Sequence comparison

more Dynamic Programming

I. Hamming distance

- defined for sequences of the same length
- Ham(S,T)=number of mismatches

```
GAGGCTCACCTGACTTCCAGGCGCGAGATGCCTCT
 ||  |||||   |||   |   ||   |||   ||||
TAGCCTCACAAGACCGCTTGGGTCGATTTGCCGAC
```

- can be computed in $O(n)$

II. LCS: Longest Common Subsequence

- longest *subsequence* common to the two strings
- LCS(ACACGA,CAAGTAGAG)=4

II. LCS: Longest Common Subsequence

- longest *subsequence* common to the two strings
- LCS(ACACGA,CAAGTAGAG)=4

ACACGA

CAAGTAGAG

II. LCS: Longest Common Subsequence

- longest *subsequence* common to the two strings
- LCS(ACACGA,CAAGTAGAG)=4

ACACGA

CAAGTAGAG

II. LCS: Longest Common Subsequence

- longest *subsequence* common to the two strings
- LCS(ACACGA,CAAGTAGAG)=4

AC**A**C**GA**

CAA**G**T**A**GAG

- d(S,T)=|S|+|T|-2·LCS(S,T)
- d(ACACGA,CAAGTAGAG)=7
- d(S,T): minimum number of letter insertions/deletions needed to transform one sequence to the other

II. LCS: Longest Common Subsequence

- longest *subsequence* common to the two strings
- LCS(ACACGA,CAAGTAGAG)=4

```
ACA-CG-A---
 ||   | |
-CAA-GTAGAG
```

- d(S,T)=|S|+|T|-2·LCS(S,T)
- d(ACACGA,CAAGTAGAG)=7
- d(S,T): minimum number of letter insertions/deletions needed to transform one sequence to the other

III. Levenshtein (edit) distance

- minimum number of edit operations (letter substitutions, insertions, deletions) required to transform S into T
- edit(ACACGA,CAAGTAGAG)=6

```
ACACG-A---
  ||  |  |
-CAAGTAGAG
```

Bioinformatics: “CIGAR strings”

part of SAM format

```
RefPos:    1  2  3  4  5  6  7     8  9 10 11 12 13 14 15 16 17 18 19
Reference: C  C  A  T  A  C  T     G  A  A  C  T  G  A  C  T  A  A  C
Read :                  A  C  T  A  G  A  A     T  G  G  C  T
```

POS: 5

CIGAR: 3M1I3M1D5M

IV. Sequence alignment (weighted edit distance)

- Given two sequences RDISLVKNAGI and RNILVSDAKNVGI

```
R D I S L V - - - K N A G I
|   |   | |       | |   | |
R N I - L V S D A K N V G I
```

- 3 types of columns corresponding to 3 elementary evolutionary events
  - matches
  - substitution (mismatch)
  - insertion, deletion (*indel*)
- Assign a *score* (positive or negative) to each "event".
- *Alignment score* = sum of scores over all columns.
- *Optimal alignment* = one that maximizes the score

Sequence alignment: scoring

- scoring function:

Mismatch :		Match :		Indel :
		G, N	: 6	
DN	: 1	R, K	: 5	−5
AV, LD	: 0	A, I, L, S, V	: 4	

```
R D I S L V - - - K N A G I
|   |   | |       | |   | |
R N I - L V S D A K N V G I
```

Score=19

```
R D I - - S L V K N A - - - G I
|   |     |         |         | |
R N I L V S - - - D A K N V G I
```

Score=−11

```
R D I - - S L V K N A G I
|   |     |     | |   | |
R N I L V S D A K N V G I
```

Score=25

Sequence alignment: scoring

- BLOSUM62 matrix for protein sequences

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
C	9																				C
S	-1	4																			S
T	-1	1	5																		T
P	-3	-1	-1	7																	P
A	0	1	0	-1	4																A
G	-3	0	-2	-2	0	6															G
N	-3	1	0	-2	-2	0	6														N
D	-3	0	-1	-1	-2	-1	1	6													D
E	-4	0	-1	-1	-1	-2	0	2	5												E
Q	-3	0	-1	-1	-1	-2	0	0	2	5											Q
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8										H
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5									R
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5								K
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5							M
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4						I
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4					L
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4				V
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6			F
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		Y
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	W
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Example of an alignment

Sequences producing significant alignments: Score (bits) E value

gnl|Pfam|pfam00155 aminotran_1, Aminotransferases class-I 338 4e-94

gnl|Pfam|pfam00155, aminotran_1, Aminotransferases class-I

Add query to multiple alignment, display up to 10 sequences most similar to the query

```
            Length = 428
            Score =  338 bits (857), Expect = 4e-94

Query:  23   KSTWFSEVQMGPPDAILGVTEAFKKDTNPKKIN----LGAGAYRDDNTQPFVLPSVREAE  78
Sbjct:  1    LSRNATFNSHGQDSSYFLGWQEYEKNPYHEVHNTNGIIQMGLAENQLCFDLLESWLAKNP  60

Query:  79   KRVVSRS-------LDKEYATIIGIPEFYNKAIELALGKGSKRLAAKHNVTAQSISGTGA  131
Sbjct:  61   EAAAFKKNGESIFAELALFQDYHGLPAFKKAMVDFMAEIRGNKVTFDPNHLVLTAGATSA  120

Query:  132  LRIGAAFLAKFWQGNREIYIPSPSWGNHV-AIFEHAGLPVNRYRYYDKDTCALDFGGLIE  190
Sbjct:  121  NETFIFCLADPGE---AVLIPTPYYPGFDRDLKWRTGVEIVPIHCTSSNGFQITETALEE  177

Query:  191  DLKKIPE---KSIVLLHACAHNPTGVDPTLEQWREISALVKKRNLYPFIDMAYQGFATGD  247
Sbjct:  178  AYQEAEKRNLRVKGVLVTNPSNPLGTTMTRNELYLLLSFVEDKGIHLISDEIYSGTAFSS  237

Query:  248  IDRDAQAVRTFEAD---------GHDFCLAQSFAKNMGLYGERAGAFTVLCSDEEEAARV  298
Sbjct:  238  P--SFISVMEVLKDRNCDENSEVWQRVHVVYSLSKDLGLPGFRVGAIYSNDDMVVAAATK  295

Query:  299  M-------SQVKILIRGLYSNP---PVHGARIAAEILNNEDLRAQWLKDVKLMADRIIDV  348
Sbjct:  296  MSSFGLVSSQTQHLLSAMLSDKKLTKNYIAENHKRLKQRQKKLVSGLQKSG-ISCLNGNA  354

Query:  349  RTKLKDNLIKLGSSQNWDHIVNQIGMFCFTGLKPEQVQK-LIKDHSVYLTNDGRVSMAGV  407
Sbjct:  355  GLFCWVDMRHLLR----SNTFEAEMELWKKIVYEVHLNISPGSSCHCTEPGWFRVCFANL  410

Query:  408  TSKNVEYLAESIHKVTK  424
Sbjct:  411  PERTLDLAMQRLKAFVG  427
```

Longest Common Subsequence

- consider score match:1, indel: 0, mismatch: -1

```
-AGGCTCACCTGACT-CCAGGC-CGA--TGCC---
 || ||||| ||| | || ||| ||||
TAG-CTCAC--GAC-GC--GG-TCGATTTGCCGAC
```

- optimal alignment ~ *longest common subsequence* (LCS)
- Score(S,T)=LCS(S,T)

Levenshtein distance

- consider score match:0, indel: -1, mismatch: -1

```
-AGGCTCACCTGACTCCAGGCCGA--TGCC---
 ||  |||||   |||  |   ||  |||   ||||
TAG-CTCAC--GACGC--GGTCGATTTGCCGAC
```

- optimal alignment ~ *Levenshtein (edit) distance*
- edit(S,T) = -Score(S,T)

Computing optimal alignment

- assume d is indel penalty (usually $d < 0$), $s(x, y)$ score of aligning letters x and y (match or mismatch), $S[1..n]$ and $T[1..m]$ are input strings
- *Idea*: compute $\text{Score}(i, j)$: maximum score between $S[1..i]$ and $T[1..j]$

Computing optimal alignment

- assume d is indel penalty (usually $d < 0$), $s(x, y)$ score of aligning letters x and y (match or mismatch), $S[1..n]$ and $T[1..m]$ are input strings
- *Idea*: compute $\text{Score}(i, j)$: maximum score between $S[1..i]$ and $T[1..j]$

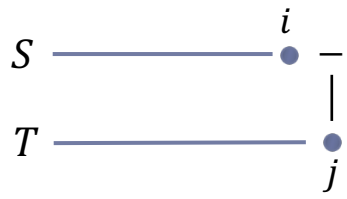
Case 1

$\text{Score}(i, j) = \text{Score}(i-1, j-1) + s(S[i], T[j])$

Case 2

$\text{Score}(i, j) = \text{Score}(i-1, j) + d$

Case 3



$\text{Score}(i, j) = \text{Score}(i, j-1) + d$

Computing optimal alignment

- assume d is indel penalty (usually $d < 0$), $s(x, y)$ score of aligning letters x and y (match or mismatch), $S[1..n]$ and $T[1..m]$ are input strings
- *Idea*: compute $\text{Score}(i, j)$: maximum score between $S[1..i]$ and $T[1..j]$

$$\text{Score}(i,j) = \max \begin{cases} \text{Score}(i-1, j-1) + s(S[i], T[j]) \\ \text{Score}(i-1, j) + d \\ \text{Score}(i, j-1) + d \end{cases}$$

- initialization: $\text{Score}(0,0) = 0, \text{Score}(0, j) = jd, \text{Score}(i, 0) = id$
- resulting score: $\text{Score}(n, m)$

Computing optimal alignment

- assume d is indel penalty (usually $d < 0$), $s(x, y)$ score of aligning letters x and y (match or mismatch), $S[1..n]$ and $T[1..m]$ are input strings
- *Idea*: compute $\text{Score}(i, j)$: maximum score between $S[1..i]$ and $T[1..j]$

$$\text{Score}(i,j) = \max \begin{cases} \text{Score}(i-1, j-1) + s(S[i], T[j]) \\ \text{Score}(i-1, j) + d \\ \text{Score}(i, j-1) + d \end{cases}$$

- initialization: $\text{Score}(0,0) = 0, \text{Score}(0,j) = jd, \text{Score}(i,0) = id$
- resulting score: $\text{Score}(n, m)$
- *Implementation*: **Dynamic Programming**!

Example

Scoring function: $s(x, x) = 2, s(x, y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	1	2	3	4	5	6	7	8
A	1								
C	2								
T	3								
G	4								
T	5								
A	6								
T	7								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	1	2	3	4	5	6	7	8
0	0	-2	-4	-6	-8	-10	-12	-14	-16
A 1	-2								
C 2	-4								
T 3	-6								
G 4	-8								
T 5	-10								
A 6	-12								
T 7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	1	2	3	4	5	6	7	8
0	0	-2	-4	-6	-8	-10	-12	-14	-16
A 1	-2	2							
C 2	-4								
T 3	-6								
G 4	-8								
T 5	-10								
A 6	-12								
T 7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	1	2	3	4	5	6	7	8
0	0	-2	-4	-6	-8	-10	-12	-14	-16
A 1	-2	2	0						
C 2	-4								
T 3	-6								
G 4	-8								
T 5	-10								
A 6	-12								
T 7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	1	2	3	4	5	6	7	8
0	0	-2	-4	-6	-8	-10	-12	-14	-16
A 1	-2	2	0	-2					
C 2	-4								
T 3	-6								
G 4	-8								
T 5	-10								
A 6	-12								
T 7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	1	2	3	4	5	6	7	8
0	0	-2	-4	-6	-8	-10	-12	-14	-16
A 1	-2	2	0	-2	-4				
C 2	-4								
T 3	-6								
G 4	-8								
T 5	-10								
A 6	-12								
T 7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
A	1	-2	2	0	-2	-4	-6	-8	-10	-12
C	2	-4								
T	3	-6								
G	4	-8								
T	5	-10								
A	6	-12								
T	7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
A	1	-2	2	0	-2	-4	-6	-8	-10	-12
C	2	-4	0	4	2	0	-2	-4	-6	-8
T	3	-6								
G	4	-8								
T	5	-10								
A	6	-12								
T	7	-14								

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
A	1	-2	2	0	-2	-4	-6	-8	-10	-12
C	2	-4	0	4	2	0	-2	-4	-6	-8
T	3	-6	-2	2	3	1	-1	0	-2	-4
G	4	-8	-4	0	4	5	3	1	-1	-3
T	5	-10	-6	-2	2	3	4	5	3	1
A	6	-12	-8	-4	0	1	2	3	7	5
T	7	-14	-10	-6	-2	-1	0	4	5	9

Example

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
A	1	-2	2	0	-2	-4	-6	-8	-10	-12
C	2	-4	0	4	2	0	-2	-4	-6	-8
T	3	-6	-2	2	3	1	-1	0	-2	-4
G	4	-8	-4	0	4	5	3	1	-1	-3
T	5	-10	-6	-2	2	3	4	5	3	1
A	6	-12	-8	-4	0	1	2	3	7	5
T	7	-14	-10	-6	-2	-1	0	4	5	9

Score(S,T)

How to recover the alignment?

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

		A	C	G	G	C	T	A	T
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
A	1	-2	2	0	-2	-4	-6	-8	-10	-12
C	2	-4	0	4	2	0	-2	-4	-6	-8
T	3	-6	-2	2	3	1	-1	0	-2	-4
G	4	-8	-4	0	4	5	3	1	-1	-3
T	5	-10	-6	-2	2	3	4	5	3	1
A	6	-12	-8	-4	0	1	2	3	7	5
T	7	-14	-10	-6	-2	-1	0	4	5	9

Score(S,T)

How to recover the alignment?

Scoring function: $s(x,x) = 2, s(x,y) = -1$ for $x \neq y$, $d = -2$

ACGGCTAT
ACTG-TAT

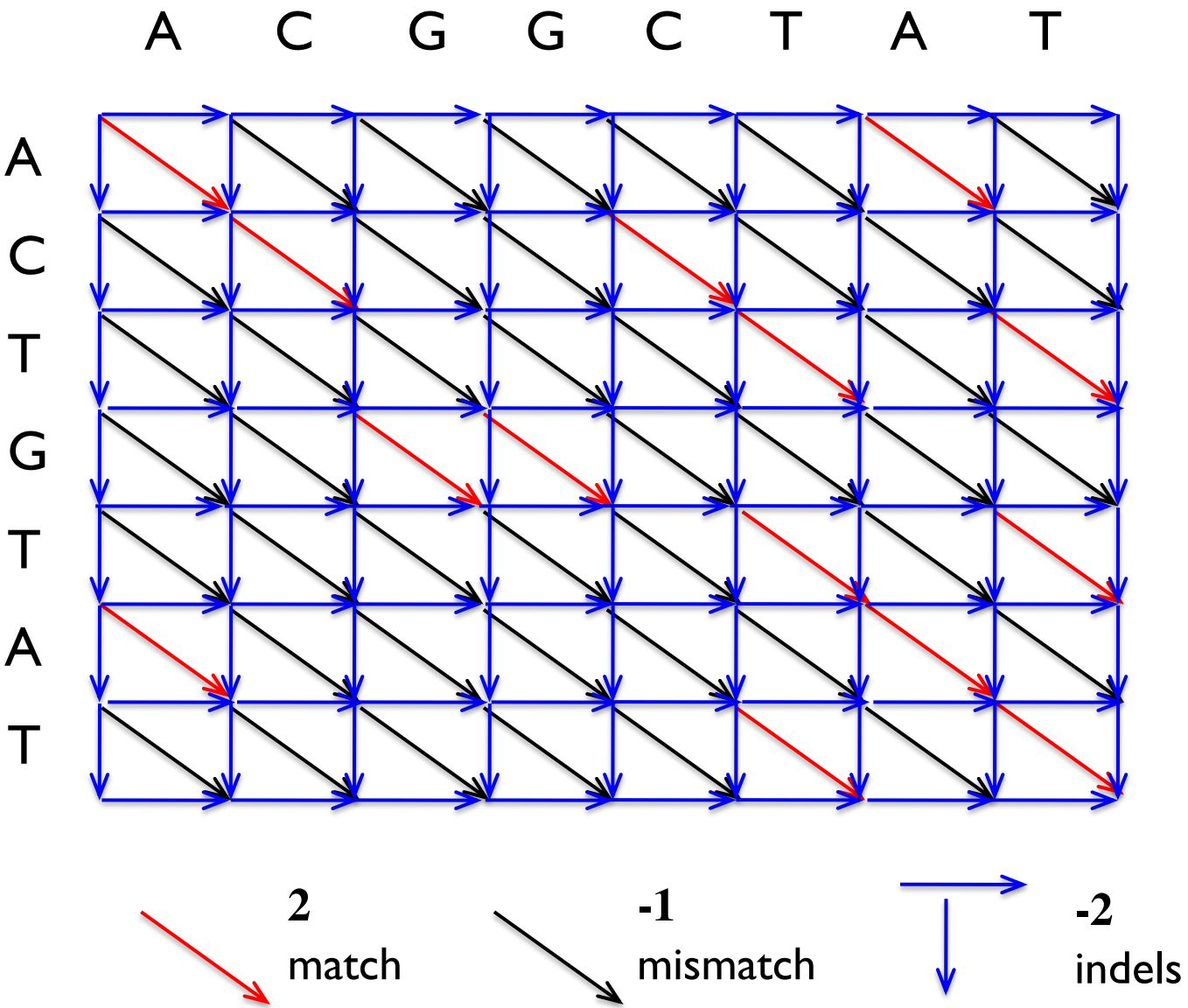
A C G G C T A T

		0	1	2	3	4	5	6	7	8
	0	0	-2	-4	-6	-8	-10	-12	-14	-16
A	1	-2	2	0	-2	-4	-6	-8	-10	-12
C	2	-4	0	4	2	0	-2	-4	-6	-8
T	3	-6	-2	2	3	1	-1	0	-2	-4
G	4	-8	-4	0	4	5	3	1	-1	-3
T	5	-10	-6	-2	2	3	4	5	3	1
A	6	-12	-8	-4	0	1	2	3	7	5
T	7	-14	-10	-6	-2	-1	0	4	5	9

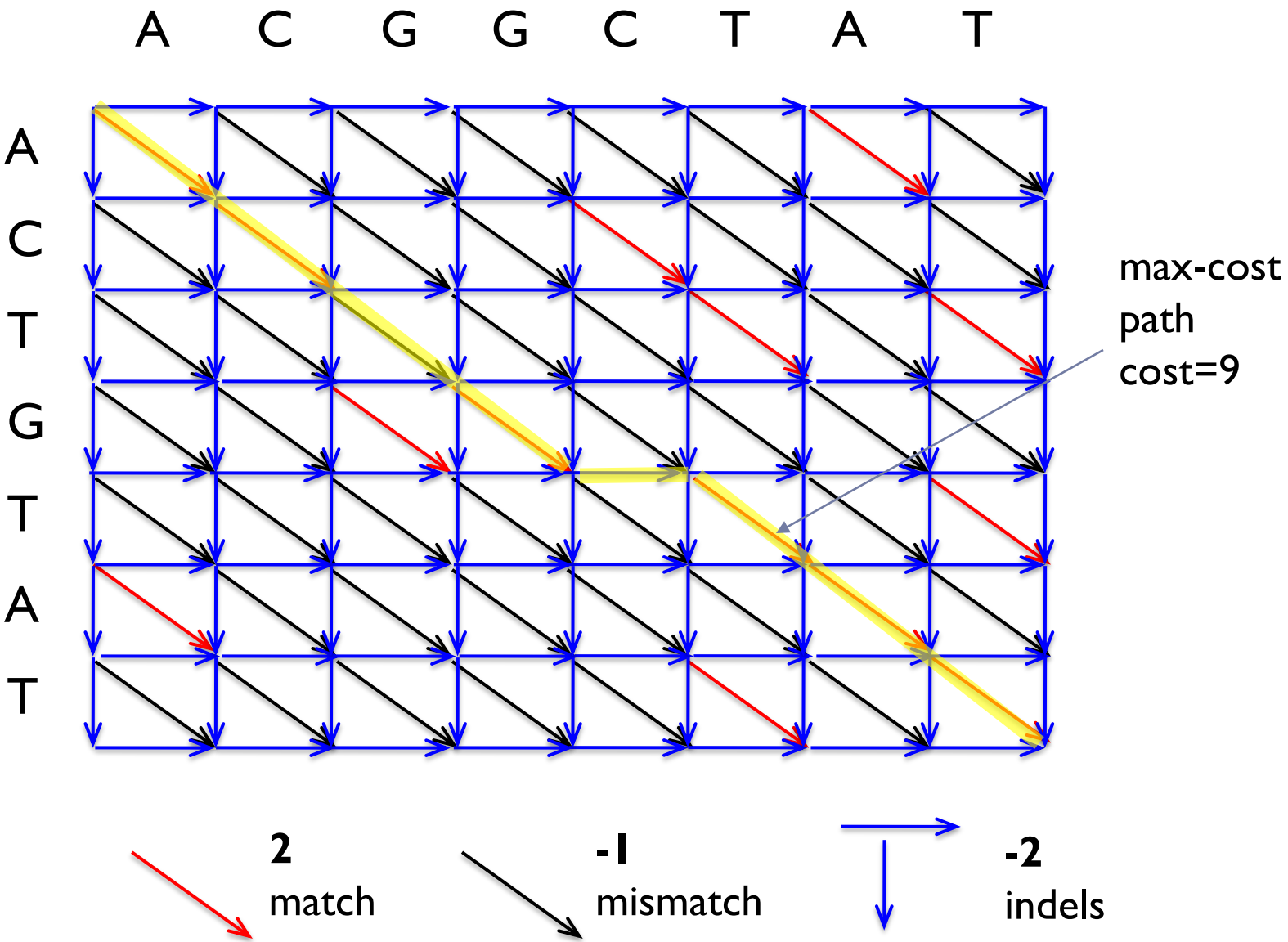
Score(S,T)

Quiz 7

Alignment: graph formulation



Alignment: graph formulation



Shortest path view (flashback)

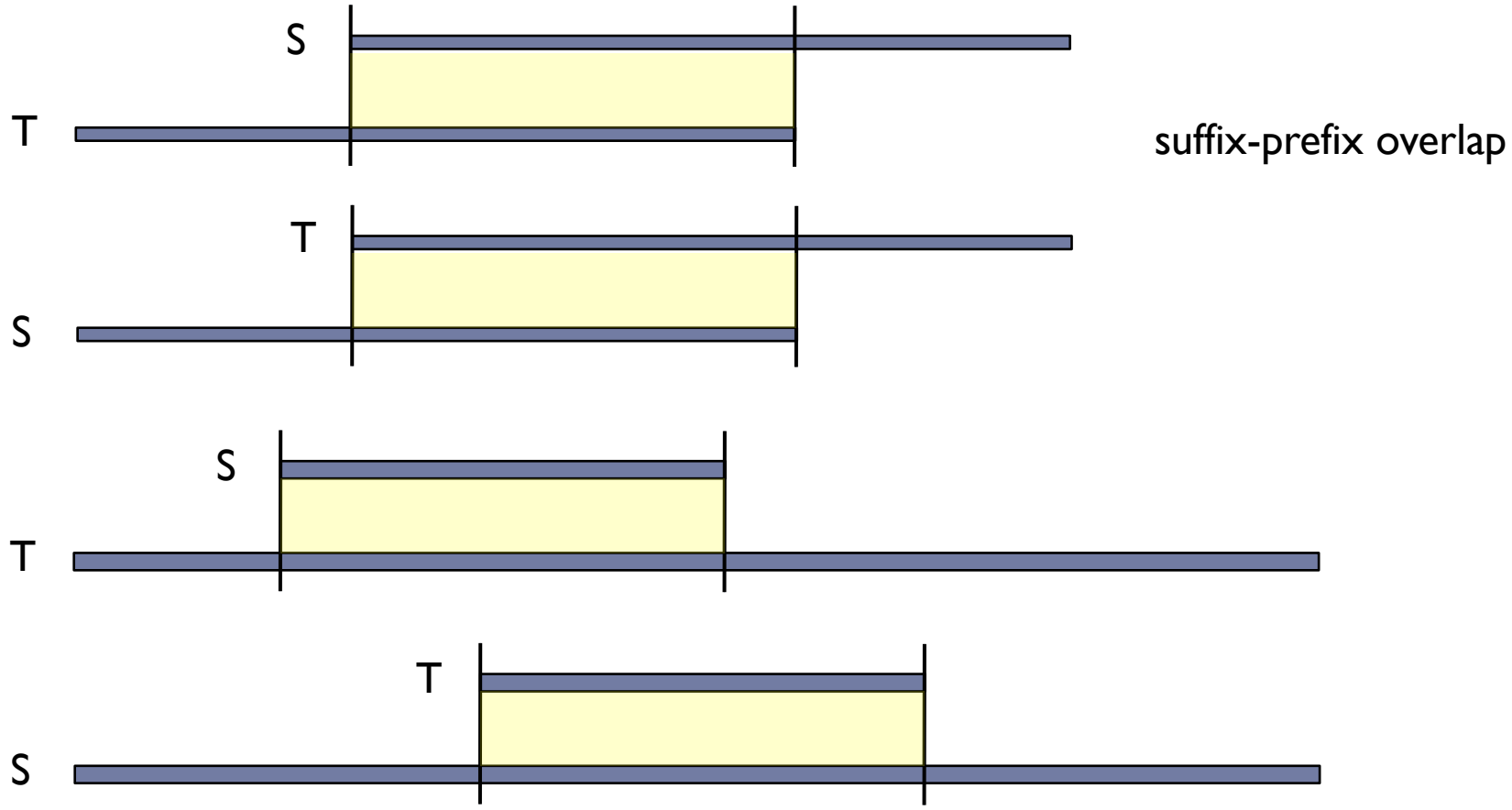
- Previous DP solution implies that *on this graph*, the single-source shortest/longest paths problem can be solved in time $O(nm)$
- … i.e. in time $O(|V| + |E|)$ as the graph has $O(nm)$ nodes and $O(nm)$ edges
- Not a surprise! We knew this already (well, almost), cf lecture on shortest paths in DAGs
- However, no need here for topological sort because of the regular graph structure

Comments

- algorithm known as Needleman-Wunsch algorithm (1970)
- note that optimal alignment is generally not unique
- the problem considered is called *global alignment*
- both time and space complexity is $O(n^2)$
- space complexity is $O(n)$ if only the optimum score has to be computed (e.g. line-by-line, keep two lines at a time)
- time can be reduced to $O(\frac{n^2}{log^2 n})$ (assuming RAM model) [Masek, Paterson 80] using "four-russians technique" (another solution in [Crochemore, Landau, Ziv-Ukelson 03])
- proved to be unlikely solvable in time $O(n^{2-\varepsilon})$ [Abboud, Williams, Weimann 14] (by reduction from 3SUM to some versions of alignment problem); similar result for LCS [Abboud, Backurs, Williams 15]

End-space free alignment

- Compute the best alignment of S and T such that spaces at string borders contribute 0



End-space free alignment: example

Scoring: $s(x,x) = 4, s(x,y) = -1$ for $x \neq y, \quad d = -2$

S = GAACTGCG

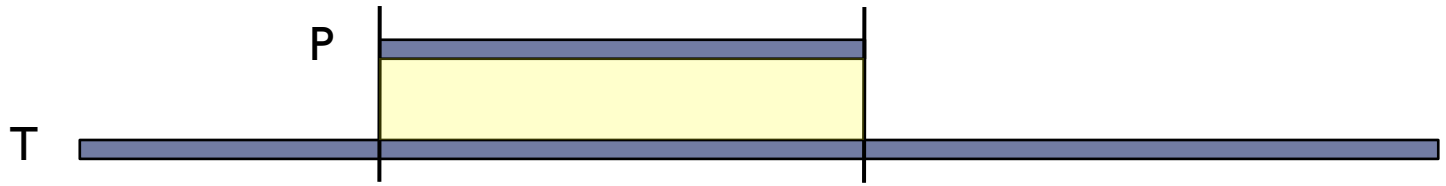
T = CAAGAC

```
GAACTGCG
|  | |
CAAG-AC
```

$score = 10$

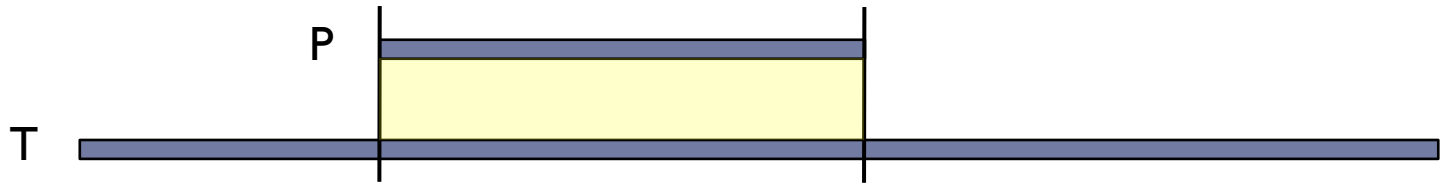
Approximate string matching

- Compute all alignments such that $\text{Score}(P, T[i..j]) \geq \delta$



Approximate string matching

- Compute all alignments such that $\text{dist}(P, T[i..j]) \leq k$



- Particular cases
  - edit distance: $O(kn)$ [Landau&Vishkin 85, Galil&Park 89, …]
  - Hamming distance: $O(n \log m)$ [Fischer&Paterson 73], $O(nk)$ [Galil&Giancarlo 86], $O(n\sqrt{k \log k})$ [Amir&Lewenstein&Porat 04], …

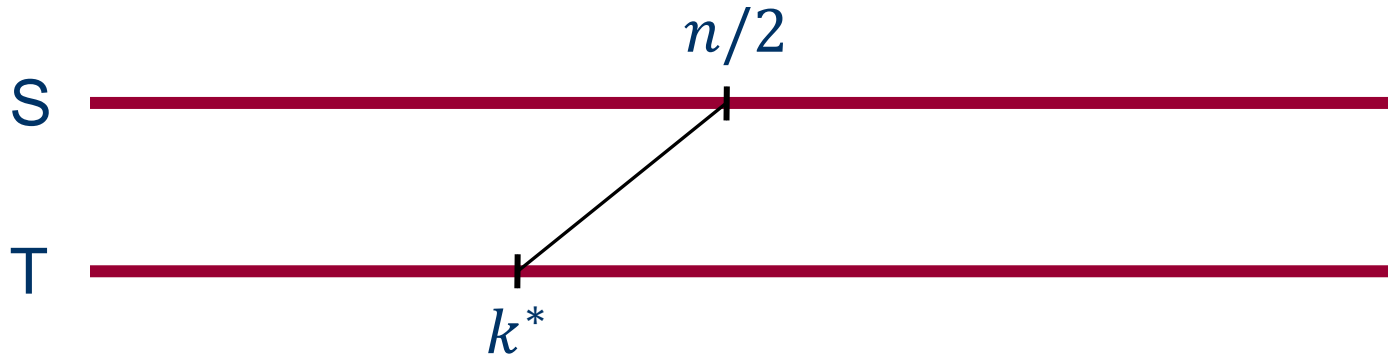
Computing alignment in linear space

- Hirschberg (1975) proposed a nice trick in order to compute the optimal alignment *in linear space* (at the price of doubling the time)

- Linear space is trivially sufficient if we only need to compute the *optimum score*:
  - fill entries of DP matrix line-by-line
  - keeping only the previous line is sufficient
- But how to do this if we need an *optimal alignment*?
  - if we don't keep the whole matrix, traceback becomes impossible

Hirschberg: main idea

- If we cut one sequence into two equal part, to which cut in the second sequence will it correspond?



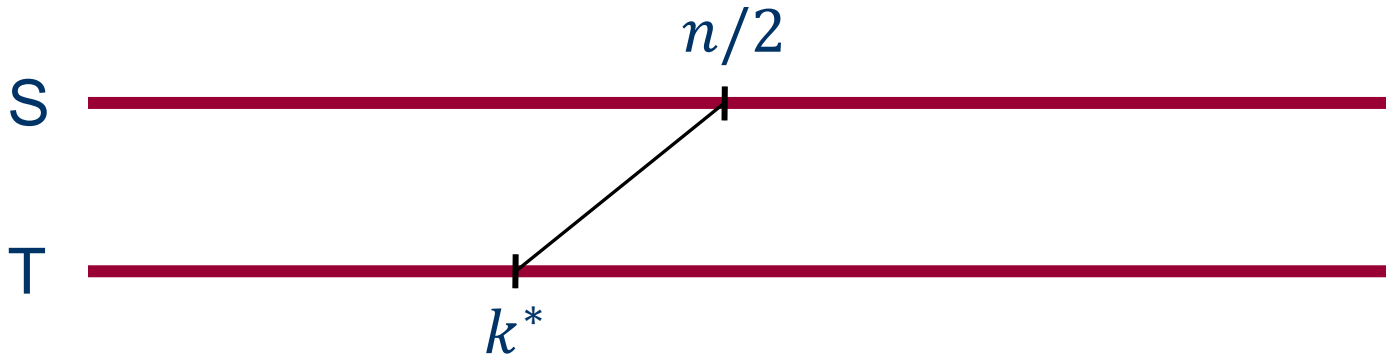
Example: $s(x,x) = 4, s(x,y) = -1$ for $x \neq y, \quad d = -2$

S = GAACTGCG

T = CAACAC

Hirschberg: main idea

- If we cut one sequence into two equal part, to which cut in the second sequence will it correspond?



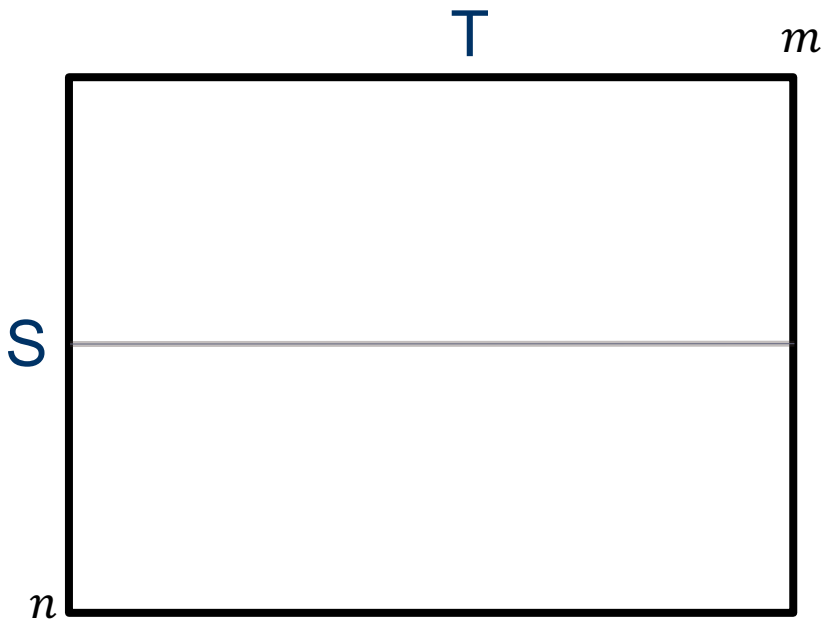
Example: $s(x,x) = 4, s(x,y) = -1$ for $x \neq y, \quad d = -2$

S = GAAC|TGCG

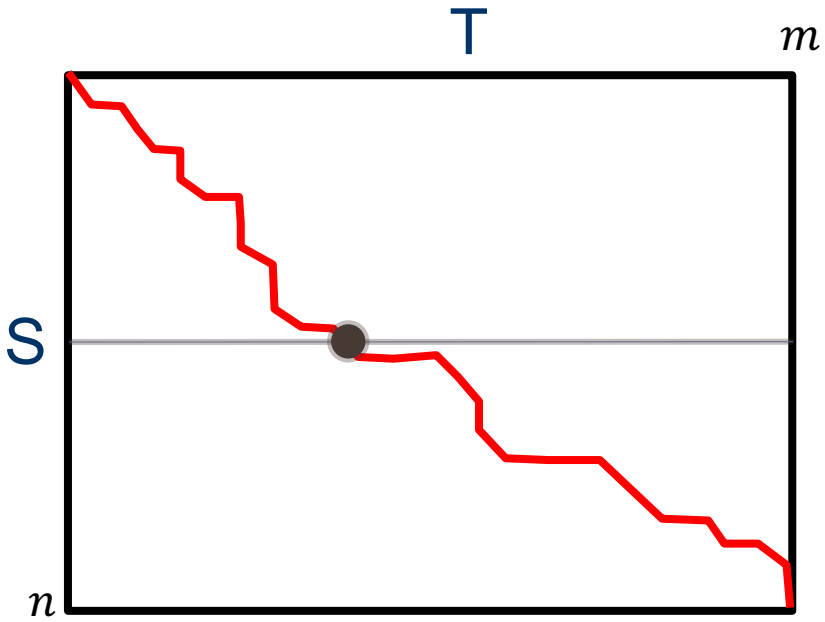
T = CAAC|AC

```
GAACTGCG
|||   |
CAAC-A-C-
```

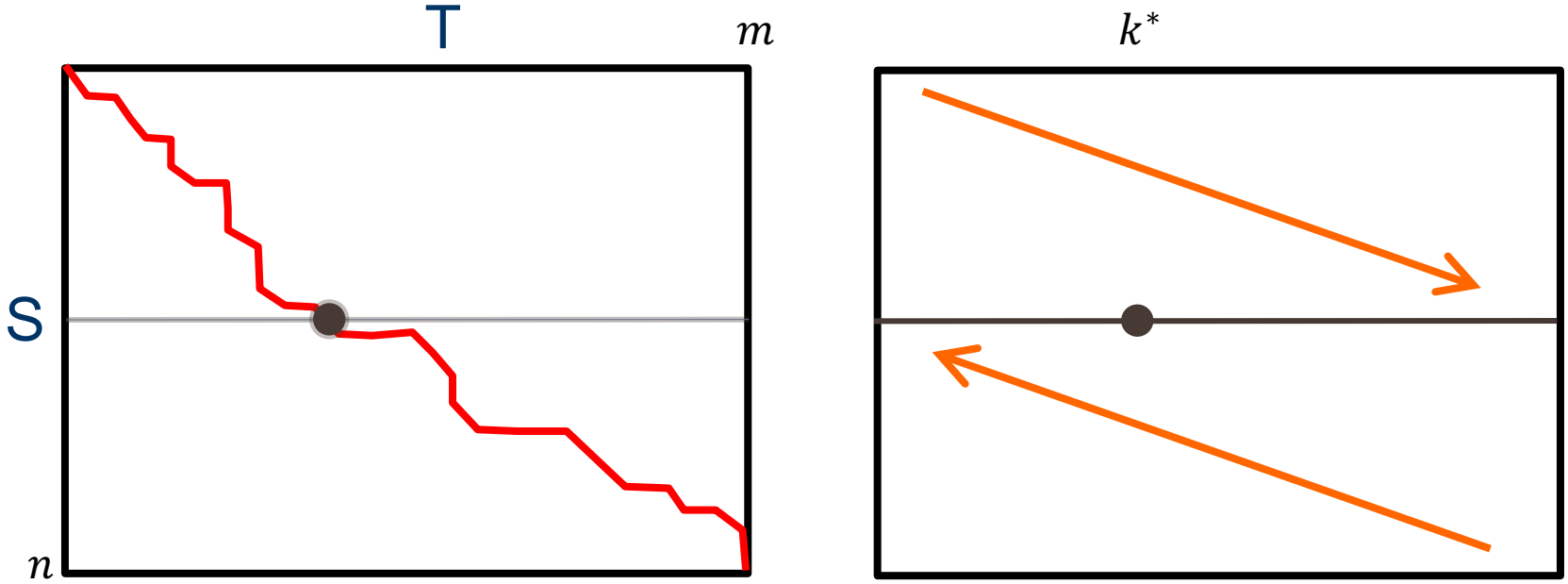
Hirschberg explained on graphs



Hirschberg explained on graphs

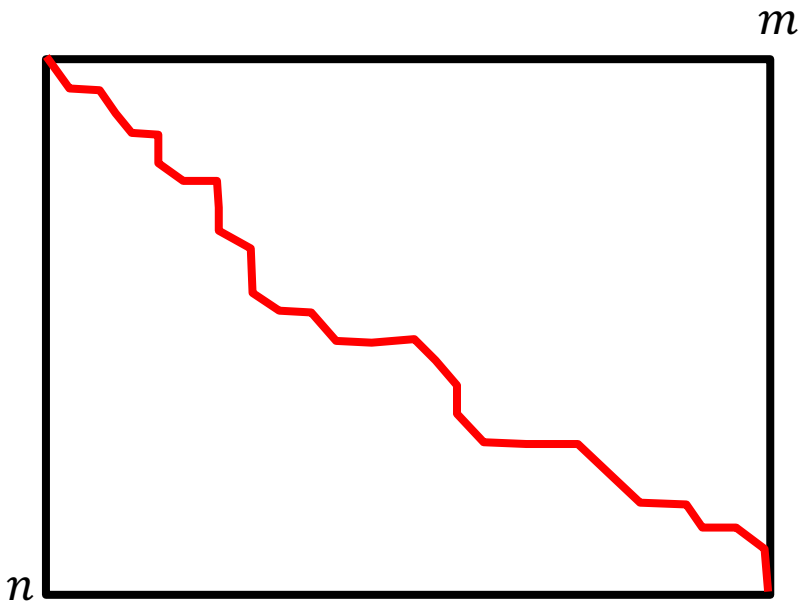


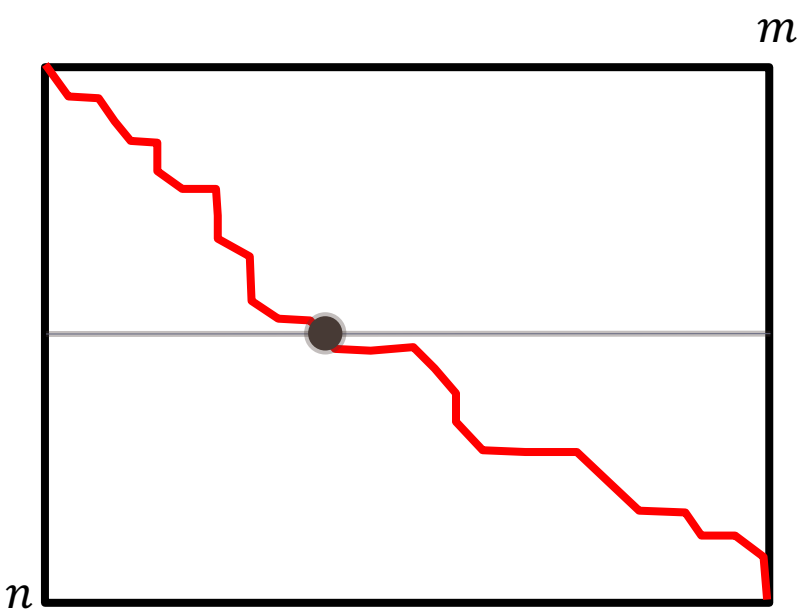
Hirschberg explained on graphs

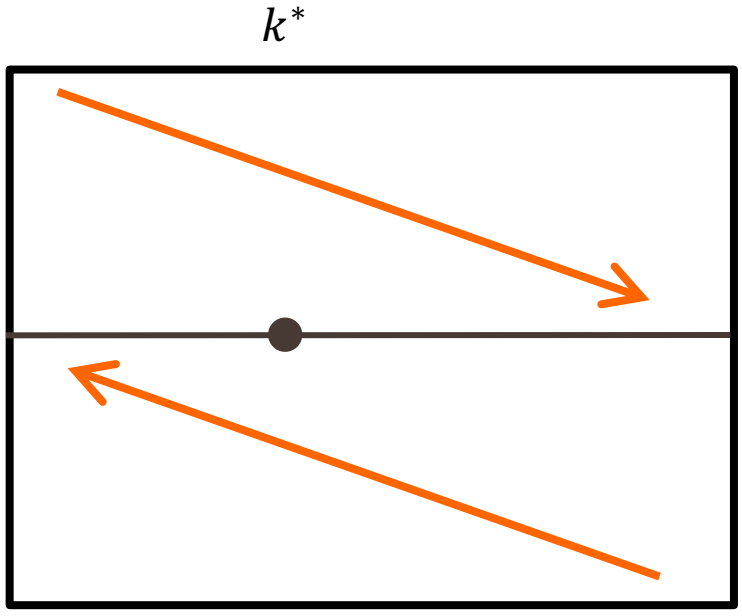


computing k^* (cf. bidirectional search):

- for all $0 \leq k \leq m$, compute (forward) the max score $\text{Score}(n/2, k)$ of a path from $(0,0)$ to $(n/2, k)$
- for all $0 \leq k \leq m$, compute (backward) the max score $\text{Score}^{\text{R}}(n/2, k)$ of a path from $(n/2, k)$ to (n, m)
- choose $k^* = \text{argmax}_k(\text{Score}(n/2, k) + \text{Score}^{\text{R}}(n/2, m - k))$

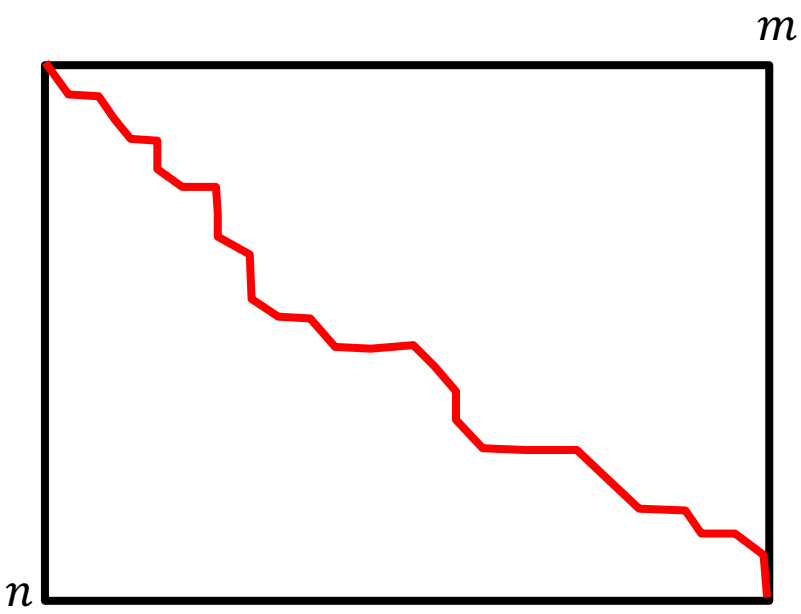

m
n

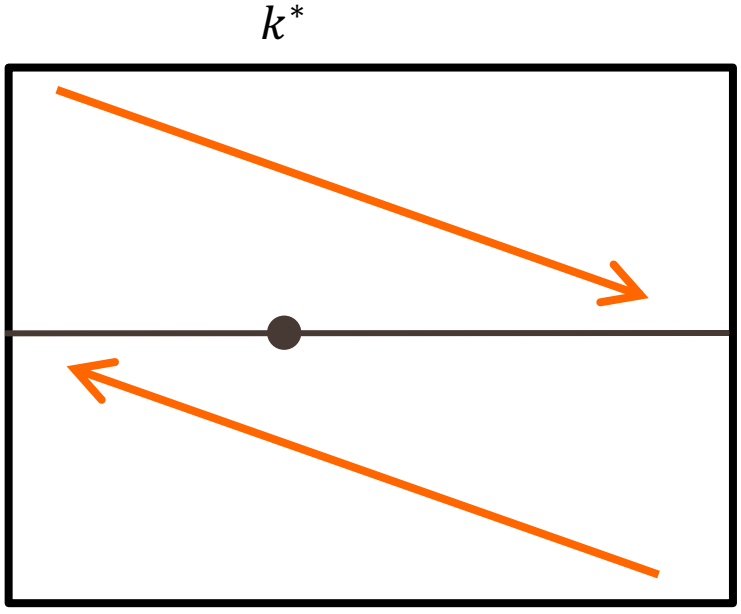


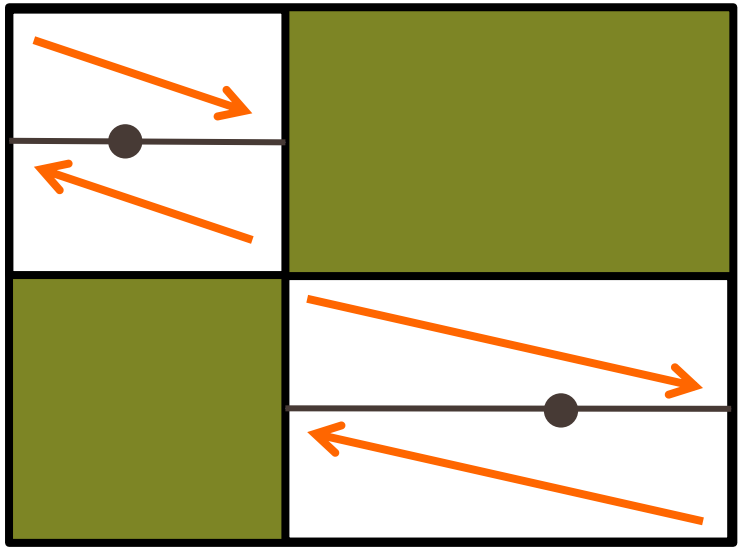


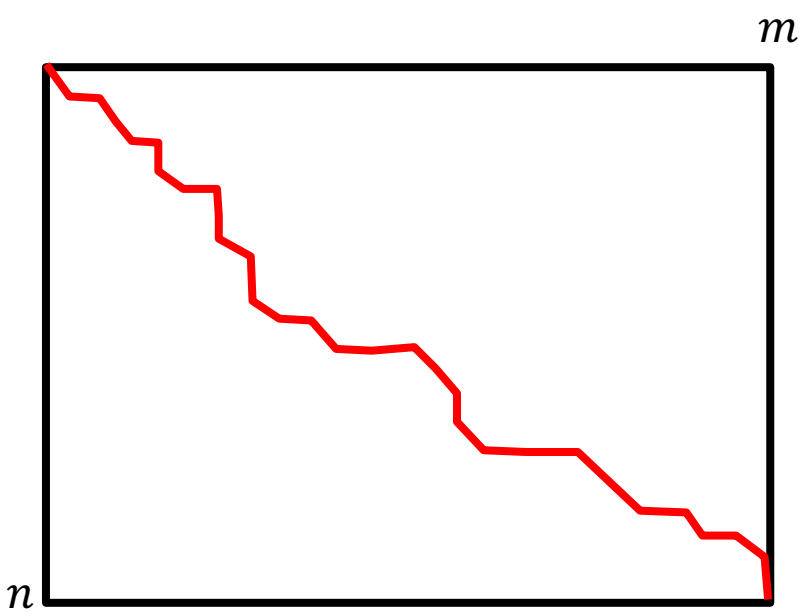
compute

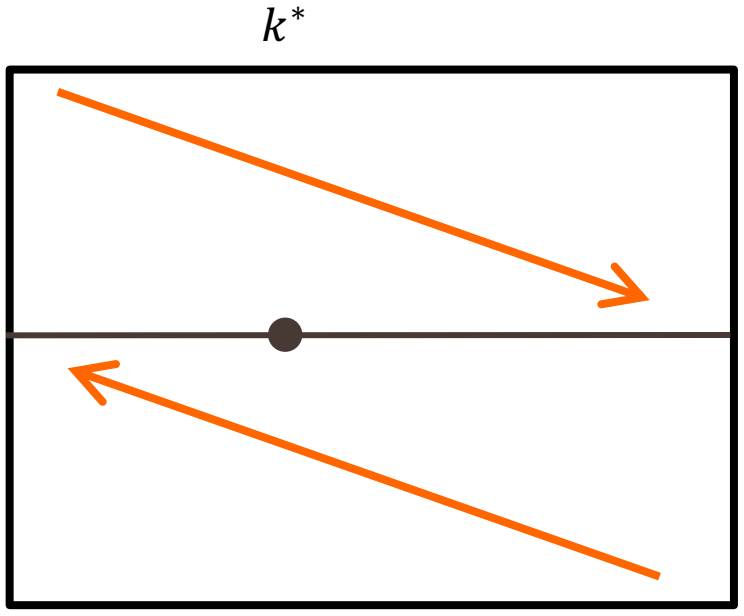
$k^* = \text{argmax}_k(\text{Score}(n/2, k) + \text{Score}^{\text{R}}(n/2, m - k))$

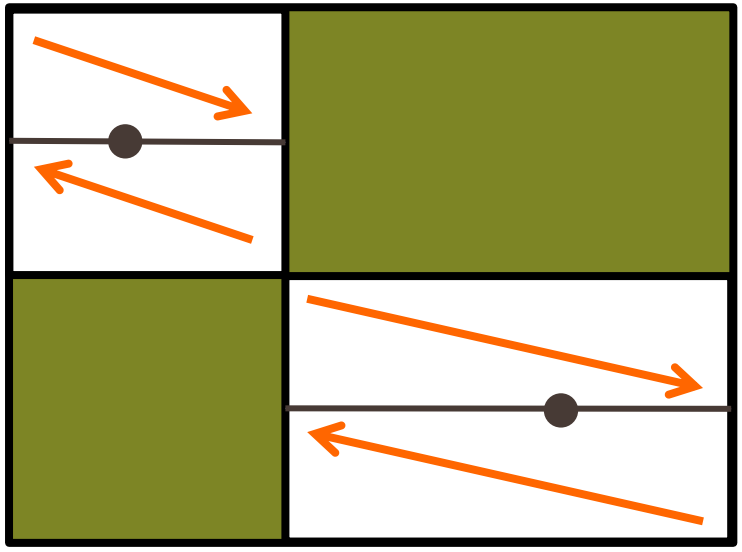

m
n

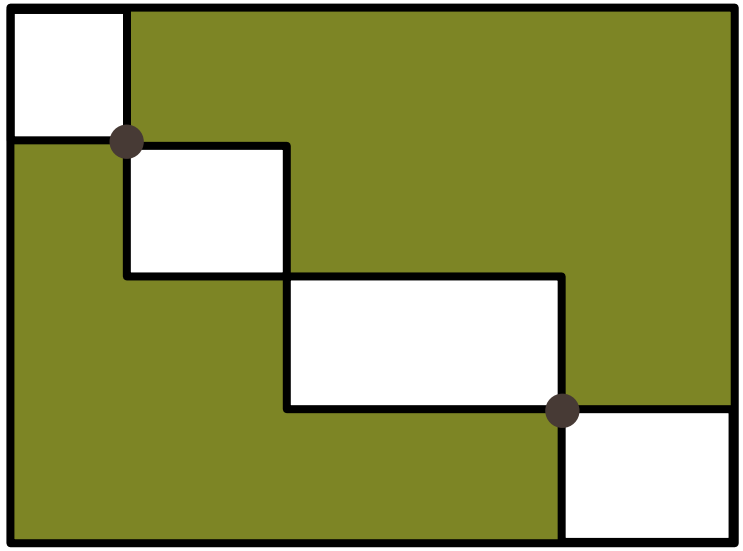

k*




m
n


k*



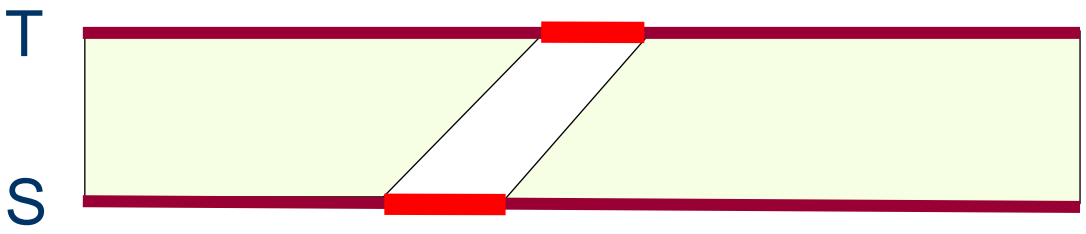


Resulting time complexity

- assume that computing the optimal Score on a $p \times q$ matrix takes time $c \cdot pq$
- computing the first "cut" takes $2 \cdot c \cdot \frac{n}{2} m = c \cdot nm$
- the first halving results in time $c \cdot \frac{n}{2} \cdot k^* + c \cdot \frac{n}{2}(m - k^*) = \frac{1}{2} c \cdot nm$
- all recursive calls take time $c \cdot nm + \frac{1}{2} c \cdot nm + \frac{1}{4} c \cdot nm + \cdots \leq 2c \cdot nm$

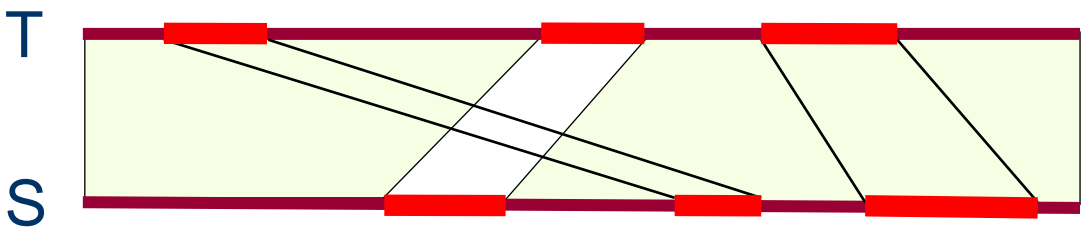
Local alignment

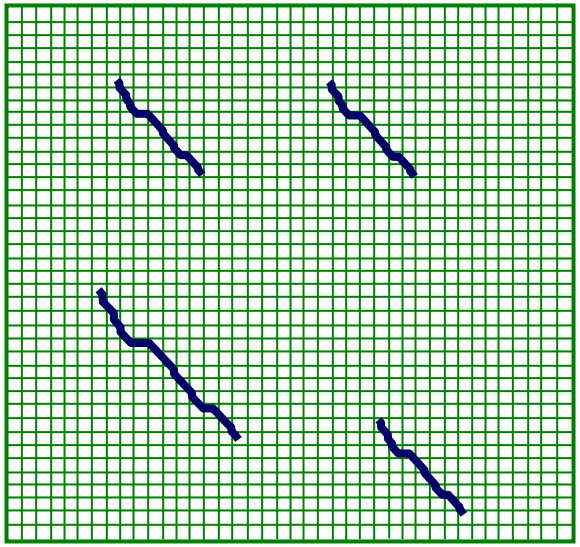
- Biologists are mostly interested in *local alignments* that may ignore arbitrary prefixes and suffixes of input sequences



Local alignment

- Biologists are mostly interested in *local alignments* that may ignore arbitrary prefixes and suffixes of input sequences





- *Problem*: Compute **all** *significant* local alignments, i.e. all alignments of score above a threshold

Smith-Waterman algorithm (1981)

- Assume matches are scored positively and mismatches/indels are scored negatively
- $\text{Score}(i, j)$: maximum score over all substrings of S that end at position i and all substrings of T that end at position j
- initialization: $\text{Score}(0, j) = \text{Score}(j, 0) = 0$

$$\text{Score}(i, j) = \max \begin{cases} 0 \\ \text{Score}(i-1, j-1) + s(S[i], T[j]) \\ \text{Score}(i-1, j) + d \\ \text{Score}(i, j-1) + d \end{cases}$$

Smith-Waterman: example

EAWACQGKL vs ERDAWCQPGKWY

$s(x,x) = 1, s(x,y) = -3$ for $x \neq y, d = -1$

T	j	0	1	2	3	4	5	6	7	8	9	10	11	12
i		$y[j]$	E	R	D	A	W	C	Q	P	G	K	W	Y
0	$x[i]$	0	0	0	0	0	0	0	0	0	0	0	0	0
1	E	0	1	0	0	0	0	0	0	0	0	0	0	0
2	A	0	0	0	0	1	0	0	0	0	0	0	0	0
3	W	0	0	0	0	0	2	1	0	0	0	0	1	0
4	A	0	0	0	0	1	1	0	0	0	0	0	0	0
5	C	0	0	0	0	0	0	2	1	0	0	0	0	0
6	Q	0	0	0	0	0	0	1	3	2	1	0	0	0
7	G	0	0	0	0	0	0	0	2	1	3	2	1	0
8	K	0	0	0	0	0	0	0	1	0	2	4	3	2
9	L	0	0	0	0	0	0	0	0	0	1	3	2	1

resulting local alignment:

$$\begin{pmatrix} \mathtt{A} & \mathtt{W} & \mathtt{A} & \mathtt{C} & \mathtt{Q} & \mathtt{-} & \mathtt{G} & \mathtt{K} \\ \mathtt{A} & \mathtt{W} & \mathtt{-} & \mathtt{C} & \mathtt{Q} & \mathtt{P} & \mathtt{G} & \mathtt{K} \end{pmatrix}$$

Comments

- Score matrix is important
- There exists a statistical model (Karlin&Altschul 90) that allows to relate the score of a local alignment and the probability for this alignment to appear in random sequences (p-value)
- The average value of score matrix should be negative

More complex gap penalty systems

- Affine gap penalty: $h + q \cdot i$

  h: gap opening penalty

  q: gap extension penalty

  $O(mn)$ algorithm [Gotoh 82]
- Convex gap penalty

  $O(mn \cdot \log n)$
- Arbitrary gap penalty

  $O(mn^2 + nm^2)$