Algorithms in Genome Analysis, Spring 2023

Veli Mäkinen

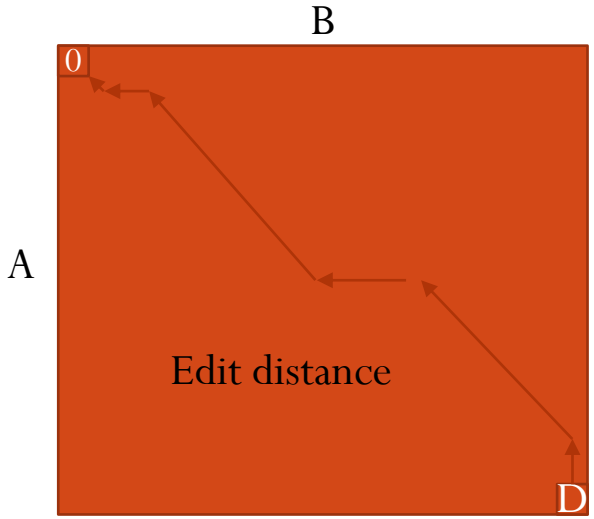
Week 3

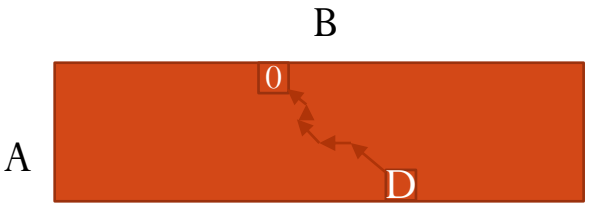
Alignments – some more advanced topics

Connection to edit distance

- An alignment can be interpreted as editing instructions to convert A into B:
  - A[i] is aligned with B[j] → Substitute A[i] with B[j]
  - A[i] is aligned with a gap ”-” → Delete A[i]
  - Gap ”-” is aligned with B[j] → Insert B[j]
- D[i,j]=min(
  - D[i-1,j-1]+(A[i]=B[j]?0:1),
  - D[i-1,j]+1,
  - D[i,j-1]+1)
- D[0,j]=j, D[i,0]=i as initialization, D(A,B)=D[|A|,|B|] as finalization
- Here D(A,B) is the unit cost edit distance.

More variations of the theme





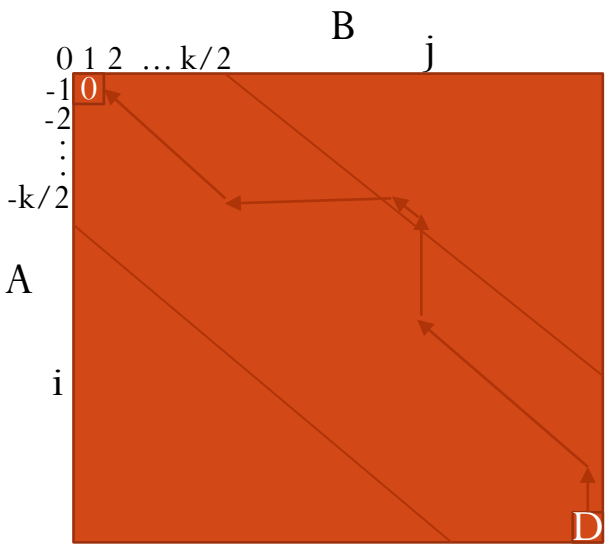
Approximate string matching

Shortest detour

Speeding-up edit distance computation

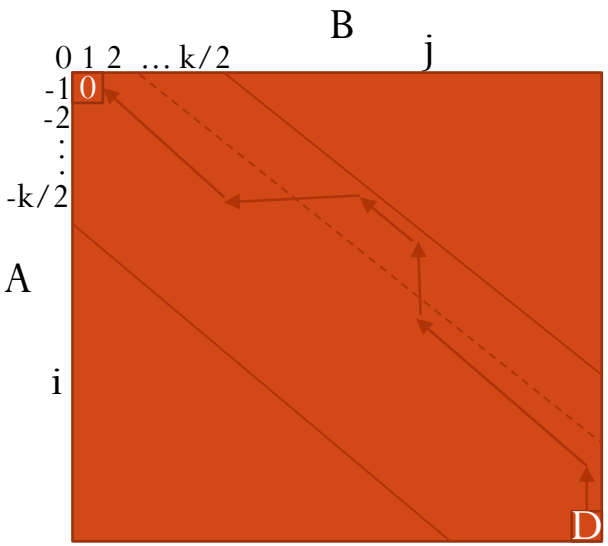
O(kn) time, where k is a threshold

- Assume |A|=|B|=n for simplicity of exposition.
- Consider a *diagonal zone* $i - j \in [-\frac{k}{2}, -\frac{k}{2} + 1, \dots, 0,1,2, \dots, \frac{k}{2}]$.
- If traceback to D[n,n] uses a cell outside the diagonal zone, it corresponds to an alignment with at least k/2+1 deletions and k/2+1 insertions, and the total cost is at least k+2.
- To decide if D(A,B)≤k, it is thus sufficient to do computation inside the diagonal zone: O(kn) time.



O(dn) time, where d is D(A,B)

- We can use doubling: Run computation with k=1, k=2, k=4, …
- As soon as D[n,n]≤k, we know that any traceback path that goes outside the diagonal zone will have cost greater than D[n,n]. That is, d=D(A,B)=D[n,n].
- As we didn't stop earlier, k/2<d.
- The running time is
  - $\sum_{x=0}^{1+\log_2 d} 2^x n < 2^{2+\log_2 d}\, n$
  - $= O(dn)$
- Algorithm and its analysis extend to the general case, where |A|≠|B|.



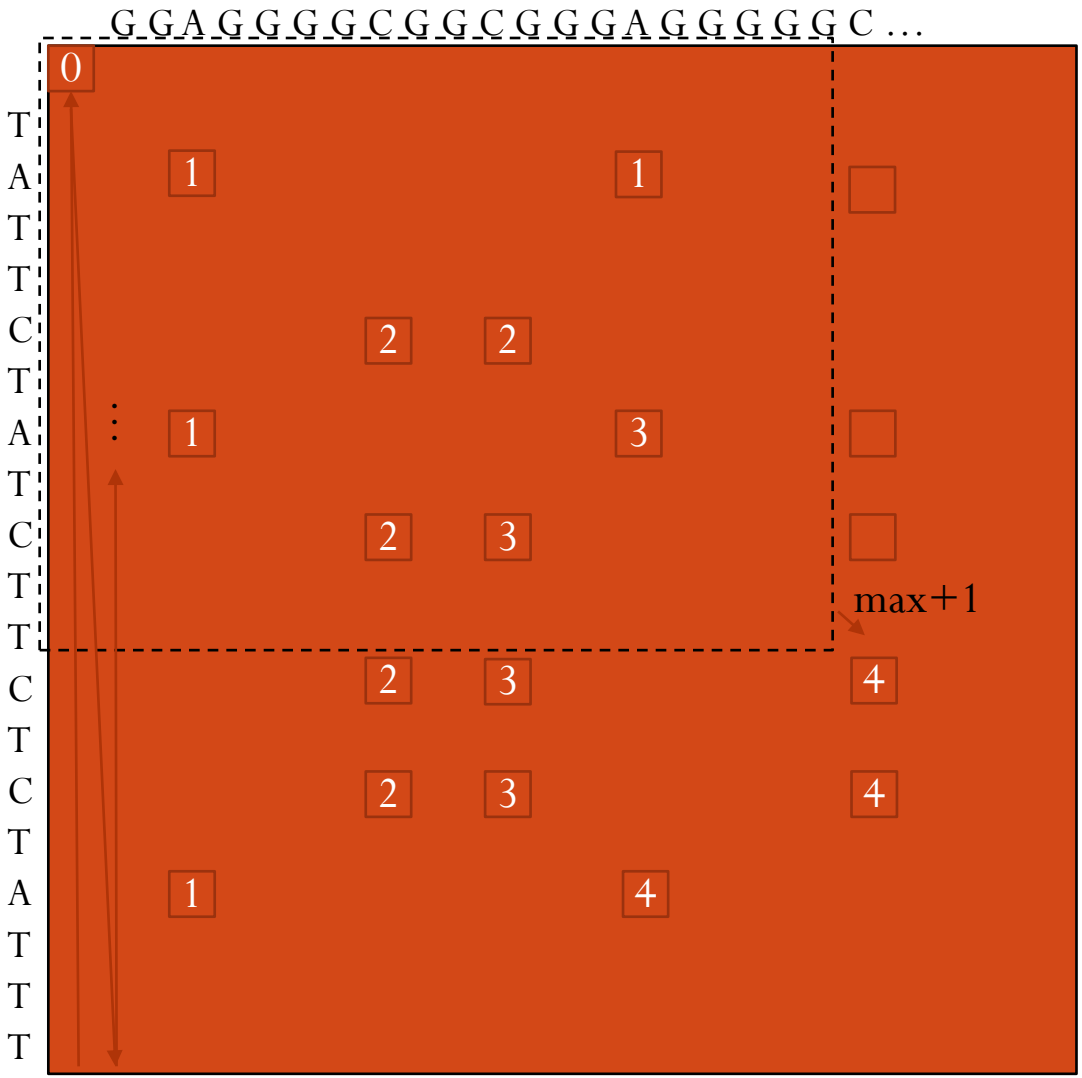
Longest Common Subsequence

Sparse dynamic programming

Longest Common Subsequence (LCS)

- LCS(A,B) is a longest sequence that can be obtained both by deleting characters from A and by deleting characters from B
- E.g. LCS(”AGCTAG”,”ACCACC”)=”ACA”
- Consider edit distance $D_{id}(A,B)$ with insertions and deletions only
- $D_{id}[i,j]=\min($
  - $D_{id}[i-1,j-1]+(A[i]=B[j]?0:\infty)$,
  - $D_{id}[i-1,j]+1$,
  - $D_{id}[i,j-1]+1$)
- $D_{id}[0,j]=j$, $D_{id}[i,0]=i$ as initialization, $D_{id}(A,B)= D_{id}[|A|,|B|]$ as finalization
- $|LCS(A,B)|=(|A|+|B|-D_{id}(A,B))/2$ (proof as exercise)

Sparse dynamic programming for LCS

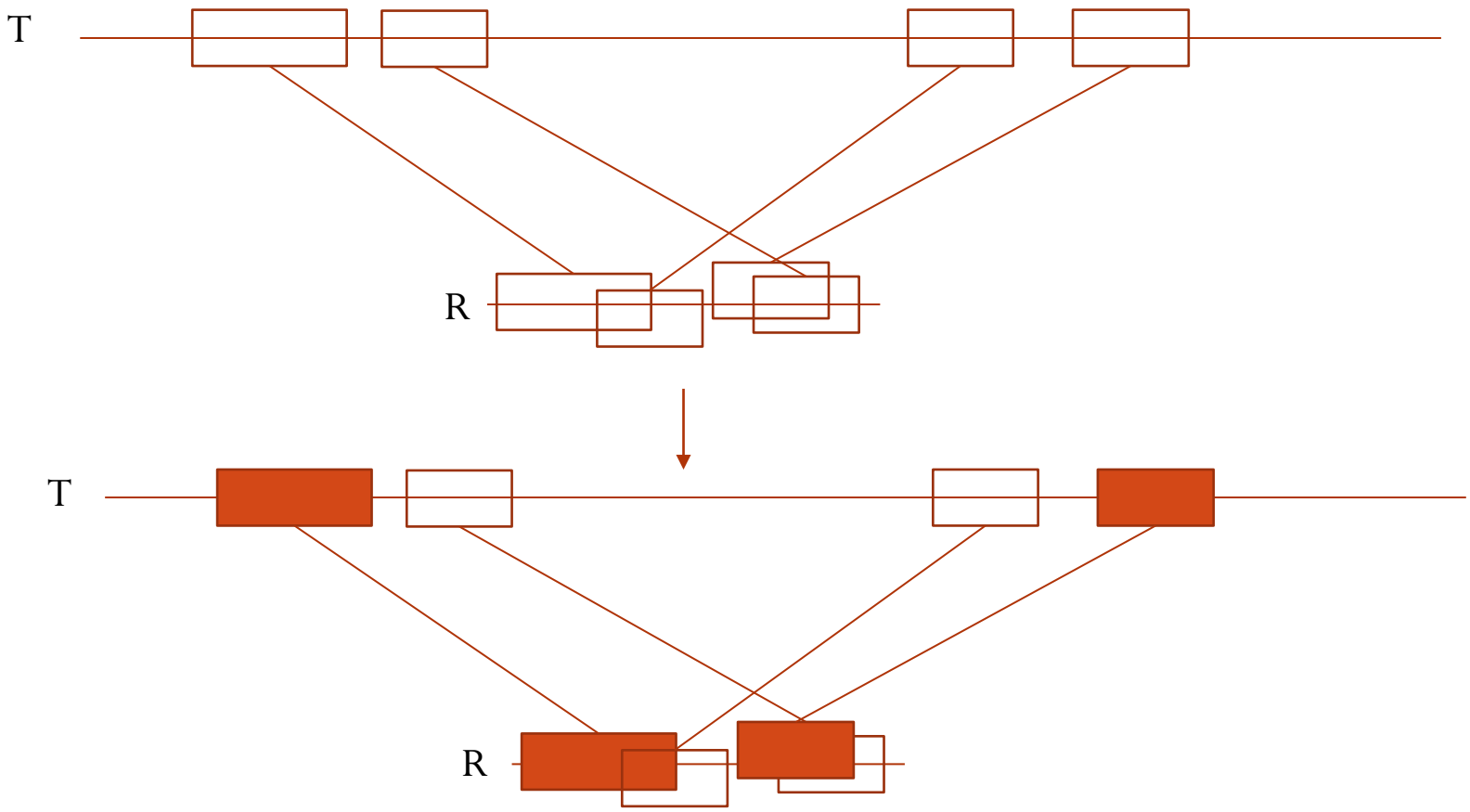


- Compute L(i,j)= |LCS(A[1..i],B[1..j])| for $(i,j) \in M$, $M = \{(0,0)\} \cup \{(i,j) | A[i] = B[j]\}$
- L(i,j)=$1 + \max_{\substack{(i',j') \in M, \\ i'<i, j'<j}} L(i',j')$
- We compute the values in reverse column order and add (key,value) pairs (i,L(i,j)) into a search tree T.
- L(i,j)=1+T.rangemax(0,i-1)
- A standard balanced binary search tree can be used for supporting the operations in O(log |A|) time.
- Running time O(|M|log |A|), assuming M given.

Co-linear chaining

Like LCS computation but the set of matches replaced by a set of alignment anchors

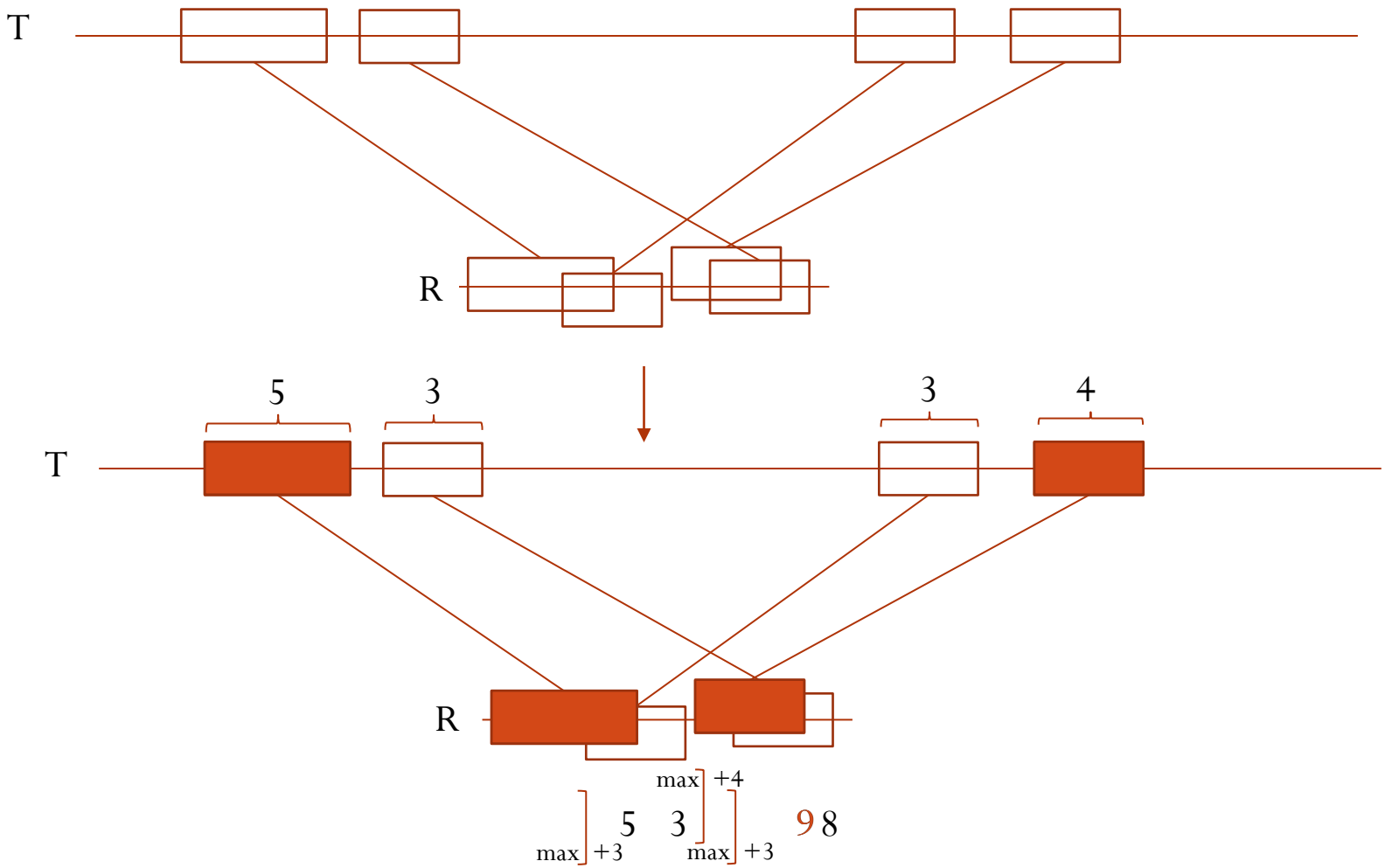
Co-linear chaining (CLC)



Co-linear chaining (CLC)

- **Alignment anchors** = e.g., set of N minimizer matches or MEMs between read R and reference T
- **Chain** = Subset of anchors forming a linear order in both R and T
- **Objective**: Score of chain, e.g, coverage of R
- **Key facts**:
  - Many variants: different ways of handling overlaps of anchors, assigning penalties to gaps
  - Many algorithms: Most variants can be solved in O(N log N) time or slightly worse running time
  - Can be applied to the alignment of both DNA (variant calling) and RNA long-reads (spliced alignment, transcript prediction)
- Course book gives an O(N log N) algorithm for allowing overlaps and for optimizing coverage of R
- Next slide illustrates a simplication of it: no overlaps allowed
  - It uses the same search tree as in the sparse dynamic programming LCS solution
  - In fact, if alignment anchors = set of matches, this algorithm solves LCS

Co-linear chaining (CLC)



Affine gap penalties

Sparse dynamic programming --> Gotoh's algorithm

Global alignment with affine gaps

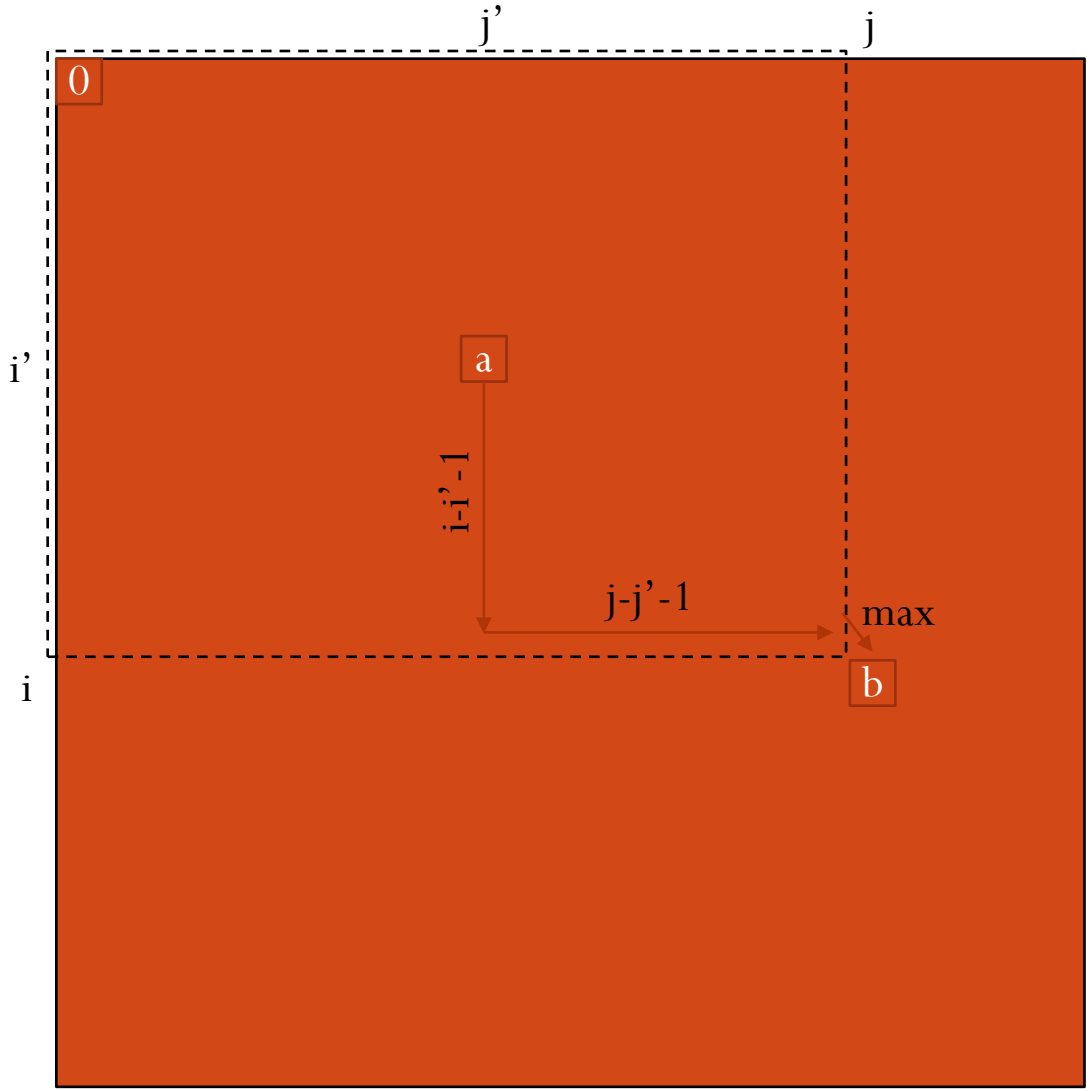
- Consider global alignment where a run of gaps of length g is penalized with $-\alpha + (g-1)\beta$, rather than with $-gd$.

```
ACA-GA-T-AA
ACAG--G-GAA
```

$$\underbrace{\qquad\qquad}_{g\,=\,6}$$

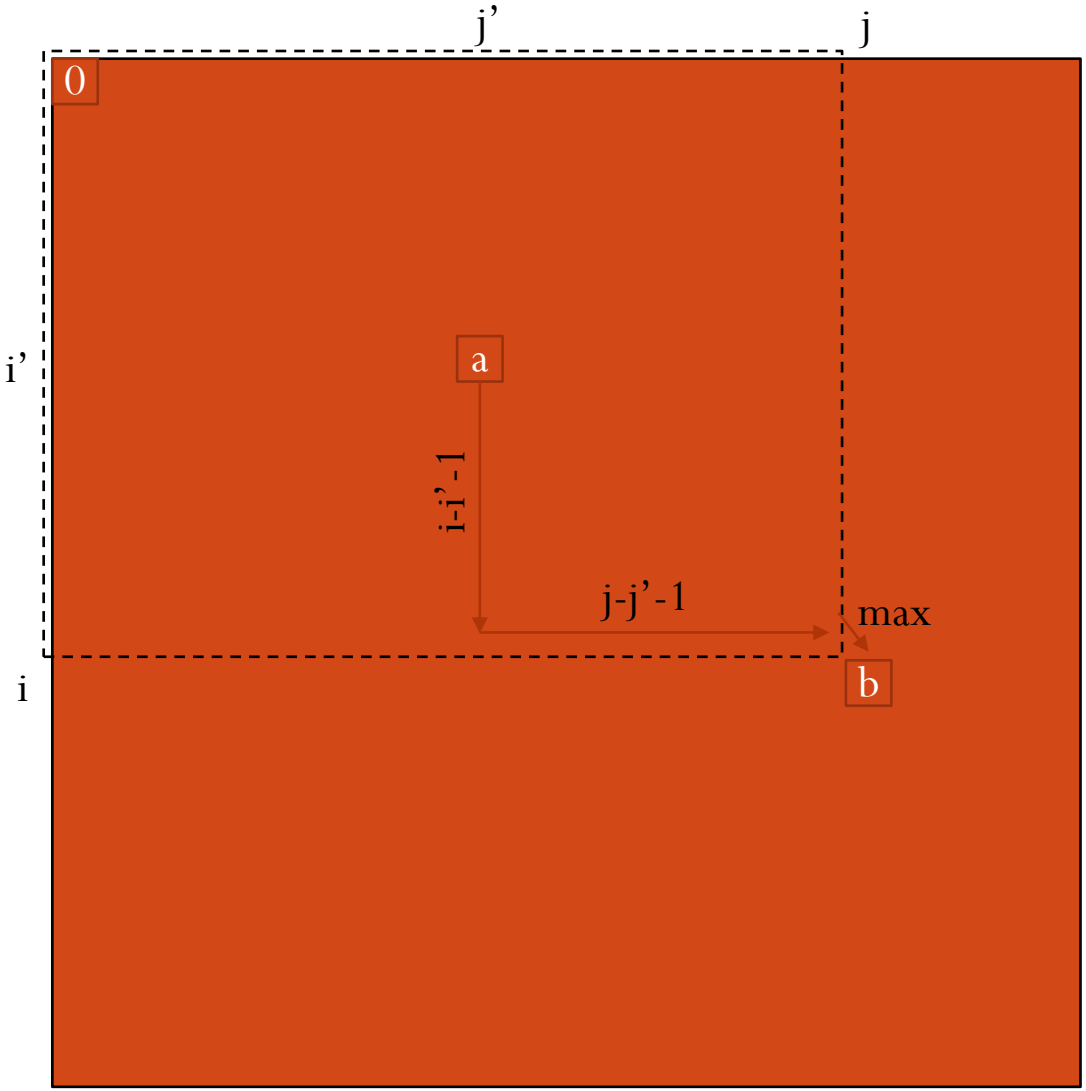
- This looks like LCS computation…

Global alignment with affine gaps



- $S[i,j] = s(A[i],B[\mathrm{j}]) + \max(\mathrm{S}[\mathrm{i}-1,\mathrm{j}-1], \max_{i'<i,j'<j,(i',j')\neq(i-1j-1)} S[i',j'] - \alpha - \beta((i-i'-1) + (j-j'-1)-1))$
- $S[i,j] = s(A[i],B[\mathrm{j}]) + \max(\mathrm{S}[\mathrm{i}-1,\mathrm{j}-1], -\alpha - \beta(i+j-3) + \max_{i'<i,j'<j,(i',j')\neq(i-1j-1)} S[i',j'] + \beta(i'+j'))$

Global alignment with affine gaps



- One can proceed as in the LCS algorithm adding (key,value) pairs $(\mathrm{i}', S[i', j'] + \beta(i' + j'))$ to a search tree, and querying the tree for max value in a range adding
$$-\alpha - \beta(i + j - 3)$$
- This yields an O(|A||B|log |A|) time algorithm.
- Now that we are not storing a sparse set, search tree becomes obsolete.
- Instead, we can keep some simple row and column maxima values to obtain O(|A||B|) time (see course book).

Gotoh's algorithm

- Even simpler than the one derived through LCS connection.
- Idea: Compute two tables, one storing optimal score for alignments ending with a match and the other for alignments ending with a gap.
- M[i,j]=S(A[1..i],B[i..j] | match)
- G[i,j]=S(A[1..i],B[i..j] | gap)
- M[i,j]=s(A[i],B[j])+max(M[i-1,j-1],G[i-1,j-1])
- G[i,j]=$\max M[i-1,j]-\alpha, M[i,j-1]-\alpha,$
  $G[i-1,j]-\beta, G[i,j-1]-\beta)$
- These can be evaluated in synchronization in O(|A||B|) time.