Dynamic-Programming Strategies for Analyzing Biomolecular Sequences

Kun-Mao Chao

Department of Life Science

National Yang-Ming University, Taiwan

2002 in Singapore

E-mail: kmchao@ym.edu.tw

WWW: http://www.ym.edu.tw/~kmchao

1881

1991

2002

2112

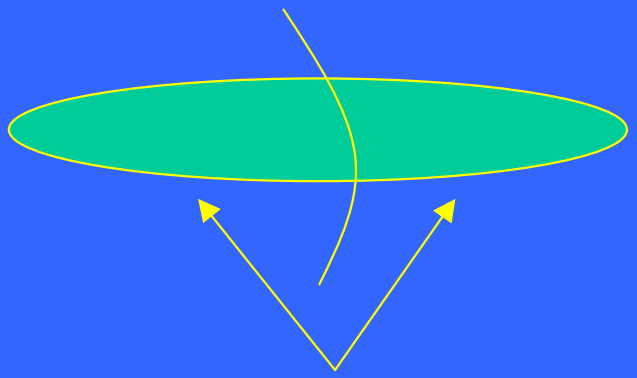
Dynamic Programming

- Dynamic programming is a class of solution methods for solving sequential decision problems with a compositional cost structure.
- Richard Bellman was one of the principal founders of this approach.

Two key ingredients

- Two key ingredients for an optimization problem to be suitable for a dynamic-programming solution:

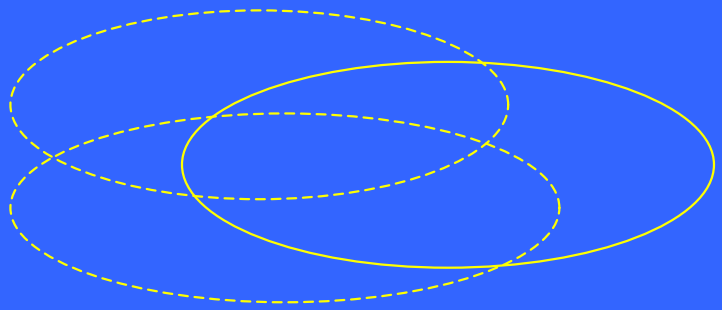
1. optimal substructures



Each substructure is optimal.

(Principle of optimality)

2. overlapping subproblems



Subproblems are dependent.

(otherwise, a divide-and-conquer approach is the choice.)

Three basic components

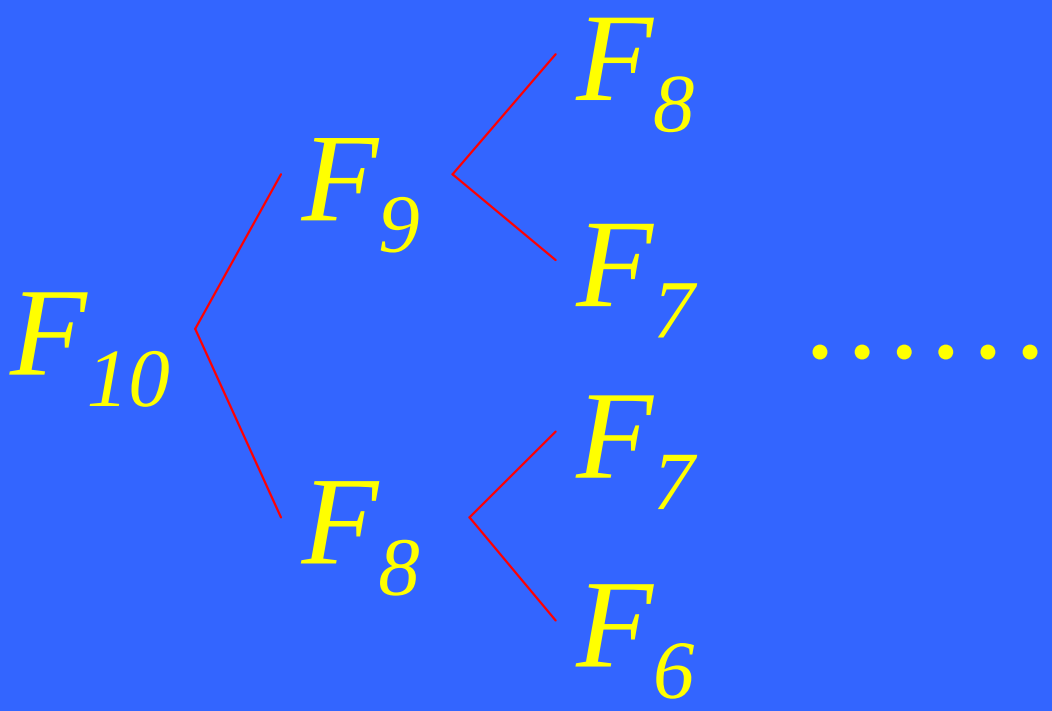
- The development of a dynamic-programming algorithm has three basic components:
  - The recurrence relation (for defining the value of an optimal solution);
  - The tabular computation (for computing the value of an optimal solution);
  - The traceback (for delivering an optimal solution).

Fibonacci numbers

The *Fibonacci numbers* are defined by the following recurrence:

$$F_0 = 0$$
$$F_1 = 1$$
$$F_i = F_{i-1} + F_{i-2} \quad \text{for } i > 1.$$

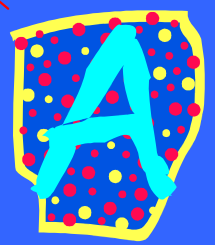
How to compute F_{10} ?



Tabular computation

- The tabular computation can avoid recompuation.

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
0	1	1	2	3	5	8	13	21	34	55



Maximum-sum interval

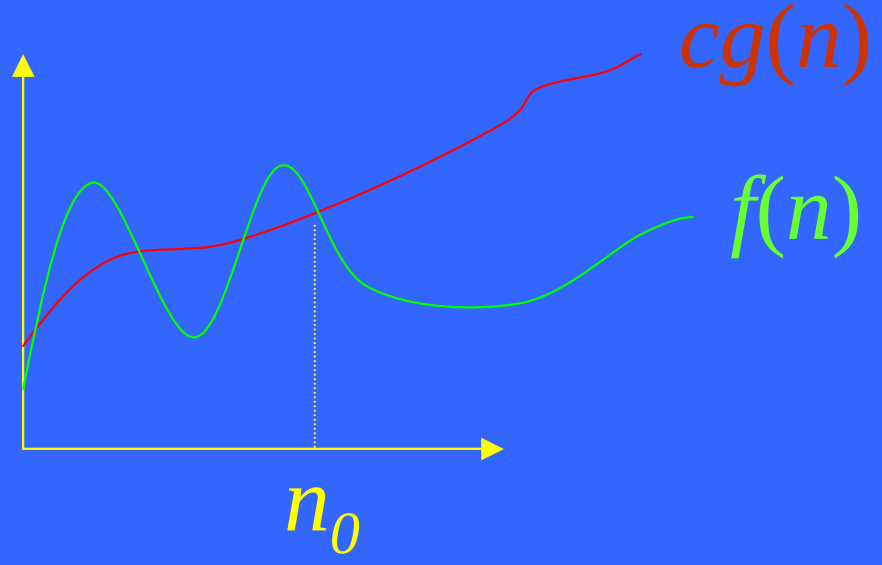
- Given a sequence of real numbers $a_1 a_2 \ldots a_n$, find a consecutive subsequence with the maximum sum.

```
9 –3 1 7 –15 2 3 –4 2 –7 6 –2 8 4 -9
```

For each position, we can compute the maximum-sum interval starting at that position in $O(n)$ time. Therefore, a naive algorithm runs in $O(n^2)$ time.

O-notation: an asymptotic upper bound

- $f(n) = O(g(n))$ iff there exist two positive constant c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$



How functions grow?

function / n	$30n$	$92n \log n$	$26n^2$	$0.68n^3$	2^n
100	0.003 sec.	0.003 sec.	0.0026 sec.	0.00068 sec.	4×10^{16} yr.
100,000	3.0 sec.	2.6 min.	3.0 days	22 yr.	

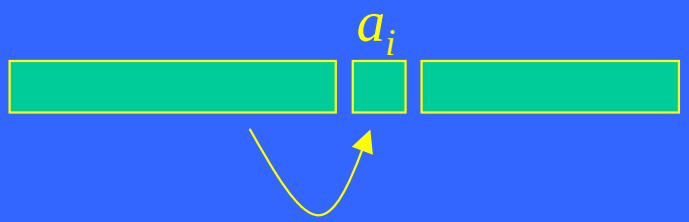
(Assume one million operations per second.)

For large data sets, algorithms with a complexity greater than $O(n \log n)$ are often impractical!

Maximum-sum interval (The recurrence relation)

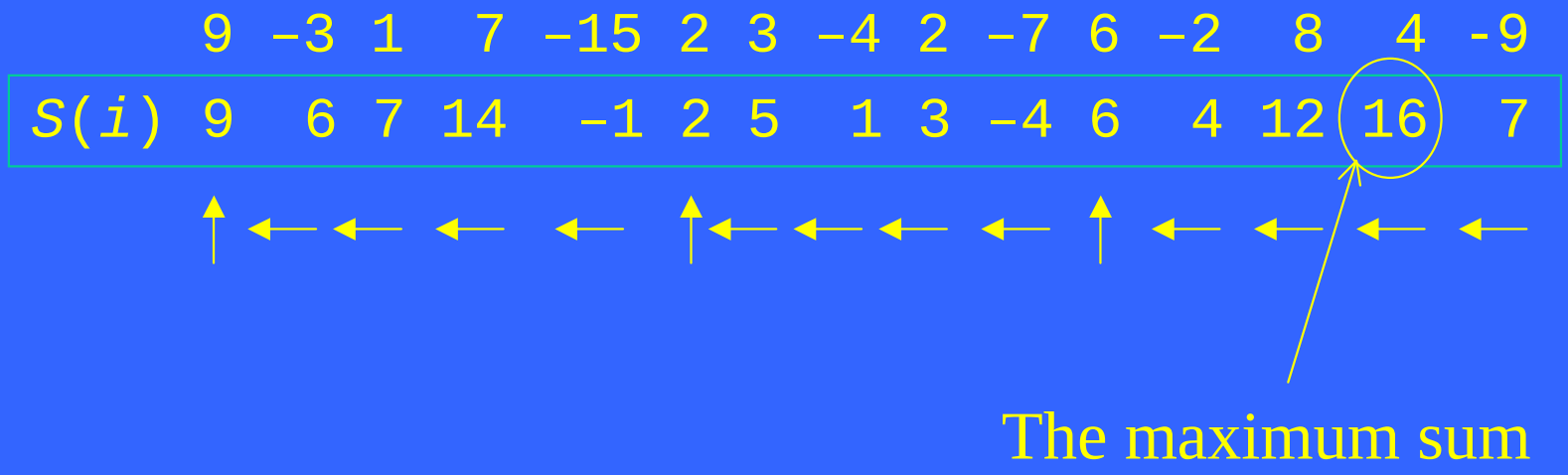
- Define $S(i)$ to be the maximum sum of the intervals ending at position i.

$$S(i) \leftarrow a_i + \max\begin{cases} S(i-1) \\ 0 \end{cases}$$

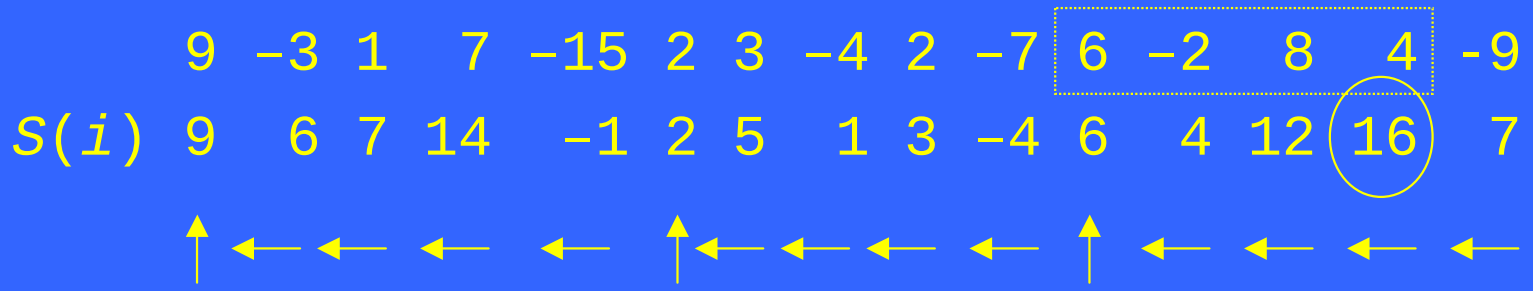


If $S(i\text{-}1) < 0$, concatenating a_i with its previous interval gives less sum than a_i itself.

Maximum-sum interval (Tabular computation)



Maximum-sum interval (Traceback)



The maximum-sum interval: 6 -2 8 4

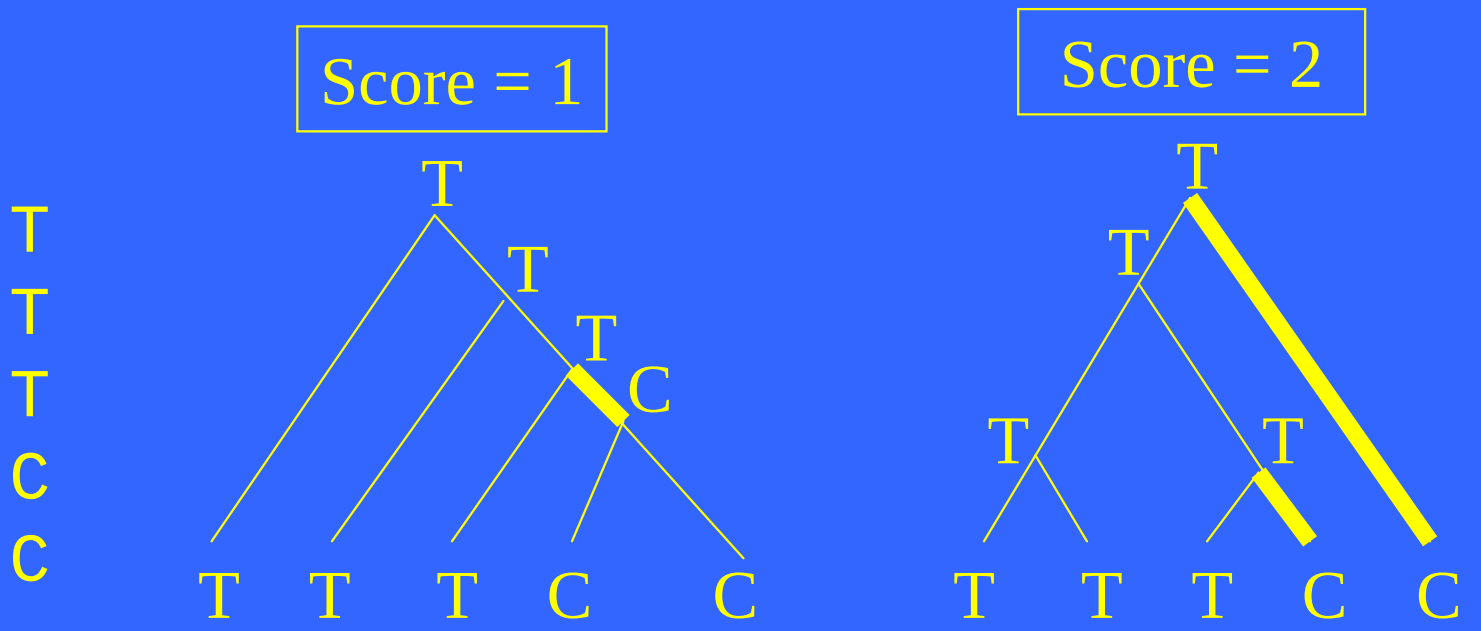
Defining scores for alignment columns

- *infocon* [Stojanovic *et al.*, 1999]
  - Each column is assigned a score that measures its information content, based on the frequencies of the letters both within the column and within the alignment.

```
CGGATCAT–GGA
CTTAACATTGAA
GAGAACATAGTA
```

Defining scores (cont'd)

- *phylogen* [Stojanovic *et al.*, 1999]
  - columns are scored based on the evolutionary relationships among the sequences implied by a supplied phylogenetic tree.



Two fundamental problems we recently solved (joint work with Lin and Jiang)

- Given a sequence of real numbers of length n and an upper bound U, find a consecutive subsequence of length at most U with the maximum sum --- an $O(n)$-time algorithm.

$U = 3$

9 –3 1 7 –15 2 3 –4 2 –7 6 –2 8 4 -9

Two fundamental problems we recently solved (joint work with Lin and Jiang)

- Given a sequence of real numbers of length n and a lower bound L, find a consecutive subsequence of length at least L with the maximum average. --- an $O(n \log L)$-time algorithm.

  $L = 4$

  3 2 14 6 6 2 10 2 6 6 14 2 1

Another example

Given a sequence as follows:

2, 6.6, 6.6, 3, 7, 6, 7, 2

and $L = 2$, the highest-average interval is the squared area, which has the average value 20/3.

2, 6.6, 6.6, 3, 7, 6, 7, 2

C+G rich regions

- Our method can be used to locate a region of length at least L with the highest C+G ratio in $O(n \log L)$ time.

```
ATGACTCGAGCTCGTCA
00101011011011010
```

←

Search for an interval of length at least L with the highest average.

Length-unconstrained version

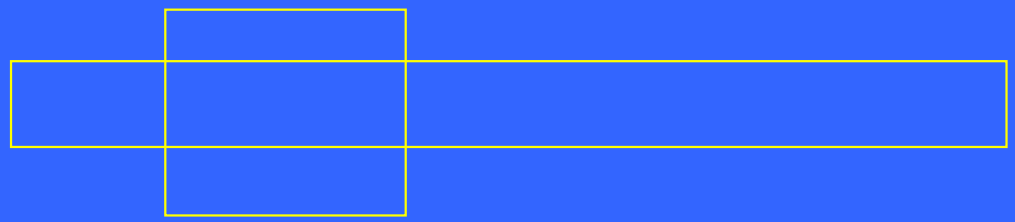
- Maximum-average interval

3 2 14 6 6 2 10 2 6 6 14 2 1

The maximum element is the answer. It can be done in $O(n)$ time.

A naive algorithm

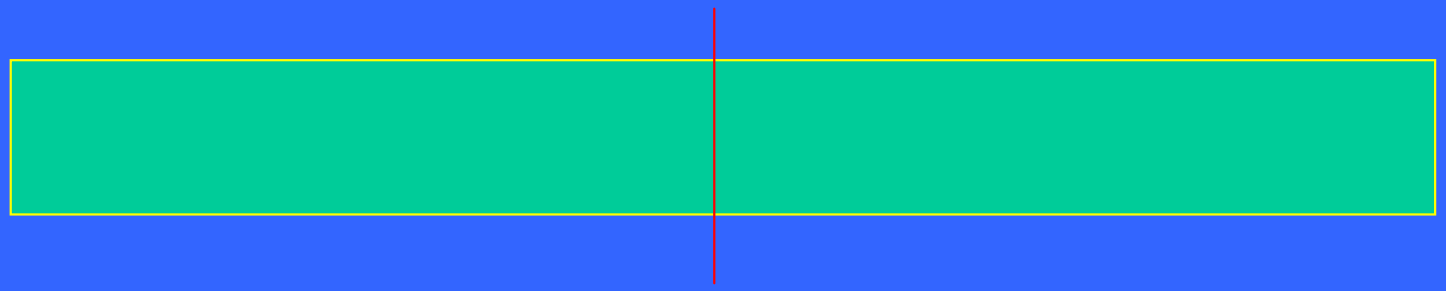
- A simple shift algorithm can compute the highest-average interval of a fixed length in O(n) time



- Try L, $L+1$, $L+2$, ..., n. In total, O(n^2).

A pigeonhole principle

- Notice that the length of an optimal interval is bounded by $2L$, we immediately have an O(nL)-time algorithm.



We can bisect a region of length >= $2L$ into two segments, where each of them is of length >= L.

Future Development

- Best *k* (nonintersecting) subsequences?
- Max-average with both upper and lower length bounds
- General (gapped) local alignment with length upper bound.
- Measurement of goodness?

Longest increasing subsequence(LIS)

- The longest increasing subsequence is to find a longest increasing subsequence of a given sequence of distinct integers $a_1 a_2 \ldots a_n$.

  e.g. 9 2 5 3 7 11 8 10 13 6

  2 3 7

  5 7 10 13 } are increasing subsequences.

  We want to find a longest one.

  9 7 11

  3 5 11 13 } are not increasing subsequences.

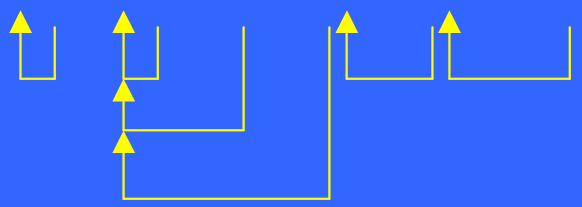
A naive approach for LIS

- Let $L[i]$ be the length of a longest increasing subsequence ending at position i.

$$L[i] = 1 + \max_{j = 0..i-1}\{L[j] \mid a_j < a_i\}$$

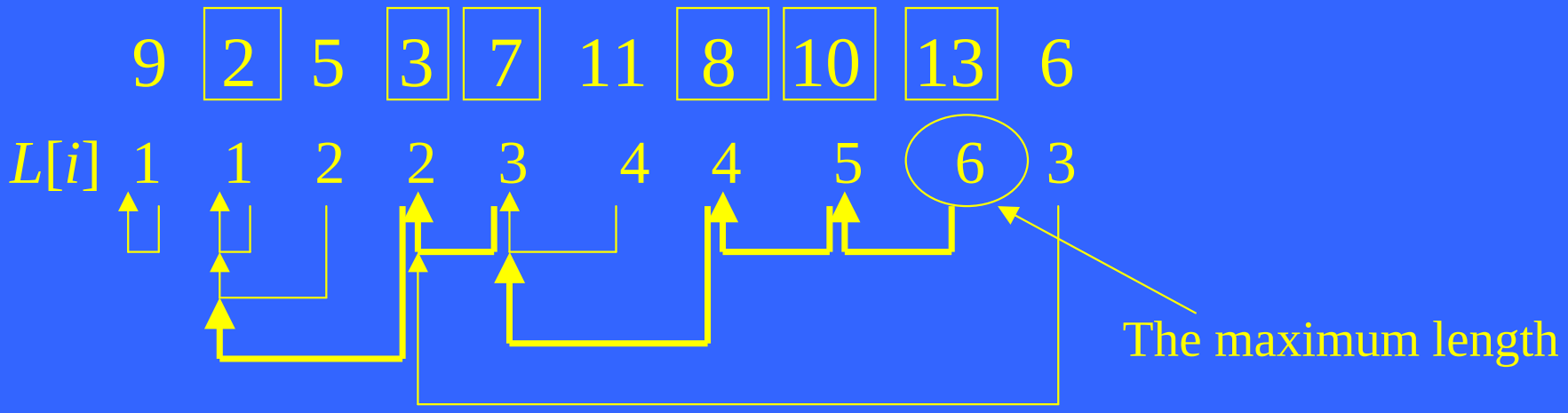
(use a dummy a_0 = minimum, and $L[0]=0$)

	9	2	5	3	7	11	8	10	13	6
$L[i]$	1	1	2	2	3		4			?



A naive approach for LIS

$$L[i] = 1 + \max_{j = 0..i-1} \{L[j] \mid a_j < a_i\}$$

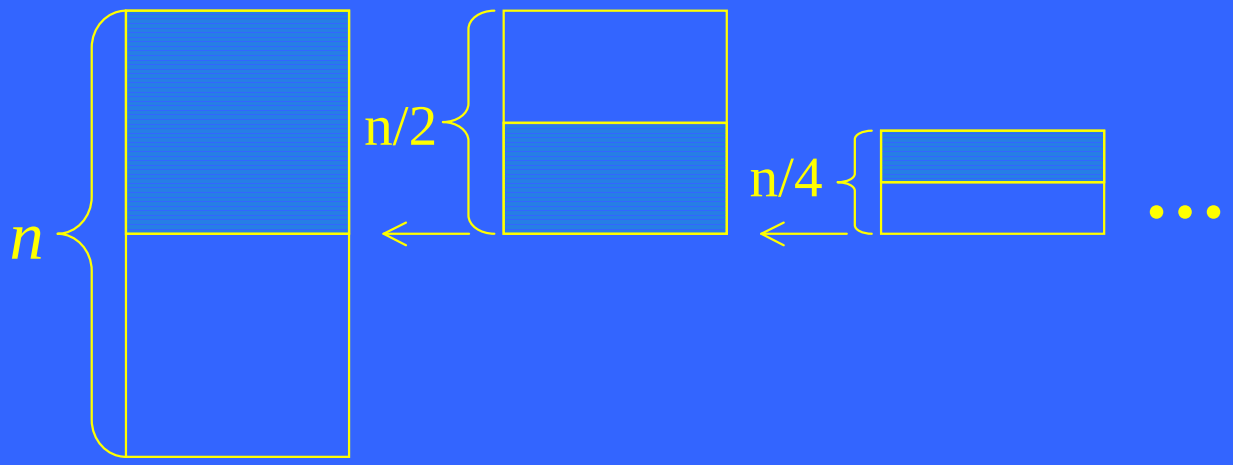


The subsequence 2, 3, 7, 8, 10, 13 is a longest increasing subsequence.

This method runs in $O(n^2)$ time.

Binary search

- Given an ordered sequence $x_1 x_2 \ldots x_n$, where $x_1 < x_2 < \ldots < x_n$, and a number y, a binary search finds the largest x_i such that $x_i < y$ in $O(\log n)$ time.



Binary search

- How many steps would a binary search reduce the problem size to *1*?

$$\underbrace{n \quad n/2 \quad n/4 \quad n/8 \quad n/16 \quad \ldots \quad 1}$$

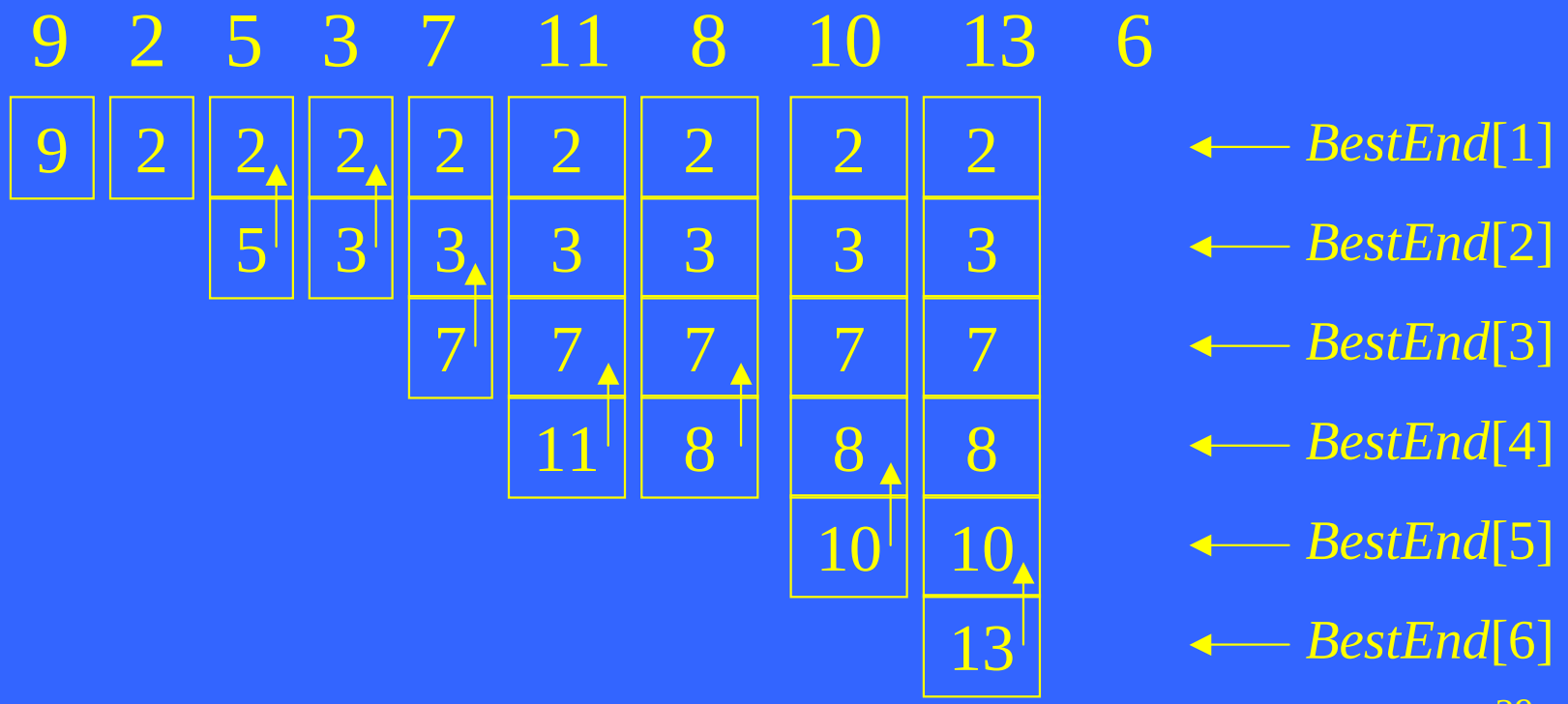
How many steps? $O(\log n)$ steps.

$$n/2^s = 1$$

$$\Rightarrow s = \log_2 n$$

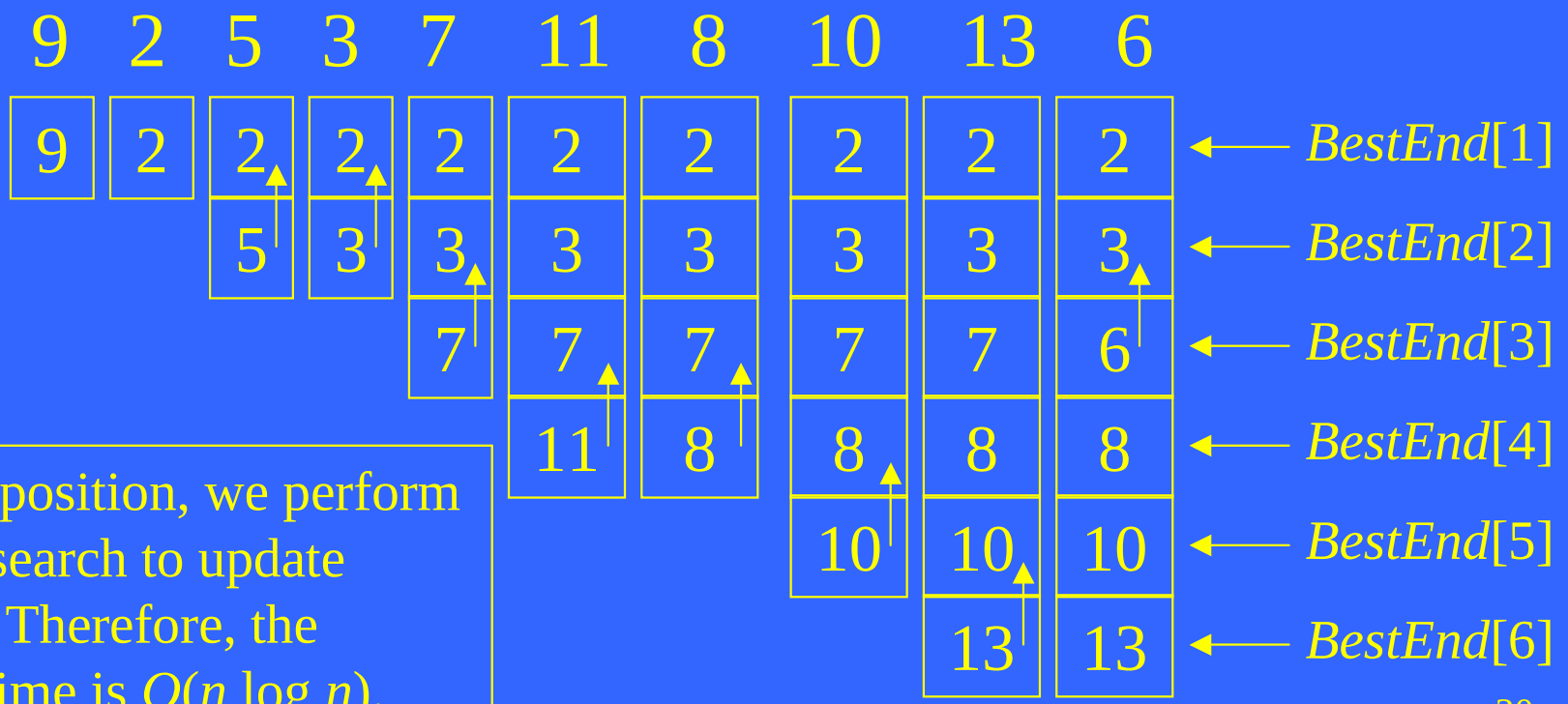
An $O(n \log n)$ method for LIS

- Define *BestEnd*[*k*] to be the smallest number of an increasing subsequence of length *k*.



An $O(n \log n)$ method for LIS

- Define *BestEnd*[k] to be the smallest number of an increasing subsequence of length k.



For each position, we perform a binary search to update *BestEnd*. Therefore, the running time is $O(n \log n)$.

Longest Common Subsequence (LCS)

- A subsequence of a sequence *S* is obtained by deleting zero or more symbols from *S*. For example, the following are all subsequences of “president”: pred, sdn, predent.
- The longest common subsequence problem is to find a maximum-length common subsequence between two sequences.

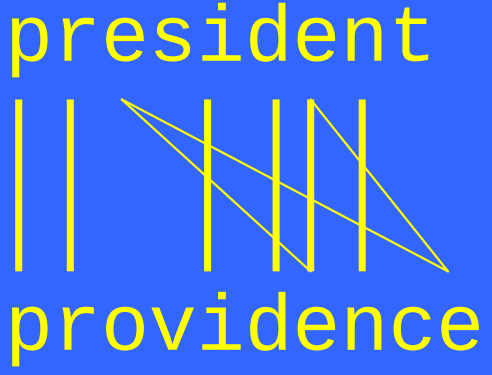
LCS

For instance,

Sequence 1: president

Sequence 2: providence

Its LCS is priden.



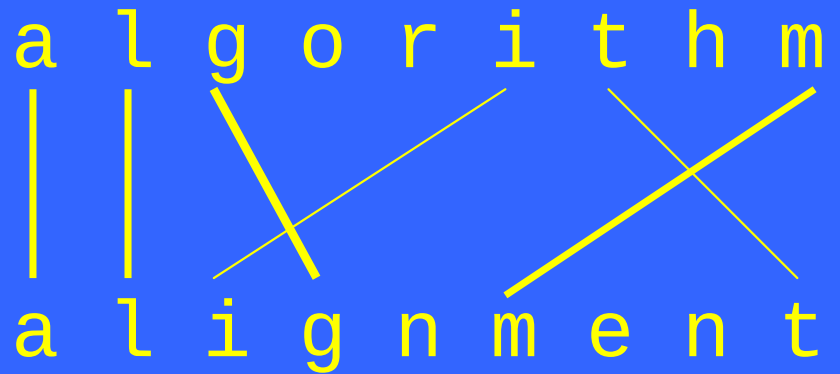
LCS

Another example:

Sequence 1: algorithm

Sequence 2: alignment

One of its LCS is algm.



How to compute LCS?

- Let A=$a_1a_2\ldots a_m$ and $B=b_1b_2\ldots b_n$.
- *len*(*i*, *j*): the length of an LCS between
  $a_1a_2\ldots a_i$ and $b_1b_2\ldots b_j$
- With proper initializations, *len*(*i*, *j*)can be computed as follows.

$$len(i,j)=\begin{cases}0 & \text{if } i=0 \text{ or } j=0,\\ len(i-1,j-1)+1 & \text{if } i,j>0 \text{ and } a_i=b_j,\\ \max(len(i,j-1),len(i-1,j)) & \text{if } i,j>0 \text{ and } a_i\neq b_j.\end{cases}$$

procedure *LCS-Length(A, B)*

1. **for** $i \leftarrow 0$ **to** m **do** $len(i,0) = 0$
2. **for** $j \leftarrow 1$ **to** n **do** $len(0,j) = 0$
3. **for** $i \leftarrow 1$ **to** m **do**
4. **for** $j \leftarrow 1$ **to** n **do**
5. **if** $a_i = b_j$ **then** $\begin{cases} len(i,j) = len(i-1, j-1) + 1 \\ prev(i,j) = " \nwarrow " \end{cases}$
6. **else if** $len(i-1, j) \geq len(i, j-1)$
7. **then** $\begin{cases} len(i,j) = len(i-1, j) \\ prev(i,j) = " \uparrow " \end{cases}$
8. **else** $\begin{cases} len(i,j) = len(i, j-1) \\ prev(i,j) = " \leftarrow " \end{cases}$
9. **return** *len* and *prev*

i		*j*	0	1	2	3	4	5	6	7	8	9	10
				p	*r*	*o*	*v*	*i*	*d*	*e*	*n*	*c*	*e*
0			0	0	0	0	0	0	0	0	0	0	0
1	*p*		0	1	1	1	1	1	1	1	1	1	1
2	*r*		0	1	2	2	2	2	2	2	2	2	2
3	*e*		0	1	2	2	2	2	2	3	3	3	3
4	*s*		0	1	2	2	2	2	2	3	3	3	3
5	*i*		0	1	2	2	2	3	3	3	3	3	3
6	*d*		0	1	2	2	2	3	4	4	4	4	4
7	*e*		0	1	2	2	2	3	4	5	5	5	5
8	*n*		0	1	2	2	2	3	4	5	6	6	6
9	*t*		0	1	2	2	2	3	4	5	6	6	6

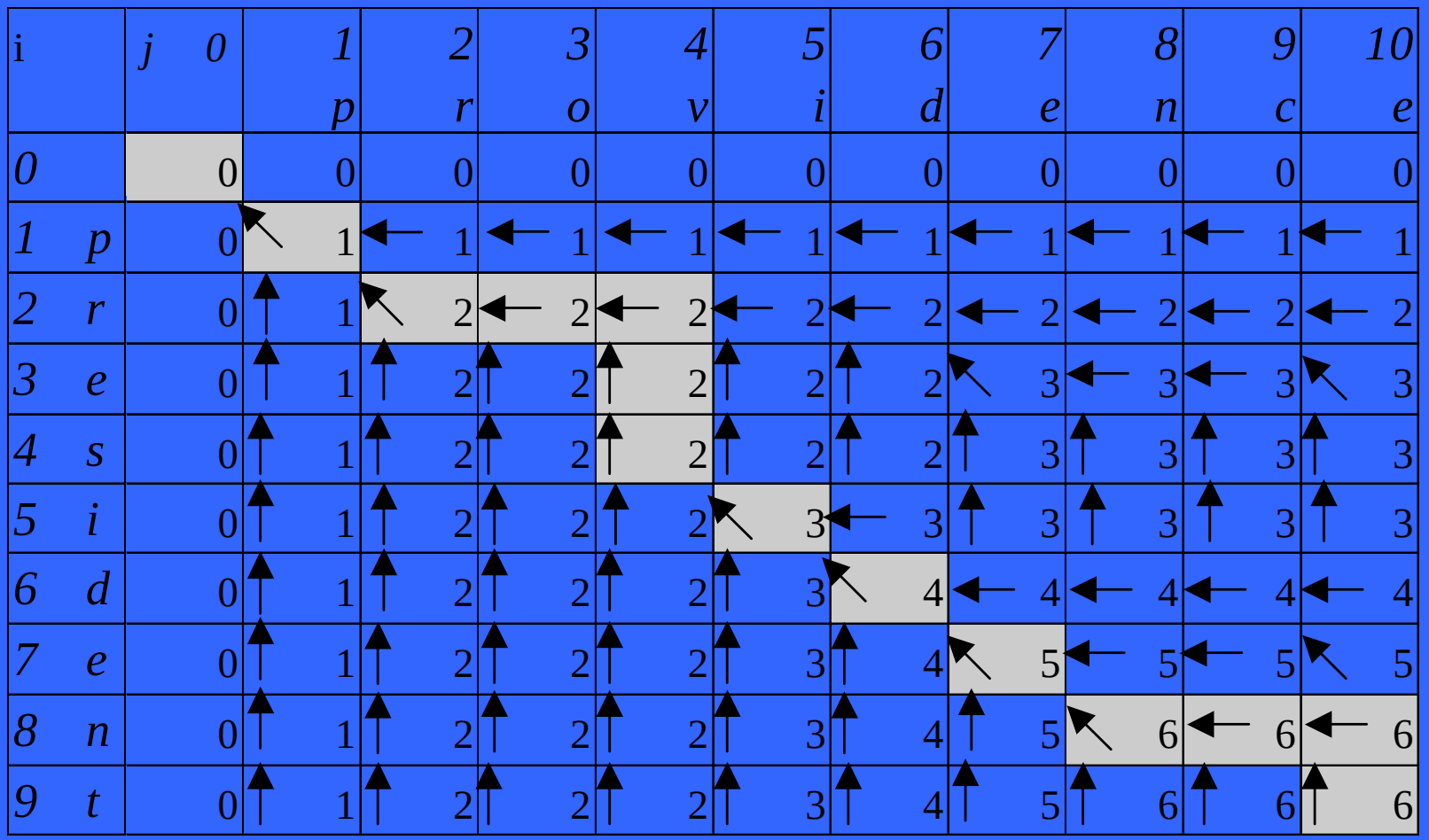
procedure *Output-LCS(A, prev, i, j)*

1 **if** $i = 0$ **or** $j = 0$ **then return**

2 **if** *prev(i, j)=”* ↖ *“* **then** $\begin{bmatrix} Output-LCS(A, prev, i-1, j-1) \\ \text{print} \quad a_i \end{bmatrix}$

3 **else if** *prev(i, j)=”* ↑ *“* **then** *Output-LCS(A, prev, i-1, j)*

4 **else** *Output-LCS(A, prev, i, j-1)*

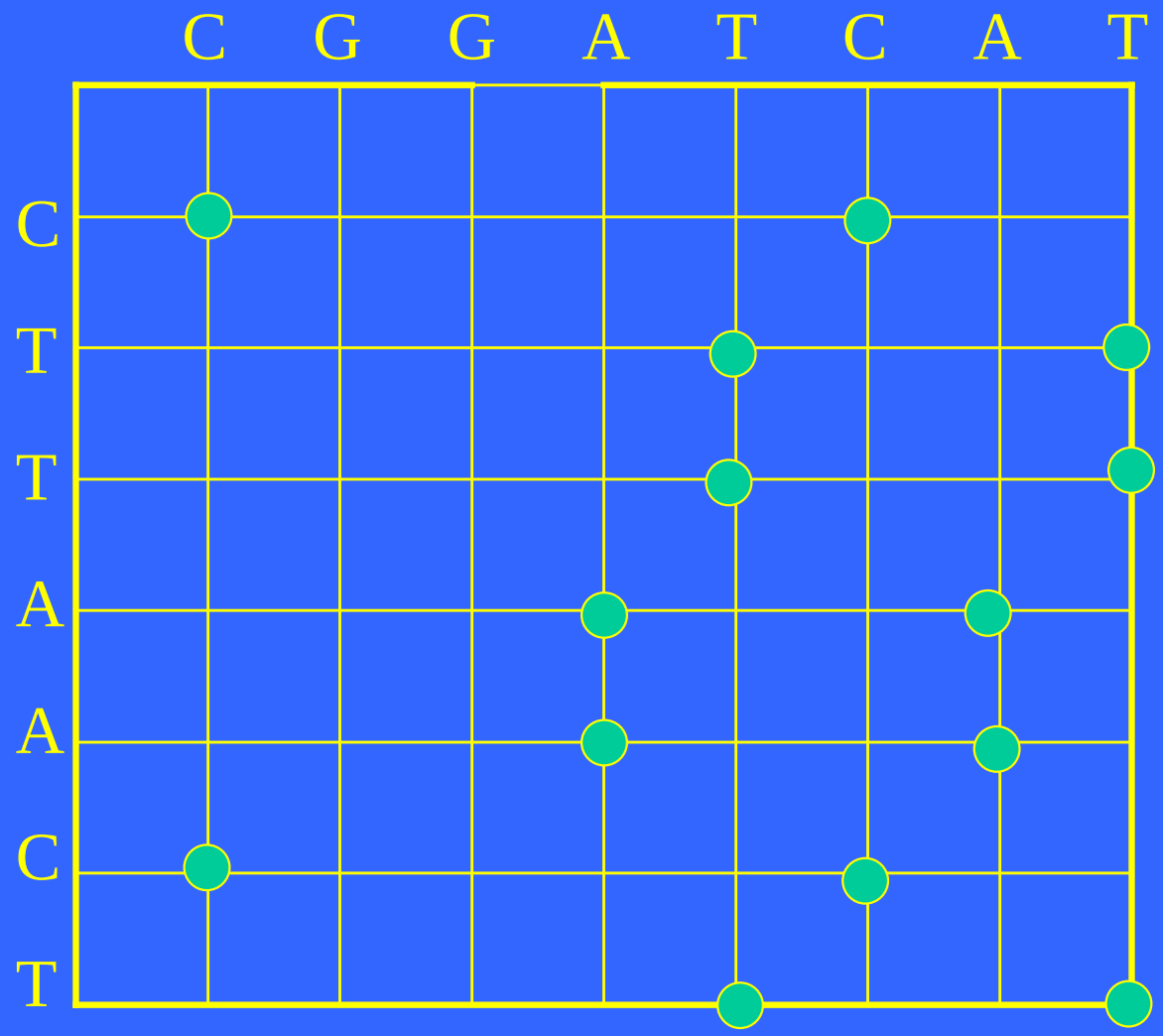


Output: *priden*

Dot Matrix

Sequence A：CTTAACT

Sequence B：CGGATCAT



Pairwise Alignment

Sequence A: CTTAACT
Sequence B: CGGATCAT

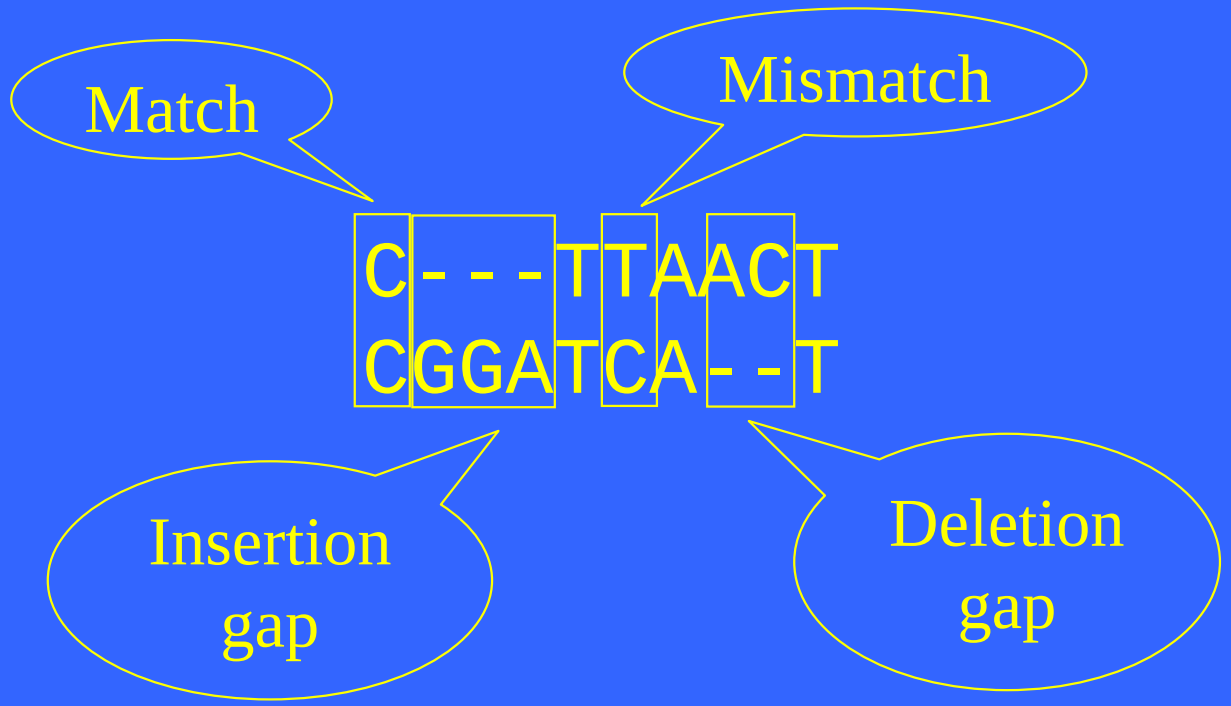
An alignment of A and B:

```
C---TTAACT  <-- Sequence A
CGGATCA--T  <-- Sequence B
```

Pairwise Alignment

Sequence A: CTTAACT
Sequence B: CGGATCAT

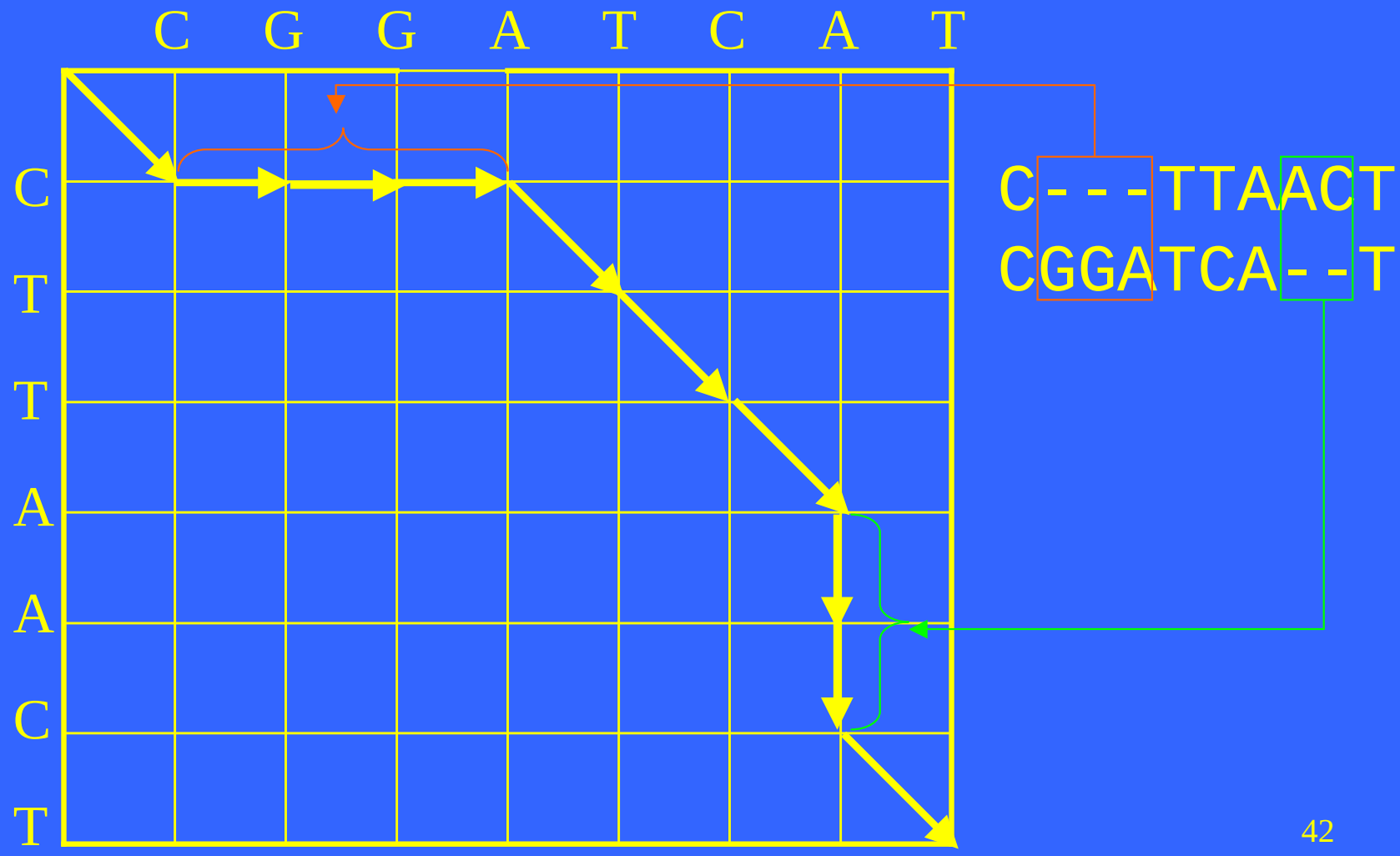
An alignment of A and B:



Alignment Graph

Sequence A: CTTAACT

Sequence B: CGGATCAT



A simple scoring scheme

- Match: +8 ($w(x, y) = 8$, if $x = y$)
- Mismatch: -5 ($w(x, y) = -5$, if $x \neq y$)
- Each gap symbol: -3 ($w(-,x)=w(x,-)=-3$)

C	-	-	-	T	T	A	A	C	T	
C	G	G	A	T	C	A	-	-	T	
+8	-3	-3	-3	+8	-5	+8	-3	-3	+8	= +12

Alignment score

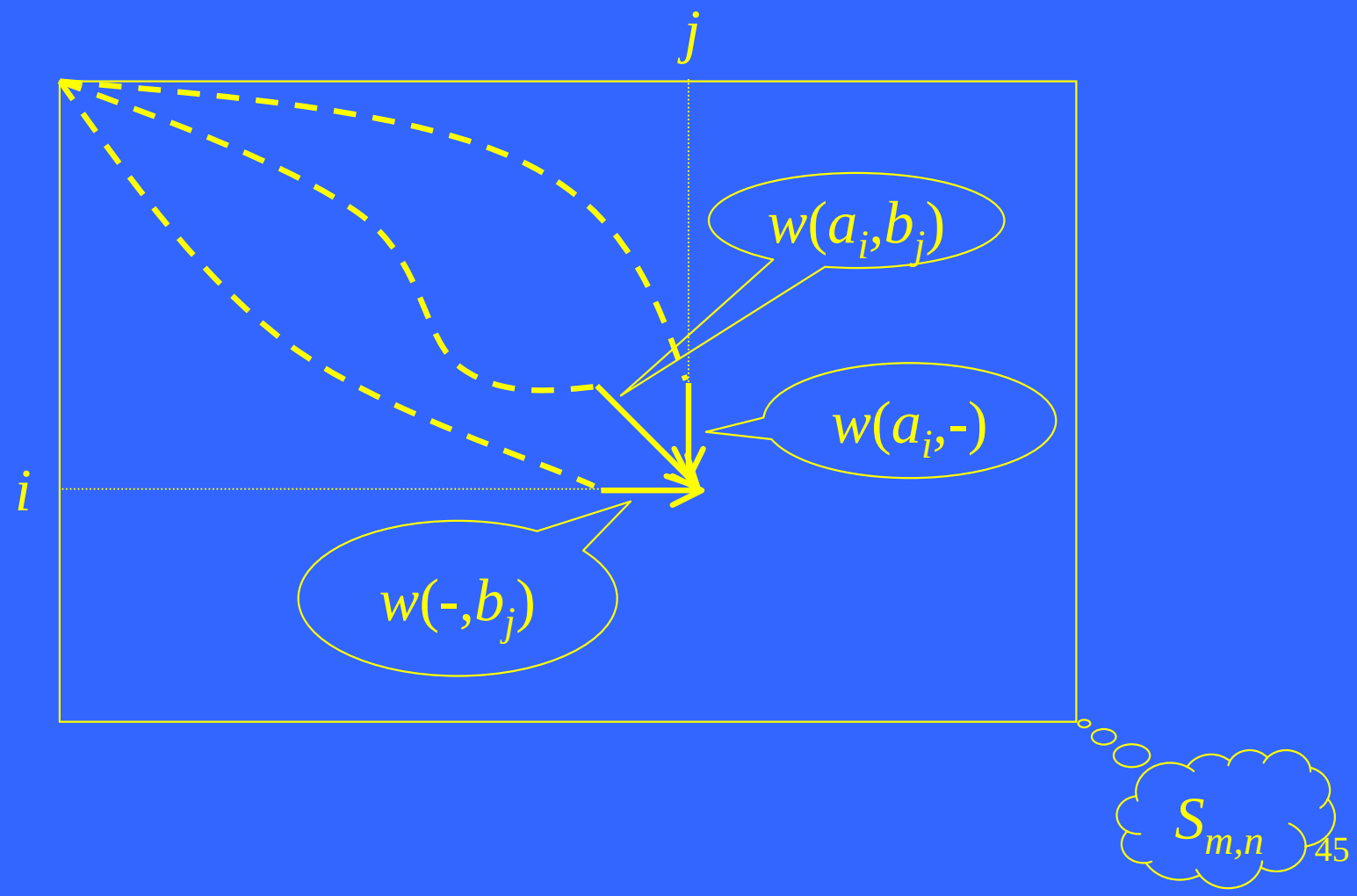
An optimal alignment

-- the alignment of maximum score

- Let A=$a_1a_2\ldots a_m$ and $B=b_1b_2\ldots b_n$.
- $S_{i,j}$: the score of an optimal alignment between
  $a_1a_2\ldots a_i$ and $b_1b_2\ldots b_j$
- With proper initializations, $S_{i,j}$ can be computed as follows.

$$s_{i,j} = \max\begin{cases} s_{i-1,j} + w(a_i,-) \\ s_{i,j-1} + w(-,b_j) \\ s_{i-1,j-1} + w(a_i,b_j) \end{cases}$$

Computing $S_{i,j}$



Initializations

		C	G	G	A	T	C	A	T
	0	-3	-6	-9	-12	-15	-18	-21	-24
C	-3								
T	-6								
T	-9								
A	-12								
A	-15								
C	-18								
T	-21								

$S_{3,5} = ?$

		C	G	G	A	T	C	A	T
	0	-3	-6	-9	-12	-15	-18	-21	-24
C	-3	8	5	2	-1	-4	-7	-10	-13
T	-6	5	3	0	-3	7	4	1	-2
T	-9	2	0	-2	-5	?			
A	-12								
A	-15								
C	-18								
T	-21								

$S_{3,5} = ?$

		C	G	G	A	T	C	A	T
	0	-3	-6	-9	-12	-15	-18	-21	-24
C	-3	8	5	2	-1	-4	-7	-10	-13
T	-6	5	3	0	-3	7	4	1	-2
T	-9	2	0	-2	-5	5	-1	-4	9
A	-12	-1	-3	-5	6	3	0	7	6
A	-15	-4	-6	-8	3	1	-2	8	5
C	-18	-7	-9	-11	0	-2	9	6	3
T	-21	-10	-12	-14	-3	8	6	4	14

optimal score

C T T A A C – T

C G G A T C A T

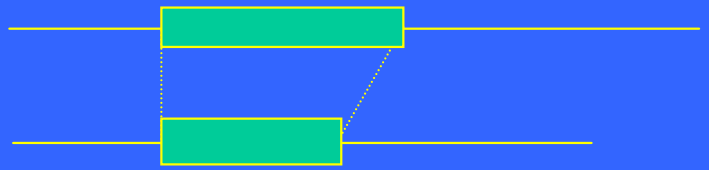
8 – 5 –5 +8 -5 +8 -3 +8 = 14

		C	G	G	A	T	C	A	T
	0	-3	-6	-9	-12	-15	-18	-21	-24
C	-3	8	5	2	-1	-4	-7	-10	-13
T	-6	5	3	0	-3	7	4	1	-2
T	-9	2	0	-2	-5	5	-1	-4	9
A	-12	-1	-3	-5	6	3	0	7	6
A	-15	-4	-6	-8	3	1	-2	8	5
C	-18	-7	-9	-11	0	-2	9	6	3
T	-21	-10	-12	-14	-3	8	6	4	14

Global Alignment vs. Local Alignment

- global alignment:

- local alignment:



An optimal local alignment

- $S_{i,j}$: the score of an optimal local alignment ending at a_i and b_j
- With proper initializations, $S_{i,j}$ can be computed as follows.

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + w(a_i, -) \\ s_{i,j-1} + w(-, b_j) \\ s_{i-1,j-1} + w(a_i, b_j) \end{cases}$$

Match: 8

Mismatch: -5

Gap symbol: -3

local alignment

		C	G	G	A	T	C	A	T
	0	0	0	0	0	0	0	0	0
C	0	8	5	2	0	0	8	5	2
T	0	5	3	0	0	8	5	3	13
T	0	2	0	0	0	8	5	2	11
A	0	0	0	0	8	5	3	?	
A	0								
C	0								
T	0								

local alignment

Match: 8

Mismatch: -5

Gap symbol: -3

		C	G	G	A	T	C	A	T
	0	0	0	0	0	0	0	0	0
C	0	8	5	2	0	0	8	5	2
T	0	5	3	0	0	8	5	3	13
T	0	2	0	0	0	8	5	2	11
A	0	0	0	0	8	5	3	13	10
A	0	0	0	0	8	5	2	11	8
C	0	8	5	2	5	3	13	10	7
T	0	5	3	0	2	13	10	8	18

The best score

A – C - T
A T C A T
8-3+8-3+8 = 18

		C	G	G	A	T	C	A	T
	0	0	0	0	0	0	0	0	0
C	0	8	5	2	0	0	8	5	2
T	0	5	3	0	0	8	5	3	13
T	0	2	0	0	0	8	5	2	11
A	0	0	0	0	8	5	3	13	10
A	0	0	0	0	8	5	2	11	8
C	0	8	5	2	5	3	13	10	7
T	0	5	3	0	2	13	10	8	18

The best score

Affine gap penalties

- Match: +8 ($w(x, y) = 8$, if $x = y$)
- Mismatch: -5 ($w(x, y) = -5$, if $x \neq y$)
- Each gap symbol: -3 ($w(-,x)=w(x,-)=-3$)
- Each gap is charged an extra gap-open penalty: -4.

		-4						-4			
C	-	-	-	T	T	A	A	C	T		
C	G	G	A	T	C	A	-	-	T		
+8	-3	-3	-3	+8	-5	+8	-3	-3	+8	=	+12

Alignment score: 12 – 4 – 4 = 4

Affine gap panalties

- A gap of length k is penalized $x + k \cdot y$.

gap-open penalty

gap-symbol penalty

Three cases for alignment endings:

1. ...x
   ...x
   } an aligned pair

2. ...x
   ...-
   } a deletion

3. ...-
   ...x
   } an insertion

Affine gap penalties

- Let $D(i, j)$ denote the maximum score of any alignment between $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$ ending with a deletion.
- Let $I(i, j)$ denote the maximum score of any alignment between $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$ ending with an insertion.
- Let $S(i, j)$ denote the maximum score of any alignment between $a_1 a_2 \ldots a_i$ and $b_1 b_2 \ldots b_j$.

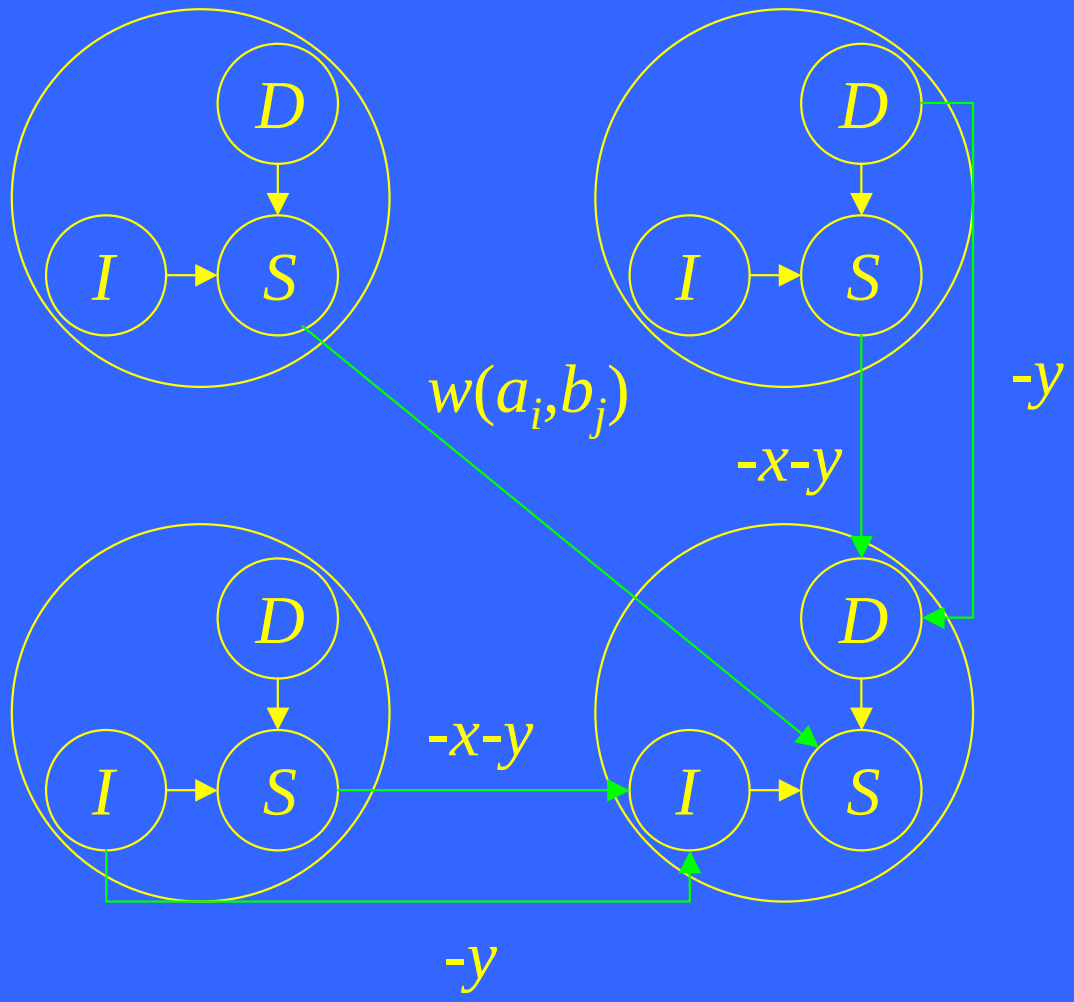
Affine gap penalties

$$D(i,j) = \max\begin{cases} D(i-1,j) - y \\ S(i-1,j) - x - y \end{cases}$$

$$I(i,j) = \max\begin{cases} I(i,j-1) - y \\ S(i,j-1) - x - y \end{cases}$$

$$S(i,j) = \max\begin{cases} S(i-1,j-1) + w(a_i, b_j) \\ D(i,j) \\ I(i,j) \end{cases}$$

Affine gap penalties



k best local alignments

- Smith-Waterman
  (Smith and Waterman, 1981; Waterman and Eggert, 1987)
- FASTA
  (Wilbur and Lipman, 1983; Lipman and Pearson, 1985)
- BLAST
  (Altschul et al., 1990; Altschul et al., 1997)

k best local alignments

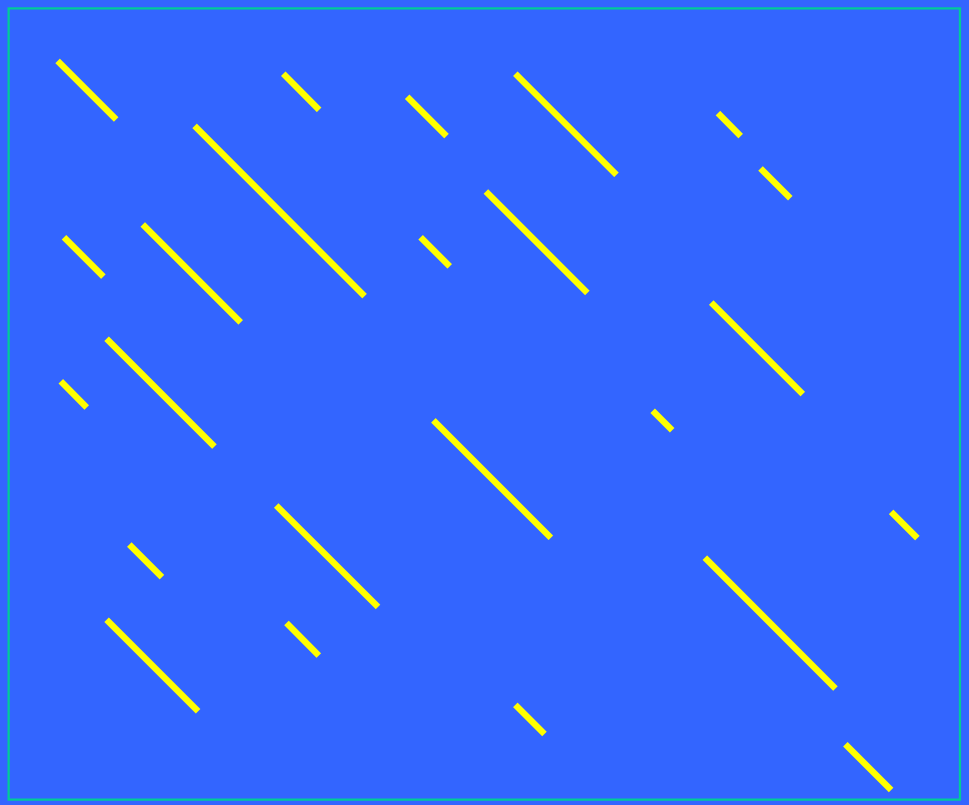
- Smith-Waterman
  (Smith and Waterman, 1981; Waterman and Eggert, 1987)
  - linear-space version：sim (Huang and Miller, 1991)
  - linear-space variants：sim2 (Chao et al., 1995); sim3 (Chao et al., 1997)
- FASTA
  (Wilbur and Lipman, 1983; Lipman and Pearson, 1985)
  - linear-space band alignment (Chao et al., 1992)
- BLAST
  (Altschul et al., 1990; Altschul et al., 1997)
  - restricted affine gap penalties (Chao, 1999)

FASTA

1) Find runs of identities, and identify regions with the highest density of identities.
2) Re-score using PAM matrix, and keep top scoring segments.
3) Eliminate segments that are unlikely to be part of the alignment.
4) Optimize the alignment in a band.

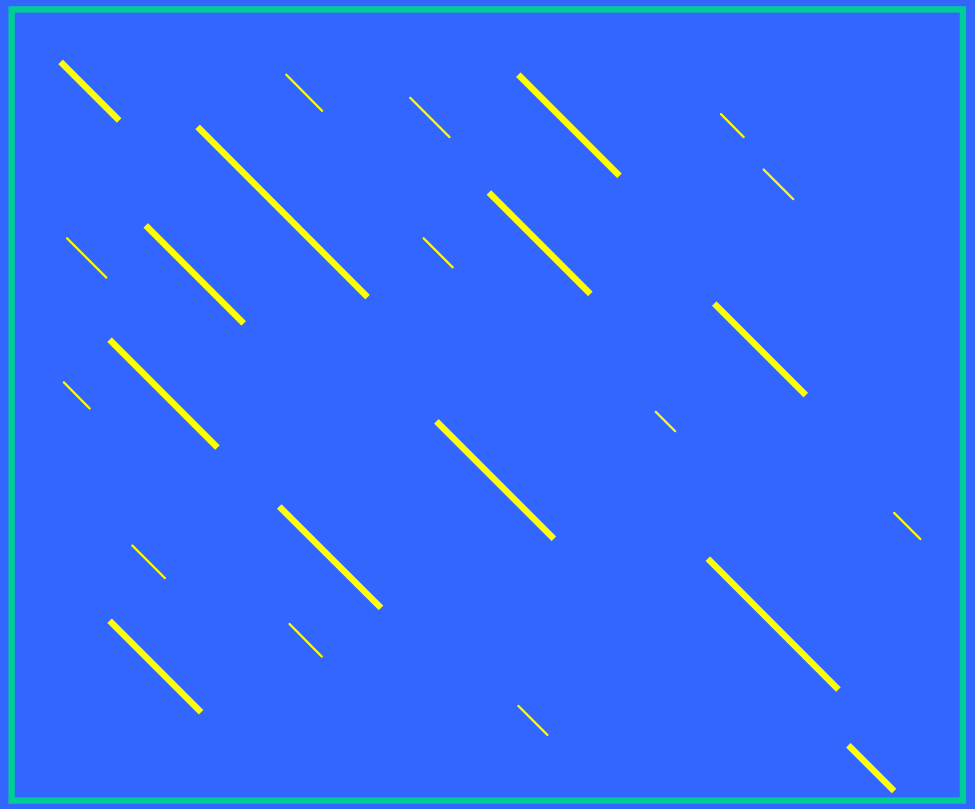
FASTA

Step 1: Find runes of identities, and identify regions with the highest density of identities.



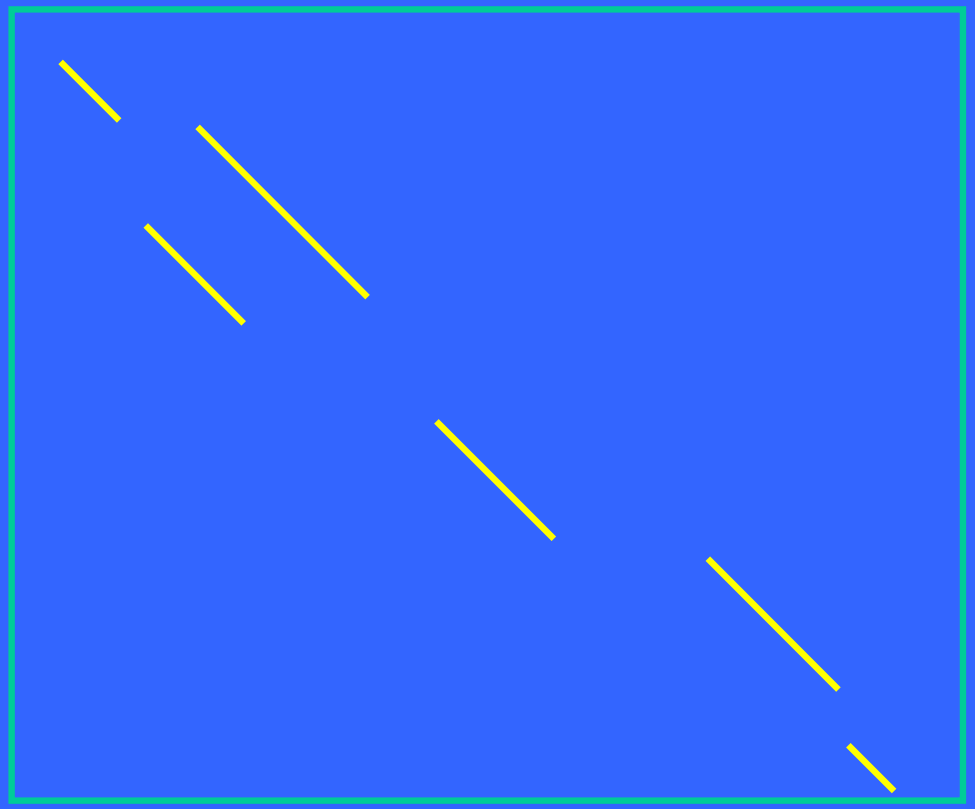
FASTA

Step 2: Re-score using PAM matrix, and keep top scoring segments.



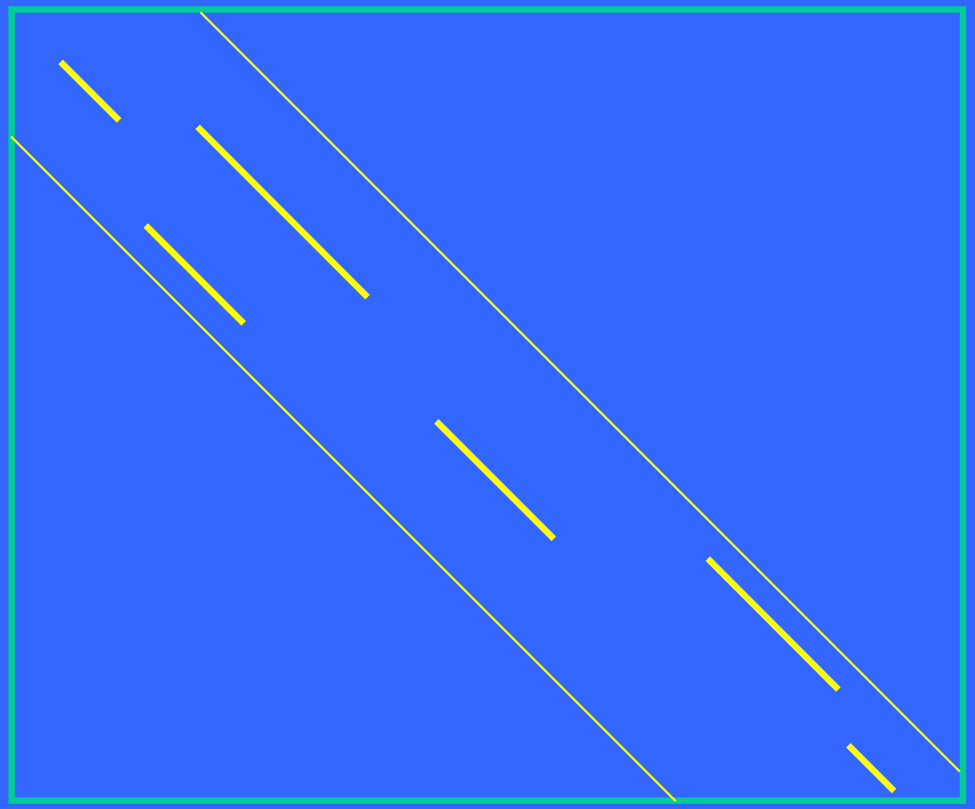
FASTA

Step 3: Eliminate segments that are unlikely to be part of the alignment.



FASTA

Step 4: Optimize the alignment in a band.



BLAST

1) Build the hash table for Sequence A.
2) Scan Sequence B for hits.
3) Extend hits.

BLAST

Step 1: Build the hash table for Sequence A. (3-tuple example)

For DNA sequences:

```
Seq. A = AGATCGAT
         12345678
```

For protein sequences:

```
Seq. A = ELVIS
```

Add *xyz* to the hash table if $Score(xyz, ELV) \geqq T$;
Add *xyz* to the hash table if $Score(xyz, LVI) \geqq T$;
Add *xyz* to the hash table if $Score(xyz, VIS) \geqq T$;

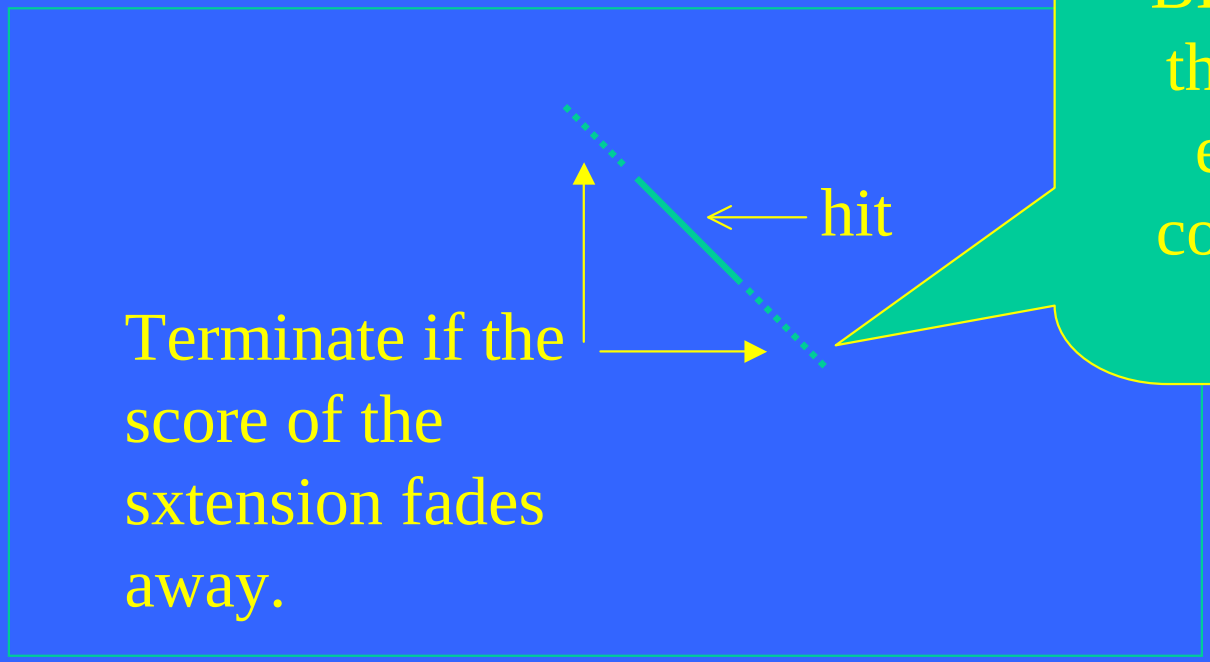
BLAST

Step2: Scan sequence B for hits.

BLAST

Step2: Scan sequence B for hits.

Step 3: Extend hits.



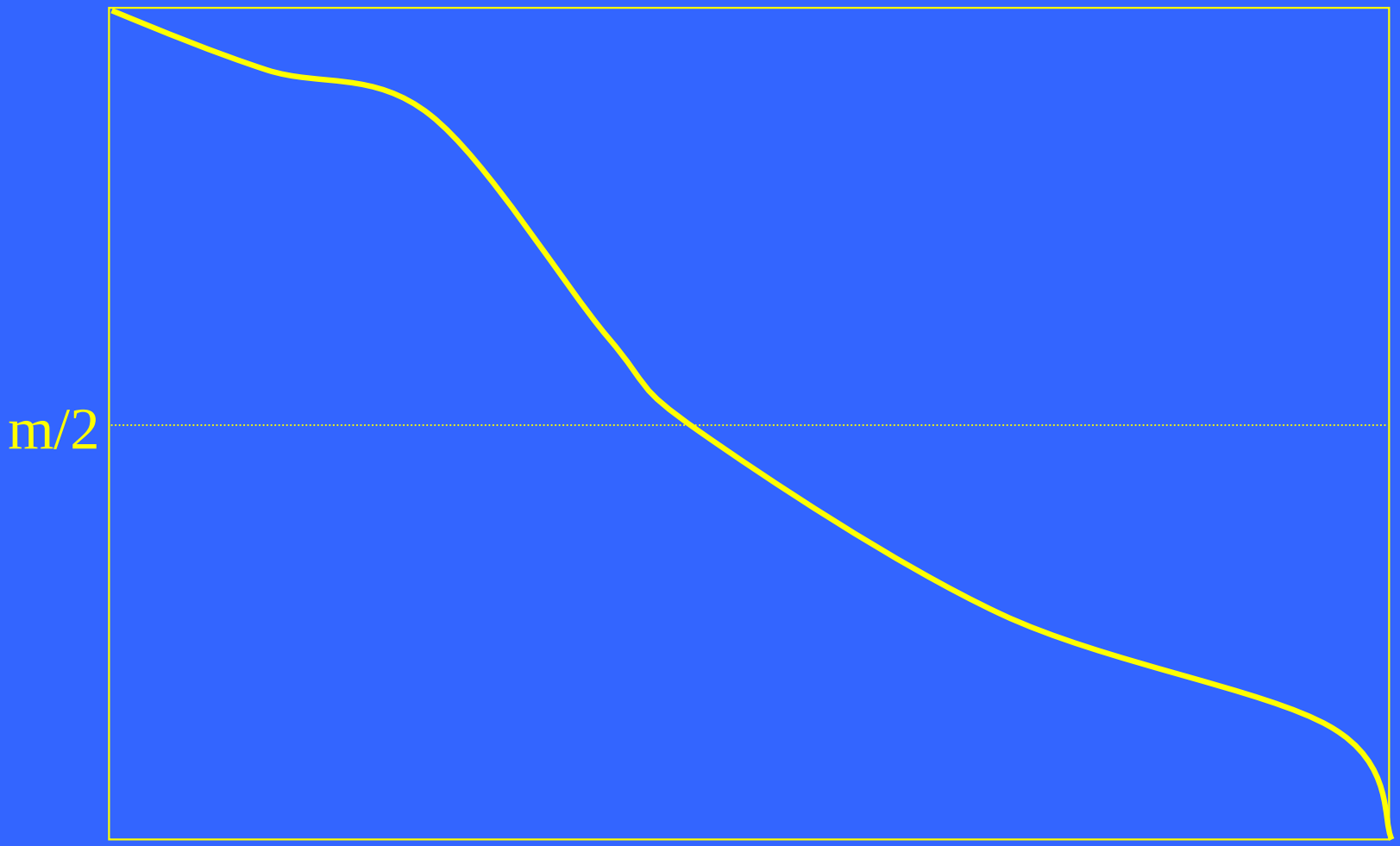
BLAST 2.0 saves the time spent in extension, and considers gapped alignments.

Remarks

- Filtering is based on the observation that a good alignment usually includes short identical or very similar fragments.
- The idea of filtration was used in both FASTA and BLAST.

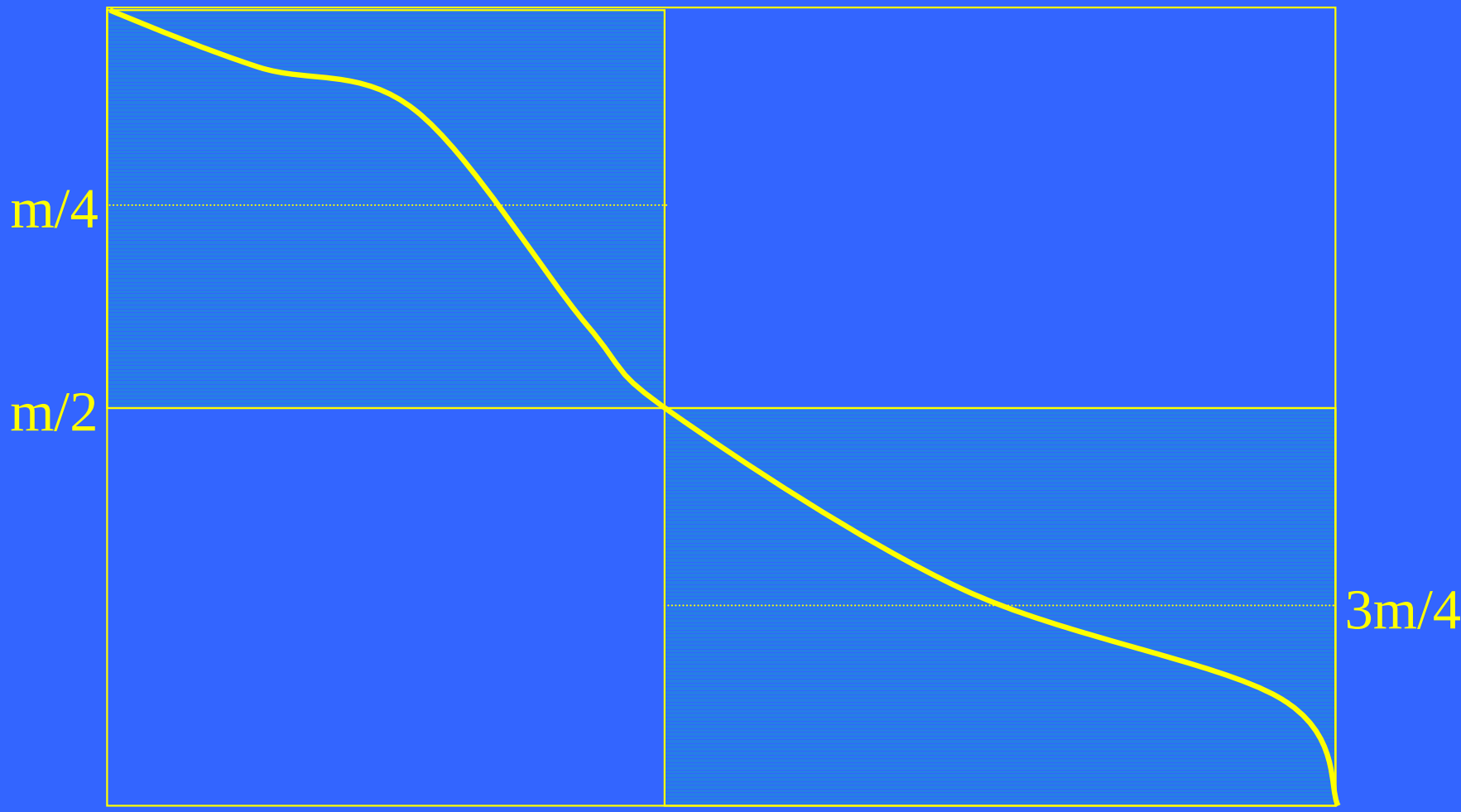
Linear-space ideas

Hirschberg, 1975; Myers and Miller, 1988



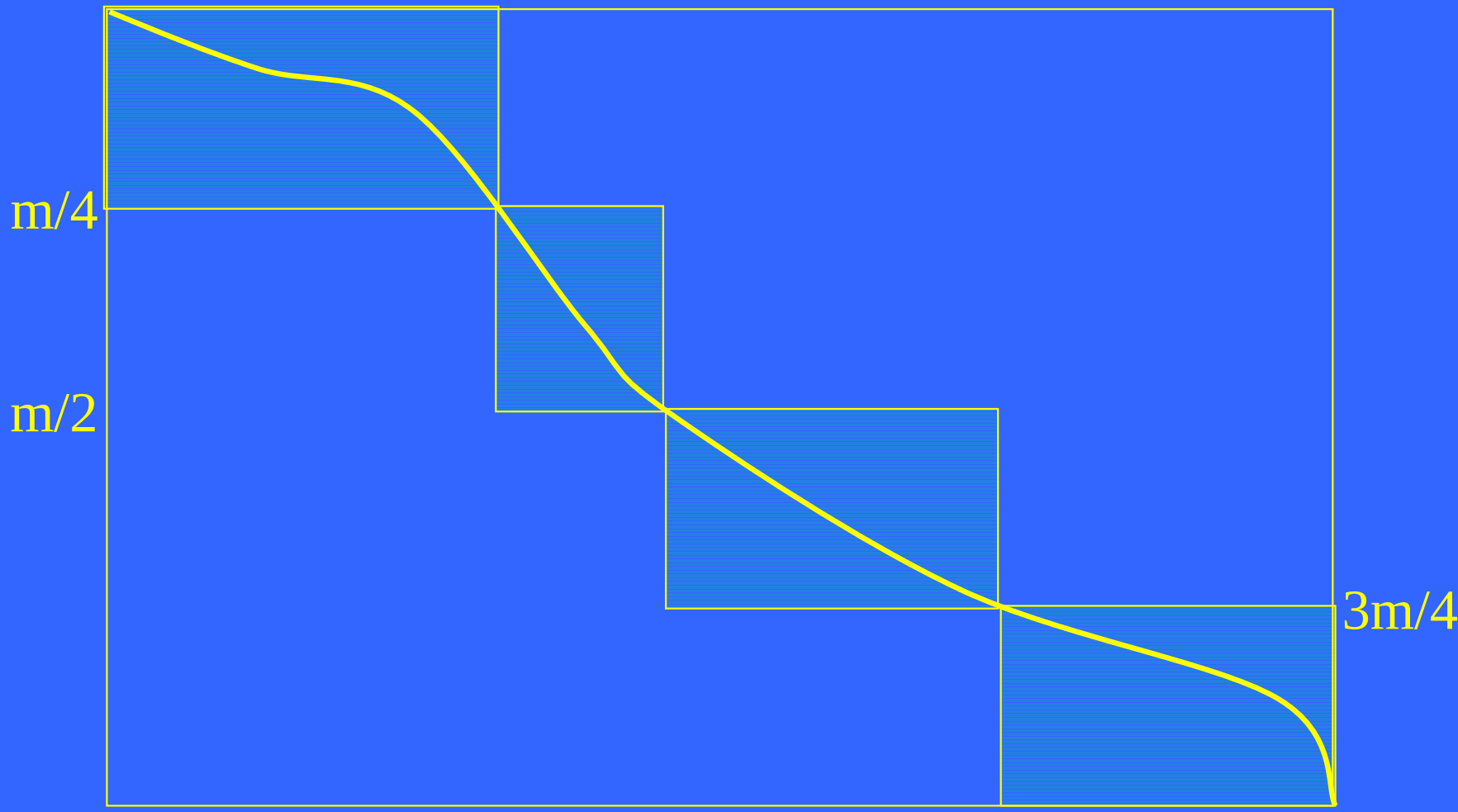
Two subproblems

½ original problem size



Four subproblems

¼ original problem size



Time and Space Complexity

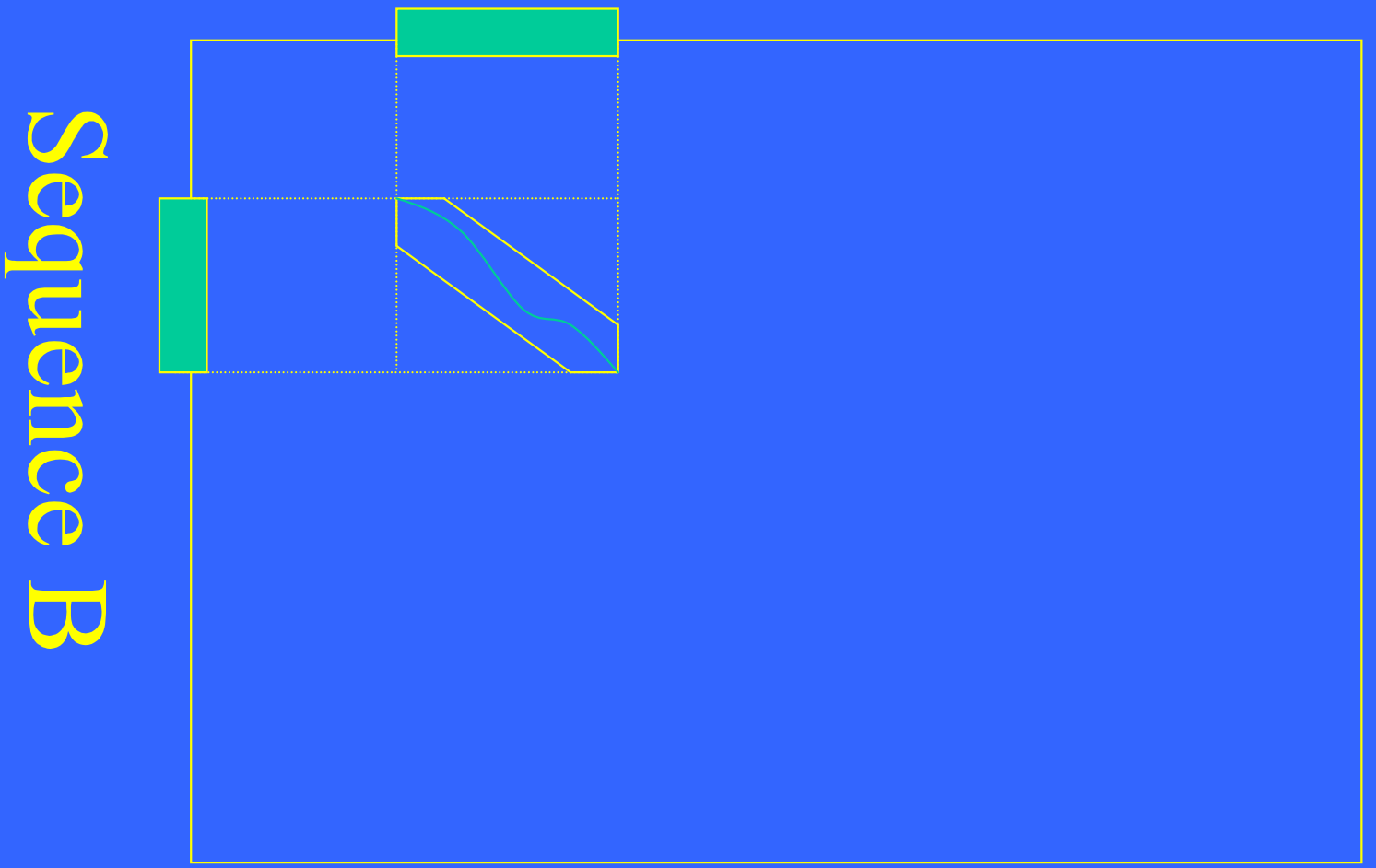
- Space: O(M+N)
- Time:

$$O(MN)*(\underbrace{1+ \tfrac{1}{2} + \tfrac{1}{4} + \ldots}_{2}) = O(MN)$$

Band Alignment

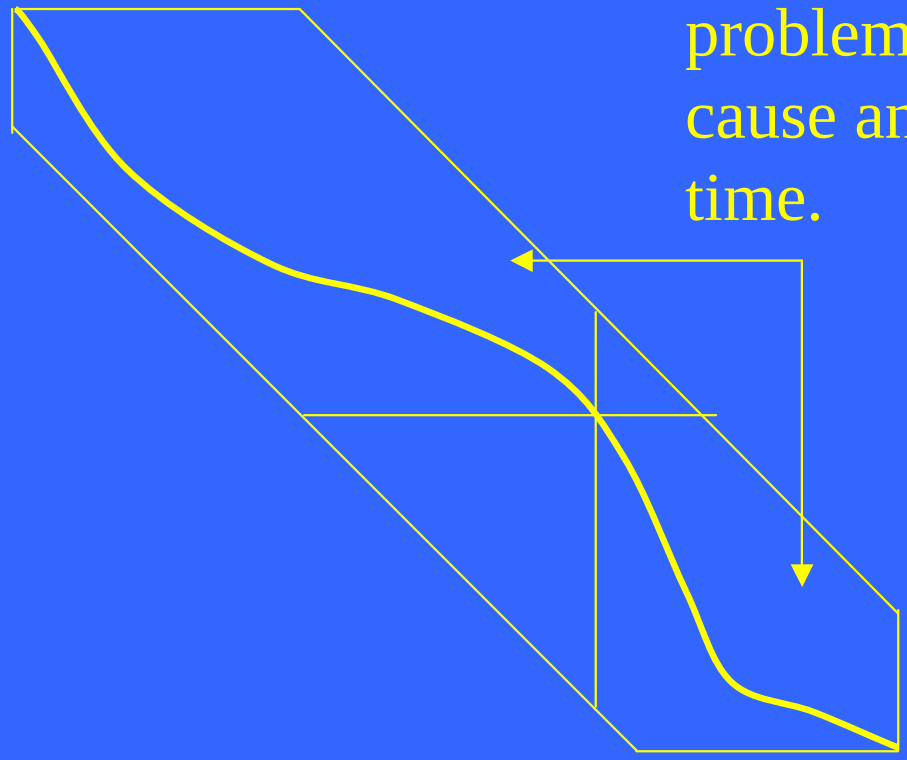
(Joint work with W. Pearson and W. Miller)

Sequence A



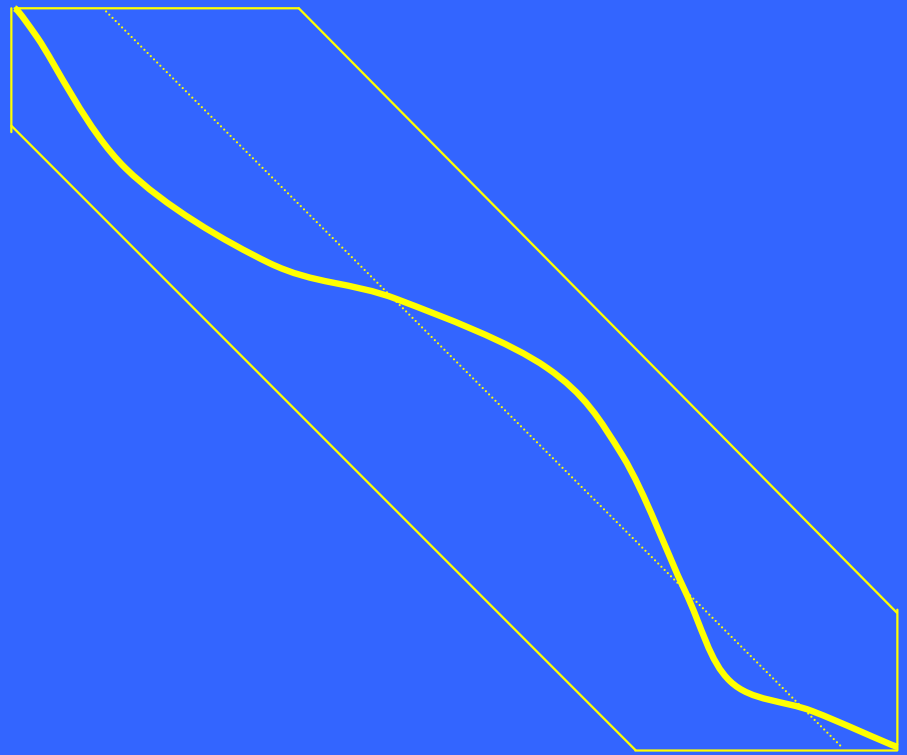
Sequence B

Band Alignment in Linear Space



The remaining subproblems are no longer only half of the original problem. In worst case, this could cause an additional log *n* factor in time.

Band Alignment in Linear Space



Multiple sequence alignment (MSA)

- The multiple sequence alignment problem is to simultaneously align more than two sequences.

```
Seq1: GCTC        GC-TC
Seq2: AC          A---C
Seq3: GATC        G-ATC
```

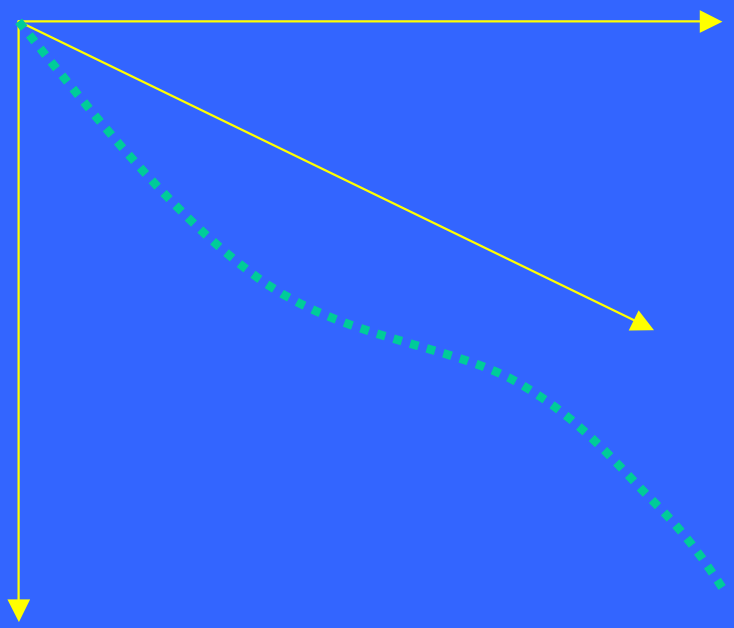
How to score an MSA?

- *Sum-of-Pairs* (SP-score)

$$\text{Score}\begin{pmatrix}\texttt{GC-TC}\\ \texttt{A---C}\\ \texttt{G-ATC}\end{pmatrix} = \text{Score}\begin{pmatrix}\texttt{GC-TC}\\ \texttt{A---C}\end{pmatrix} + \text{Score}\begin{pmatrix}\texttt{GC-TC}\\ \texttt{G-ATC}\end{pmatrix} + \text{Score}\begin{pmatrix}\texttt{A---C}\\ \texttt{G-ATC}\end{pmatrix}$$

MSA for three sequences

- an $O(n^3)$ algorithm

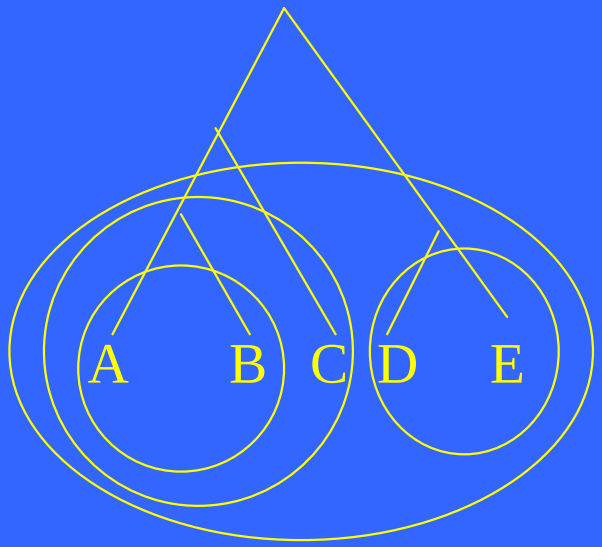


General MSA

- For k sequences of length n: $O(n^k)$
- NP-Complete (Wang and Jiang)
- The exact multiple alignment algorithms for many sequences are not feasible.
- Some approximation algorithms are given. (*e.g.*, 2- l/k for any fixed l by Bafna *et al.*)

Progressive alignment

- A heuristic approach proposed by Feng and Doolittle.
- It iteratively merges the most similar pairs.
- “Once a gap, always a gap”



The time for progressive alignment in most cases is roughly the order of the time for computing all pairwise alignment, i.e., $O(k^2n^2)$.

Concluding remarks

- Three essential components of the dynamic-programming approach:
  - the recurrence relation
  - the tabular computation
  - the traceback
- The dynamic-programming approach has been used in a vast number of computational problems in bioinformatics.