University of BRISTOL | Department of Computer Science

PROGRAMMING and ALGORITHMS II

INTRODUCTION TO

DYNAMIC PROGRAMMING II

Dr Tilo Burghardt

Unit Code COMS10001

Algorithm Design Paradigms

DIVIDE & CONQUER

Break down a problem into **independent** sub-problems of related type, solve them separately and combine the solutions.

GREEDY APPROACH

Use a sequence of **locally optimal** decisions incrementally to build up a solution.

DYNAMIC PROGRAMMING

Break down a problem into **overlapping** sub-problems of related type, build up solutions from larger and larger sub-solutions.

Recap: Knapsack Problem

University of BRISTOL | Department of Computer Science

Item i	Value v_i	Weight w_i
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

weight limit w

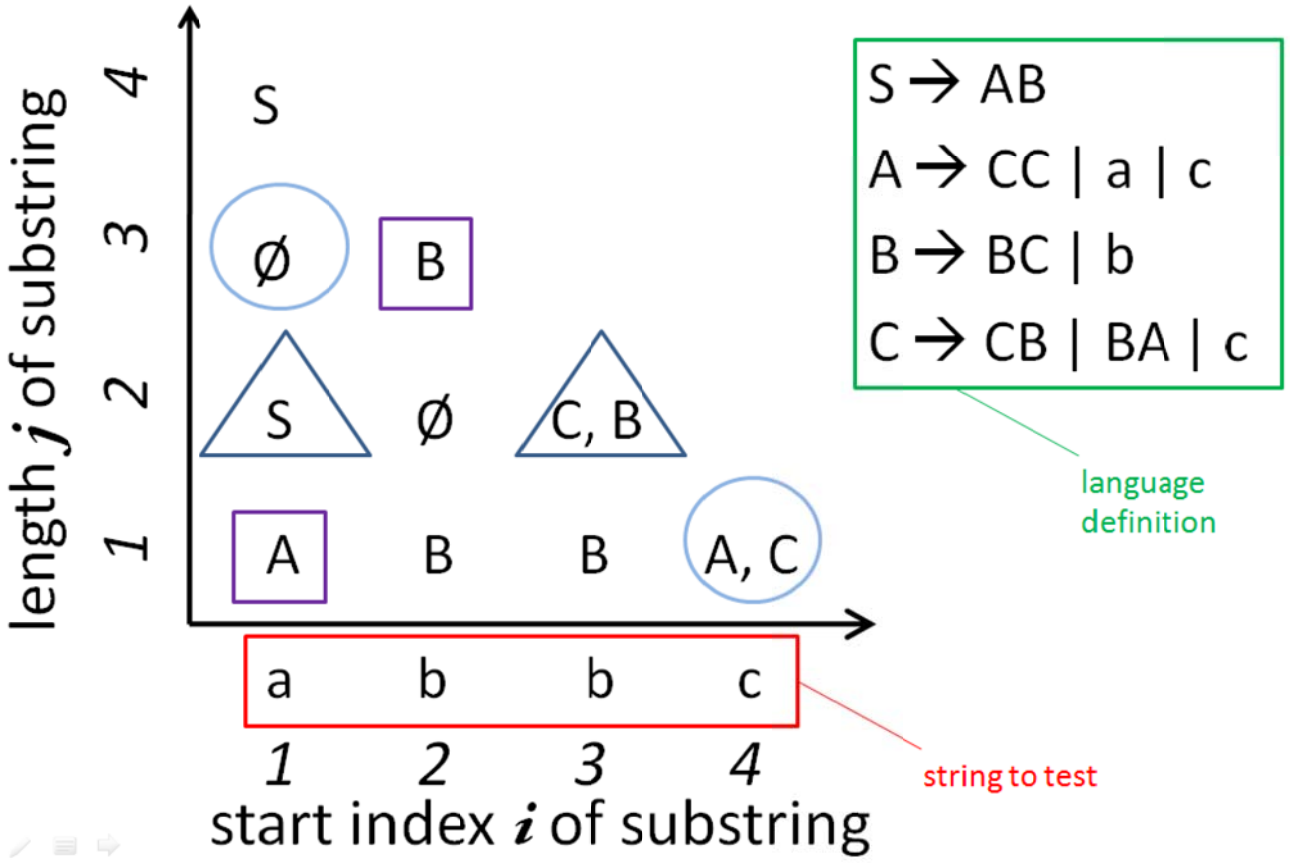
item subsets {1,..,i}	{ }	1	2	3	4	5	6	7	8	9	10	11
{ }	0	0	0	0	0	0	0	0	0	0	0	0
	0	1	1	1	1	1	1	1	1	1	1	1
	0	1	6	7	7	7	7	7	7	7	7	7
	0	1	6	7	7	18	19	24	25	25	25	25
	0	1	6	7	7	18	22	24	28	29	29	40
	0	1	6	7	7	18	22	28	29	34	34	40

$$best(i,w) = \begin{cases} 0 & if\,(i=0) \\ best(i-1,w) & if\,(w_i > w) \\ \max[best(i-1,w), v_i + best(i-1, w-w_i)] & otherwise \end{cases}$$

Recap: CYK Algorithm

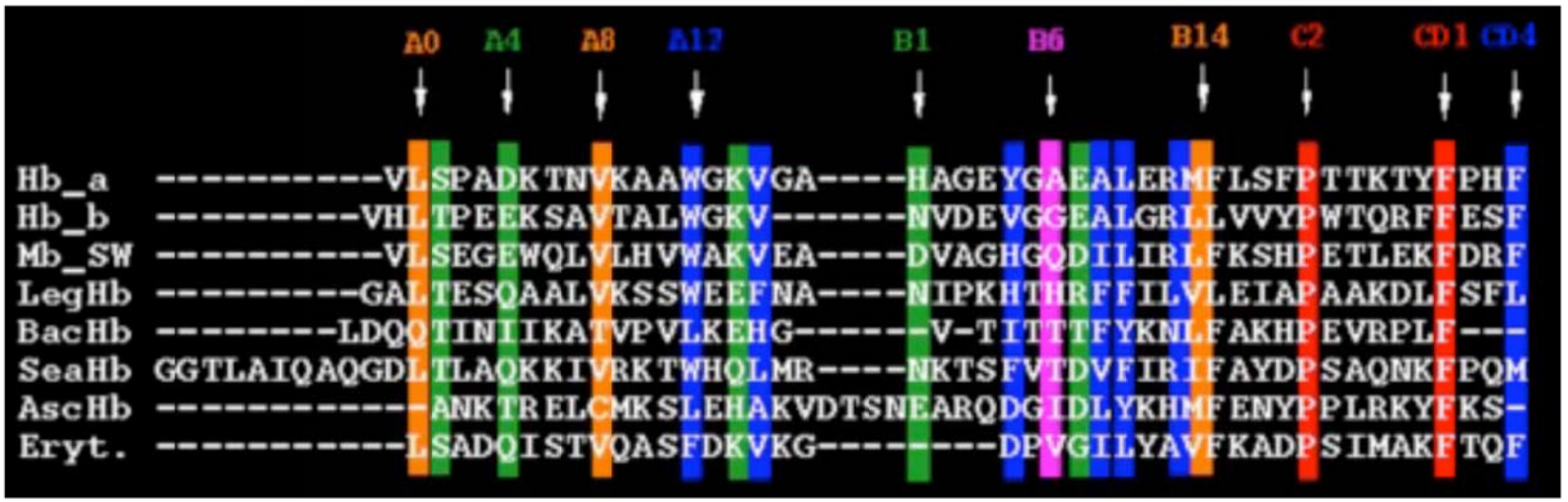
$$table(i,1) = \{R \mid \exists_{(R \to w[i]) \in P}\}$$

$$table(i,j) = \bigcup_{x=1}^{x=j-1} \{R \mid \exists_{(R \to AB) \in P} : A \in table(i,x) \wedge B \in table(i+x, j-x)\}$$



Motivation: DNA Similarity

- In 1984 R Doolittle et al. found similarities between a cancer-causing gene and a normal growth factor gene searching a database
- Today, finding sequence similarities with genes of known function is a common approach to infer the function of newly sequenced genes



The LCS Problem

Given: two strings $x[1...m]$ and $y[1...n]$

Goal: find a longest subsequence of characters (not necessarily continuous, but in correct order) common to both strings

Example: $x = A\ B\ C\ B\ D\ A\ B$

$y = B\ D\ C\ A\ B\ A$

$\rightarrow LCS(x, y) = B\ C\ B\ A$

Brute Force Approach

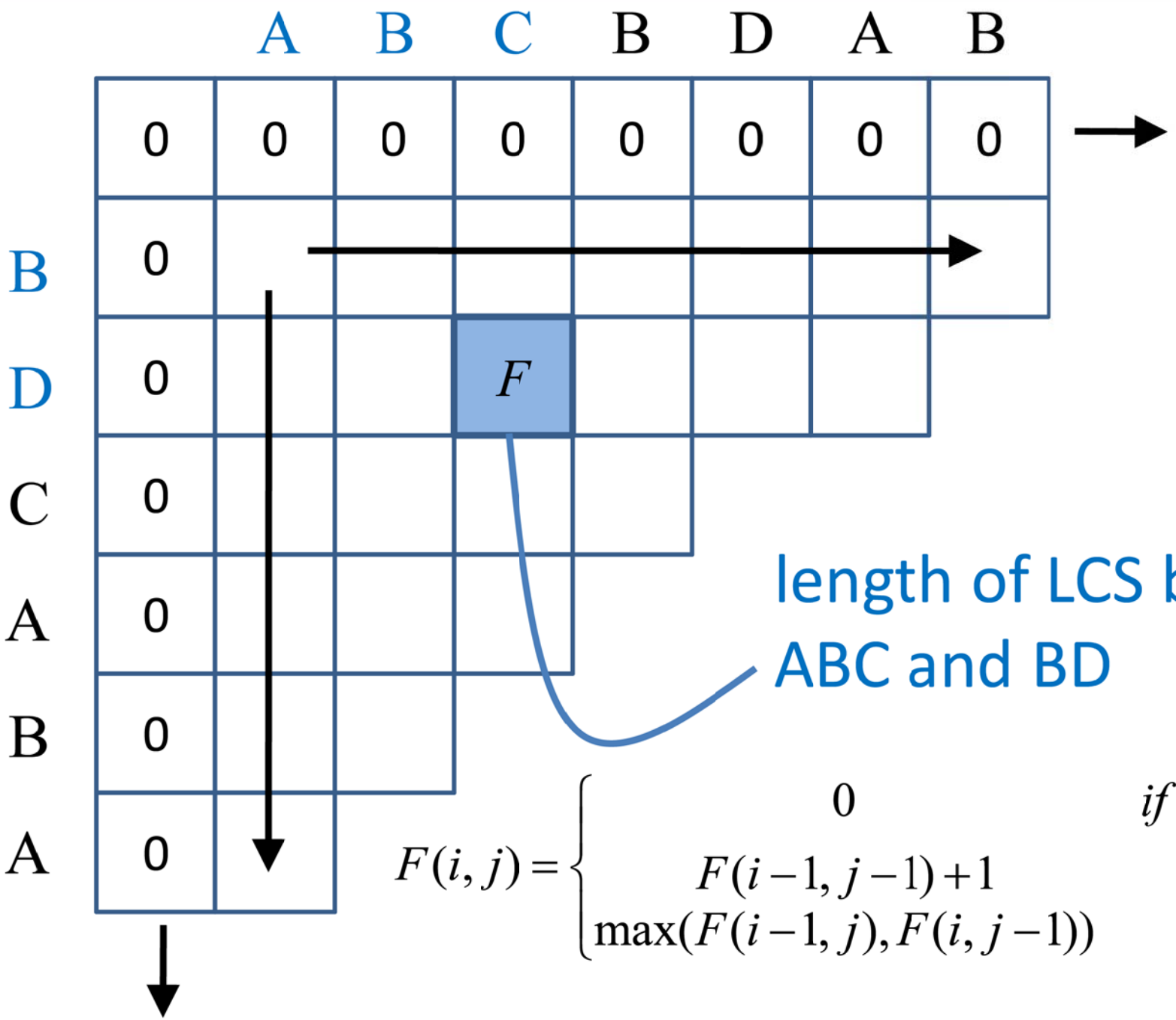
- generate every subsequence of $x[1...m]$ and check if it is also a subsequence of $y[1...n]$

→ a single check will take up to n steps, this will give $O(n)$ for checking a subsequence

→ there are 2^m sub-sequences to check

→ Worst-case runtime in $O(n2^m)$

Bottom-up Approach

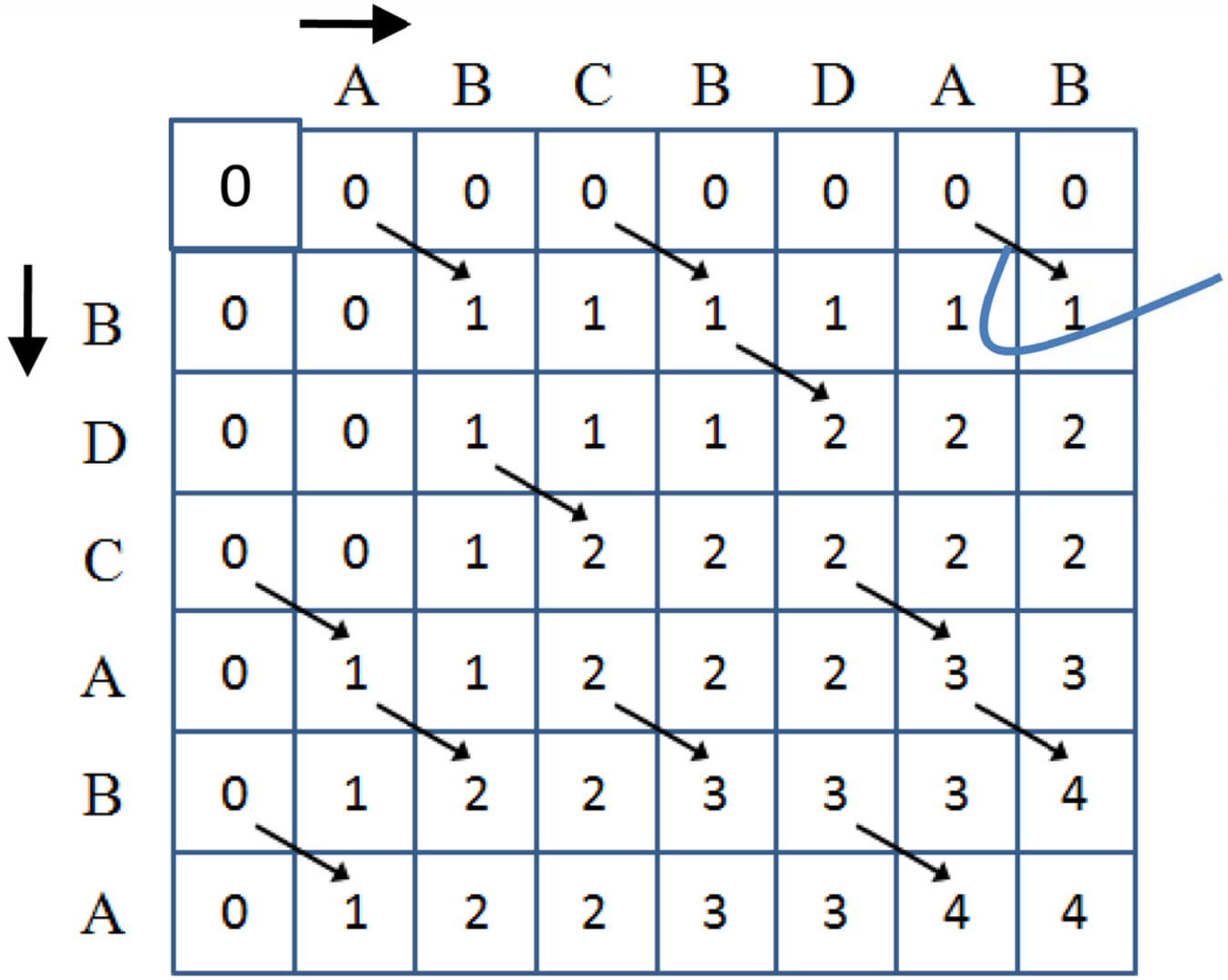


length of LCS between ABC and BD

$$F(i,j) = \begin{cases} 0 & if\ (i=0)\ or\ (j=0) \\ F(i-1,j-1)+1 & if\ x[i]=y[j] \\ \max(F(i-1,j),F(i,j-1)) & otherwise \end{cases}$$

Computing Sub-solutions

University of BRISTOL | Department of Computer Science



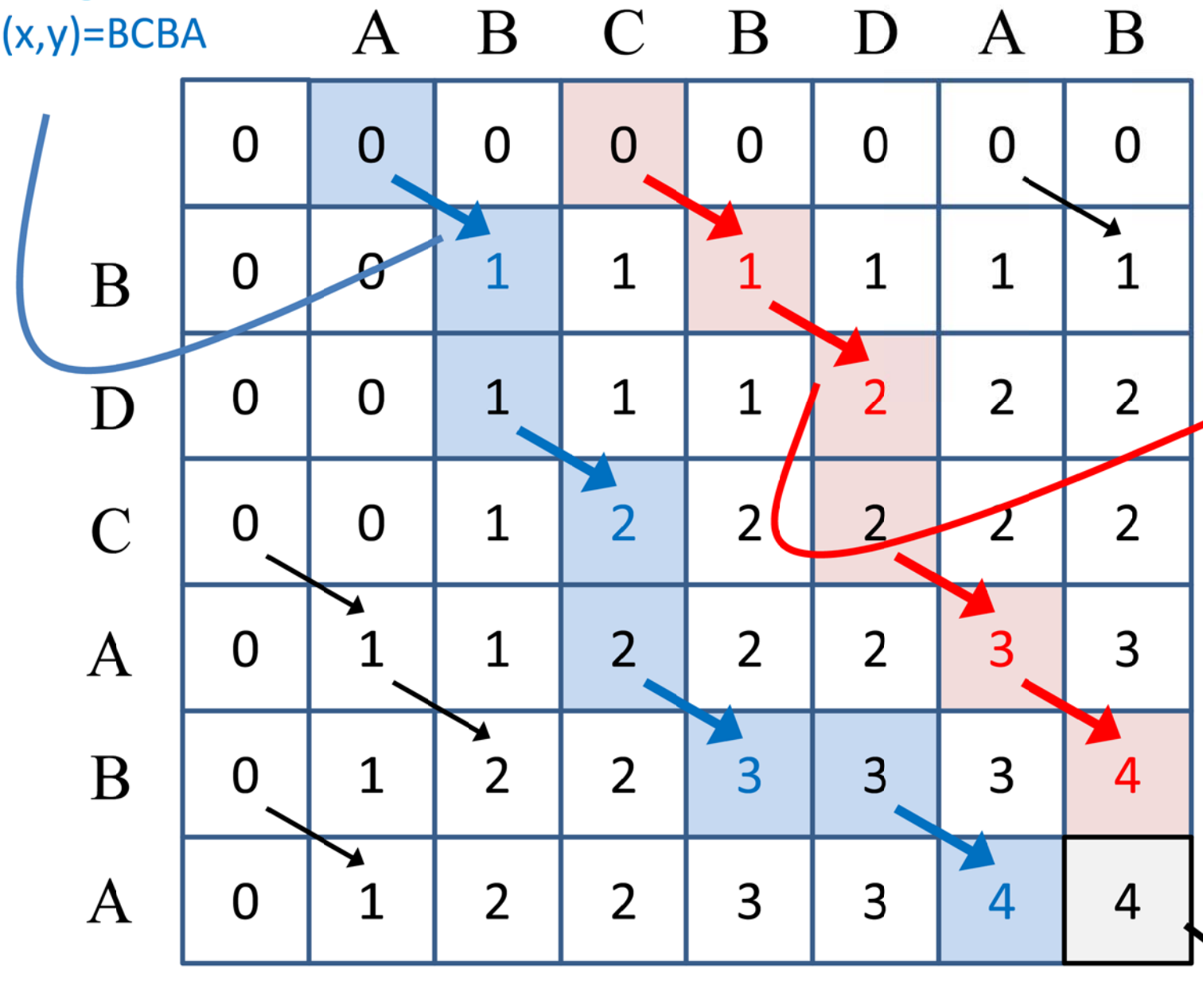
		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

diagonal arrows indicate character match (case 2 expansion of dynamic programming table)

$$F(i,j) = \begin{cases} 0 & if\ (i=0)\ or\ (j=0) \\ F(i-1,j-1)+1 & if\ x[i]=y[j] \\ \max(F(i-1,j),F(i,j-1)) & otherwise \end{cases}$$

Tracing LCS Solutions

Matching alternative 1:
$LCS_1(x,y)$=BCBA

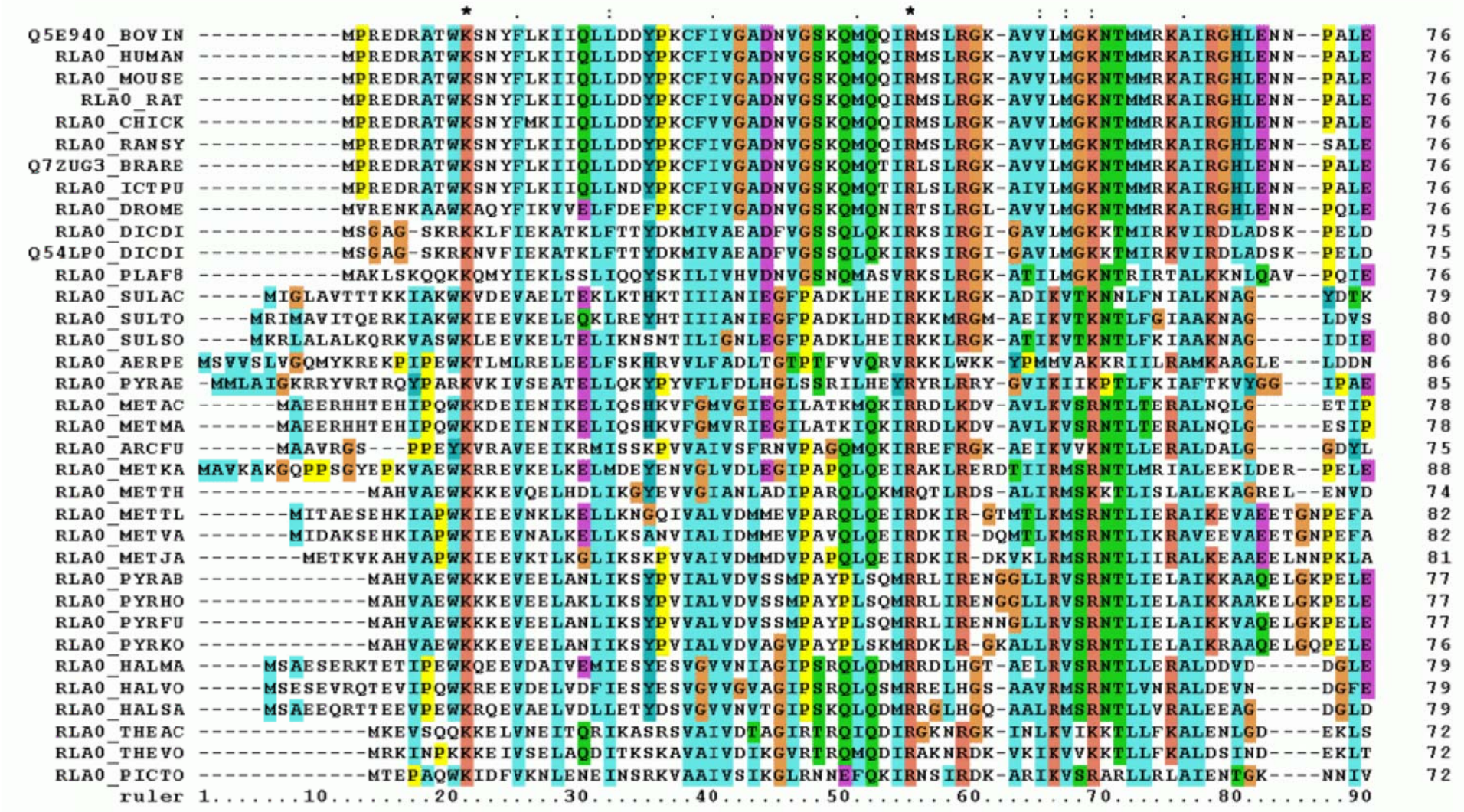


Matching alternative 2:
$LCS_2(x,y)$=BDAB

Overall, LCS has 4 characters in any case.

Sequence Alignment

University of BRISTOL | Department of Computer Science

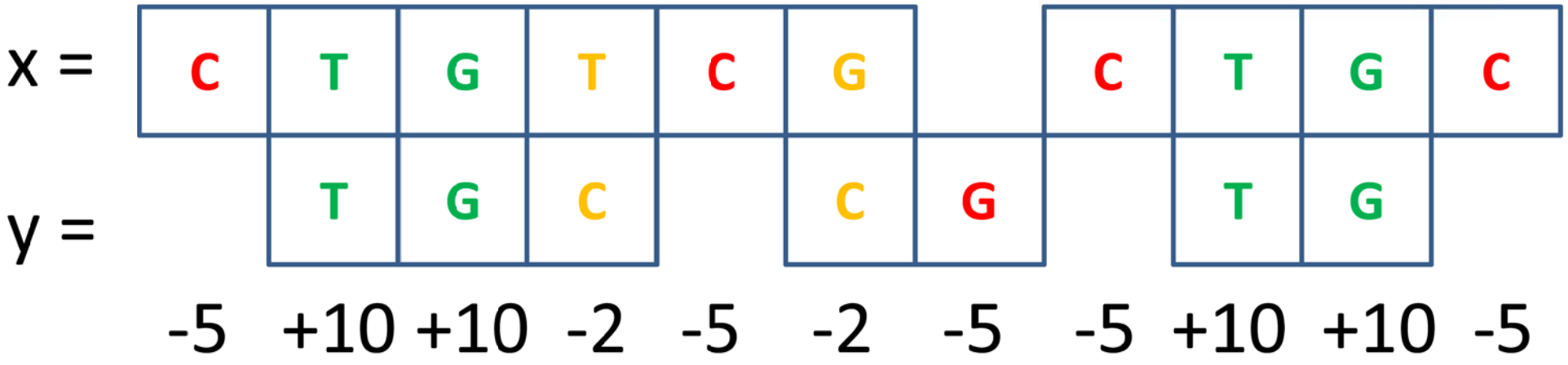


First 90 positions of a protein multiple sequence alignment of instances of the acidic ribosomal protein P0 (L10E) from several organisms.

Alignment Scoring

University of BRISTOL | Department of Computer Science

- Example Scoring: **MATCH = +10**
  MISMATCH = -2
  GAP = -5
- Example Alignment:



Total Score = 11

Needleman-Wunsch Algorithm

Task: find the highest scoring overall alignment

Idea: extend the concept of LCS by allowing for different scores for matches and mismatches as well as a gap penalties:

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + match\,score(x_i, y_j) \\ F(i-1,j) - gap\ penalty \\ F(i,j-1) - gap\ penalty \end{cases}$$

Recurrence Relations

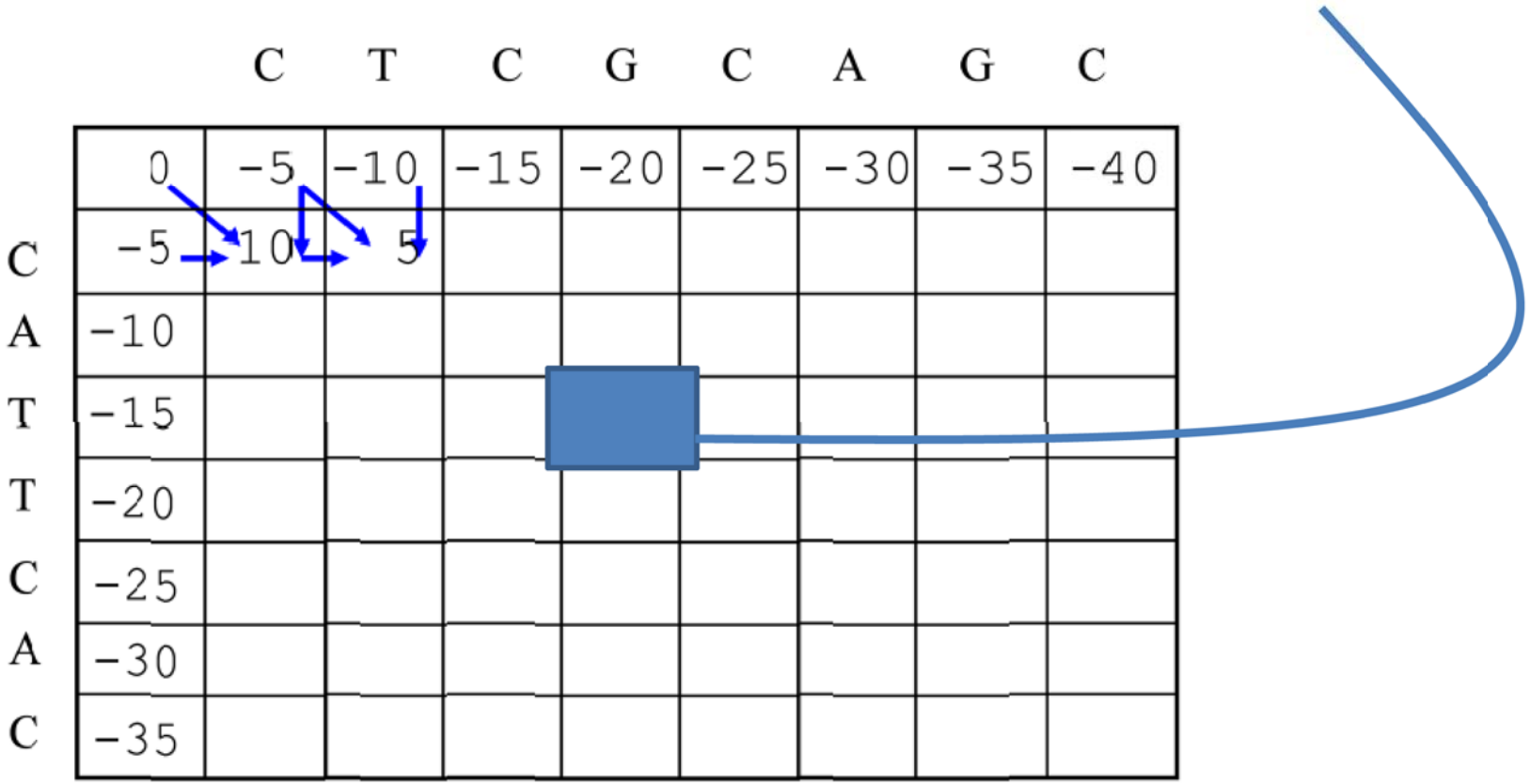
$$F(i,j) = \max \begin{cases} F(i-1,j-1) + score(x_i, y_j) \\ F(i-1,j) + 0 \\ F(i,j-1) + 0 \end{cases}$$

$$F(i,j) = \max \begin{cases} F(i-1,j-1) + match\,score(x_i, y_j) \\ F(i-1,j) - gap\ penalty \\ F(i,j-1) - gap\ penalty \end{cases}$$

Alignment Scoring Table

University of BRISTOL | Department of Computer Science

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + match\,score(x_i, y_j) \\ F(i-1, j) - gap\ penalty \\ F(i, j-1) - gap\ penalty \end{cases}$$



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

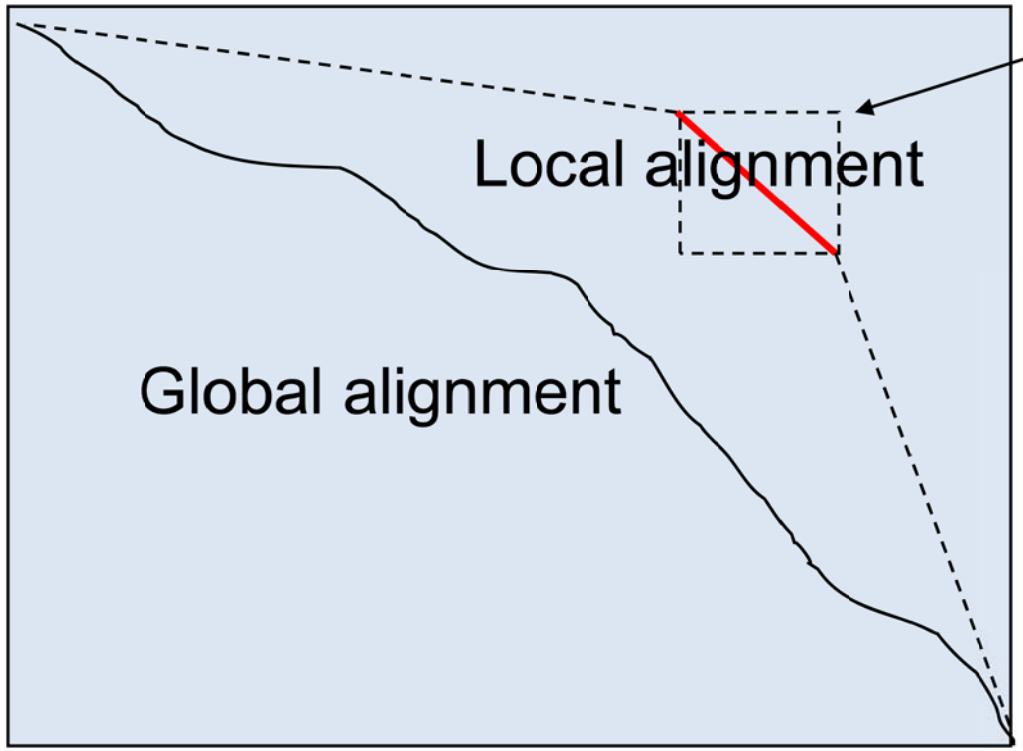
Alignment Scoring Example

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + match\,score(x_i, y_j) \\ F(i-1, j) - gap\ penalty \\ F(i, j-1) - gap\ penalty \end{cases}$$

	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

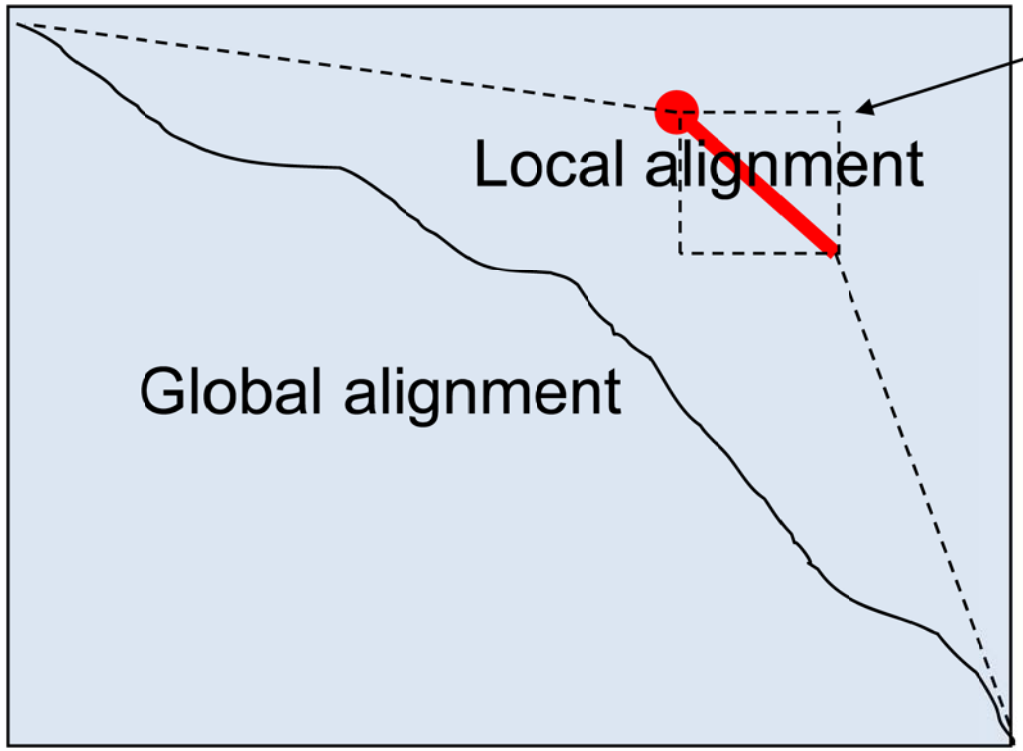
+10 for match, -2 for mismatch, -5 for space

Global vs Local Alignment



Computation of a regional Global Alignment to get Local Alignment

Global vs Local Alignment



Idea: suppress penalty from outside window

→ limit reach of penalties

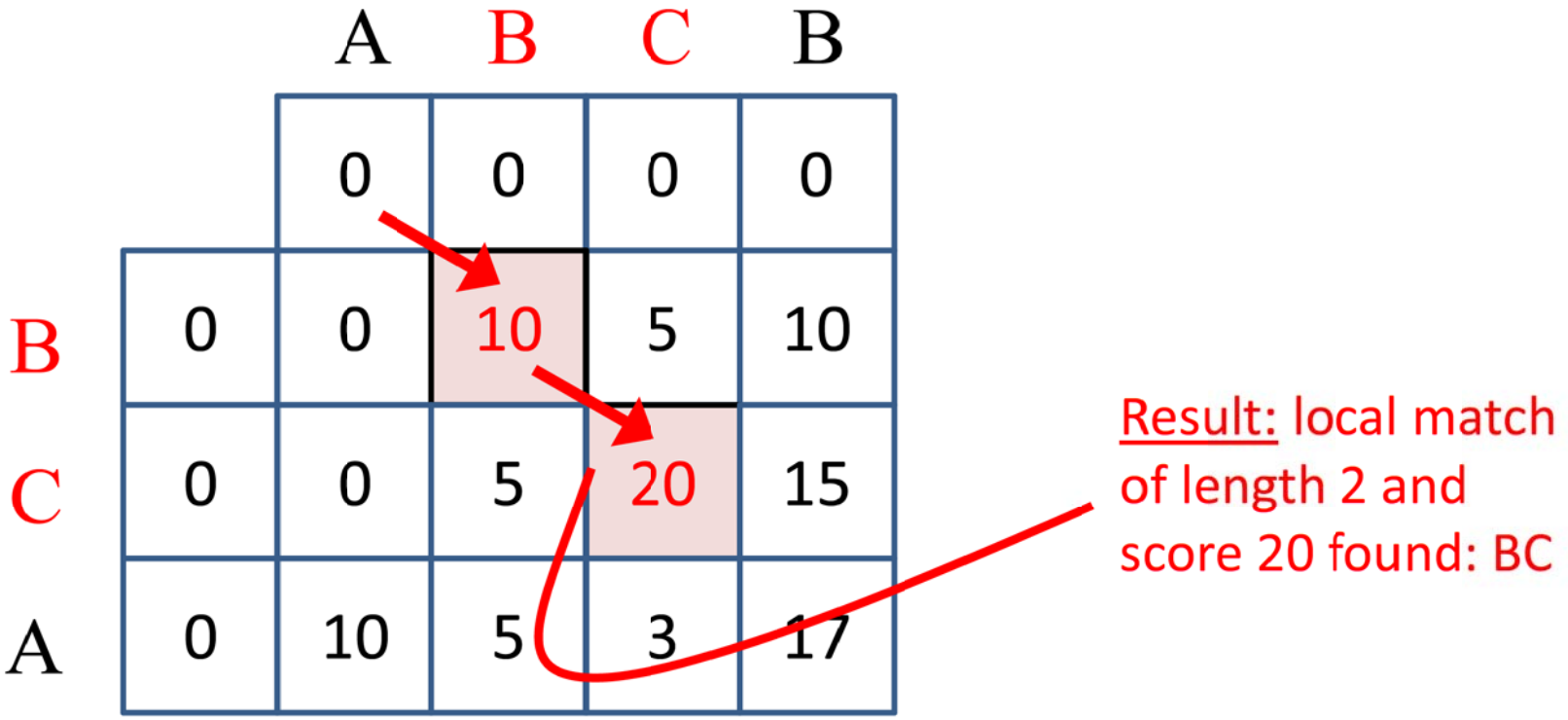
Waterman-Smith Algorithm

Key Idea: we extend the algorithm by Needleman-Wunsch via enforcing a positive matrix:

$$F(i,j) = \max \begin{cases} F(i-1, j-1) + match\ score(x_i, y_j) \\ F(i-1, j) - gap\ penalty \\ F(i, j-1) - gap\ penalty \\ 0 \end{cases}$$

Local Alignment Example

University of BRISTOL | Department of Computer Science



$$F(i,j) = \max \begin{cases} F(i-1, j-1) + match\,score(x_i, y_j) \\ F(i-1, j) - gap\ penalty \\ F(i, j-1) - gap\ penalty \\ 0 \end{cases}$$

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

Dynamic Programming Stages

Step1: Determine Structure

Characterize the structure of an optimal solution by showing that it can be decomposed into optimal sub-problems.

Step2: Find Recurrence Relation

Recursively define the value of an optimal solution by expressing it in terms of optimal solutions for smaller problems.

Step 3: Bottom-up Computation

Compute the value of an optimal solution in a bottom-up fashion by using a table structure.

Step 4: Construction of Optimal Solution

Construct an optimal solution from computed information.

Algorithm Design Paradigms

DIVIDE & CONQUER

Break down a problem into **independent** sub-problems of related type, solve them separately and combine the solutions.

GREEDY APPROACH

Use a sequence of **locally optimal** decisions incrementally to build up a solution.

DYNAMIC PROGRAMMING

Break down a problem into **overlapping** sub-problems of related type, build up solutions from larger and larger sub-solutions.