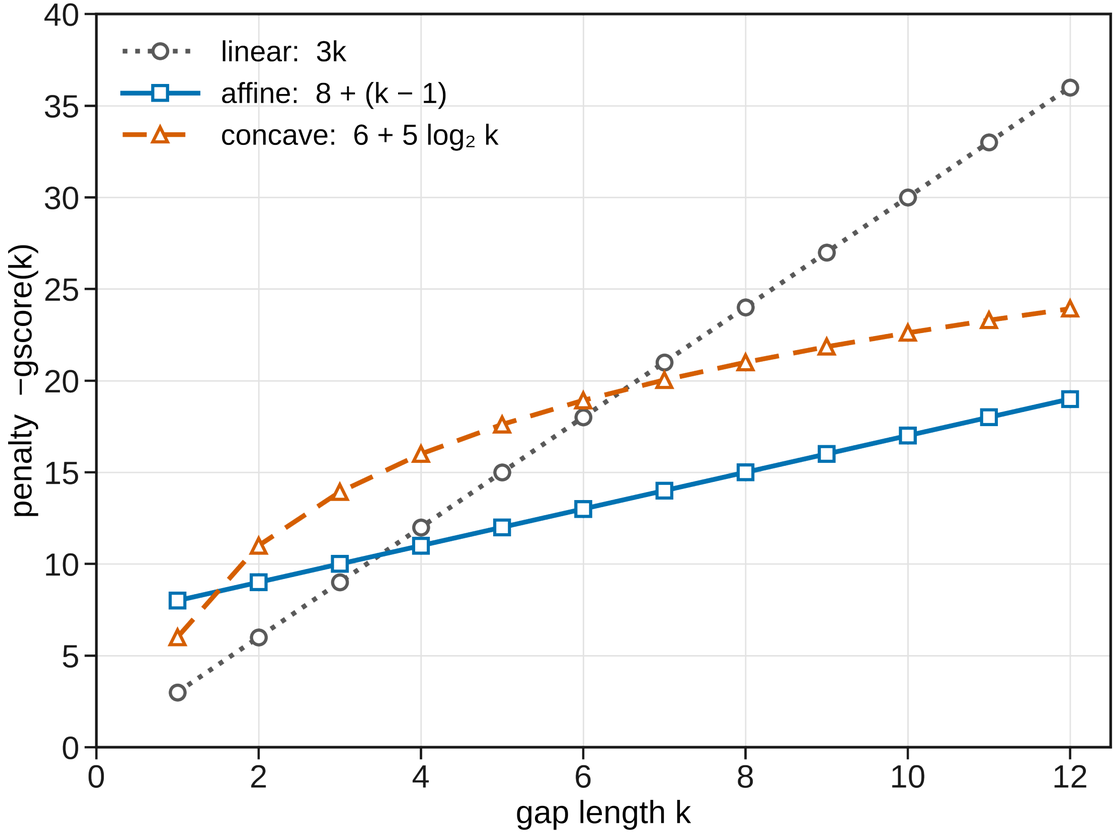
```
A A A G A A T T C A
|   |   |   |   | |
A - A - A - T - C A
10 −7 10 −7 10 −7 10 −7 10 10
```

(b) One run of four: 32

```
A A A G A A T T C A
| | |       | | |
A A A - - - - T C A
10 10 10 −7 −7 −7 −7 10 10 10
```

- With a fixed score per gap character both score **32** – and so do **18** other alignments: a 20-way tie.
- But (b) needs *one* event – deleting `GAAT` from *x* (or inserting it into *y*) – where (a) needs four.
- So score a run of k gaps with a function **gscore(k)** that grows *more slowly* than k · g.

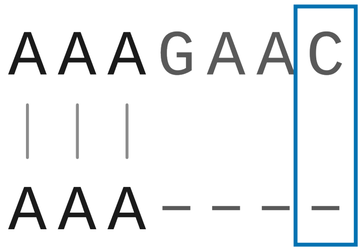
Gap penalty functions



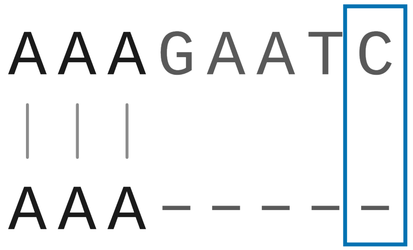
- **General**: gscore(k) may be any function of k. Here we plot the *penalty*, −gscore(k).
- **Affine**: a price to open the gap, then a smaller price per extra character.
- **Concave**: each extra character costs less than the one before. (Some texts, e.g. Gusfield, call these *convex* gap weights.)
- *Linear is the special case we already know how to solve.*

Why the Lecture 8 recurrence breaks

(a) the last x character ends a run of 4



(b) …or a run of 5



- Both last columns are "x_i over a gap". Under linear gaps each costs g – but now it costs gscore(4) − gscore(3) in (a) and gscore(5) − gscore(4) in (b).
- To score the last column we must know **how long the run is that it ends**. OPT(i, j) is a single number; it doesn't say.
- *Worse, the best alignment of a prefix may end in a match while the best alignment of the longer prefix needs one that ends in a gap.*

Three tables, one per kind of last column

M(i, j)

$x_1 \dots x_i$ so far	x_i
$y_1 \dots y_j$ so far	y_j

last column is a (mis)match

X(i, j)

$x_1 \dots x_i$ so far	–
$y_1 \dots y_j$ so far	y_j

last column is a gap in x

Y(i, j)

$x_1 \dots x_i$ so far	x_i
$y_1 \dots y_j$ so far	–

last column is a gap in y

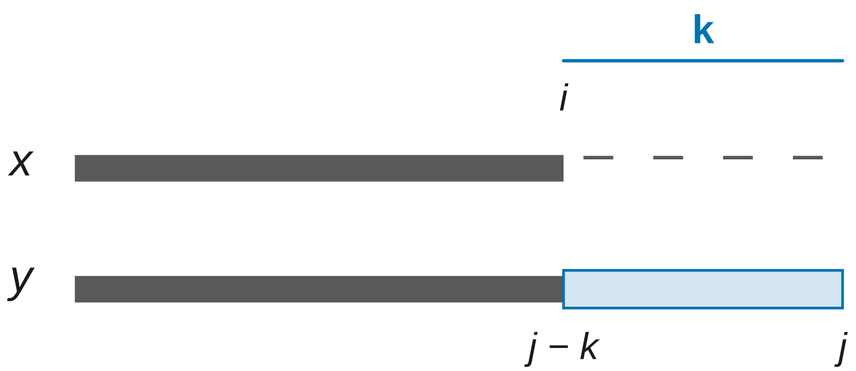
- Each table holds the best score of $x_1 \dots x_i$ against $y_1 \dots y_j$ **among alignments that end that way**.
- The answer is max{ M(m, n), X(m, n), Y(m, n) }.

M: the last column is a match or mismatch

$$M(i, j) = s(x_i, y_j) + \max \begin{cases} M(i-1, j-1) \\ X(i-1, j-1) \\ Y(i-1, j-1) \end{cases}$$

- By definition the last column is x_i over y_j: add its score.
- Before it, the alignment of $x_1 \ldots x_{i-1}$ and $y_1 \ldots y_{j-1}$ may end in **any** way, so take the best of the three tables.

X and Y: choose the length of the last run



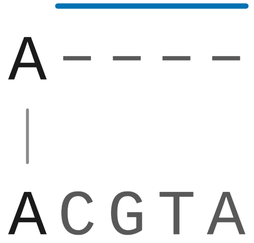
$$X(i,j) = \max \begin{cases} M(i, j-k) + \text{gscore}(k) \\ Y(i, j-k) + \text{gscore}(k) \end{cases} \qquad 1 \le k \le j$$

$$Y(i,j) = \max \begin{cases} M(i-k, j) + \text{gscore}(k) \\ X(i-k, j) + \text{gscore}(k) \end{cases} \qquad 1 \le k \le i$$

- X: the alignment ends with $y_{j-k+1} \ldots y_j$ against a run of k gaps in *x*. Score the whole run at once: gscore(k).
- We don't know k, so try every k – that loop is the new cost.
- Before the run: anything except another gap in *x* (next slide). Y is the mirror image.

General gaps: why X never follows X

(a) one run of 4: gscore(4)



(b) cut into 1 + 3: gscore(1) + gscore(3)



- If X(i, j) could build on X(i, j−k), one run of gaps could be cut into two pieces and scored as two runs – (b).
- For a concave penalty the split version scores lower or equal; for other penalties it could score higher. Either way it is the wrong score for that alignment.
- Nothing is lost by forbidding it: the longer run is already covered by a larger k.

General gaps: base cases and running time

- **Base cases**: M(0, 0) = 0; X(0, j) = gscore(j); Y(i, 0) = gscore(i). Every other border cell is −∞ – no alignment ends that way.
- **Answer**: max{ M(m, n), X(m, n), Y(m, n) }; trace back from whichever table attains it.
- **Time**: 3(m+1)(n+1) cells. An M cell takes O(1), but an X or Y cell tries every k: O(n) or O(m). Total **O(mn(m + n))** – $O(n^3)$ when m = n.
- **Space**: O(mn).

Affine gaps

Open once, then extend cheaply

Three kinds – Semi-global – Local – General gaps – **Affine gaps**

Affine gap penalties

$$\text{gscore}(k) = \mathbf{g}_{\text{open}} + (k-1)\cdot \mathbf{g}_{\text{ext}}$$

alignment	linear (gap −7)	affine (−10, −1)
`AAAGAATTCA / A–A–A–T–CA`	32	20
`AAAGAATTCA / AAA––––TCA`	32	**47**

- Here $g_{open} = -10$ and $g_{ext} = -1$ (with match +10, mismatch −5). Opening is expensive; extending is cheap.
- Now (b) wins outright: **47** is the unique optimum, and the 20-way tie is gone.
- *Conventions differ: some books write open + k · extend, so their "open" is our $g_{open} - g_{ext}$. Check before you compare numbers.*

Affine gaps: only open vs extend matters

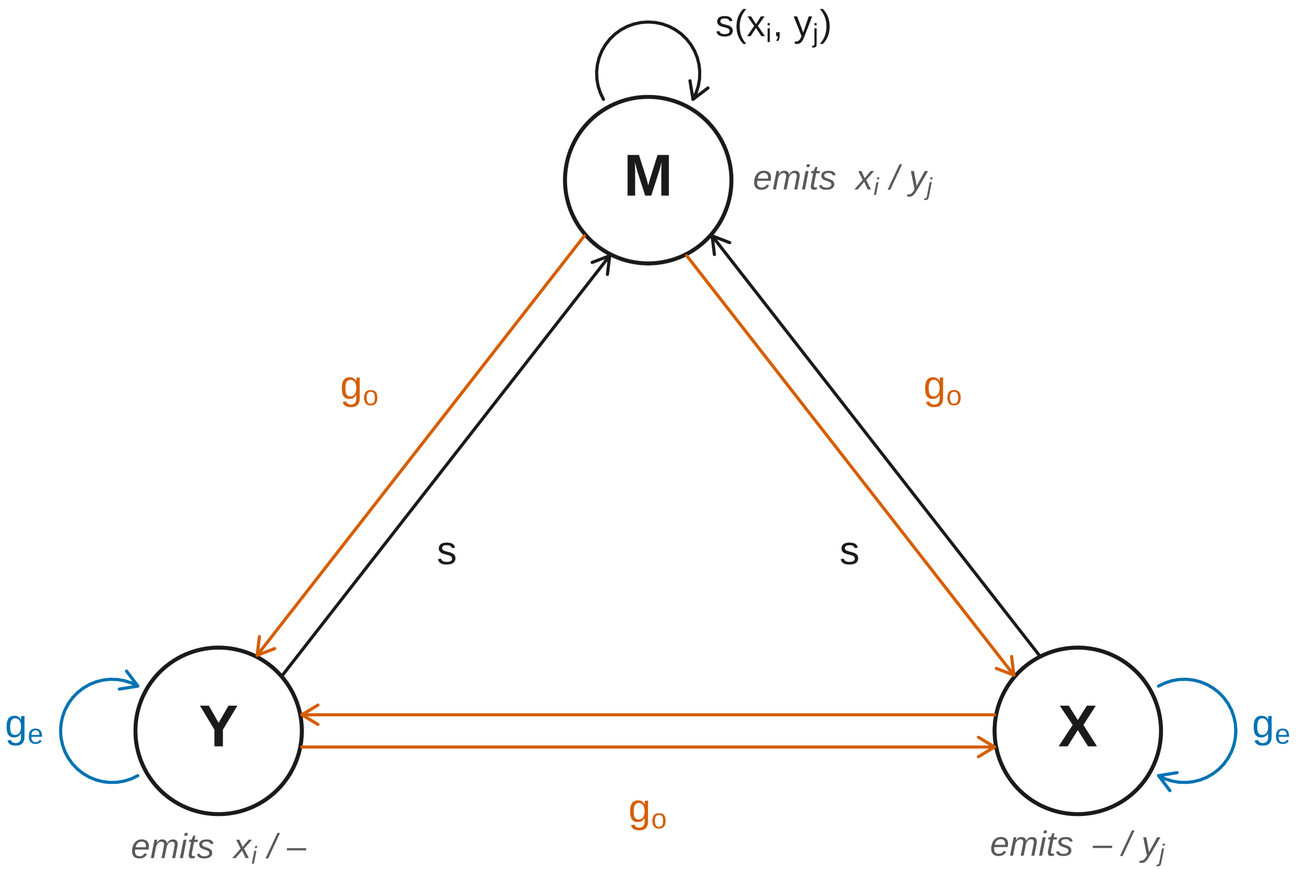
$$M(i,j) = s(x_i, y_j) + \max \begin{cases} M(i-1, j-1) \\ X(i-1, j-1) \\ Y(i-1, j-1) \end{cases}$$

$$X(i,j) = \max \begin{cases} g_o + M(i, j-1) & \textit{open} \\ g_e + X(i, j-1) & \textit{extend} \\ g_o + Y(i, j-1) & \textit{open} \end{cases}$$

$$Y(i,j) = \max \begin{cases} g_o + M(i-1, j) & \textit{open} \\ g_o + X(i-1, j) & \textit{open} \\ g_e + Y(i-1, j) & \textit{extend} \end{cases}$$

- $g_o = g_{open}$, $g_e = g_{ext}$. X and Y now look back **one step**, not k steps.
- A gap in *x* that follows a gap in *x* **extends** it; after anything else it **opens** a new run.
- The table it came from records exactly the one fact we need.

The same thing as a state machine



- States = the three tables; each step emits one alignment column.
- Entering X or Y from elsewhere costs g_o; staying costs g_e; entering M scores the (mis)match.
- An alignment = a walk through the machine; the DP finds the best-scoring walk that emits x and y.
- *Pair hidden Markov models are this picture with probabilities.*

Affine gaps: base cases

- **$M(0, 0) = 0$**: the empty alignment. Everything starts here.
- $M(i, 0) = M(0, j) = -\infty$ for $i, j \geq 1$: an alignment with an empty string cannot end in a (mis)match.
- $X(0, j) = g_o + (j - 1) \cdot g_e$: all of $y_1 \ldots y_j$ against one run of gaps. $X(i, 0) = -\infty$ for all i (including $X(0, 0)$).
- $Y(i, 0) = g_o + (i - 1) \cdot g_e$, and $Y(0, j) = -\infty$ – the mirror image.
- **Why $X(0, 0) = -\infty$**: otherwise $X(0, 1)$ could "extend" a run that does not exist and pay g_e instead of g_o.

Example: CARTS vs CAT, affine gaps

M · ends x_i / y_j

		–	C	A	R	T	S
		0	1	2	3	4	5
T	3	−∞	−13	−7	8	5	−3
A	2	−∞	−12	10	−7	−8	−9
C	1	−∞	5	−12	−13	−14	−15
–	0	0	−∞	−∞	−∞	−∞	−∞

X · ends – / y_j

		–	C	A	R	T	S
		0	1	2	3	4	5
T	3	−12	−6	0	−10	−11	−12
A	2	−11	−5	−15	−16	−17	−18
C	1	−10	−20	−21	−22	−23	−24
–	0	−∞	−∞	−∞	−∞	−∞	−∞

Y · ends x_i / –

		–	C	A	R	T	S
		0	1	2	3	4	5
T	3	−∞	−22	−16	−10	−2	−3
A	2	−∞	−21	−15	0	−1	−2
C	1	−∞	−20	−5	−6	−7	−8
–	0	−∞	−10	−11	−12	−13	−14

- Scores: match +5, mismatch −2, $g_o = -10$, $g_e = -1$. Shaded: the cells visited by the traceback of `CARTS / CAT--`.
- At (5, 3), Y and M both hold **−3**: two optimal alignments, one ending in a gap and one in a mismatch.
- *−∞ marks alignments that cannot exist (e.g. X along the bottom row).*

Reading the alignment back out

in table	came from	column added	score added
M(i, j)	M, X or Y at (i−1, j−1)	x_i / y_j	$s(x_i, y_j)$
X(i, j)	M or Y at (i, j−1) · X at (i, j−1)	– / y_j	$g_o \cdot g_e$
Y(i, j)	M or X at (i−1, j) · Y at (i−1, j)	x_i / –	$g_o \cdot g_e$

(a) M → M → M → Y → Y

C A **R** T S
| | ×
C A **T** – –
M M M Y Y

(b) M → M → Y → Y → M

C A R T **S**
| | ×
C A – – **T**
M M Y Y M

(a) 5 + 5 − 2 + (−10 − 1) = −3 (b) 5 + 5 + (−10 − 1) − 2 = −3

The letter under each column is the table the traceback was in when it added that column.

Why not one table plus arrows?

(a) Best two ways to align ART with CAR

```
ART
× × ×
CAR
```

−6 ends in a mismatch

```
−ART
 | |
CAR−
```

−10 ends in a gap

(b) Add S against a gap

```
ARTS
× × ×
CAR−
```

$-6 + g_o = -16$

```
−ARTS
 | |
CAR−−
```

$-10 + g_e = -11$

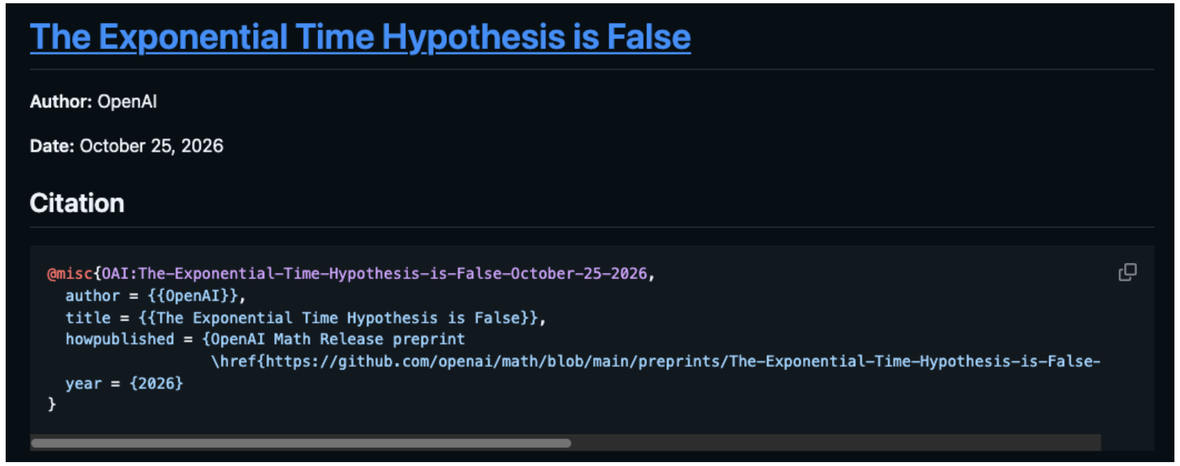
- One table keeps only the winner at each cell: −6 at (3, 3). Extending it with a gap must *open* one, so it reports **−16**.
- The true optimum, `−ARTS / CAR−−`, builds on the *loser* (−10), because its gap can be *extended*: **−11**.
- Y(3, 3) = −10 keeps that runner-up. Each table is the best score *given how the alignment ends* – the one fact the future depends on.

Affine gaps: running time

- **3(m+1)(n+1) cells**, each a max of three: **O(mn) time** — the same as linear gaps (Gotoh, 1982).
- Space O(mn) to trace back; O(n) if only the score is needed (keep two columns of each table).
- General gaps: O(mn(m + n)). Affine buys nearly the same modelling power for a constant factor of 3 — which is why almost every aligner uses it.
- *Local + affine: add the 0 case to M (start a new alignment) and take the best M anywhere.*

Side note: can we beat $O(n^2)$?

- Reading the strings takes $\Omega(n)$; the DP takes $O(n^2)$. Is there anything in between?
- **Four Russians** (Arlazarov, Dinic, Kronrod, Faradzev, 1970): precompute every small block of the table. Masek & Paterson (1980) used it for edit distance in **$O(n^2 / \log n)$**.
- **Probably not much better**: Backurs & Indyk (STOC 2015) showed that an $O(n^{2-\varepsilon})$ algorithm for edit distance would refute the Strong Exponential Time Hypothesis (SETH).
- *That is a conditional lower bound – ~~most researchers believe SETH, but it is not proven~~.*



The Exponential Time Hypothesis is False

Author: OpenAI

Date: October 25, 2026

Citation

```
@misc{OAI:The-Exponential-Time-Hypothesis-is-False-October-25-2026,
  author = {{OpenAI}},
  title = {{The Exponential Time Hypothesis is False}},
  howpublished = {OpenAI Math Release preprint
                  \href{https://github.com/openai/math/blob/main/preprints/The-Exponential-Time-Hypothesis-is-False-
  year = {2026}
}
```

- *Note: ETH false does not necessarily imply SETH is false but this result disproves both.*

Theorem 1.1 refutes ETH, SETH, Gap-ETH, and NSETH.

https://github.com/openai/math/tree/main/preprints/The-Exponential-Time-Hypothesis-is-False-October-25-2026

Recap

variant	base cases	recurrence	answer	time
global	$i \cdot g,\ j \cdot g$	max of 3	OPT(m, n)	O(mn)
fitting (y in x)	OPT(i, 0) = 0	max of 3	max over top row	O(mn)
local	all 0	max of 3 and 0	max anywhere	O(mn)
general gaps	M(0,0) = 0, $-\infty$ …	3 tables, try every k	max of 3 at (m, n)	O(mn(m+n))
affine gaps	M(0,0) = 0, $-\infty$ …	3 tables, open / extend	max of 3 at (m, n)	O(mn)

Main idea: each variant changes *where paths may start*, *where they may end*, or *what the table must remember* — the DP itself stays the same.