Pairwise alignment

Outline of the chapter:

1. Scoring matrices
2. Global, local, and overlap alignment of two sequences using dynamic programming
3. Affine gap penalty
4. Modifications that reduce the space and time complexity

Alignments with more complex gap models

The linear gap penalty model is not ideal for biological sequences, because it counts each insertion or deletion as an individual "evolutionary event":

Linear gap penalty:	`GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL`
	`GSAQVKGHGKK--------VA--D----A-SALSDLHAHKL`
Affine gap penalty:	`GSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKL`
	`GSAQVKGHGKKVADA---------------SALSDLHAHKL`

Want to make it expensive to open a gap, but less expensive to extend a gap.

Alignments with more complex gap models (2)

Consider the *general gap model*:

A function $\gamma(i, j, k)$ is used to calculate the penalty for a *run* of k gaps. This might be quite sophisticated and depend, say, on the details of the sequence that is spanned by the gaps.

Calculation (for global alignment):

Init.: $F(0,0) = 0$, $F(i,0) = \gamma(i,0,i)$ and $F(0,j) = \gamma(0,j,j)$

Recurrence:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + s(x_i, y_j), \\ \max_{1 \le k \le j} \{F(i, j-k) - \gamma(i,j,k)\}, \\ \max_{1 \le k \le i} \{F(i-k, j) - \gamma(i,j,k)\}. \end{cases}$$

Alignments with more complex gap models (3)

Drawback: The general gap model requires *cubic* time, because for each cell (i, j) we need to inspect $i + j + 1$ predecessors.

Affine gap penalties

In practice one uses a *affine gap penalty* to evaluate a run of k consecutive gaps, given by

$$\gamma(k) = d + (k-1)e,$$

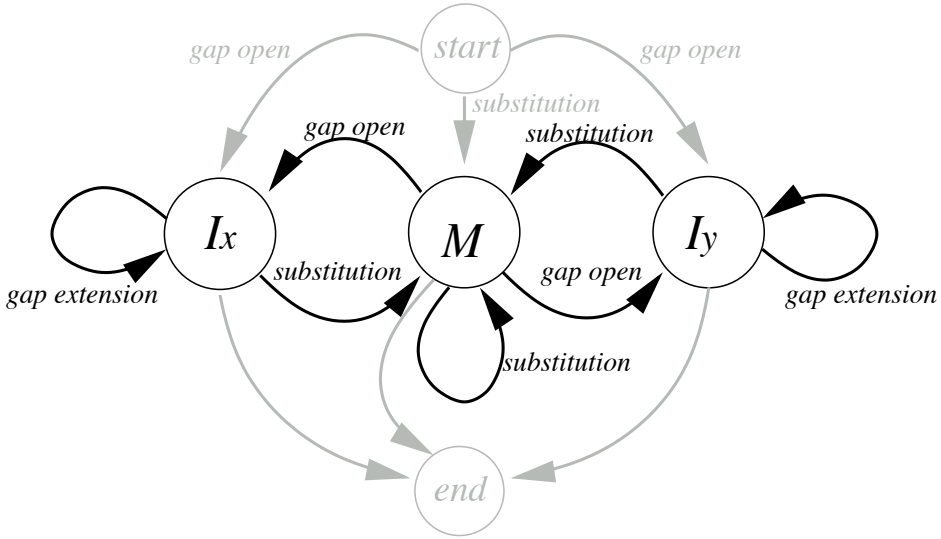
with d the *gap-open* penalty and e the *gap-extension* penalty, with $d > e$.

Gotoh modified the Needleman-Wunsch so as to use affine gap costs.

This achieves *quadratic* time, using three matrices M, I_x and I_y.

Automaton description

We can interpret the three matrices M, I_x and I_y as three different *states* in a finite state automaton ("match", "insertion in x" and "insertion in y"). This automaton emits two aligned characters on each transition across an edge:



We do not want a gap in one sequence to be immediately followed by a gap in the other sequence, hence there are no transitions between I_x and I_y.

Affine gap penalties (2)

There are three main types of transitions:

Case 1 x_i aligns to y_j:	Case 2 x_i aligns to a gap:	Case 3 y_j aligns to a gap:
I G A (...) x_i L G V (...) y_j	A I G A (L ...) x_i A G V y_j (- ...) -	S G A x_i (- ...) - S L G V (P ...) y_j

1. $M(i,j)$ is the best score up to (i,j), given that x_i is aligned to y_j,
2. $I_x(i,j)$ is the best score up to (i,j), given that x_i is aligned to a gap, and
3. $I_y(i,j)$ is the best score up to (i,j), given that y_j is aligned to a gap.

Recursion global alignment with affine gap costs

Initialization:

$$M(0,0) = 0 \text{ and } I_x(0,0) = I_y(0,0) = -\infty$$

$$M(i,0) = I_y(i,0) = -\infty \text{ and } I_x(i,0) = -(d + (i-1)e) \quad \text{for } i = 1, \ldots, n$$

$$M(0,j) = I_x(0,j) = -\infty \text{ and } I_y(0,j) = -(d + (j-1)e) \quad \text{for } j = 1, \ldots, m$$

			x_i					
			0	T	T	A	G	T
y_j	0	M						
		I_x						
		I_y						
	T	M						
		I_x						
		I_y						
	T	M						
		I_x						
		I_y						
	G	M						
		I_x						
		I_y						

Recursion:

$$
\begin{aligned}
I_x(i,j) &= \max \begin{cases} M(i-1,j) - d, \\ I_x(i-1,j) - e; \end{cases} \\
I_y(i,j) &= \max \begin{cases} M(i,j-1) - d, \\ I_y(i,j-1) - e; \end{cases} \\
M(i,j) &= \max \begin{cases} M(i-1,j-1) + s(x_i, y_j), \\ I_x(i-1,j-1) + s(x_i, y_j), \\ I_y(i-1,j-1) + s(x_i, y_j). \end{cases}
\end{aligned}
$$

Example global alignment with affine gap costs (3)

Consider $X = \texttt{TTAGT}$ and $Y = \texttt{TTG}$, $s(a, a) = 1$, $s(a, b) = -1$, $\gamma(k) = d + (k-1)e$ with $d = 4$ and $e = 1$ for the match, mismatch, gap-open and gap-extension scores, respectively:

		0	T	T	A	G	T
0	M	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	I_x	$-\infty$	-4	-5	-6	-7	-8
	I_y	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
T	M	$-\infty$	$+1$	-3	-6	-7	-6
	I_x	$-\infty$	$-\infty$	-3	-4	-5	-6
	I_y	-4	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
T	M	$-\infty$	-3	+2	-4		
	I_x	$-\infty$	$-\infty$	-7	-2		
	I_y	-5	-3	-7	-10		
G	M	$-\infty$	-6				
	I_x	$-\infty$	$-\infty$				
	I_y	-6	-4				

no insertion form x to y
repeat at home with yt vids

Example global alignment with affine gap costs (4)

Consider $X = \texttt{TTAGT}$ and $Y = \texttt{TTG}$, $s(a, a) = 1$, $s(a, b) = -1$, $\gamma(k) = d + (k-1)e$ with $d = 4$ and $e = 1$ for the match, mismatch, gap-open and gap-extension scores, respectively:

		0	T	T	A	G	T
	M	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
0	I_x	$-\infty$	-4	-5	-6	-7	-8
	I_y	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	M	$-\infty$	$+1$	-3	-6	-7	-6
T	I_x	$-\infty$	$-\infty$	-3	-4	-5	-6
	I_y	-4	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
	M	$-\infty$	-3	$+2$	-4	-5	-4
T	I_x	$-\infty$	$-\infty$	-7	-2	-3	-4
	I_y	-5	-3	-7	-10	-11	-10
	M	$-\infty$	-6	-4	$+1$	-1	-4
G	I_x	$-\infty$	$-\infty$	-10	-8	-3	-4
	I_y	-6	-4	-2	-8	-9	-8

Last cell F(n,m) shows the optimal alignment

Simplifying the affine-gap algorithm

This can be simplified to only two matrices, M and I, where

- M corresponds to an alignment of two symbols, and
- I corresponds to an insertion in one of the two sequences.

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

$$I(i,j) = \max \begin{cases} M(i-1,j) - d, \\ I(i-1,j) - e, \\ M(i,j-1) - d, \\ I(i,j-1) - e. \end{cases}$$

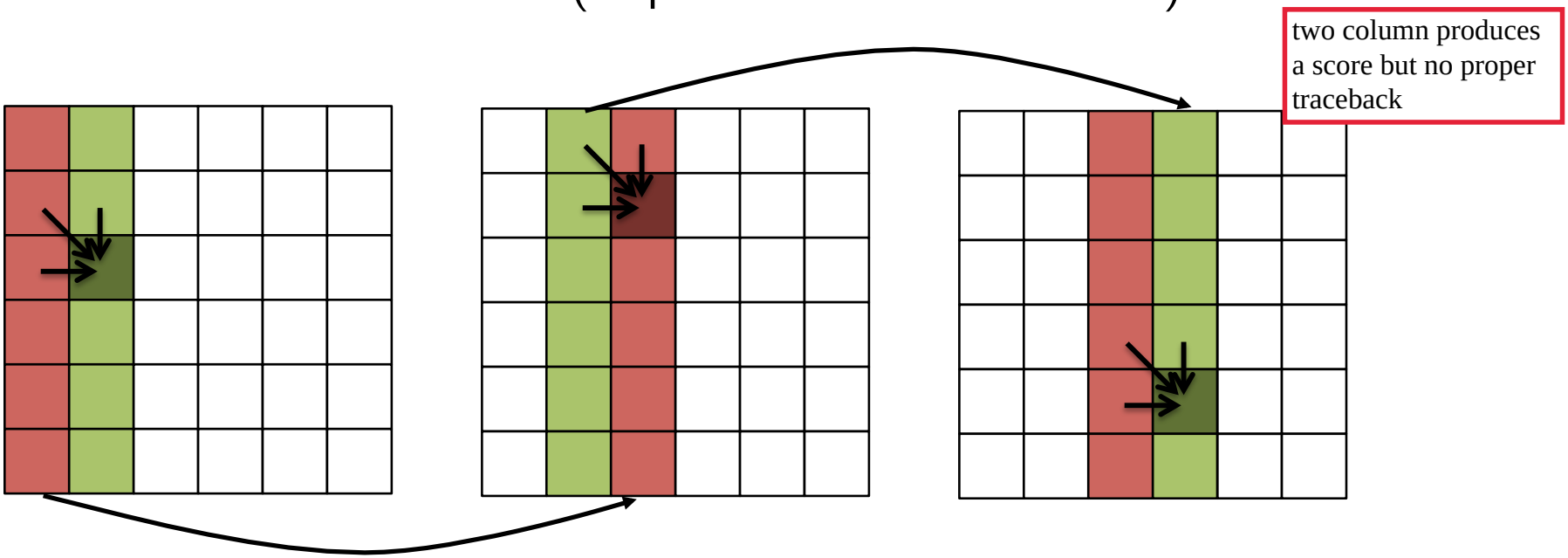
Produces same results as above as long as $\min\{s(a,b)\} > -2e$.

```
XXX   X-   ---
---     -Y   YYY
gapaextension of -2e
```

Alignment in linear space

Can we compute a best alignment between two sequences $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_m)$ using only linear space?

The *best score* of an alignment is easy to compute in linear space. The computation of $F(i, j)$ requires only that we store the values of the previous column and the current column (or previous and current row):



Let's call this the "two column trick".

Divide and conquer

Can we compute a *best alignment* between two sequences $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, m)$ using only linear space?

We saw that the best score can easily be computed using only linear space.

How to perform *trace-back* using only linear space? We will discuss this for the case of global alignments and linear gaps (it also works for all other cases).

```
Two column trick
P previous column i-1
Q current column i

Init
for j =0 to n:
P[j] = -j * d

Recursion
for i = 1-m:
Q[0] = -i * d
for j  = 0 to n:
Q[j] = max  P[j-1] + s(a,b)
            P[j]  - d
            Q[j-1] -d
swap P and Q unless game over
```

Divide and conquer (2)

Idea: Use divide-and-conquer.

Consider the middle column $q = \lfloor \frac{n}{2} \rfloor$ of the F matrix. If we had a method for determining at which cell (q, r) a best scoring alignment passes through the q^{th} column, then that would give us *one element* of the traceback, and we could then divide the dynamic-programming problem into two parts:

- align from $(0, 0)$ to (q, r), and then
- align from (q, r) to (n, m).

$y \setminus x$	0	1	...	$\boldsymbol{q}$	...	n
0						
1						
...						
$\boldsymbol{r}$				(q, r)		
...						
m						

Finally, concatenate the two solutions to obtain the final result.

Divide and conquer (3)

How to determine r, the row in which a best path crosses the q-th column?

Idea: use the two-column trick to compute values of $c(i,j)$, where for $i > q$, we define $c(i,j)$ such that $(q, c(i,j))$ is on an optimal path from $(1,1)$ to (i,j):

Computation of c

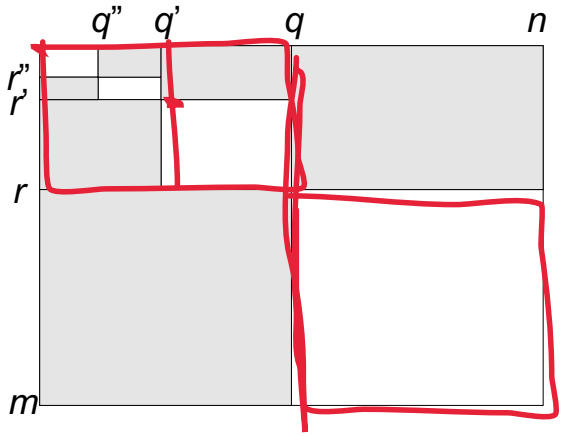
We initialize as follows: $c(q,j) = j$ for $j = 0\ldots,n$.
Then, as we compute $F(i,j)$ column for column, if $i > q$, we set $c(i,j) = c(i',j')$, where (i',j') is the cell from which $F(i,j)$ is obtained.

Note that $c(i,j)$ is computed from the values in the previous and current column only. Hence, we can apply the "two column trick".

The desired value is $r = c(n,m)$.

Divide and conquer (4)

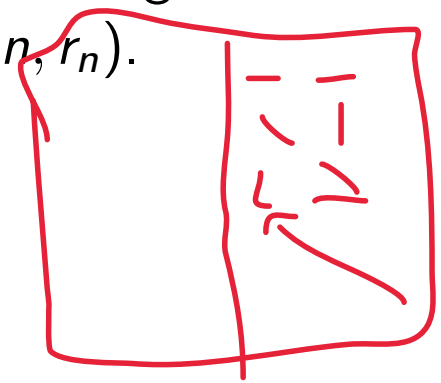
We can now use (q, r) to recurse, as indicated here:



once past middle column compute the value c: optimal path
r = c value at the end
divide the problem into smaller squares always checking for middle line
- only works for global alignment in this case

If we calculate both c and F using the two-column trick, then the algorithm only requires linear space on each subproblem.

We obtain the actual alignment from the sequence of pairs $(1, r_1), (2, r_2), \ldots (n, r_n)$.



compute value c from the middle column
check divide and conquer again
middle column as start of the traceback
initialize from the middle column and get best value for F

Divide and conquer (5)

What is the time complexity?

- In the initial problem, we process $1 \times nm$ cells.
- In the first level of recursion, we process $\frac{1}{2}nm$ cells.
- In the next level of recursion, we process $\frac{1}{4}nm$ cell.
- etc.

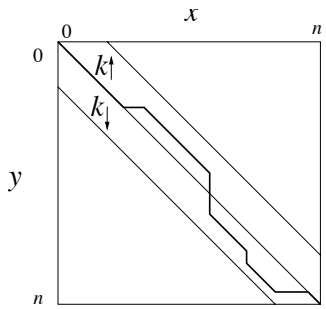
The total number of cells processed is given by $\sum_{i=0}^{n} \frac{1}{2^i} < 2$.

Thus, the algorithm is only twice as slow as the quadratic-space one.

Alignment in linear time: Banded global alignment

To simplify the following discussion, assume $n = m$.

Idea: Instead of computing the whole matrix F, use only a *band* of cells along the main diagonal:



We call $2k$ the *width* of the band.

Clearly, the time complexity of the banded algorithm will be $O(kn)$.

Questions: Will this algorithm produce an optimal global alignment? What should k be set to?

Algorithm (KBand)

Input: *Two sequences X and Y of equal length n, integer k*
Output: *Best score α_k of global alignment, for band of width $2k$*
Initialization: *Set* $(0, i) = 0$.
Set $F(i, 0) = -i \cdot d$ *for all* $i = 1, 2, \ldots, k$.
Set $F(0, j) = -j \cdot d$ *for all* $j = 1, 2, \ldots, k$.

for $i = 1$ **to** n **do**
  for $h = -k$ **to** k **do**
    $j = i + h$
    if $1 \leq j \leq n$ **then**
      $F(i, j) = F(i - 1, j - 1) + s(x_i, y_j)$
      if *insideBand*$(i - 1, j, k)$ **then**
        $F(i, j) = \max\{F(i, j), F(i - 1, j) - d\}$
      if *insideBand*$(i, j - 1, k)$ **then**
        $F(i, j) = \max\{F(i, j), F(i, j - 1) - d\}$
return $F(n, n)$

To test whether (i, j) is inside the band, we use:
$$\textit{insideBand}(i, j, k) = (-k \leq i - j \leq k).$$

Searching for high-identity alignments

We can use the KBand algorithm as a fast method for finding high-identity alignments:

If we know that the two input sequences are highly similar and we have a bound b on the number of gaps that will occur in the best alignment, then the KBand algorithm with $k = b/2$ will compute an optimal alignment.

For example, in forensics, one must sometimes determine whether a sample of human mtDNA obtained from a victim matches a sample obtained from a relative (or from a hair brush etc). If two such sequences differ by more than a few base-pairs or gaps, then they are not considered a match.

Optimal alignments using KBand

Let X and Y be two DNA sequences of the same length n. Let M be the match score and d the gap penalty.

Question: Let α_k be the best score obtained using the KBand algorithm for a given k. Can we tell whether α_k is equal to the optimal alignment score α?

Lemma (Sufficient condition for optimality)

If $\alpha_k \geq M(n-(k+1)) - 2(k+1)d$, then $\alpha_k = \alpha$.

Proof If there exists an optimal alignment with score α that does not leave the band, then clearly $\alpha_k = \alpha$. Else, all optimal alignments leave the band somewhere. This requires insertion of at least $k+1$ gaps in each sequence, and allows only at most $n-(k+1)$ matches, giving the desired bound. □

Optimal alignment using repeated KBand

The following algorithm computes an optimal alignment by repeated application of the KBand algorithm, with larger and larger k:

Algorithm (Repeated KBand)

Input: *two sequences X and Y of the same length n*
Output: *an optimal global alignment of x and y*

Initialize $k = 1$
repeat
    compute α_k *using KBand*
    if $\alpha_k \geq M(n-k-1) - 2(k+1)d$ **then**
        return α_k
    $k = 2k$
end

We omit details of the traceback.

Analysis of time complexity

The algorithm terminates when:

$$\alpha_k \geq M(n-k-1) - 2(k+1)d \quad \Leftrightarrow$$

$$\alpha_k - Mn + M + 2d \geq -(M+2d)k \quad \Leftrightarrow$$

$$Mn - \alpha_k - (M+2d) \leq (M+2d)k \quad \Leftrightarrow$$

$$\frac{Mn-\alpha_k}{M+2d} - 1 \leq k \qquad (*)$$

At termination, the total complexity is given by:

$$n + 2n + 4n + \cdots + kn \leq 2kn.$$

So far, this doesn't look better than nn.

We will show that k will be smaller for more similar sequences.

When the algorithm stops for k, then $(*)$ implies:

$$\frac{k}{2} < \frac{Mn - \alpha_{k/2}}{M + 2d} - 1,$$

because the algorithm didn't stop for $\frac{k}{2}$.

There are two cases: If $\alpha_{k/2} = \alpha_k = \alpha$, then (by replacing $\frac{\alpha}{2}$ by α in the previous equation) we obtain:

$$\frac{k}{2} < \frac{Mn - \alpha}{M + 2d} - 1 \qquad \Leftrightarrow$$

$$k < 2\left(\frac{Mn - \alpha}{M + 2d} - 1\right).$$

Analysis of time complexity (3)

Otherwise, $\alpha_{\frac{k}{2}} < \alpha_k = \alpha$. Then any optimal alignment must have more than $\frac{k}{2}$ gaps, and thus

$$\alpha \leq M\left(n - \left(\frac{k}{2} + 1\right)\right) \leq M\left(n - \frac{k}{2} - 1\right) + 2\left(\frac{k}{2} + 1\right)d,$$

where we added the positive term on the right so that solving for k gives use the following condition:

$$k \leq 2\left(\frac{Mn - \alpha}{M - 2d} - 1\right).$$

In both cases, k is bounded by a linear function that *decreases* as α increases, hence:

The more similar the sequences, the faster the KBand algorithm will run.

Summary

We saw that global-, local- and overlap alignment can be efficiently performed using dynamic programming.

The affine gap model places gaps in runs, without an increase of running time.

The dynamic programs can be run in linear space.

The use of a band can reduce the runtime to linear for very similar sequences.

The Repeated KBand algorithm has a runtime that depends on the similarity of the two input sequences.