Space-efficient Alignment

Hirschberg's algorithm

CMSC 423 · Fall 2026

Same O(mn) time, O(m + n) space, and still the alignment itself.

UNIVERSITY OF MARYLAND

Follows Kleinberg & Tardos, Algorithm Design, §6.7.

Figures redrawn.

From Lecture 9 to today

Lecture 9

Which parts should align? What does a gap cost?

semi-global · local · affine gaps

→

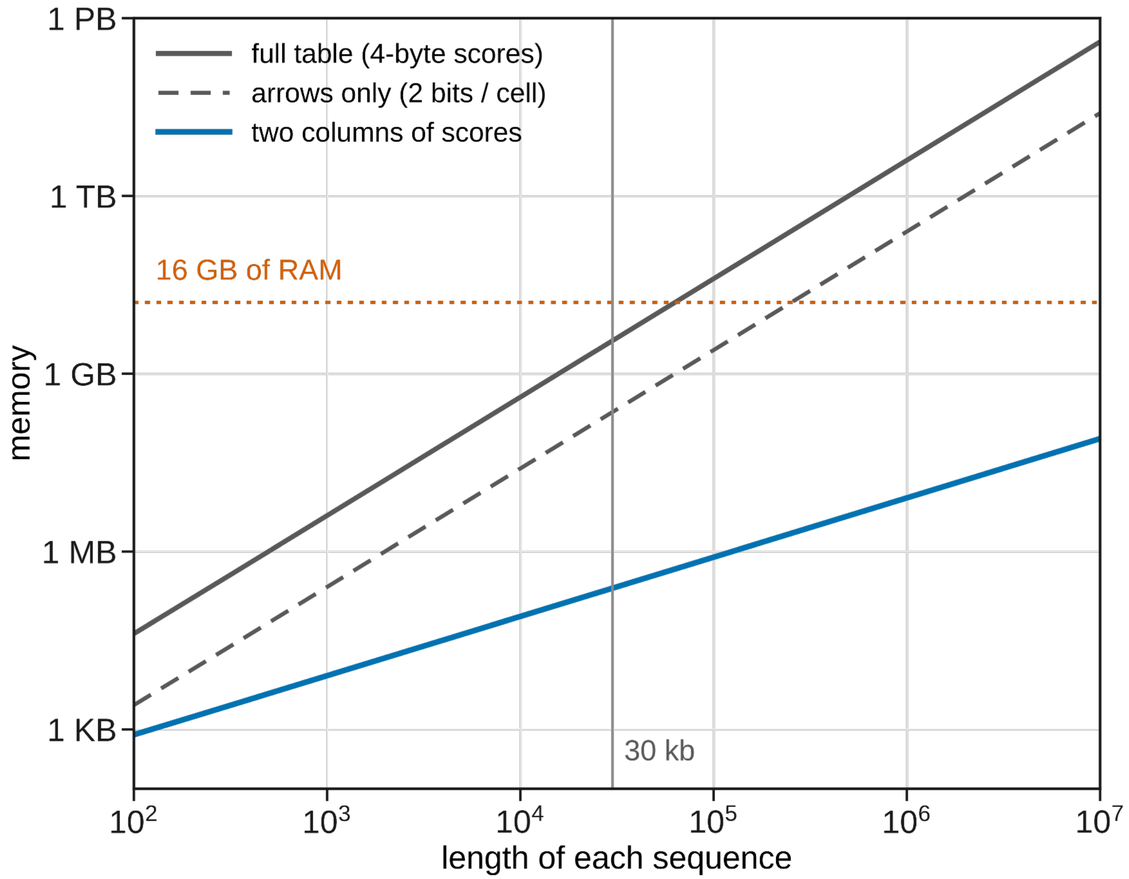
Today

Can we align long sequences without the m × n table?

linear-space scoring · Hirschberg's algorithm

- Every DP so far fills an (m+1) × (n+1) table: **O(mn) time** and **O(mn) space**.
- Time: O(mn) is close to the best possible we've been able to achieve so far.
- Space usually runs out first, and it *can* be fixed: **O(m + n) space at the same O(mn) time**, by combining DP with divide and conquer.

Space runs out before time does



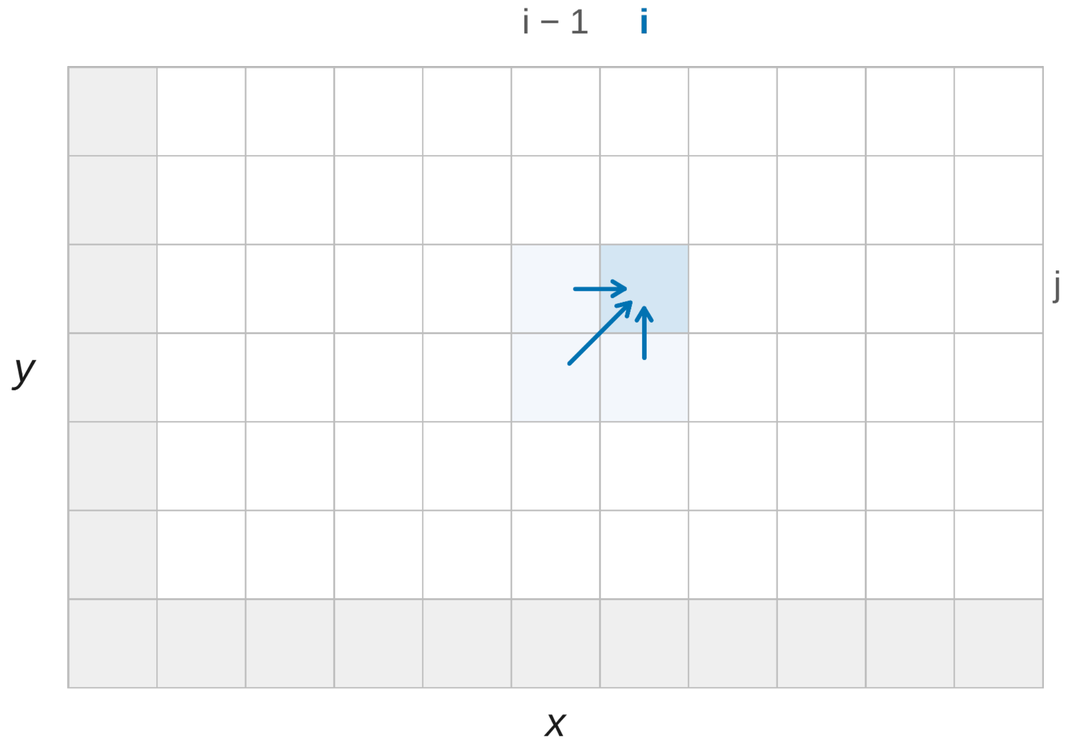
- Two 30 kb sequences: **9×10^8 cells**.
- The full table of 4-byte scores: **≈ 3.6 GB**. Even 2-bit traceback arrows alone: ≈ 225 MB.
- Two columns of scores: **≈ 240 KB**.
- 9×10^8 cell updates are seconds of compute. It is the *memory* that grows out of reach (n^2 against n on the plot).

Score in linear space

What does one cell need?

Score in linear space – Forward & backward – Divide & conquer – Analysis – In practice

One cell needs only its three neighbours

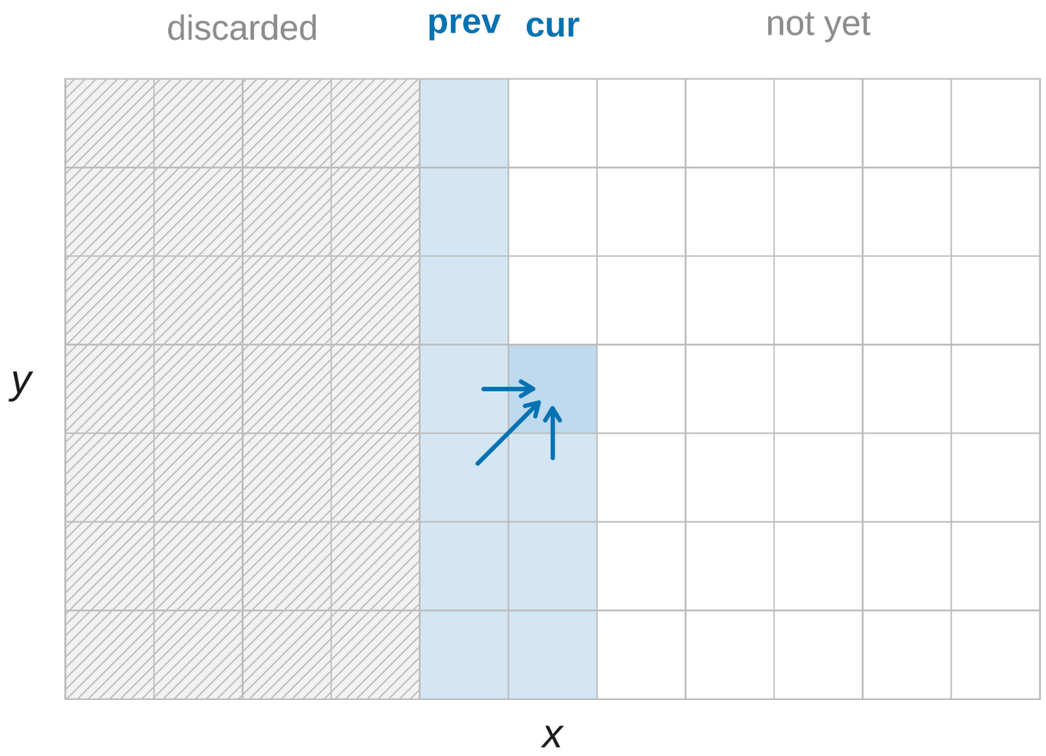


$$\mathrm{OPT}(i, j) = \max \begin{cases} s(x_i, y_j) + \mathrm{OPT}(i-1, j-1) \\ g + \mathrm{OPT}(i-1, j) \\ g + \mathrm{OPT}(i, j-1) \end{cases}$$

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

- Two neighbours are in column i − 1; one is just below, in column i.
- Once column i is finished, columns 0 … i − 2 are *never read again*.

Fill column by column, keep two columns



- Fill column i bottom to top from column i − 1. Then **cur** becomes **prev** and the old one is dropped.
- Two arrays of n + 1 numbers: **O(n) space**, still O(mn) time.
- Rows work as well. Keep the line along the shorter string: **O(min(m, n))**.

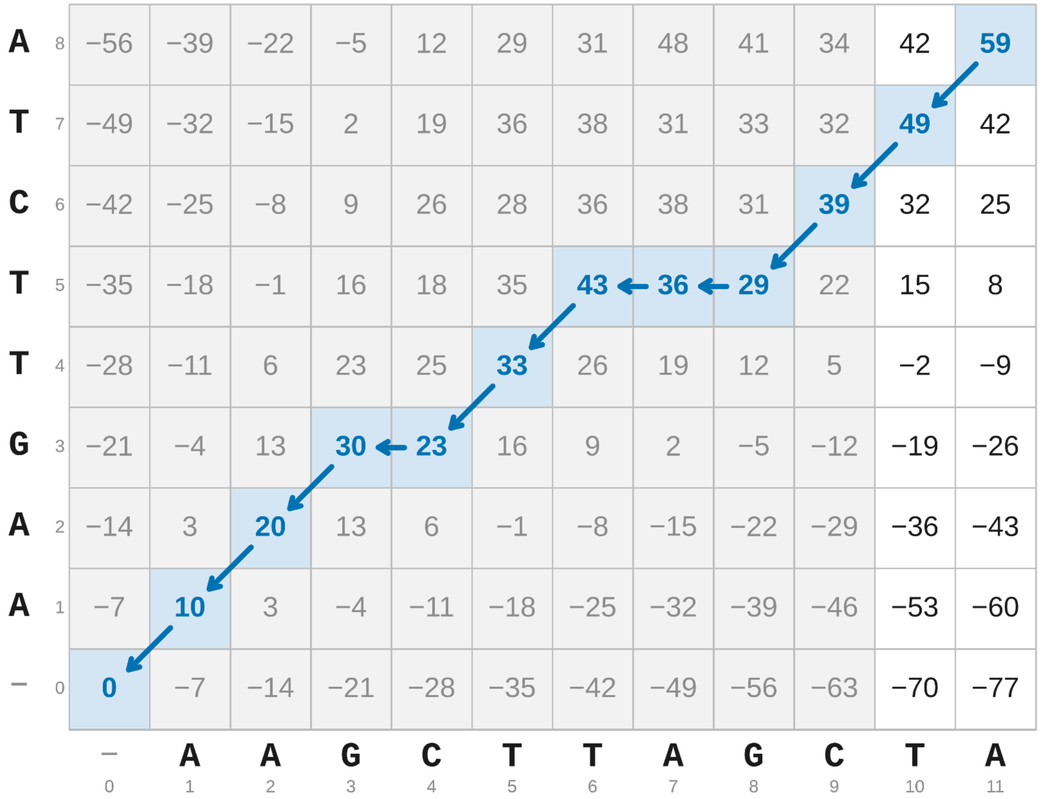
The score in linear space

ScoreOnly(x, y)

```
prev[j] ← j · g                          for j = 0 … n
for i = 1 … m:
    cur[0] ← i · g
    for j = 1 … n:
        cur[j] ← max( prev[j-1] + s(x[i], y[j]),
                      prev[j]   + g,
                      cur[j-1]  + g )
    swap(prev, cur)
return prev                ▷ OPT(m, j) for every j
```

- It returns the **whole last column**, OPT(m, j) for every j, not just OPT(m, n). We will need that.
- Example: x = AAGCTTAGCTA, y = AAGTTCTA (scores +10 / −5 / −7). The last column runs from −77 up to OPT(11, 8) = **59**.
- O(mn) time, O(n) space.

But the traceback needs every column



		–	A	A	G	C	T	T	A	G	C	T	A
		0	1	2	3	4	5	6	7	8	9	10	11
A	8	−56	−39	−22	−5	12	29	31	48	41	34	42	**59**
T	7	−49	−32	−15	2	19	36	38	31	33	32	**49**	42
C	6	−42	−25	−8	9	26	28	36	38	31	**39**	32	25
T	5	−35	−18	−1	16	18	35	**43**	**36**	**29**	22	15	8
T	4	−28	−11	6	23	25	**33**	26	19	12	5	−2	−9
G	3	−21	−4	13	**30**	**23**	16	9	2	−5	−12	−19	−26
A	2	−14	3	**20**	13	6	−1	−8	−15	−22	−29	−36	−43
A	1	−7	**10**	3	−4	−11	−18	−25	−32	−39	−46	−53	−60
–	0	**0**	−7	−14	−21	−28	−35	−42	−49	−56	−63	−70	−77

- The score-only pass ends holding columns 10 and 11. The traceback starts at (11, 8) and walks back to (0, 0) *through every column* — the grey ones are gone.
- Keeping only the arrows still needs Θ(mn) memory: ≈ 225 MB for 30 kb.
- **Idea**: find the path without the table, one cell at a time — starting with where it crosses the *middle* column.

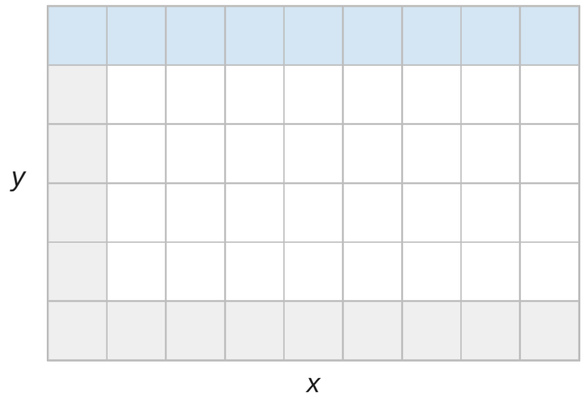
Forward & backward

What a column of the table means

Score in linear space – **Forward & backward** – Divide & conquer – Analysis – In practice

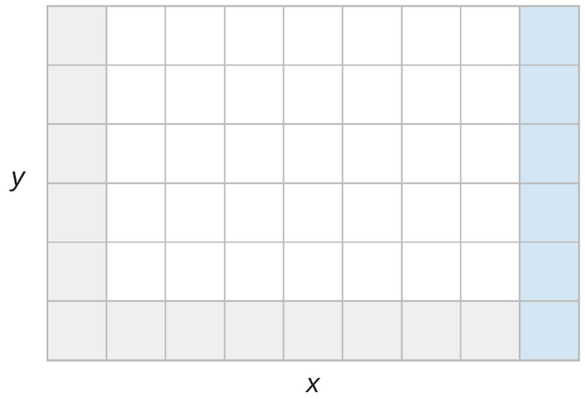
What one column of the table holds

(a) Top row



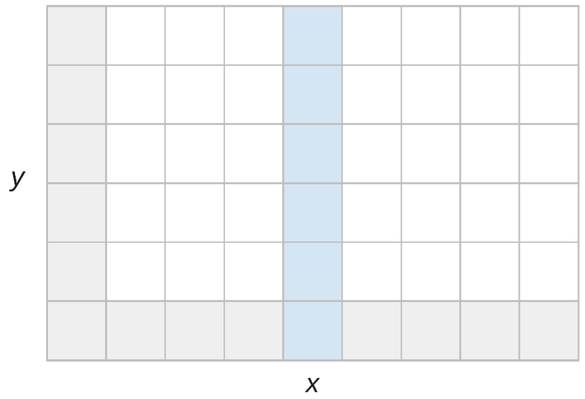
OPT(i, n): $x_1 \ldots x_i$
against all of y

(b) Last column



OPT(m, j): all of x
against $y_1 \ldots y_j$

(c) Any column i

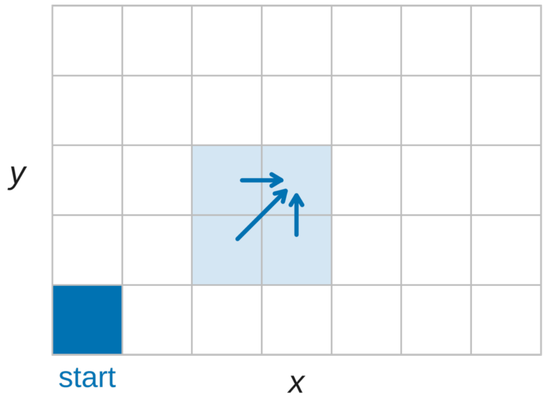


OPT(i, j): $x_1 \ldots x_i$
against $y_1 \ldots y_j$, every j

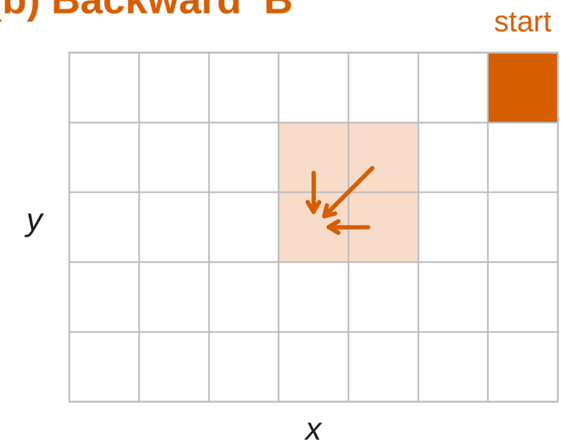
- The score-only pass can stop at *any* column i and hand back (c): run it on $x_1 \ldots x_i$ alone.
- That is the best score of a *prefix* of x against *every* prefix of y – in O(n) space.

Run the same DP backwards: suffixes

(a) Forward F



(b) Backward B



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

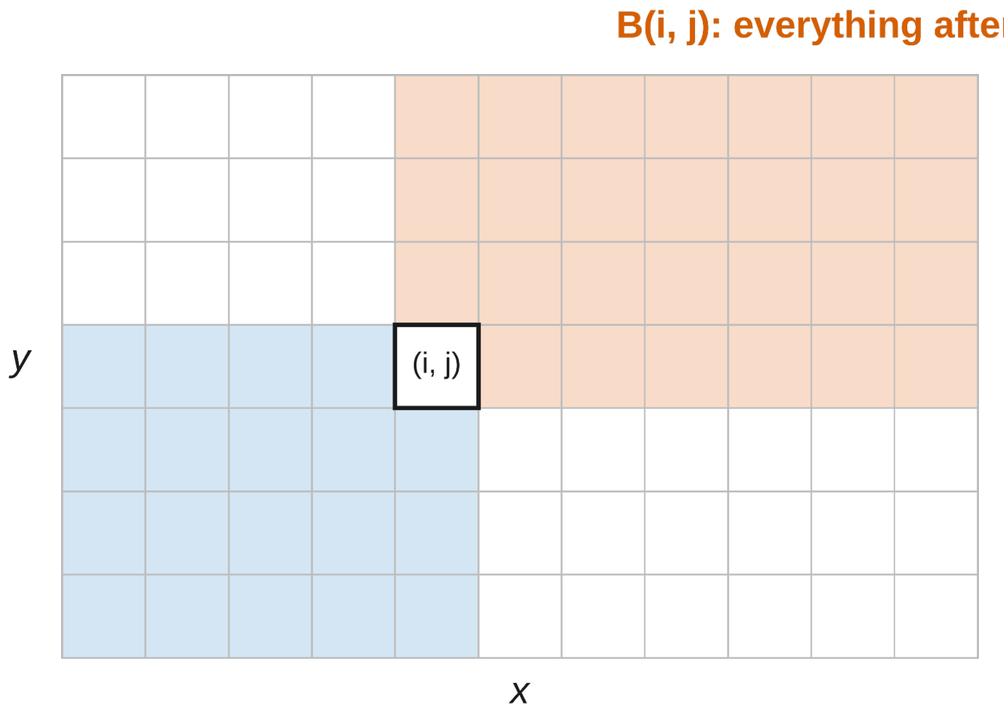
$$F(0,0) = 0,\ F(i,0) = i \cdot g,\ F(0,j) = j \cdot g$$

$$\mathbf{B}(i,j) = \max \begin{cases} s(x_{i+1}, y_{j+1}) + B(i+1, j+1) \\ g + B(i+1, j) \\ g + B(i, j+1) \end{cases}$$

$$B(m,n) = 0,\ B(i,n) = (m-i) \cdot g,\ B(m,j) = (n-j) \cdot g$$

- **F** is Lecture 9's OPT: $x_1 \ldots x_i$ vs $y_1 \ldots y_j$. **B**(i, j) is the best score of the *suffixes* $x_{i+1} \ldots x_m$ vs $y_{j+1} \ldots y_n$.
- B fills from the top-right corner, and also needs only two columns.

One cell, two questions



- Read cell (i, j) as the point *after* x_i and y_j : a node of the edit DAG. **F** looks back to (0, 0); **B** looks ahead to (m, n).
- So **F(i, j) + B(i, j)** is the best score of an alignment *forced through* (i, j).
- That is why B(i, j) starts at x_{i+1}: both tables then describe the same points.
- *Kleinberg & Tardos call them f and g; g is our gap score.*

Example: forward meets backward

			mid = 5									
		–	A	A	G	C	T	T	A	G	C	T	A
		0	1	2	3	4	5	6	7	8	9	10	11
A	8	−56	−39	−22	−5	12	**29** **−42**	−35	−28	−21	−14	−7	0
T	7	−49	−32	−15	2	19	**36** **−25**	−18	−11	−4	3	10	−7
C	6	−42	−25	−8	9	26	**28** **−8**	−1	6	13	20	3	−14
T	5	−35	−18	−1	16	18	**35** **9**	16	23	30	13	−4	−21
T	4	−28	−11	6	23	25	**33** **26**	18	25	23	6	−11	−28
G	3	−21	−4	13	30	23	**16** **28**	20	18	16	−1	−18	−35
A	2	−14	3	20	13	6	**−1** **21**	19	26	9	−8	−25	−42
A	1	−7	10	3	−4	−11	**−18** **29**	36	19	2	−15	−32	−49
–	0	0	−7	−14	−21	−28	**−35** **31**	29	12	−5	−22	−39	−56

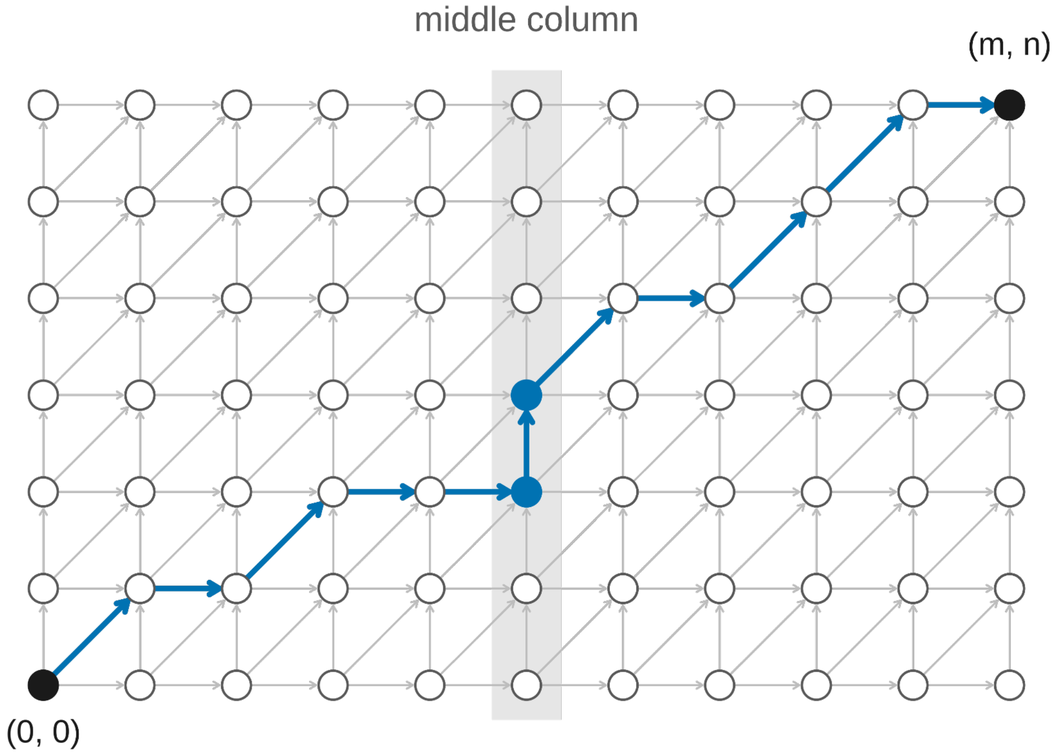
- **F**, filled from column 0 to column mid = ⌊11 / 2⌋ = 5. Only column 5 is kept.
- **B**, filled from column 11 back to column 5.
- Column 5 holds both: F(5, j) on top, B(5, j) below.
- Two score-only passes touch each cell about once (column 5 twice): O(mn) time, O(n) space.

Divide & conquer

Where does the best path cross the middle?

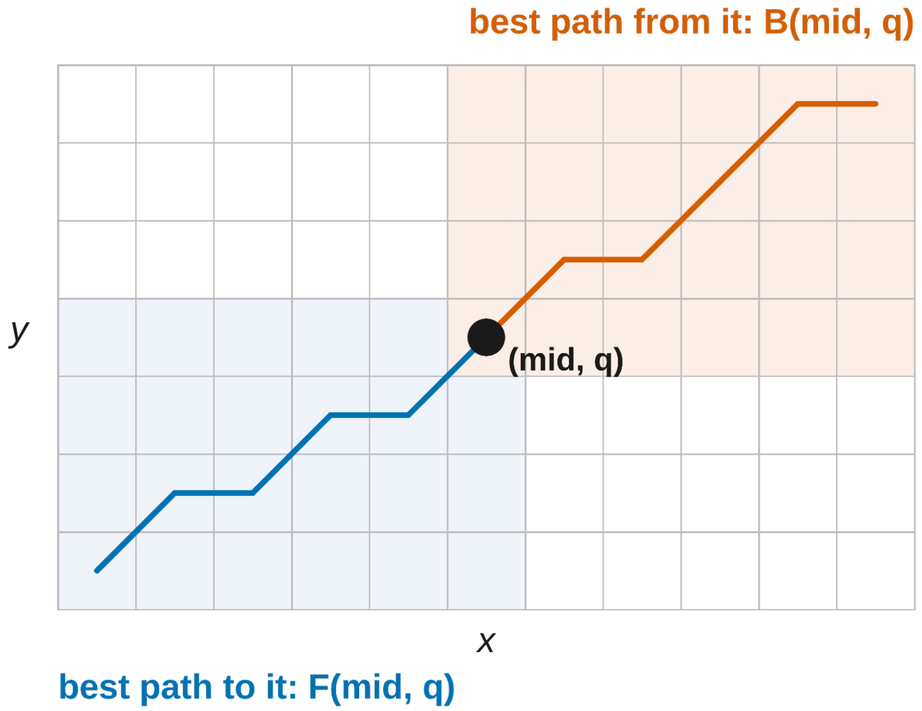
Score in linear space – Forward & backward – **Divide & conquer** – Analysis – In practice

Every path crosses the middle column



- Each edge moves at most one column to the right, so a path from (0, 0) to (m, n) **must visit column mid**.
- It may visit several cells there: a run of vertical steps is a gap in x.
- Take any of them, (mid, q). The path splits into a path (0, 0) → (mid, q) and a path (mid, q) → (m, n).

Where the best path crosses

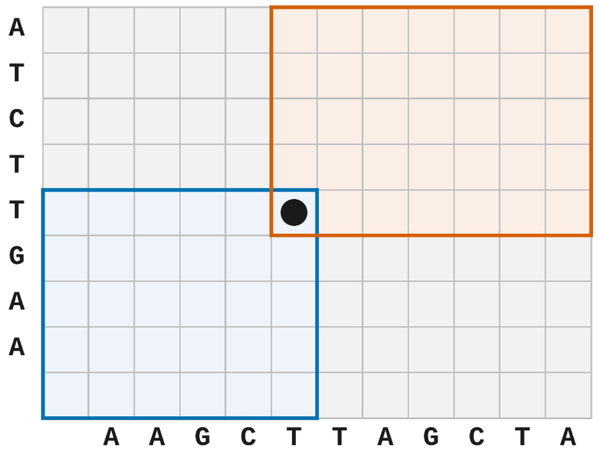


$$OPT(m, n) = \max_q [\ \mathbf{F(mid, q)} + \mathbf{B(mid, q)}\ ]$$

- **≤** The optimal path crosses at some (mid, q). Its first part scores at most F(mid, q), its second at most B(mid, q).
- **≥** For any q, the best path to (mid, q) followed by the best path from it is an alignment of x and y.
- So a q that maximises the sum is a cell some optimal path goes through. Both columns cost O(mn) time and O(n) space.

Example: choose q

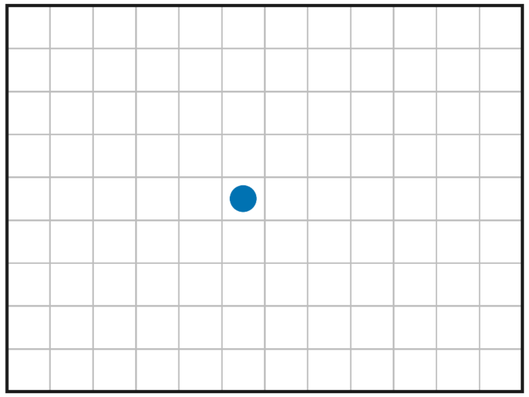
	0	1	2	3	4	5	6	7	8
y_j	–	A	A	G	T	T	C	T	A
F(5, j)	−35	−18	−1	16	**33**	35	28	36	29
B(5, j)	31	29	21	28	**26**	9	−8	−25	−42
sum	−4	11	20	44	**59**	44	20	11	−13



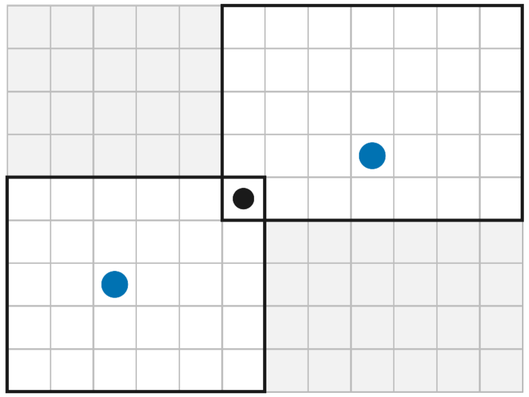
- The largest sum is **59 at q = 4** – exactly OPT(11, 8), and the only q that reaches it.
- So the optimal alignment puts `AAGCT` against `AAGT`, and `TAGCTA` against `TCTA`.
- Two independent, smaller problems: 5 × 4 and 6 × 4. Everything grey is ruled out.

Recurse on the two rectangles

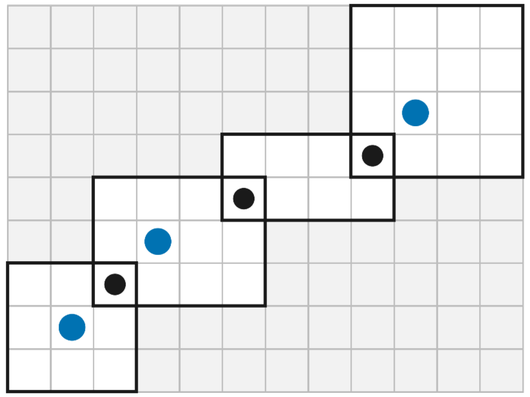
(a) Level 0



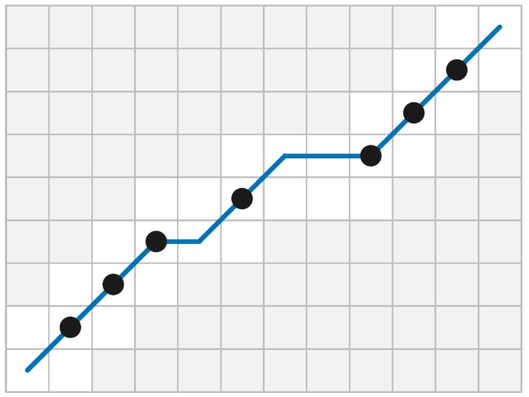
(b) Level 1



(c) Level 2



(d) The path



- Each split fixes one cell of the optimal path (**blue dot**) and leaves two rectangles. Everything grey can no longer be on the path.
- Recurse until a rectangle has one column or one row; solve that with the plain DP – its table has at most two columns or two rows.
- Here: 7 splits, 8 base cases. Joining the pieces gives the whole path (d).

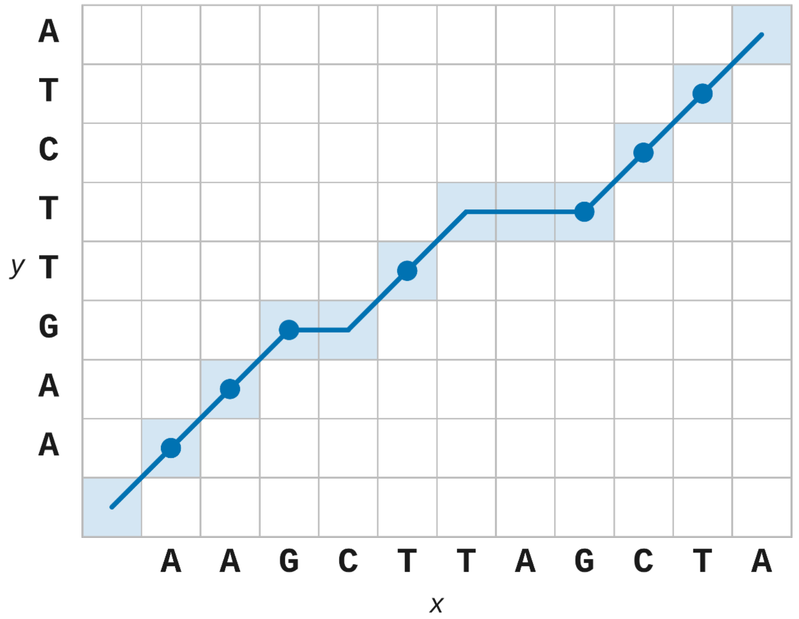
Hirschberg's algorithm

```
Hirschberg(x, y):                          ▷ returns an optimal path
    m ← |x|,  n ← |y|
    if m ≤ 1 or n ≤ 1:
        return plain DP + traceback        ▷ table ≤ 2 wide
    mid ← ⌊m / 2⌋
    F ← ScoreOnly(x[1..mid], y)             ▷ forward column
    B ← backward ScoreOnly(x[mid+1..m], y)
    q ← the j maximising F[j] + B[j]
    P₁ ← Hirschberg(x[1..mid],   y[1..q])
    P₂ ← Hirschberg(x[mid+1..m], y[q+1..n])
    return P₁ joined to P₂                 ▷ they share (mid, q)
```

- Pass *index ranges*, not copies of substrings.
- Ties: every best q lies on some optimal path. We take the smallest.
- Hirschberg (1975) did this for the longest common subsequence; it carries over to any linear-gap score.

After Kleinberg & Tardos, §6.7; Hirschberg (1975)

Example: the result




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

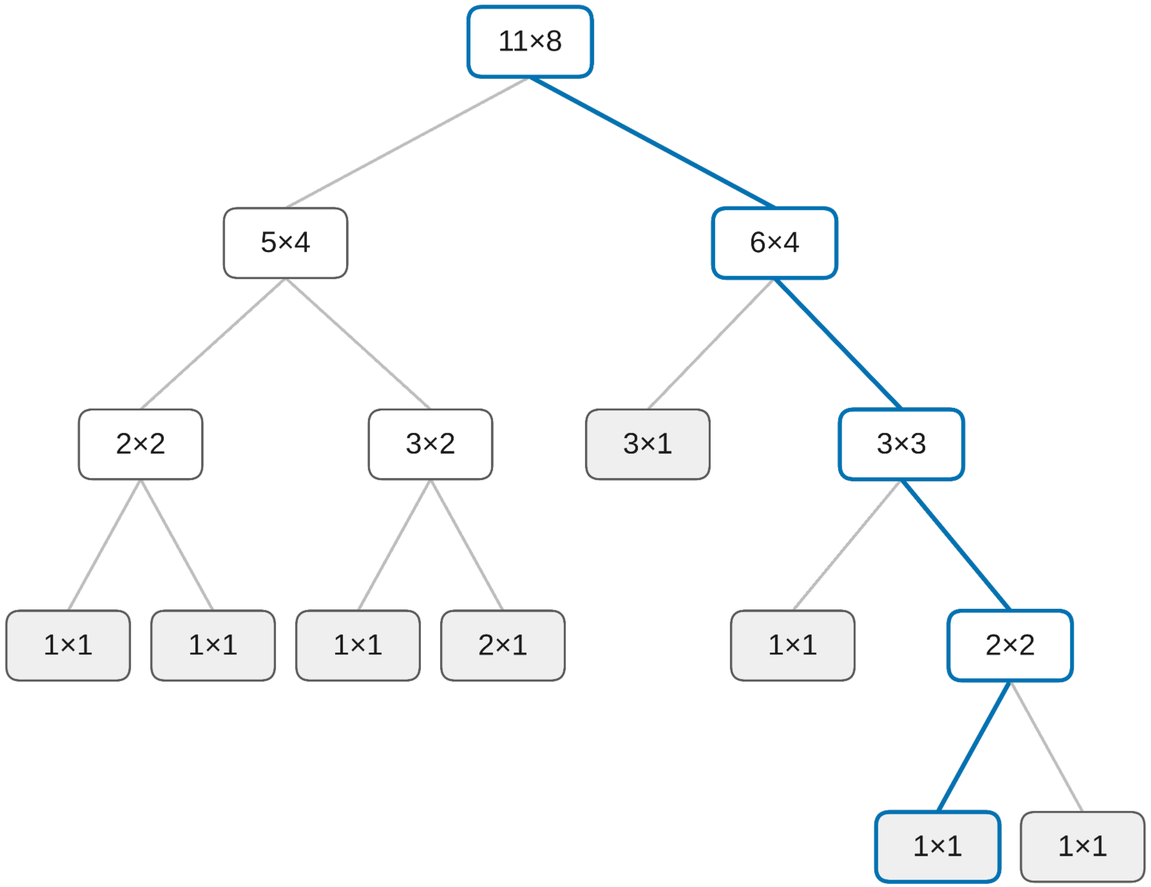
- 8 matches and 3 gaps: $8 \cdot 10 - 3 \cdot 7 = \mathbf{59} = \mathrm{OPT}(11, 8)$.
- The same alignment the full table's traceback finds – here the optimum is unique.
- With ties, Hirschberg returns *an* optimal alignment, not necessarily the one the full traceback would pick.

Analysis

How much space? How much time?

Score in linear space – Forward & backward – Divide & conquer – **Analysis** – In practice

Space: O(m + n)



grey: base case (plain DP)

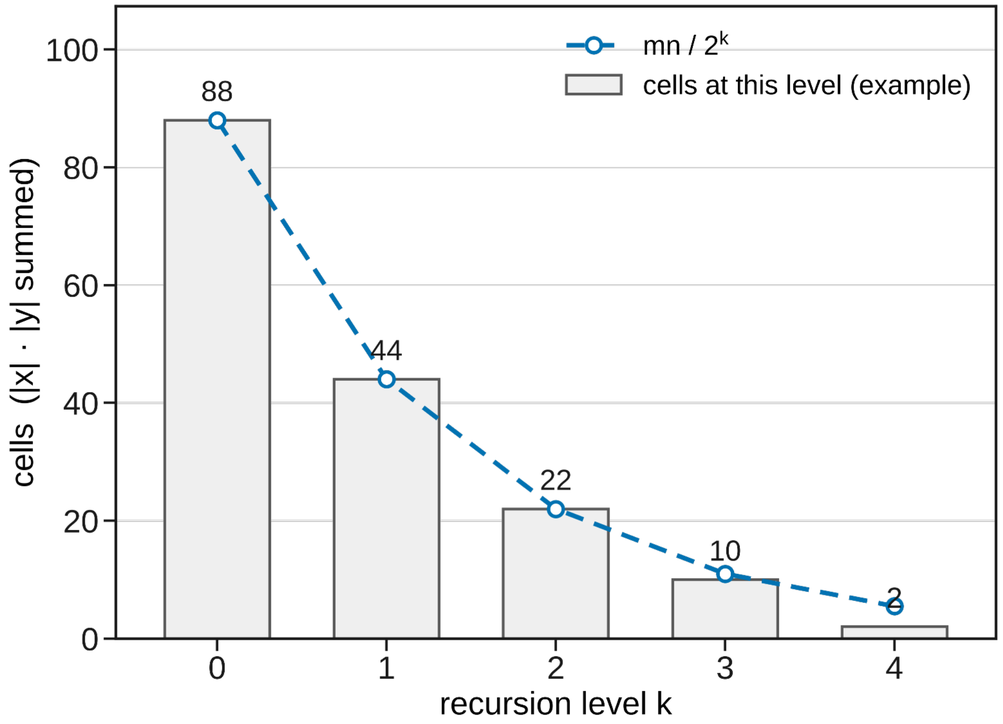
- **Score passes**: F's column plus B's two working columns, 3(n + 1) numbers, freed before recursing – O(n).
- **Stack**: only the **blue** chain is alive at once, depth ≈ $\log_2$ m, a few indices per frame.
- **Path**: at most m + n + 1 cells; base cases: a table at most two wide.
- *Example: never more than 27 = 3(n + 1) scores alive; depth 5.*

Time: every level does half the work

$T(m, n) \le c \cdot mn + T(\lfloor m/2 \rfloor, q) + T(\lceil m/2 \rceil, n - q)$

$T(m, 1) \le cm, \quad T(1, n) \le cn$

- $c \cdot mn$: the two score passes touch each cell about once.
- q depends on the strings, so the master theorem does not apply directly. But the two children's areas add up to *about half* whatever q is: $mid \cdot q + (m - mid)(n - q) \approx mn / 2$.
- So the levels cost $mn + mn/2 + mn/4 + \ldots \le$ **$2mn$**. The example: 88, 44, 22, 10, 2.



Proof by induction

Claim: **T(m, n) ≤ k · mn** with k = 2c. Assume m is a power of 2 (otherwise only the constants change).

```
T(m, n) ≤ c·mn + T(m/2, q) + T(m/2, n − q)
        ≤ c·mn + k·(m/2)·q + k·(m/2)·(n − q)      ▷ induction
        = c·mn + k·(m/2)·n
        = (c + k/2) · mn
        = k · mn                                  ▷ k = 2c
```

- Base cases: T(m, 1) ≤ cm ≤ k · m · 1, and T(1, n) likewise.
- *Pieces with q = 0 or q = n are a single row of gaps: O(m + n) more per level, a lower-order term.*
- The q terms cancel: that is the *area halves* fact from the last slide.
- So Hirschberg is O(mn), about twice the work of the plain DP.

Twice the work, a sliver of the space

method	time	space	returns
full table + traceback	O(mn)	O(mn)	the alignment
two columns (score only)	O(mn)	O(min(m, n))	the score (a whole column)
Hirschberg	O(mn), ≈ 2×	O(m + n)	the alignment

- Measured on a random 300 × 300 DNA pair: Hirschberg evaluated 2.02 × (m+1)(n+1) cells, with at most 3(n + 1) scores alive.
- For two 30 kb sequences: ≈ 360 KB of scores instead of ≈ 3.6 GB.

In practice

Other alignments, and faster in practice

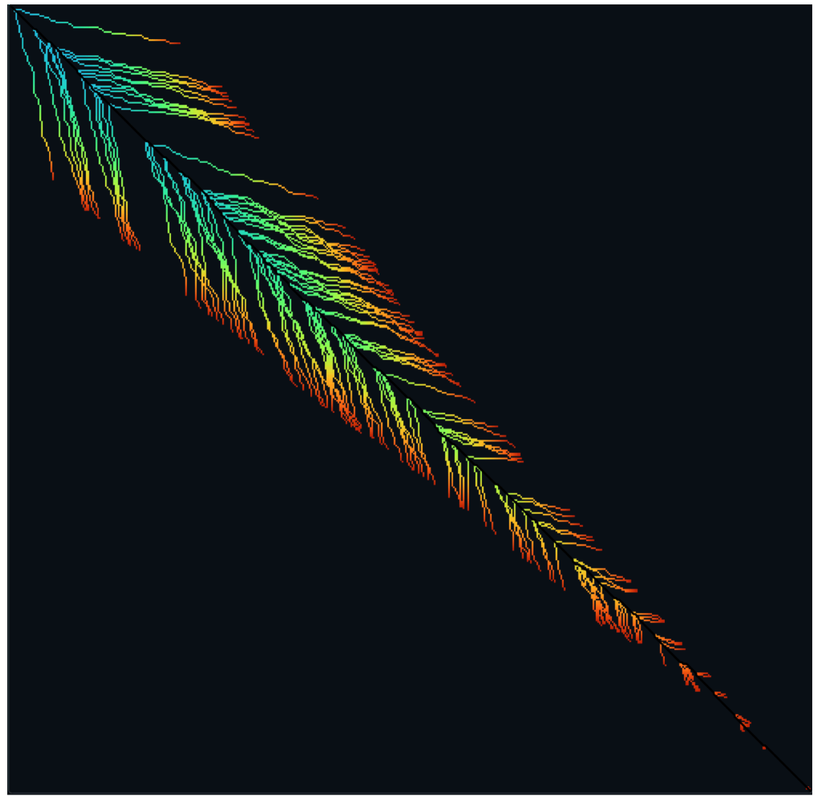
Score in linear space – Forward & backward – Divide & conquer – Analysis – **In practice**

The same trick for other alignments

- **Local alignment**: a forward local score pass finds where the best local alignment ends; a backward pass from that cell finds where it starts. Then run global Hirschberg on just those two substrings.
- **Affine gaps** (Myers & Miller, 1988): carry the M, X, Y columns forward and backward. The split must also allow the best path to cross the middle column *inside a gap*, so that the gap is opened only once.
- **Semi-global**: change the base cases and the end cell as in Lecture 9; the split argument is unchanged.
- *Same bounds in each case: O(mn) time, O(m + n) space.*

Faster when the sequences are similar

- O(mn) is the worst case. Similar sequences have a small optimal penalty s, and the best path stays near the diagonal.
- **WFA** (wavefront alignment) only visits states with penalty ≤ s: **O(ns) time**.
- **BiWFA** (Marco-Sola et al., Bioinformatics 2023) runs WFA from both ends until they meet – Hirschberg's idea – in **O(s) space**.
- **A*PA** (Groot Koerkamp & Ivanov, Bioinformatics 2024): A* with a seed heuristic; on long, similar sequences it explores fewer states still.



States explored by diagonal transition (WFA): they hug the diagonal.

Image: R. Groot Koerkamp, A*PA repository (MPL-2.0)

Recap

- **Score in linear space**: keep two columns. It returns a whole column, not just one number.
- **Forward and backward**: F (prefixes) and B (suffixes) on the same cells; F(i, j) + B(i, j) = the best path through (i, j).
- **Divide**: the best path crosses the middle column at the q that maximises F(mid, q) + B(mid, q).
- **Conquer**: recurse on two rectangles of total area ≈ mn / 2: O(mn) time (≈ 2×), O(m + n) space.

Main idea: divide and conquer turns a score-only DP into one that also finds the alignment: about 2× the time, for a factor of about min(m, n) less space.