

BMEG3102 Bioinformatics

Lecture 2. Sequence Alignment and Searching (1/2)



Qi Dou

Email: qidou@cuhk.edu.hk

Office: Room 1014, 10/F, SHB

BMEG3102 Bioinformatics

The Chinese University of Hong Kong



1. Problems related to sequences

- Core: Sequence alignment

2. Sequence alignment

- Problem components
- Difficulty
- Methods (focused on optimal methods in this lecture)
 - Global alignment
 - Local alignment



Part 1

PROBLEMS RELATED TO SEQUENCES



- Many biological objects are represented by sequences (text strings)
 - DNA sequences (A, C, G, T)
 - E.g., TATACATTAG
 - RNA sequences (A, C, G, U)
 - E.g., UAUACAUUAG
 - Protein sequences (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y)
 - E.g., YTL
 - ...



- General assumption: Sequences with more similar text strings also have more similar biological properties
- Why?



- General assumption: Sequences with more similar text strings also have more similar biological properties
- Why?
 - Evolutionarily: Either
 - They have diverged from a common ancestor for a shorter time or they have still preserved their original properties
 - “Selective pressure” – conservation suggests importance (changes are unfavorable)
 - At the molecular level:
 - Structures are more similar
 - Containing similar functional units (domains)
 - ...



1. Biological question: “How similar are the genomes of humans and mice?”
 - Computational question: Given two sequences r and s , compute their similarity, $\text{sim}(r, s)$



1. Biological question: “How similar are the genomes of humans and mice?”
 - Computational question: Given two sequences r and s , compute their similarity, $\text{sim}(r, s)$

2. Biological question: “This gene causes obesity in mice. Do humans have the same gene?”
 - Computational question: Given a sequence r (the mouse gene) and a database D of sequences (all human genes), find sequences s in D where $\text{sim}(r, s)$ is above a threshold



1. Biological question: “How similar are the genomes of humans and mice?”
 - Computational question: Given two sequences r and s , compute their similarity, $\text{sim}(r, s)$

2. Biological question: “This gene causes obesity in mice. Do humans have the same gene?”
 - Computational question: Given a sequence r (the mouse gene) and a database D of sequences (all human genes), find sequences s in D where $\text{sim}(r, s)$ is above a threshold
 - The simplest way is to compute $\text{sim}(r, s)$ for each s one by one



3. Biological question: “We know some mutations of this gene cause sickle-cell anemia. We have the sequences of 100 patients and 100 normal people. Let’s find out the disease-causing mutations.”
 - Computational question: Given two sets of sequences of different lengths, find an ***alignment*** that maximizes the overall similarity (so that nucleotides in a column were likely originated from the same nucleotide in their ancestral sequence). Then look for mutations that are unique to one group.



3. Biological question: “We know some mutations of this gene cause sickle-cell anemia. We have the sequences of 100 patients and 100 normal people. Let’s find out the disease-causing mutations.”

—Computational question: Given two sets of sequences of different lengths, find an ***alignment*** that maximizes the overall similarity (so that nucleotides in a column were likely originated from the same nucleotide in their ancestral sequence). Then look for mutations that are unique to one group.

Patients ACGCGT
CGCGT
ACGCGA

Controls AGCTT
ACGCTT
ACGCTA



3. Biological question: “We know some mutations of this gene cause sickle-cell anemia. We have the sequences of 100 patients and 100 normal people. Let’s find out the disease-causing mutations.”

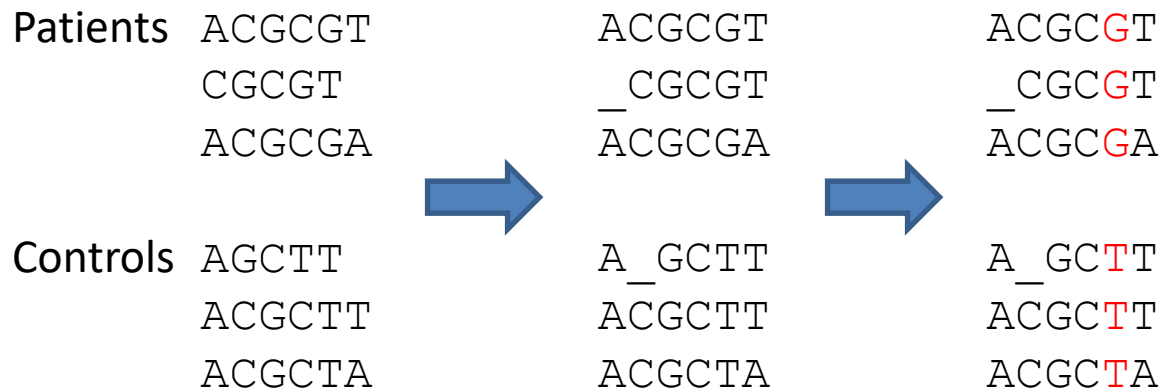
—Computational question: Given two sets of sequences of different lengths, find an ***alignment*** that maximizes the overall similarity (so that nucleotides in a column were likely originated from the same nucleotide in their ancestral sequence). Then look for mutations that are unique to one group.

Patients	ACGCGT		ACGCGT
	CGCGT		_CGCGT
	ACGCGA		ACGCGA
Controls	AGCTT		A_GCTT
	ACGCTT		ACGCTT
	ACGCTA		ACGCTA



3. Biological question: “We know some mutations of this gene cause sickle-cell anemia. We have the sequences of 100 patients and 100 normal people. Let’s find out the disease-causing mutations.”

—Computational question: Given two sets of sequences of different lengths, find an **alignment** that maximizes the overall similarity (so that nucleotides in a column were likely originated from the same nucleotide in their ancestral sequence). Then look for mutations that are unique to one group.



Performing the alignment makes it easy to compute the similarity between two sequences.



- In all these problems, we need to get either
 - The similarity of sequences OR
 - The alignment of sequences
- Similarity can be easily computed after performing an alignment
 - We will study how it can be done in general
- Therefore, sequence alignment is a core topic in the study of sequences



Part 2

SEQUENCE ALIGNMENT



- Definition:
 - Given a set of sequences, an alignment is the same set of sequences with zero or more gaps inserted into them so that
 1. They all have the same length afterwards
 2. For each column (also called a position or a site), at least one of the resulting sequences is not a gap



- Definition:
 - Given a set of sequences, an alignment is the same set of sequences with zero or more gaps inserted into them so that
 1. They all have the same length afterwards
 2. For each column (also called a position or a site), at least one of the resulting sequences is not a gap
- Example and terminology:

```
CGGTCACCTTGA
CGGTCCTTGT
```



- Definition:
 - Given a set of sequences, an alignment is the same set of sequences with zero or more gaps inserted into them so that
 1. They all have the same length afterwards
 2. For each column (also called a position or a site), at least one of the resulting sequences is not a gap
- Example and terminology:

CGGTCACTTGA
CGGTCCTTGT

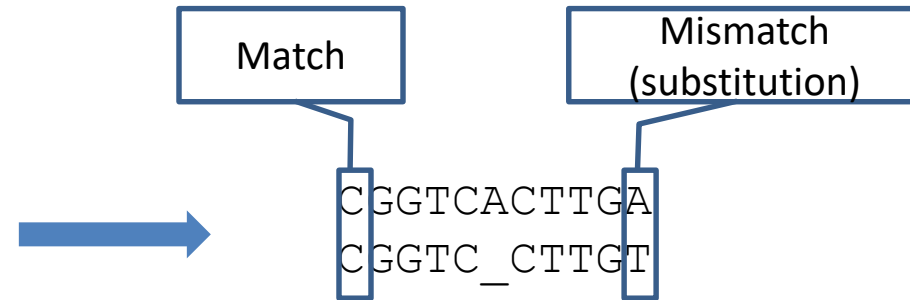


CGGTCACTTGA
CGGTC _CTTGT



- Definition:
 - Given a set of sequences, an alignment is the same set of sequences with zero or more gaps inserted into them so that
 1. They all have the same length afterwards
 2. For each column (also called a position or a site), at least one of the resulting sequences is not a gap
- Example and terminology:

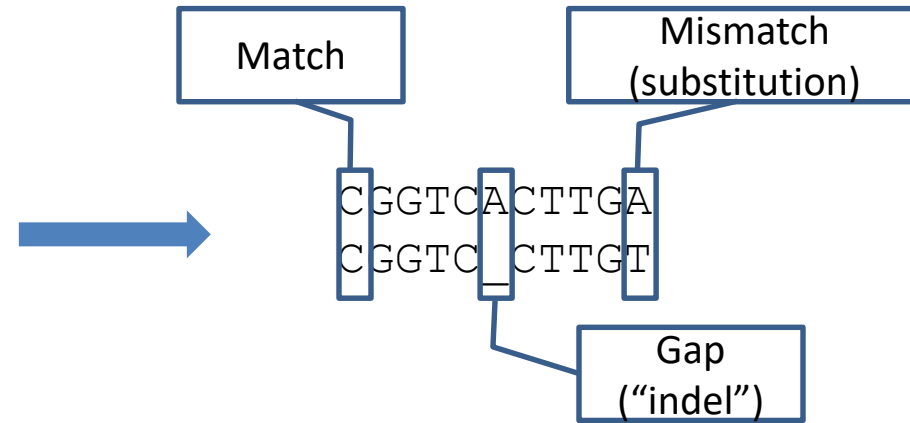
CGGTCACCTTGA
CGGTCCTTGT





- Definition:
 - Given a set of sequences, an alignment is the same set of sequences with zero or more gaps inserted into them so that
 1. They all have the same length afterwards
 2. For each column (also called a position or a site), at least one of the resulting sequences is not a gap
- Example and terminology:

CGGTCACTTGA
CGGTCCTTGT

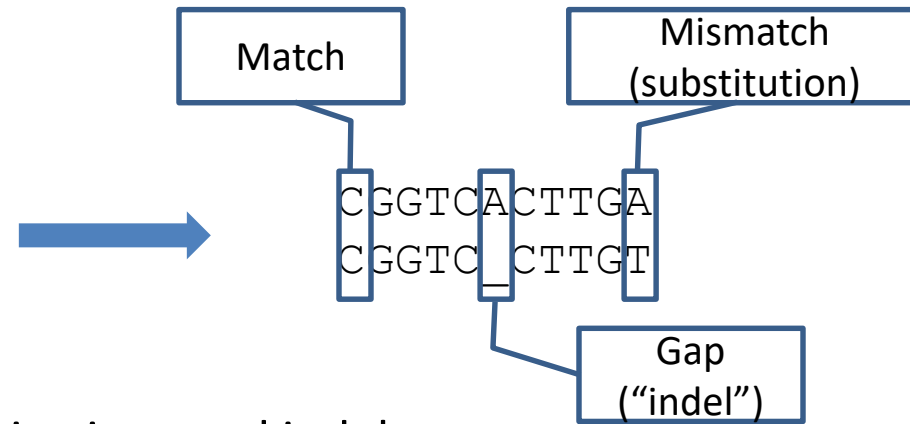




- Definition:
 - Given a set of sequences, an alignment is the same set of sequences with zero or more gaps inserted into them so that
 1. They all have the same length afterwards
 2. For each column (also called a position or a site), at least one of the resulting sequences is not a gap

- Example and terminology:

CGGTCACTTGA
CGGTCCTTGT



- A good alignment is one with few substitutions and indels
 - the sequence alignment problem is to find out the optimal alignment, i.e., the one with the highest score (to be defined)



- Number of sequences

- 2 sequences: Pairwise sequence alignment

TACCG
CAC_T

- >2 sequences: Multiple sequence alignment

TACCG
CAC_T
T_CCT



- Number of sequences
 - 2 sequences: Pairwise sequence alignment
 - >2 sequences: Multiple sequence alignment
- Which part to align
 - Whole sequences: Global alignment
 - Parts of sequences: Local alignment

TACCG
CAC_T

TACCG
CAC_T
T_CCT

1 TACCG 5
1 CAC_T 4

2 AC 3
2 AC 3



Sequence alignment problems

- Number of sequences
 - 2 sequences: Pairwise sequence alignment
 - >2 sequences: Multiple sequence alignment
- Which part to align
 - Whole sequences: Global alignment
 - Parts of sequences: Local alignment
- How to compute similarity
 - Ways to compute substitution scores
 - Ways to compute gap penalties

TACCG
CAC_T

TACCG
CAC_T
T_CCT

1 TACCG 5
1 CAC_T 4

2 AC 3
2 AC 3



- Consider a simple setting:
 - 2 sequences (pairwise alignment)
 - Align whole sequences (global alignment)
 - Alignment score
 - Match: +1 alignment score;
 - Otherwise (mismatch or indel): -1 alignment score

An example



- ACG **vs.** AGG

An example



- ACG **vs.** AGG
 - A possible alignment:
AC_G
AGG_
Alignment score = -2

An example



- ACG vs. AGG

- A possible alignment:

- AC _ G

- AGG _

- Alignment score = -2

- Another possible alignment:

- ACG

- AGG

- Alignment score = 1



- ACG vs. AGG

- A possible alignment:

- AC _ G

- AGG _

- Alignment score = -2

- Another possible alignment:

- ACG

- AGG

- Alignment score = 1

- Questions:

1. How can we find out the optimal alignment(s), i.e., the one(s) with highest alignment score?
2. If the two sequences have lengths m and n respectively, how many possible alignments are there? (If the number is small, we can simply enumerate all possible alignments in a brute-force manner to find out the best one(s).)



- Example:

ACG

AGG



- Example:

ACG
AGG

No gaps
(1 case)

ACG
AGG



- Example:

ACG
AGG

No gaps
(1 case)

ACG
AGG

1 gap
(12 cases)

_ACG	_ACG	_ACG	A_CG	A_CG	A_CG
A_GG	AG_G	AGG_	_AGG	AG_G	AGG_
AC_G	AC_G	AC_G	ACG_	ACG_	ACG_
_AGG	A_GG	AGG_	_AGG	A_GG	AG_G

Note: Here we consider them as different alignments



- Example:

Note: Here we consider them as different alignments

ACG
AGG

No gaps
(1 case)

ACG
AGG

1 gap
(12 cases)

<u> </u> ACG	<u> </u> ACG	<u> </u> ACG	A <u> </u> CG	A <u> </u> CG	A <u> </u> CG
A <u> </u> GG	AG <u> </u> G	AGG <u> </u>	<u> </u> AGG	AG <u> </u> G	AGG <u> </u>
AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	ACG <u> </u>	ACG <u> </u>	ACG <u> </u>
<u> </u> AGG	A <u> </u> GG	AGG <u> </u>	<u> </u> AGG	A <u> </u> GG	AG <u> </u> G

2 gaps (30 cases)

<u> </u> <u> </u> ACG	<u> </u> <u> </u> ACG	<u> </u> <u> </u> ACG	<u> </u> A <u> </u> CG	<u> </u> A <u> </u> CG	<u> </u> A <u> </u> CG	<u> </u> AC <u> </u> G	<u> </u> AC <u> </u> G	<u> </u> AC <u> </u> G	<u> </u> ACG <u> </u>
AG <u> </u> <u> </u> G	AG <u> </u> <u> </u> G	AGG <u> </u> <u> </u>	A <u> </u> <u> </u> G <u> </u> G	A <u> </u> <u> </u> GG <u> </u>	AGG <u> </u> <u> </u>	A <u> </u> <u> </u> GG	A <u> </u> <u> </u> GG	AG <u> </u> <u> </u> G	A <u> </u> <u> </u> GG
<u> </u> ACG <u> </u>	<u> </u> ACG <u> </u>	A <u> </u> <u> </u> CG	A <u> </u> <u> </u> CG	A <u> </u> <u> </u> CG	A <u> </u> C <u> </u> G	A <u> </u> C <u> </u> G	A <u> </u> C <u> </u> G	A <u> </u> CG <u> </u>	A <u> </u> CG <u> </u>
A <u> </u> G <u> </u> G	AG <u> </u> <u> </u> G	<u> </u> AG <u> </u> G	<u> </u> AGG <u> </u>	AGG <u> </u> <u> </u>	<u> </u> A <u> </u> GG	<u> </u> AGG <u> </u>	AG <u> </u> <u> </u> G	<u> </u> A <u> </u> GG	<u> </u> AG <u> </u> G
A <u> </u> CG <u> </u>	AC <u> </u> <u> </u> G	AC <u> </u> <u> </u> G	AC <u> </u> <u> </u> G	AC <u> </u> G <u> </u>	AC <u> </u> G <u> </u>	AC <u> </u> G <u> </u>	ACG <u> </u> <u> </u>	ACG <u> </u> <u> </u>	ACG <u> </u> <u> </u>
AG <u> </u> <u> </u> G	<u> </u> <u> </u> AGG	<u> </u> AGG <u> </u>	A <u> </u> GG <u> </u>	<u> </u> AGG	<u> </u> AG <u> </u> G	A <u> </u> G <u> </u> G	<u> </u> AGG	<u> </u> A <u> </u> GG	A <u> </u> <u> </u> GG



- Example:

Note: Here we consider them as different alignments

ACG
AGG

No gaps
(1 case)

ACG
AGG

1 gap
(12 cases)

<u> </u> ACG	<u> </u> ACG	<u> </u> ACG	A <u> </u> CG	A <u> </u> CG	A <u> </u> CG
A <u> </u> GG	AG <u> </u> G	AGG <u> </u>	<u> </u> AGG	AG <u> </u> G	AGG <u> </u>
AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	ACG <u> </u>	ACG <u> </u>	ACG <u> </u>
<u> </u> AGG	A <u> </u> GG	AGG <u> </u>	<u> </u> AGG	A <u> </u> GG	AG <u> </u> G

2 gaps (30 cases)

<u> </u> ACG	<u> </u> ACG	<u> </u> ACG	A <u> </u> CG	A <u> </u> CG	A <u> </u> CG	AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	ACG <u> </u>
AG <u> </u> G	AG <u> </u> G	AGG <u> </u>	A <u> </u> G <u> </u> G	A <u> </u> GG <u> </u>	AGG <u> </u>	A <u> </u> GG	A <u> </u> GG	AG <u> </u> G	A <u> </u> GG
<u> </u> ACG	<u> </u> ACG	A <u> </u> CG	A <u> </u> CG	A <u> </u> CG	A <u> </u> C <u> </u> G	A <u> </u> C <u> </u> G	A <u> </u> C <u> </u> G	A <u> </u> CG	A <u> </u> CG
A <u> </u> G <u> </u> G	AG <u> </u> G	<u> </u> AG <u> </u> G	<u> </u> AGG	AGG <u> </u>	<u> </u> A <u> </u> GG	<u> </u> AGG	AG <u> </u> G	<u> </u> A <u> </u> GG	<u> </u> AG <u> </u> G
A <u> </u> CG	AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	ACG	ACG	ACG
AG <u> </u> G	<u> </u> AGG	<u> </u> AGG	A <u> </u> GG	<u> </u> AGG	<u> </u> AG <u> </u> G	A <u> </u> G <u> </u> G	<u> </u> AGG	<u> </u> A <u> </u> GG	A <u> </u> GG

3 gaps (20 cases)

<u> </u> ACG	<u> </u> A <u> </u> CG	<u> </u> AC <u> </u> G	<u> </u> ACG	A <u> </u> CG	A <u> </u> C <u> </u> G	A <u> </u> CG	AC <u> </u> G	AC <u> </u> G	ACG
AGG	AG <u> </u> G	AG <u> </u> G	AG <u> </u> G	A <u> </u> GG	A <u> </u> G <u> </u> G	A <u> </u> G <u> </u> G	A <u> </u> GG	A <u> </u> G <u> </u> G	A <u> </u> GG
A <u> </u> CG	A <u> </u> C <u> </u> G	A <u> </u> CG	A <u> </u> C <u> </u> G	A <u> </u> C <u> </u> G	A <u> </u> CG	AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	ACG
<u> </u> AGG	<u> </u> AG <u> </u> G	<u> </u> AG <u> </u> G	<u> </u> A <u> </u> GG	<u> </u> A <u> </u> G <u> </u> G	<u> </u> A <u> </u> GG	<u> </u> AGG	<u> </u> AG <u> </u> G	<u> </u> A <u> </u> GG	<u> </u> AGG



- Example:

Note: Here we consider them as different alignments

ACG **63 possible** No gaps
 AGG **alignments** (1 case)
Best: no gaps
(score = 1)

ACG
AGG

1 gap
(12 cases)

<u> </u> ACG	<u> </u> ACG	<u> </u> ACG	A <u> </u> CG	A <u> </u> CG	A <u> </u> CG
<u>A</u> GG	<u>AG</u> G	<u>AGG</u>	<u> </u> AGG	<u>AG</u> G	<u>AGG</u>
AC <u> </u> G	AC <u> </u> G	AC <u> </u> G	ACG <u> </u>	ACG <u> </u>	ACG <u> </u>
<u> </u> AGG	A GG	AGG	<u> </u> AGG	A GG	AG G

2 gaps (30 cases)

<u> </u> ACG	<u> </u> ACG	<u> </u> ACG	<u> </u> A CG	<u> </u> A CG	<u> </u> A CG	<u> </u> AC G	<u> </u> AC G	<u> </u> AC G	<u> </u> ACG
<u>AG</u> G	<u>AG</u> G	<u>AGG</u>	<u>A</u> G G	<u>A</u> G G	<u>AGG</u>	<u>A</u> GG	<u>A</u> GG	<u>AG</u> G	<u>A</u> GG
<u> </u> ACG	<u> </u> ACG	A CG	A CG	A CG	A C G	A C G	A C G	A CG	A CG
<u>A</u> G G	<u>AG</u> G	<u> </u> AG G	<u> </u> AGG	<u>AGG</u>	<u> </u> A GG	<u> </u> AGG	<u>AG</u> G	<u> </u> A GG	<u> </u> AG G
A CG	AC G	AC G	AC G	AC G	AC G	AC G	ACG	ACG	ACG
<u>AG</u> G	<u> </u> AGG	<u> </u> AGG	<u>A</u> GG	<u> </u> AGG	<u> </u> AG G	<u>A</u> G G	<u> </u> AGG	<u> </u> A GG	<u>A</u> GG

3 gaps (20 cases)

<u> </u> ACG	<u> </u> A CG	<u> </u> AC G	<u> </u> ACG	<u> </u> A CG	<u> </u> A C G	<u> </u> A CG	<u> </u> AC G	<u> </u> AC G	<u> </u> ACG
<u>AGG</u>	<u>AG</u> G	<u>AG</u> G	<u>AG</u> G	<u>A</u> GG	<u>A</u> G G	<u>A</u> G G	<u>A</u> GG	<u>A</u> G G	<u>A</u> GG
A CG	A C G	A CG	A C G	A C G	A CG	AC G	AC G	AC G	ACG
<u> </u> AGG	<u> </u> AG G	<u> </u> AG G	<u> </u> A GG	<u> </u> A G G	<u> </u> A GG	<u> </u> AGG	<u> </u> AG G	<u> </u> A GG	<u> </u> AGG

Number of possible alignments [optional]



- Given first sequence of length m and second sequence of length n ($m \leq n$)
 - Minimum number of gaps added to first sequence: $n-m$
 - Maximum number of gaps added to first sequence: n
 - For x gaps inserted into the first sequence:
 - There are ${}_{x+m}C_x$ ways to insert the gaps
 - Need to insert $x+m-n$ gaps into the second sequence
 - Can only insert at places that are not gaps in the first sequence. ${}_mC_{x+m-n}$ ways
 - Therefore, the total number of possible alignments is

$$\sum_{x=n-m}^n \binom{x+m}{x} \binom{m}{x+m-n} = \sum_{x=n-m}^n \frac{(x+m)!}{x! m!} \frac{m!}{(x+m-n)! (n-x)!} = \sum_{x=n-m}^n \frac{(x+m)!}{x! (x+m-n)! (n-x)!}$$

–E.g., when $m=n=3$,

$$\sum_{x=0}^3 \frac{(x+3)!}{x! x! (3-x)!} = \frac{(0+3)!}{0! 0! (3-0)!} + \frac{(1+3)!}{1! 1! (3-1)!} + \frac{(2+3)!}{2! 2! (3-2)!} + \frac{(3+3)!}{3! 3! (3-3)!} = 1 + 12 + 30 + 20 = 63$$



So, how large is it?

$$\sum_{x=n-m}^n \binom{x+m}{x} \binom{m}{x+m-n} = \sum_{x=n-m}^n \frac{(x+m)!}{x! m!} \frac{m!}{(x+m-n)! (n-x)!} = \sum_{x=n-m}^n \frac{(x+m)!}{x! (x+m-n)! (n-x)!}$$

<i>n \ m</i>	1	2	3	4	5	6	7	8	9	10
1	3									
2	5	13								
3	7	25	63							
4	9	41	129	321						
5	11	61	231	681	1683					
6	13	85	377	1289	3653	8989				
7	15	113	575	2241	7183	19825	48639			
8	17	145	833	3649	13073	40081	108545	265729		
9	19	181	1159	5641	22363	75517	224143	598417	1462563	
10	21	221	1561	8361	36365	134245	433905	1256465	3317445	8097453

- Exponential growth (see Covington, *Journal of Quantitative Linguistics* 11(3):173-182, 2004 for the counts for other definitions of unique alignments)

How to find the best alignment then?



- If m and n are not too large (say, $<10,000$):
 - Dynamic programming (for finding optimal alignments, i.e., alignments with highest alignment score) – this lecture
 - Essentially a smart way to compare all alignments
 - Need a smart way because of the exponential number of possible alignments
- If m or n is very large (e.g., whole-genome alignment with m and n at the scale of billions):
 - Heuristic algorithms (for finding reasonably good alignments that may not be optimal) – next lecture

How to compare all alignments?



- An important concept in computer science is “divide-and-conquer”:
 1. Divide a big problem into smaller problems
 2. Solve the smaller problems
 3. Combine the results to solve the original big problem

How to compare all alignments?



- An important concept in computer science is “divide-and-conquer”:
 1. Divide a big problem into smaller problems
 2. Solve the smaller problems
 3. Combine the results to solve the original big problem
- In Step 2,
 - If a smaller problem is still too difficult to solve, we divide it further into even smaller problems
 - Otherwise, we solve it directly
- Note: Some people require the sub-problems in divide-and-conquer to be non-overlapping. Here we use the term in a more relaxed manner.

Divide-and-conquer: an example



- Suppose we want to align these sequences:
 - $r = \text{AG}$
 - $s = \text{G}$

Divide-and-conquer: an example



- Suppose we want to align these sequences:
 - $r = \text{AG}$
 - $s = \text{G}$
- Possible alignments (we want to find out the best alignment without listing them in this way):

AG	_AG	AG	AG_	A_G
G_	G__	_G	__G	_G_

Divide-and-conquer: an example



- Suppose we want to align these sequences:
 - $r = \text{AG}$
 - $s = \text{G}$
- Possible alignments (we want to find out the best alignment without listing them in this way):

AG	_AG	AG	AG_	A_G
G_	G__	_G	_G	_G_
1	2	3	3	3

Divide-and-conquer: an example



- Suppose we want to align these sequences:

– $r = \text{AG}$

– $s = \text{G}$

- Possible alignments (we want to find out the best alignment without listing them in this way):

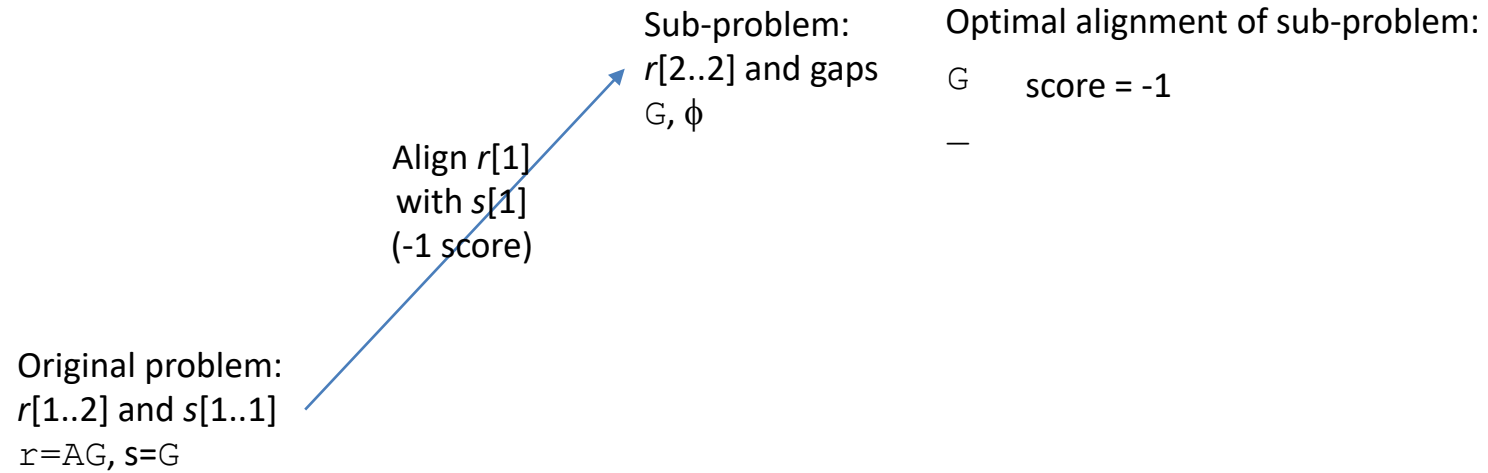
AG	_AG	AG	AG_	A_G
G_	G__	_G	_G	_G_
1	2	3	3	3

- Notation: The original problem is to align $r[1..2]$ and $s[1..1]$
- To solve this problem, we want to see which of the following is the best:
 1. Align $r[1]$ with $s[1]$, and find the best way to align the remaining $r[2..2]$ with ϕ , i.e., empty sequence)
 2. Align a gap with $s[1]$, and find the best way to align the remaining $r[1..2]$ with ϕ
 3. Align $r[1]$ with a gap, and find the best way to align the remaining $r[2..2]$ with $s[1..1]$

Recursion tree



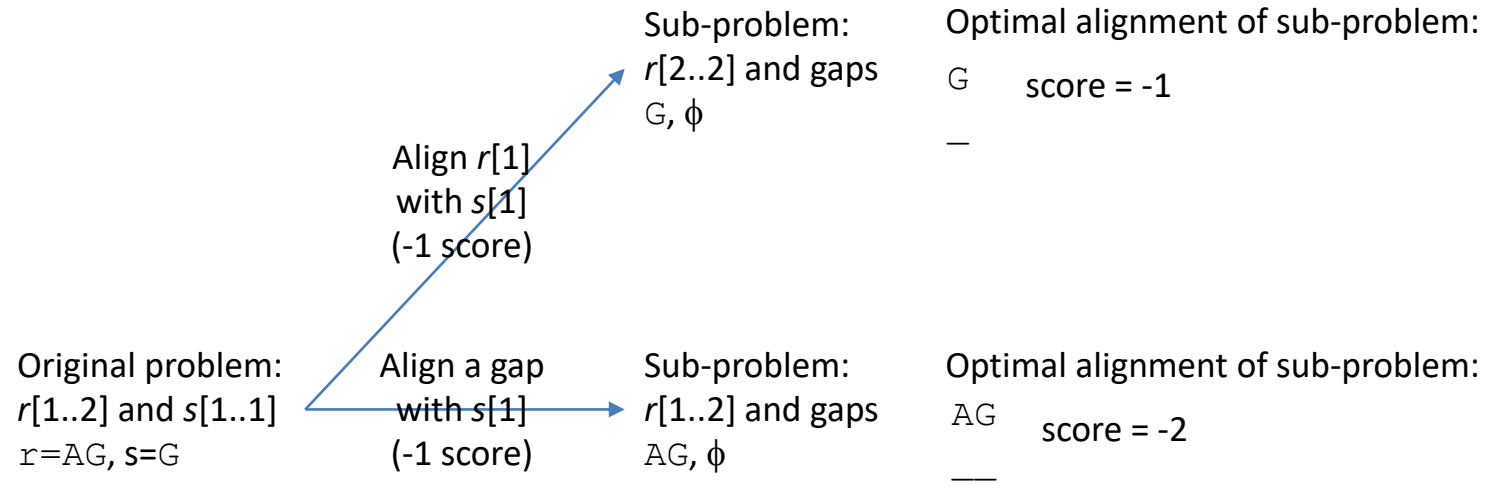
- We can summarize the process by a “tree”:



Recursion tree



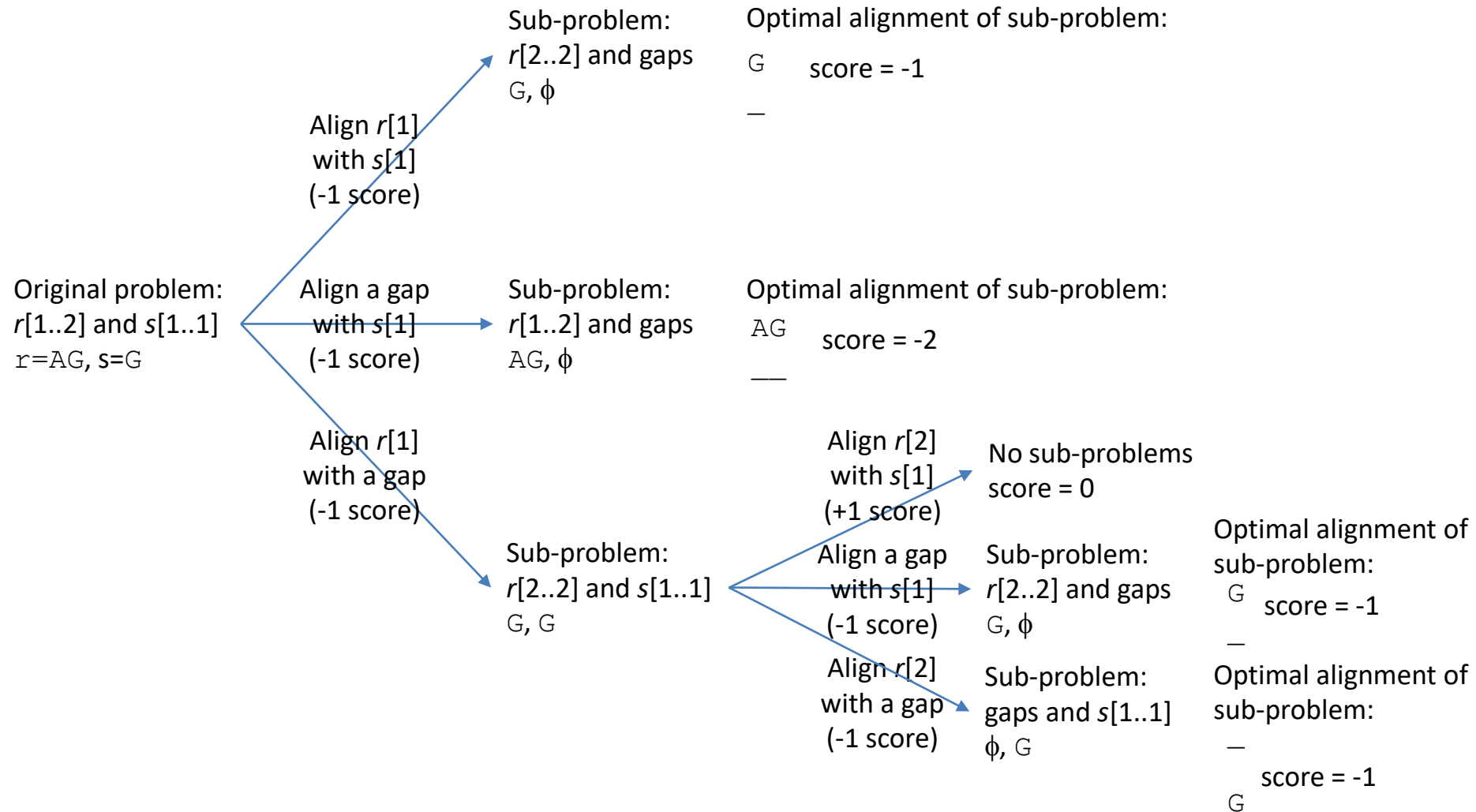
- We can summarize the process by a “tree”:



Recursion tree



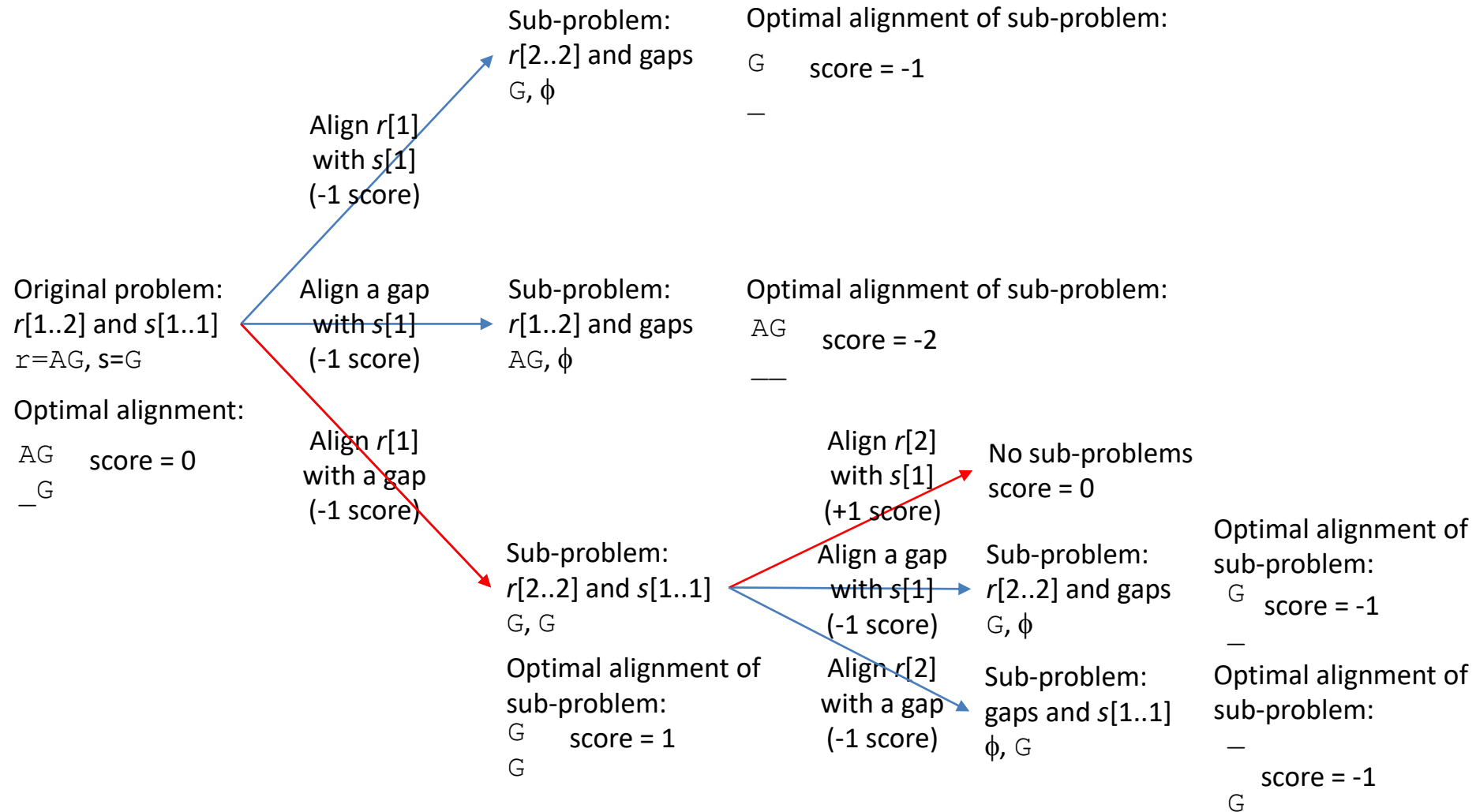
- We can summarize the process by a “tree”:



Recursion tree



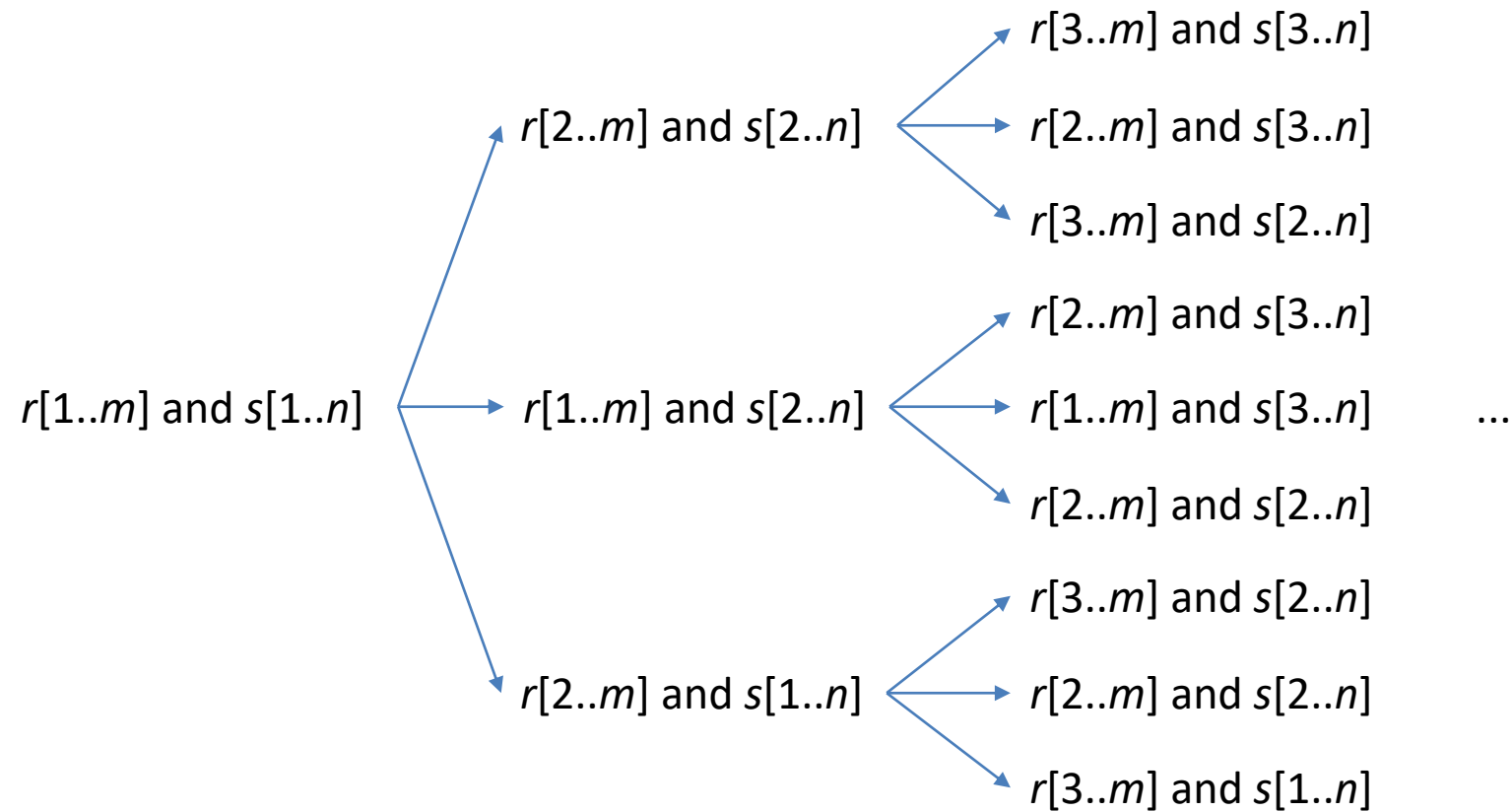
- We can summarize the process by a “tree”:

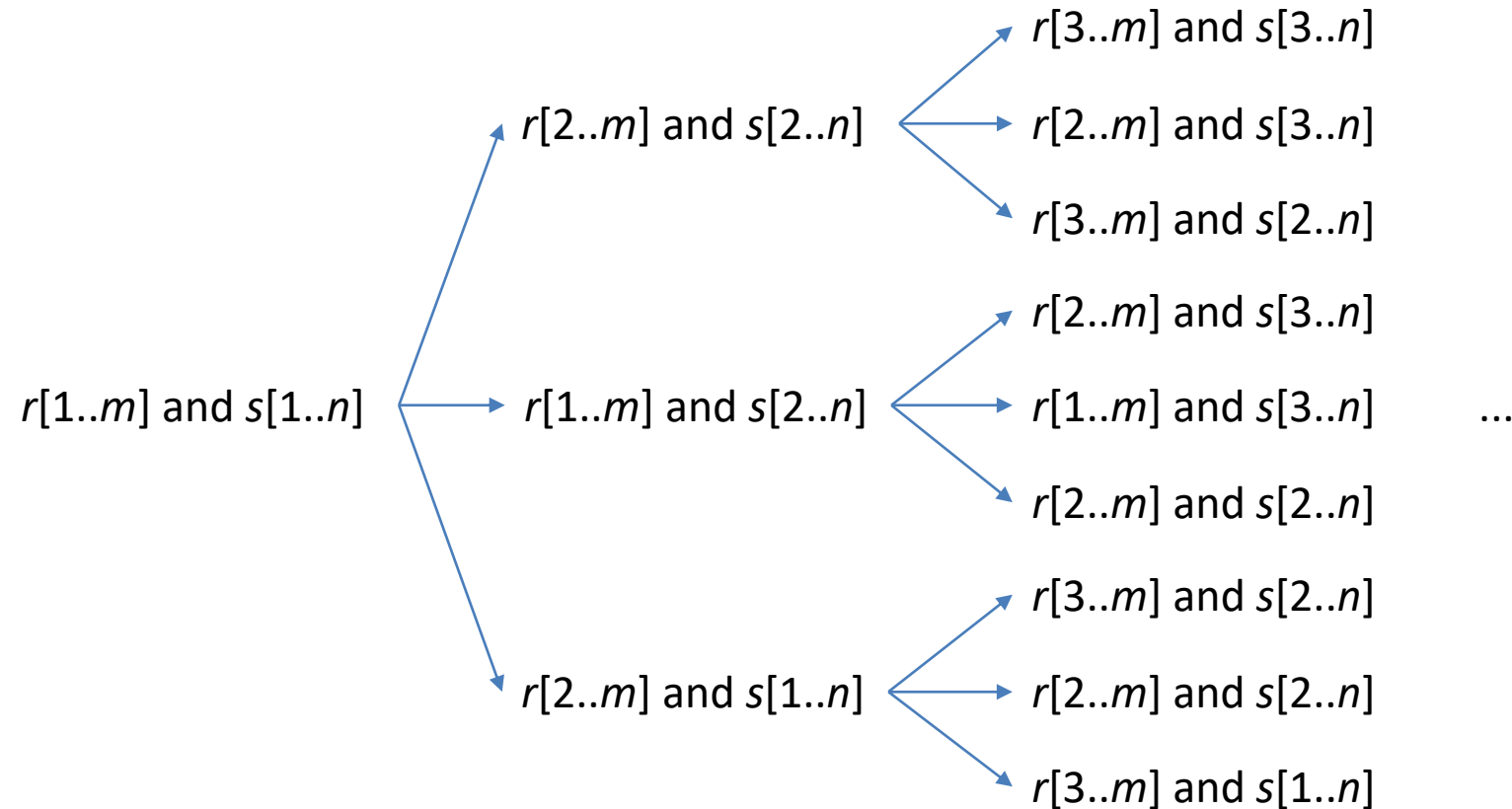




- Basically, divide-and-conquer is a systematic way to compare all possible alignments to find out the best one
- Why is it a good idea?
 - It compares groups of alignments at the same time without the need to consider individual alignments one by one
 - For example, when aligning $r=ACG$ and $s=AGG$, once we know the best alignment for $r[2..3]=CG$ and $s[3..3]=G$ is $\begin{smallmatrix} CG \\ _ G \end{smallmatrix}$ (and thus $\begin{smallmatrix} CG \\ _ G \end{smallmatrix}$ is better than $\begin{smallmatrix} CG \\ G _ \end{smallmatrix}$), we immediately know that $\begin{smallmatrix} A _ CG \\ AG _ G \end{smallmatrix}$ is better than $\begin{smallmatrix} A _ CG \\ AGG _ \end{smallmatrix}$, $\begin{smallmatrix} _ ACG \\ _ AG _ G \end{smallmatrix}$ is better than $\begin{smallmatrix} _ ACG \\ AGG _ \end{smallmatrix}$, and so on, without considering these individual alignments one by one.
- In general, the recursion tree will look like the one on the next page:

Recursion tree

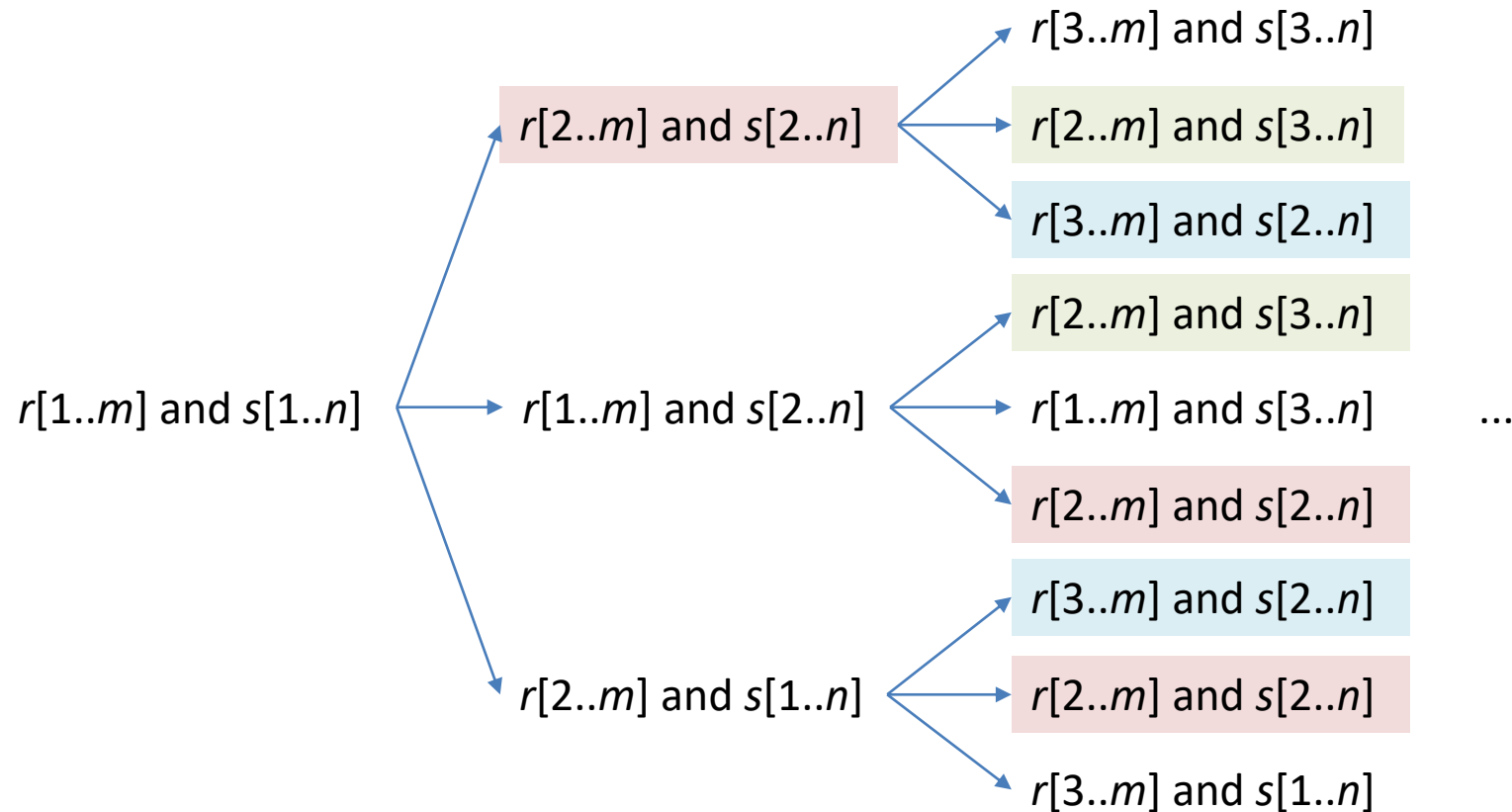




- Is it really a good idea?
 - It seems that there is an exponential number of sub-problems to solve?
 - Not really, because different branches share common problems



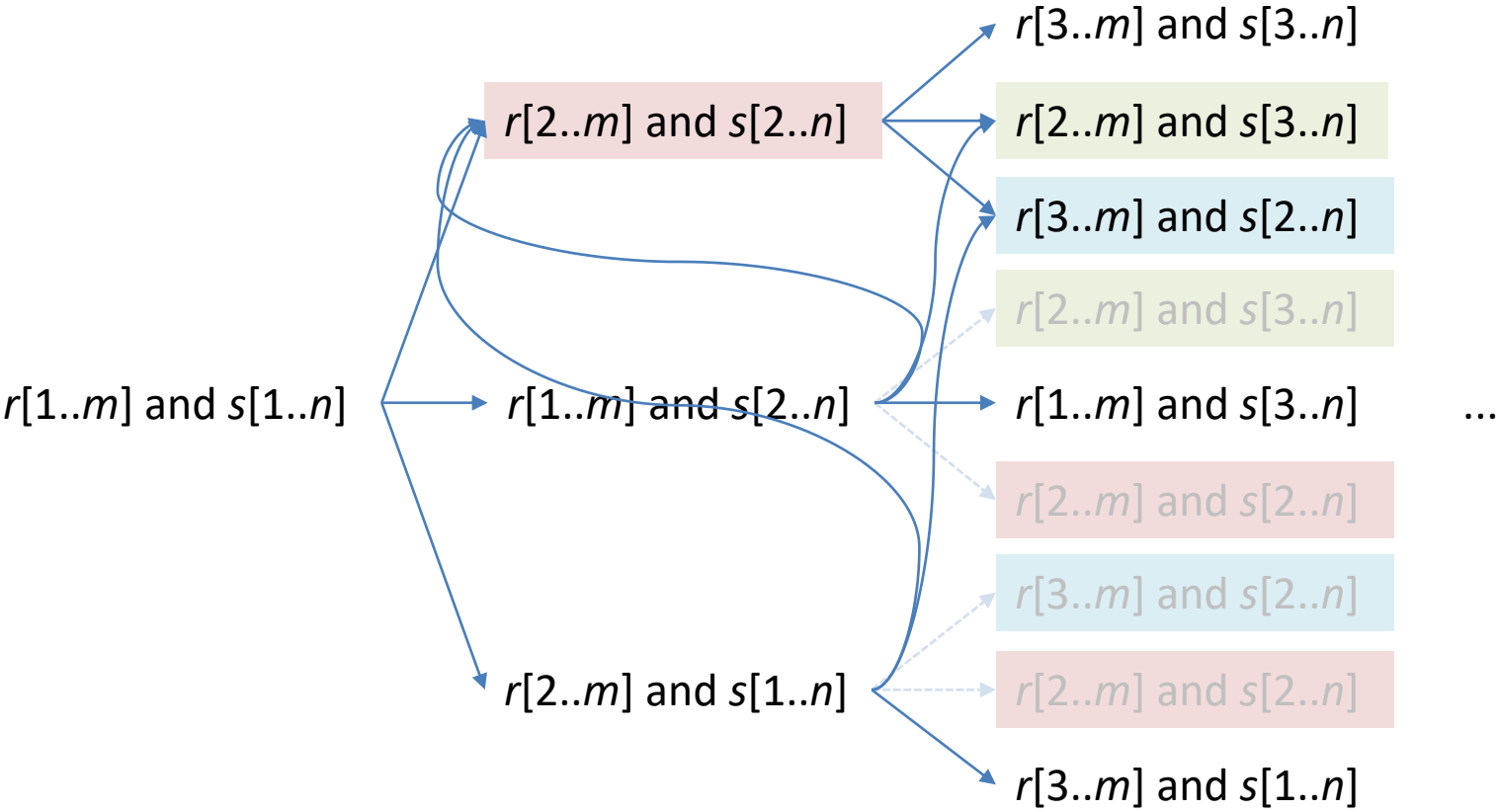
- Important observation: many sub-problems are the same





Reusing results

- Idea: Store the intermediate results and reuse them



–Note that each sub-problem involves a **suffix** of r and a **suffix** of s



- So, the key point is to store and reuse the alignment scores between the suffixes
- How many suffixes are there?
 - $m+1$ for r : $r[1..m]$, $r[2..m]$, ..., $r[m..m]$, ϕ (just gaps)
 - $n+1$ for s : $s[1..n]$, $s[2..n]$, ..., $s[n..n]$, ϕ (just gaps)
- How many alignment problems are there?
 - $(m+1)(n+1) - 1 = mn + m + n \approx mn$ for large m, n
 - -1 because it is meaningless to align two empty strings
 - Polynomial, instead of exponential
 - Much smaller than the total number of alignments
 - E.g., when $m=10$ and $n=10$:
 - » There are 8,097,453 possible alignments
 - » These alignments only share $(11)(11)-1 = 120$ unique sub-problems
 - Some are extremely easy to solve. For example, aligning ϕ and $s[1..x]$ is to simply align everything in the second sequence with gaps, for all x between 1 and n



- Based on these ideas, we are going to study a method called dynamic programming for finding optimal alignments
- It is a bit complicated when you first learn it, but after having some exercises, you should be able to solve problems using it
- Our goals:
 - Everyone: Understanding the high-level concepts
 - Everyone: Knowing how to fill in the numbers in the dynamic programming table
 - Some (hopefully most or even all) of you: Understanding why the numbers should be filled in that way



- Dynamic programming is a systematic way to reuse the results of sub-problems



- Dynamic programming is a systematic way to reuse the results of sub-problems
- Define a $(m+1) \times (n+1)$ table V , where $V(i, j)$ equals the optimal (i.e., highest) alignment score of the suffixes $r[i..m]$ and $s[j..n]$
 - $-r[m+1] = \phi$
 - $-s[n+1] = \phi$
- Fill in the values by making use of other values already filled in
- The final answer is read from the cell corresponding to the alignment of $r[1..m]$ and $s[1..n]$, i.e., $V(1, 1)$

Make sure you remember the definition of $V(i, j)$

A simple example



$r \backslash s$	A	G	G	ϕ
A	Best alignment between ACG and AGG	Best alignment between ACG and GG	Best alignment between ACG and G	Best alignment between ACG and ϕ
C	Best alignment between CG and AGG	Best alignment between CG and GG	Best alignment between CG and G	Best alignment between CG and ϕ
G	Best alignment between G and AGG	Best alignment between G and GG	Best alignment between G and G	Best alignment between G and ϕ
ϕ	Best alignment between ϕ and AGG	Best alignment between ϕ and GG	Best alignment between ϕ and G	Best alignment between ϕ and ϕ

Best alignments

(No need to construct, for illustration here only)

$r \backslash s$	A	G	G	ϕ
A	Best alignment score between ACG and AGG	Best alignment score between ACG and GG	Best alignment score between ACG and G	Best alignment score between ACG and ϕ
C	Best alignment score between CG and AGG	Best alignment score between CG and GG	Best alignment score between CG and G	Best alignment score between CG and ϕ
G	Best alignment score between G and AGG	Best alignment score between G and GG	Best alignment score between G and G	Best alignment score between G and ϕ
ϕ	Best alignment score between ϕ and AGG	Best alignment score between ϕ and GG	Best alignment score between ϕ and G	Best alignment score between ϕ and ϕ

Best alignment scores

(The V table that needs to be constructed)

A simple example



$r \backslash s$	A	G	G	ϕ
A				
C				
G				
ϕ				

$r \backslash s$	A	G	G	ϕ
A				
C		?		
G	?			
ϕ		?		

$V(i, j)$ equals the optimal (i.e., highest) alignment score of the suffixes $r[i..m]$ and $s[j..n]$

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C				CG —
G				G —
ϕ	\overline{AGG}	\overline{GG}	\overline{G}	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2
G				-1
ϕ	-3	-2	-1	0

$V(i, j)$ equals the optimal (i.e., highest) alignment score of the suffixes $r[i..m]$ and $s[j..n]$

Red arrow: Pointing from a sub-problem P1 to another sub-problem P2, where the optimal score of P2 is due to P1.

A simple example










$r \backslash s$	A	G	G	ϕ
A				ACG —
C				CG —
G			$r[3..3]$ and $s[3..3]$	G —
ϕ	\overline{AGG}	\overline{GG}	\overline{G}	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2 ↑
G				-1 ↑
ϕ	-3 ←	-2 ←	-1 ←	0 ← ↑

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C				CG —
G			$r[3..3]$ and $s[3..3]$	G —
ϕ	\overline{AGG}	\overline{GG}	\overline{G}	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2 
G				-1 
ϕ	-3 	-2 	-1 	0  

Case 1: Align $r[3]$ with $s[3]$

G

G

New sub-problem : nil

Result: +1 from diagonal (0+1=1)

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C				CG —
G			$r[3..3]$ and $s[3..3]$	G —
ϕ	\overline{AGG}	\overline{GG}	\overline{G}	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2
G				-1
ϕ	-3	-2	-1	0

Diagram illustrating the dynamic programming table with arrows indicating the optimal path (blue) and other possible paths (red).

Case 1: Align $r[3]$ with $s[3]$

G
G

New sub-problem : nil

Result: +1 from diagonal ($0+1=1$)

Case 2: Align gap with $s[3]$

\overline{G}

New sub-problem: align $r[3..3]$ and ϕ

Result: -1 from right ($-1-1=-2$)

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C				CG —
G			$r[3..3]$ and $s[3..3]$	G —
ϕ	\overline{AGG}	\overline{GG}	\overline{G}	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2
G				-1
ϕ	-3	-2	-1	-1

Diagram showing alignment paths with arrows and scores:

- Red arrows (up): $\phi \rightarrow C$ (-2), $C \rightarrow G$ (-1), $G \rightarrow \phi$ (-1)
- Blue arrows (left): $\phi \rightarrow G$ (-1), $G \rightarrow \phi$ (-1)
- Black arrow (diagonal): $\phi \rightarrow G$ (+1)

Case 1: Align $r[3]$ with $s[3]$

G
G

New sub-problem : nil

Result: +1 from diagonal ($0+1=1$)

Case 2: Align gap with $s[3]$

—
G

New sub-problem: align $r[3..3]$ and ϕ

Result: -1 from right ($-1-1=-2$)

Case 3: Align $r[3]$ with gap

G
—

New sub-problem: align ϕ and $s[3..3]$

Result: -1 from bottom ($-1-1=-2$)

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C				CG —
G			G G	G —
ϕ	— AGG	— GG	— G	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2
G				-1
ϕ	-3	-2	-1	0

Arrows indicating optimal path: Red arrows point from (G, G) to (G, ϕ) and from (ϕ , G) to (G, G). Blue arrows point from (ϕ , G) to (ϕ , ϕ) and from (ϕ , ϕ) to (G, ϕ).

Case 1: Align $r[3]$ with $s[3]$

G
G

New sub-problem : nil

Result: +1 from diagonal (0+1=1)

Case 2: Align gap with $s[3]$

—
G

New sub-problem: align $r[3..3]$ and ϕ

Result: -1 from right (-1-1=-2)

Case 3: Align $r[3]$ with gap

G
—

New sub-problem: align ϕ and $s[3..3]$

Result: -1 from bottom (-1-1=-2)

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C			$r[2..3]$ and $s[3..3]$	CG —
G			G G	G —
ϕ	— AGG	— GG	— G	

$r \backslash s$	A	G	G	ϕ
A				-3
C				-2 ↑
G				-1 ↖
ϕ	-3	-2 ←	-1 ←	0 ←

Case 1: Align $r[2]$ with $s[3]$

C
G

New sub-problem : align $r[3..3]$ and ϕ
Result: -1 from diagonal (-1-1=-2)

Case 2: Align gap with $s[3]$

—
G

New sub-problem: align $r[2..3]$ and ϕ
Result: -1 from right (-2-1=-3)

Case 3: Align $r[2]$ with gap

C
—

New sub-problem: align $r[3..3]$ and $s[3..3]$
Result: -1 from bottom (1-1=0)

A simple example



$r \backslash s$	A	G	G	ϕ
A				ACG —
C			CG _G	CG —
G			G G	G —
ϕ	AGG —	GG —	G —	

$r \backslash s$	A	G	G	ϕ
A				-3
C			0	-2 -1
G			1 -1	-1 -1
ϕ	-3	-2	-1	-1

Case 1: Align $r[2]$ with $s[3]$

C
G

New sub-problem : align $r[3..3]$ and ϕ
Result: -1 from diagonal ($-1-1=-2$)

Case 2: Align gap with $s[3]$

—
G

New sub-problem: align $r[2..3]$ and ϕ
Result: -1 from right ($-2-1=-3$)

Case 3: Align $r[2]$ with gap

C
—

New sub-problem: align $r[3..3]$ and $s[3..3]$
Result: -1 from bottom ($1-1=0$)

A simple example



$r \backslash s$	A	G	G	ϕ
A	ACG AGG	ACG ACG G_G _GG	ACG _G	ACG _
C	CG_ C_G AGG AGG	CG GG	CG _G	CG _
G	_G_ _G AGG AGG	G_ _G GG GG	G G	G _
ϕ	_AGG	_GG	_G	

$r \backslash s$	A	G	G	ϕ
A	1	$\leftarrow -1$	$\leftarrow -1$	$\leftarrow -3$
C	$\uparrow -1$	$\leftarrow -1$	$\leftarrow -1$	$\leftarrow -2$
G	$\uparrow -1$	$\leftarrow -1$	$\leftarrow -1$	$\leftarrow -1$
ϕ	$\uparrow -3$	$\leftarrow -1$	$\leftarrow -1$	$\leftarrow -1$

Final answer



$r \backslash s$	A	G	G	ϕ
A	ACG AGG	ACG G_G ACG _GG	ACG _G	ACG _
C	CG_ AGG C_G AGG _CG AGG	CG GG	CG _G	CG _
G	_G_ AGG _G AGG	G_ GG _G GG	G G	G _
ϕ	_AGG	_GG	_G	

$r \backslash s$	A	G	G	ϕ
A	1	-1	-1	-3
C	-1	-1	-1	-1
G	-1	-1	-1	-1
ϕ	-3	-1	-1	-1



- Dynamic programming is a systematic way to reuse the results of sub-problems
- Define a $(m+1) \times (n+1)$ table V , where $V(i, j)$ equals the optimal (i.e., highest) alignment score of the suffixes $r[i..m]$ and $s[j..n]$
 - $-r[m+1] = \phi$
 - $-s[n+1] = \phi$
- Fill in the values by making use of other values already filled in
- The final answer is read from the cell corresponding to the alignment of $r[1..m]$ and $s[1..n]$, i.e., $V(1, 1)$

Make sure you remember the definition of $V(i, j)$



- Value of $V(i,j)$ based on $V(i+1,j+1)$, $V(i+1,j)$ and $V(i,j+1)$:

$$V(i,j) = \max \begin{cases} V(i+1,j+1) + 1 & \text{if } r[i] = s[j] \\ V(i+1,j+1) - 1 & \text{if } r[i] \neq s[j] \\ V(i,j+1) - 1 \\ V(i+1,j) - 1 \end{cases}$$

Optimal alignment score between $r[i..m]$ and $s[j..n]$

Alignment score between $r[i]$ and $s[j]$ + Optimal alignment score between $r[i+1..m]$ and $s[j+1..n]$



- Value of $V(i,j)$ based on $V(i+1,j+1)$, $V(i+1,j)$ and $V(i,j+1)$:

$$V(i,j) = \max \begin{cases} V(i+1,j+1) + 1 & \text{if } r[i] = s[j] \\ V(i+1,j+1) - 1 & \text{if } r[i] \neq s[j] \\ V(i,j+1) - 1 \\ V(i+1,j) - 1 \end{cases}$$

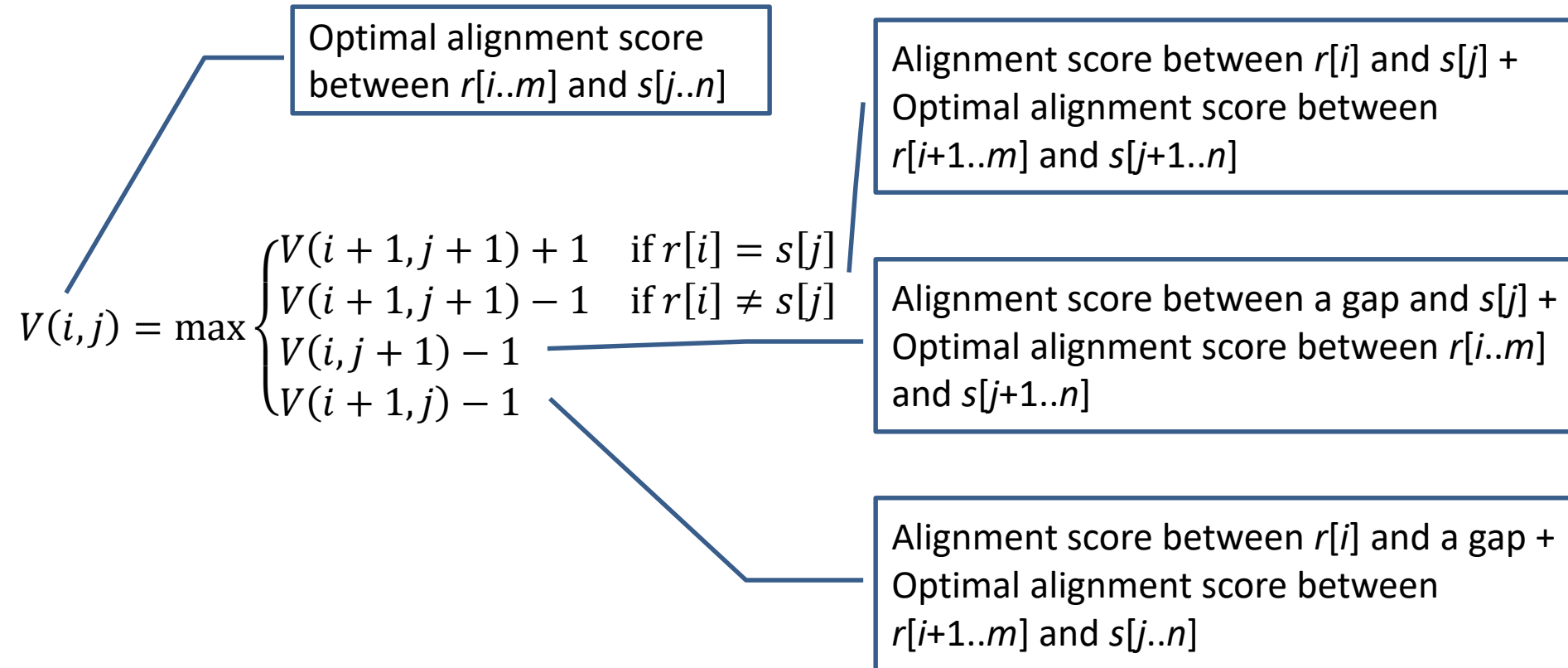
Optimal alignment score between $r[i..m]$ and $s[j..n]$

Alignment score between $r[i]$ and $s[j]$ + Optimal alignment score between $r[i+1..m]$ and $s[j+1..n]$

Alignment score between a gap and $s[j]$ + Optimal alignment score between $r[i..m]$ and $s[j+1..n]$



- Value of $V(i,j)$ based on $V(i+1,j+1)$, $V(i+1,j)$ and $V(i,j+1)$:





- Sometimes there are multiple alignments with the same best score
- Usually only the score matrix is filled. The best alignment(s) is obtained by tracing the “red arrows” backward from $V(1, 1)$
- In general, after filling the dynamic programming table V , to find out the optimal global alignment:
 - Start from the upper-left corner
 - Follow the red arrow backward
 - If there are multiple of them, each time follow one
 - Each will lead to a new set of optimal alignments
 - Until reaching the lower-right corner

Final answer



$r \backslash s$	A	G	G	ϕ
A	ACG AGG	ACG ACG G_G _GG	ACG _G	ACG _
C	CG_ C_G AGG AGG _CG _AGG	CG GG	CG _G	CG _
G	_G_ _G AGG AGG	G_ _G GG GG	G G	G _
ϕ	_AGG	_GG	_G	

$r \backslash s$	A	G	G	ϕ
A	1	-1	-1	-3
C	-1	+1 0	-1	-2
G	-1	-1	-1 1	-1
ϕ	-3	-2	-1	0

Diagram illustrating a sequence alignment matrix with scores and arrows indicating the optimal path (red arrows) and alternative paths (blue arrows).

A slightly more complex example



- r : ATGCGT
- s : ACGGCGT

$s \backslash r$	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	-2	-4	-6
T	1	2	3	3	1	-1	-3	-5
G	1	2	3	4	2	0	-2	-4
C	-1	0	1	2	3	1	-1	-3
G	-3	-2	-1	0	1	2	0	-2
T	-5	-4	-3	-2	-1	0	1	-1
ϕ	-7	-6	-5	-4	-3	-2	-1	0

Best alignment score: 3

A slightly more complex example



- r : ATGCGT
- s : ACGGCGT

$s \backslash r$	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	-2	-4	-6
T	1	2	3	3	1	-1	-3	-5
G	1	2	3	4	2	0	-2	-4
C	-1	0	1	2	3	1	-1	-3
G	-3	-2	-1	0	1	2	0	-2
T	-5	-4	-3	-2	-1	0	1	-1
ϕ	-7	-6	-5	-4	-3	-2	-1	0

How to read off the alignment?

Consider the arrow point into $V(i,j)$

- ↖ Consume both $r[i]$ and $s[j]$
- ← Consume $s[j]$ (i.e., add a gap to r to align with $s[j]$)
- ↑ Consume $r[i]$ (i.e., add a gap to s to align with $r[i]$)

Best alignment score: 3

Best alignment 1:

r A _ T G C G T
 s A C G G C G T

A slightly more complex example



- r : ATGCGT
- s : ACGGCGT

$s \backslash r$	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	-2	-4	-6
T	1	2	3	3	1	-1	-3	-5
G	1	2	3	4	2	0	-2	-4
C	-1	0	1	2	3	1	-1	-3
G	-3	-2	-1	0	1	2	0	-2
T	-5	-4	-3	-2	-1	0	1	-1
ϕ	-7	-6	-5	-4	-3	-2	-1	0

How to read off the alignment?

Consider the arrow point into $V(i,j)$

- ↖ Consume both $r[i]$ and $s[j]$
- ← Consume $s[j]$ (i.e., add a gap to r to align with $s[j]$)
- ↑ Consume $r[i]$ (i.e., add a gap to s to align with $r[i]$)

Best alignment score: 3

Best alignment 1:

r A_TGCGT

s ACGGCGT

Best alignment 2:

r AT_GCGT

s ACGGCGT

A slightly more complex example



- r : ATGCGT
- s : ACGGCGT

$s \backslash r$	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	-2	-4	-6
T	1	2	3	3	1	-1	-3	-5
G	1	2	3	4	2	0	-2	-4
C	-1	0	1	2	3	1	-1	-3
G	-3	-2	-1	0	1	2	0	-2
T	-5	-4	-3	-2	-1	0	1	-1
ϕ	-7	-6	-5	-4	-3	-2	-1	0

How to read off the alignment?

Consider the arrow point into $V(i,j)$

- ↖ Consume both $r[i]$ and $s[j]$
- ← Consume $s[j]$ (i.e., add a gap to r to align with $s[j]$)
- ↑ Consume $r[i]$ (i.e., add a gap to s to align with $r[i]$)

Best alignment score: 3

Best alignment 1:

r A_TGCGT
 s ACGGCGT

Best alignment 2:

r AT_GCGT
 s ACGGCGT

Best alignment 3:

r ATG_CGT
 s ACGGCGT

Needleman-Wunsch algorithm



- Proposed by Saul B. Needleman and Christian D. Wunsch in 1970 (Needleman and Wunsch, *J. Mol. Biol.* 48(3):443-453, 1970) for protein sequences
- Slightly more general than what we have been using so far: match, mismatch and indel scores are defined by a scoring matrix σ
 - The scoring matrix we have been using:

σ	A	C	G	T	–
A	1	-1	-1	-1	-1
C	-1	1	-1	-1	-1
G	-1	-1	1	-1	-1
T	-1	-1	-1	1	-1
–	-1	-1	-1	-1	N/A



- Here: A slight variation of the original algorithm
- Update formula:

$$V(i, j) = \max \begin{cases} V(i + 1, j + 1) + \sigma(r[i], s[j]) \\ V(i, j + 1) + \sigma(' ', s[j]) \\ V(i + 1, j) + \sigma(r[i], ' ') \end{cases}$$

- Boundary conditions:
 - $V(m+1, n+1) = 0$
 - $V(i, n+1) = V(i+1, n+1) + \sigma(r[i], ' '), \text{ for all } i \in [1, m]$
 - All characters in $r[i..m]$ are aligned with gaps
 - $V(m+1, j) = V(m+1, j+1) + \sigma(' ', s[j]), \text{ for all } j \in [1, n]$
 - All characters in $s[j..n]$ are aligned with gaps
- Optimal alignment score: $V(1, 1)$



- Need time and space proportional to mn if we ignore lower-order terms, i.e., in “Big-O notation”: $O(mn)$.



A slightly more complex example

- *r*: ATGCGT
- *s*: ACGGCGT

<i>r</i> \ <i>s</i>	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	-2	-4	-6
T	1	2	3	3	1	-1	-3	-5
G	1	2	3	4	2	0	-2	-4
C	-1	0	1	2	3	1	-1	-3
G	-3	-2	-1	0	1	2	0	-2
T	-5	-4	-3	-2	-1	0	1	-1
ϕ	-7	-6	-5	-4	-3	-2	-1	0

How to read off the alignment?
Consider the arrow point into $V(i,j)$

- ↖ Consume both $r[i]$ and $s[j]$
- ← Consume $s[j]$ (i.e., add a gap to r to align with $s[j]$)
- ↑ Consume $r[i]$ (i.e., add a gap to s to align with $r[i]$)

Best alignment score: 3

Best alignment 1:

r A _ T G C G T
s A C G G C G T

Best alignment 2:

r A T _ G C G T
s A C G G C G T

Best alignment 3:

r A T G _ C G T
s A C G G C G T



- Need time and space proportional to mn if we ignore lower-order terms, i.e., in “Big-O notation”: $O(mn)$.
- If we only need to get the alignment score, the space needed can be greatly reduced by discarding an old row/column once the values of a new row/column has been filled up.
- Many improvements and extensions have been proposed. Also possible to get the optimal alignments (in addition to alignment score) using less space.



- What would be a good scoring matrix?
 1. Gives higher scores for more similar characters
 2. Handles gaps properly
- How to construct a scoring matrix?
 - Based on evolutionary models
 - Parameters estimated from large sequence databases
- High-level introduction here. Will go slightly deeper when we study phylogenetic trees



- Example 1: DNA sequences
 - Recall that A and G have two carbon rings (the “purines”), while C and T have only one (the “pyrimidines”)
 - Milder consequences for changes to the same type (purine to purine or pyrimidine to pyrimidine): “transition”
 - More drastic consequences for changes to the other type (purine to pyrimidine or pyrimidine to purine): “transversion”

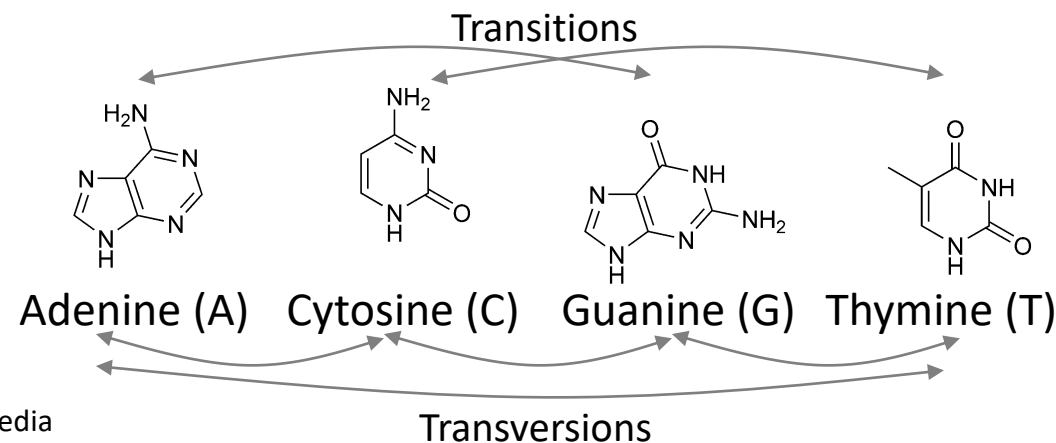


Image credit: Wikipedia



- Example 1: DNA sequences
 - Recall that A and G have two carbon rings (the “purines”), while C and T have only one (the “pyrimidines”)
 - Milder consequences for changes to the same type (purine to purine or pyrimidine to pyrimidine): “transition”
 - More drastic consequences for changes to the other type (purine to pyrimidine or pyrimidine to purine): “transversion”

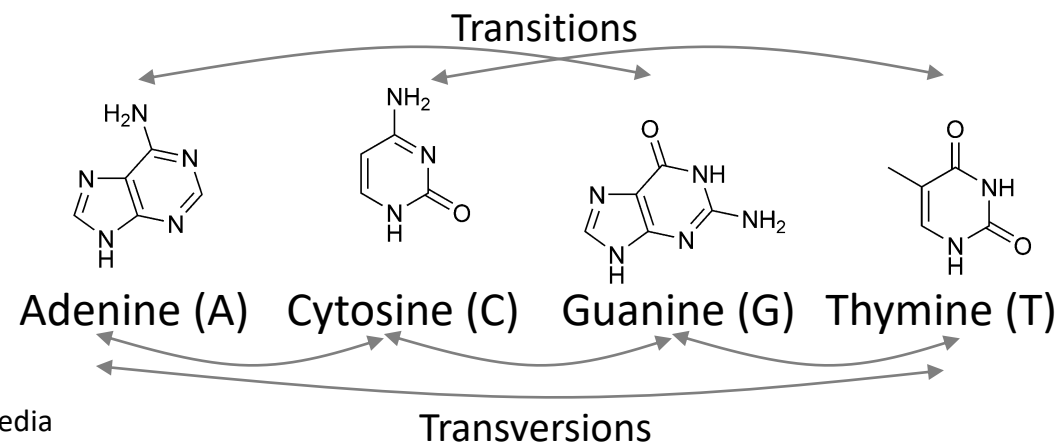


Image credit: Wikipedia

A possible scoring matrix:

σ	A	C	G	T	–
A	1	-2	-1	-2	-3
C	-2	1	-2	-1	-3
G	-1	-2	1	-2	-3
T	-2	-1	-2	1	-3
–	-3	-3	-3	-3	N/A



- Example 2: Protein sequences
 - Milder consequences if the new amino acid has properties similar to the original one (size, polarity, charge, hydrophobicity, ...)

Amino Acid	Side-chain polarity	Side-chain charge (pH 7.4)	Hydropathy index
Alanine	nonpolar	neutral	1.8
Arginine	polar	positive	-4.5
Asparagine	polar	neutral	-3.5
Aspartic acid	polar	negative	-3.5
Cysteine	polar	neutral	2.5
Glutamic acid	polar	negative	-3.5
Glutamine	polar	neutral	-3.5
Glycine	nonpolar	neutral	-0.4
Histidine	polar	positive(10%) neutral(90%)	-3.2
Isoleucine	nonpolar	neutral	4.5

Amino Acid	Side-chain polarity	Side-chain charge (pH 7.4)	Hydropathy index
Leucine	nonpolar	neutral	3.8
Lysine	polar	positive	-3.9
Methionine	nonpolar	neutral	1.9
Phenylalanine	nonpolar	neutral	2.8
Proline	nonpolar	neutral	-1.6
Serine	polar	neutral	-0.8
Threonine	polar	neutral	-0.7
Tryptophan	nonpolar	neutral	-0.9
Tyrosine	polar	neutral	-1.3
Valine	nonpolar	neutral	4.2

Note the similarity between leucine and isoleucine

Information source: Wikipedia



- DNA (see http://en.wikipedia.org/wiki/Models_of_DNA_evolution)
 - The Jukes-Cantor model: Equal probability of changing to the other three bases
 - Kimura model: Differentiating between transition and transversion
- Protein (see http://www.genome.jp/aaindex/AAindex/list_of_matrices for a list of 94 matrices)
 - PAM (Point Accepted Mutation) series: Based on substitution rate
 - BLOSUM (BLOck SUBstitution Matrix) series: Based on blocks of conserved sequences
- They are either used directly as σ , or to derive σ
- Will study them in more detail later

The BLOSUM62 matrix



- Will explain what the name means in a later lecture

	Ala A	Arg R	Asn N	Asp D	Cys C	Gln Q	Glu E	Gly G	His H	Ile I	Leu L	Lys K	Met M	Phe F	Pro P	Ser S	Thr T	Trp W	Tyr Y	Val V
Ala A	4	-1	-2	-2	0	-1	-1	0	-2	-1	-1	-1	-1	-2	-1	1	0	-3	-2	0
Arg R	-1	5	0	-2	-3	1	0	-2	0	-3	-2	2	-1	-3	-2	-1	-1	-3	-2	-3
Asn N	-2	0	6	1	-3	0	0	0	1	-3	-3	0	-2	-3	-2	1	0	-4	-2	-3
Asp D	-2	-2	1	6	-3	0	2	-1	-1	-3	-4	-1	-3	-3	-1	0	-1	-4	-3	-3
Cys C	0	-3	-3	-3	9	-3	-4	-3	-3	-1	-1	-3	-1	-2	-3	-1	-1	-2	-2	-1
Gln Q	-1	1	0	0	-3	5	2	-2	0	-3	-2	1	0	-3	-1	0	-1	-2	-1	-2
Glu E	-1	0	0	2	-4	2	5	-2	0	-3	-3	1	-2	-3	-1	0	-1	-3	-2	-2
Gly G	0	-2	0	-1	-3	-2	-2	6	-2	-4	-4	-2	-3	-3	-2	0	-2	-2	-3	-3
His H	-2	0	1	-1	-3	0	0	-2	8	-3	-3	-1	-2	-1	-2	-1	-2	-2	2	-3
Ile I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4	2	-3	1	0	-3	-2	-1	-3	-1	3
Leu L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4	-2	2	0	-3	-2	-1	-2	-1	1
Lys K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5	-1	-3	-1	0	-1	-3	-2	-2
Met M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5	0	-2	-1	-1	-1	-1	1
Phe F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6	-4	-2	-2	1	3	-1
Pro P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7	-1	-1	-4	-3	-2
Ser S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4	1	-3	-2	-2
Thr T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5	-2	-2	0
Trp W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11	2	-3
Tyr Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	-1
Val V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4



- What would be a reasonable way to handle gaps?
 - There are more single nucleotide polymorphisms (SNPs) than small indels
⇒ Higher penalty for gaps than for mismatches
 - Small indels are more common than large indels
⇒ Higher penalty for larger gaps
 - More likely to have a large gap (possibly due to a single mutation event) than multiple small gaps with the same total size (possibly due to multiple mutation events)
⇒ Higher penalty for gap opening than gap extension



- “Affine” means a straight line that may not pass through the origin
- Mathematical formula: $y = -a - bx$
 - Here, we can use it to define the gap penalty
 - y : Final gap score (a negative number)
 - x : Size of gap (i.e., number of consecutive ‘_’s)
 - $-a$: Gap opening penalty
 - $-b$: Gap size penalty



- What is the main difficulty if affine gap penalty is to be applied?
 - Without affine gap penalty, the penalty for a gap is fixed and does not depend on other positions
 - With affine gap penalty, the penalty for a gap depends on whether it is the last position of the gap (in which case gap opening cost needs to be added)
- How to deal with this issue?



- Main idea: Use different tables to store different cases, in order to calculate gap size
 - $V_0(i, j)$: Optimal alignment score between $r[i..m]$ and $s[j..n]$ with $r[i]$ aligning to $s[j]$
 - $V_1(i, j)$: Optimal alignment score between $r[i..m]$ and $s[j..n]$ with $r[i]$ aligning to a gap
 - $V_2(i, j)$: Optimal alignment score between $r[i..m]$ and $s[j..n]$ with $s[j]$ aligning to a gap
- Their update formulas are slightly different because of the affine gap penalty model
- The final optimal alignment score is the maximum of the three cases



Global alignment with affine gap penalty [optional]

- Update formulas:

- $V(i, j) = \max \{V_0(i, j), V_1(i, j), V_2(i, j)\}$
- $V_0(i, j) = V(i+1, j+1) + \sigma(r[i], s[j])$ // No gap penalty
- $V_1(i, j) = \max \{ \begin{array}{l} V_0(i+1, j) - a - b, \\ V_1(i+1, j) - b, \\ V_2(i+1, j) - a - b \end{array} \}$ // Open a new gap on the second seq.
// Extend a gap on the second seq.
// Open a new gap on the second seq.
// (and close the one on the first seq.)
- $V_2(i, j) = \max \{ \begin{array}{l} V_0(i, j+1) - a - b, \\ V_1(i, j+1) - a - b, \\ V_2(i, j+1) - b \end{array} \}$ // Open a new gap on the first seq.
// Open a new gap on the first seq.
// (and close the one on the second seq.)
// Extend a gap on the first seq.

- Boundary cases:

- $V(m+1, n+1) = V_1(m+1, n+1) = V_2(m+1, n+1) = 0$ // Nothing aligned
- $V_0(i, n+1) = -\infty$, for all $i \in [1, m+1]$ // Invalid case: $s[n+1]$ not a non-gap char.
- $V_0(m+1, j) = -\infty$, for all $j \in [1, n+1]$ // Invalid case: $r[m+1]$ not a non-gap char.
- $V_1(m+1, j) = -\infty$, for all $j \in [1, n]$ // Invalid case: $r[m+1]$ not a non-gap char.
- $V_1(i, n+1) = -a - (m-i+1)b$, for all $i \in [1, m]$ // A gap of size $(m-i+1)$ on the 2nd seq.
- $V_2(i, n+1) = -\infty$, for all $i \in [1, m]$ // Invalid case: $s[n+1]$ not a non-gap char.
- $V_2(m+1, j) = -a - (n-j+1)b$, for all $j \in [1, n]$ // A gap of size $(n-j+1)$ on the 1st seq.



1. What are the two key ideas behind dynamic programming?



1. What are the two key ideas behind dynamic programming?
2. What is the meaning of the entry at the i -th row and j -th column of the dynamic programming table for global alignment?



1. What are the two key ideas behind dynamic programming?
2. What is the meaning of the entry at the i -th row and j -th column of the dynamic programming table for global alignment?
3. How do we calculate the value of the cell on the upper-left corner for global alignment? (Match: +1 score, mismatch: -1 score, indel: -2 score)

$r \backslash s$	G	T	ϕ
A		-1	-4
T	-1	1	-2
ϕ	-4	-2	0



Local alignment

- Some sequences share common functional domains
 - The sequences are similar within the domains
 - They may not be similar outside the domains

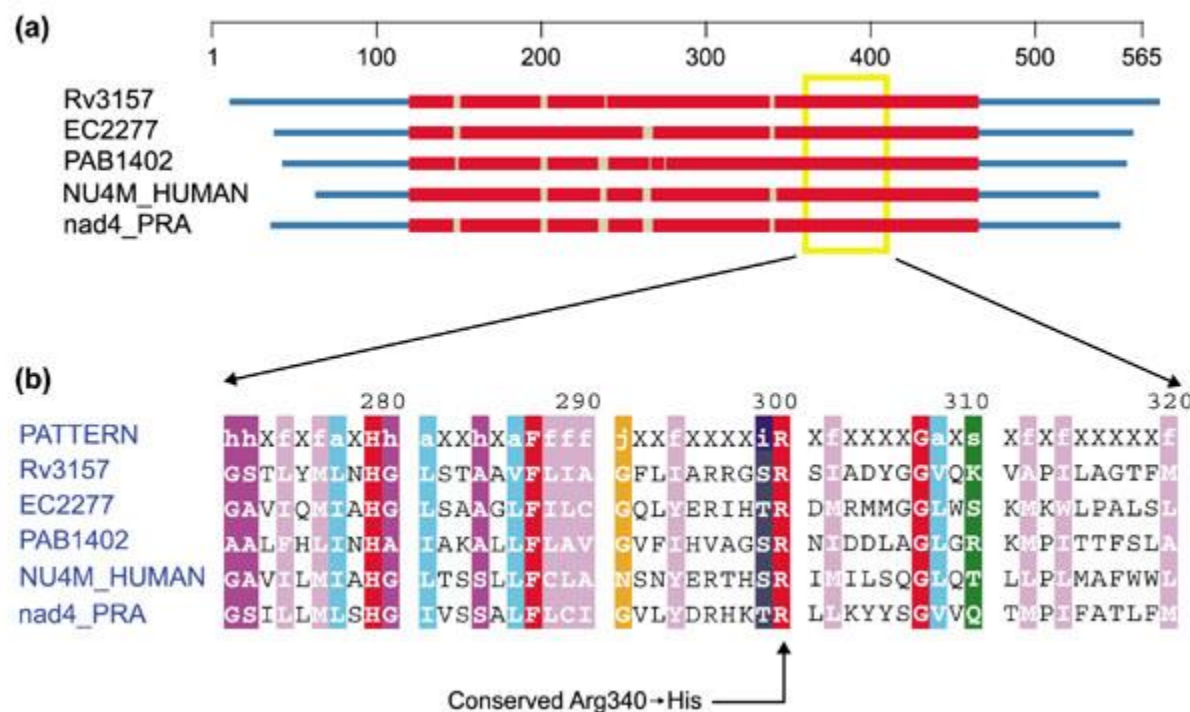


Image credit: Plasterer et al., *Genome Biology* 2:research0021 (2001)



- We may want to consider the conserved part only when aligning two sequences
 - Since we do not really know where it is, we allow some characters at the beginning and some characters at the end of each sequence to be not aligned
- Problem definition:
 - Given a set of sequences, a local alignment is **a set of subsequences each extracted from one of the original sequences**, with zero or more gaps inserted into **these subsequences** so that
 1. They all have the same length afterwards
 2. For each position, at least one of the **extracted sub-sequences** is not a gap
- Seems a much harder problem, but surprisingly it can be solved in a way similar to global alignment



- Inputs
 - Sequence r of length m
 - Sequence s of length n
- Outputs
 - The alignment(s) each aligning a **subsequences** of r and a **subsequence** of s , which has the highest score among all possible **subsequence pairs** and all possible ways to align them.
 - **text in red**: different from global alignment
 - a subsequence is a middle part of the sequence, or the empty string
 - The corresponding highest alignment score



Local alignment

- Example:

- r*: ATGCGT

- s*: ACGGCGT

- Match: +1; Otherwise: -1 (not considering affine gap penalty here)

- Global alignment (we have done it before)

- Optimal alignments:

A _TGCGT	AT _GCGT	ATG _CGT
ACGGCGT	ACGGCGT	ACGGCGT

- Optimal score: 3

- Local alignment

- GCGT (Good to show the occurrence positions: 3 GCGT 6)

- GCGT (4 GCGT 7)

- Optimal score: 4



- Proposed by Temple F. Smith and Michael S. Waterman in 1981 (Smith and Waterman, *J. Mol. Biol.* 147(1):195-197, 1981)
- A modification of the Needleman-Wunsch dynamic programming algorithm to allow unaligned regions at two ends



- Similar to global alignment, we define a table V to store and reuse results of sub-problems
- Define a $(m+1) \times (n+1)$ table V , where $V(i, j)$ equals the optimal alignment score among **all the prefixes** of $r[i..m]$ and $s[j..n]$
 - $r[m+1] = \phi$
 - $s[n+1] = \phi$
- Fill in the values by making use of other values already filled in
- Key concept: a subsequence is a prefix of a suffix

V table: Global vs. local



<i>r</i> \ <i>s</i>	A	G	G	ϕ
A				
C				
G				
ϕ				

Global alignment:
Best alignment score between
CG and AGG

Local alignment:
Best alignment score between all prefixes of CG
and AGG, i.e.,
Best alignment score among (CG and AGG), (CG
and AG), (CG and A), (CG and ϕ), (C and AGG), (C
and AG), (C and A), (C and ϕ), (ϕ and AGG), (ϕ and
AG), (ϕ and A) and (ϕ and ϕ)



- Initialization:

- $V(i, n+1) = 0$, for all $i \in [1, m]$

- All characters in $r[i..m]$ are aligned with gaps: Do not include them in the optimal alignment

- $V(m+1, j) = 0$, for all $j \in [1, n]$

- All characters in $s[j..n]$ are aligned with gaps : Do not include them in the optimal alignment

- Recursive formula:

$$V(i, j) = \max \begin{cases} V(i+1, j+1) + \sigma(r[i], s[j]) \\ V(i, j+1) + \sigma('-', s[j]) \\ V(i+1, j) + \sigma(r[i], '-') \\ 0 \end{cases}$$

- Optimal alignment score: $\max_{i,j} V(i, j)$



Local alignment example

- r : ATGCGT
- s : ACGGCGT
- Match: +1; Otherwise: -1

$s \backslash r$	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	0	0	0
T	1	2	3	3	1	0	1	0
G	1	2	3	4	2	1	0	0
C	1	2	1	2	3	1	0	0
G	0	0	1	1	1	2	0	0
T	0	0	0	0	0	0	1	0
ϕ	0	0	0	0	0	0	0	0

Alignment score: 4



Local alignment example

- *r*: ATGCGT
- *s*: ACGGCGT
- Match: +1; Otherwise: -1

<i>r</i> \ <i>s</i>	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	0	0	0
T	1	2	3	3	1	0	1	0
G	1	2	3	4	2	1	0	0
C	1	2	1	2	3	1	0	0
G	0	0	1	1	1	2	0	0
T	0	0	0	0	0	0	1	0
ϕ	0	0	0	0	0	0	0	0

Alignment score: 4

Best alignment:

```
r  3  GCGT  6
s  4  GCGT  7
```



Why does the algorithm work?

- *r*: ATGCGT
- *s*: ACGGCGT
- Match: +1; Otherwise: -1

<i>r</i> \ <i>s</i>	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	0	0	0
T	1	2	3	3	1	0	1	0
G	1	2	3	4	2	1	0	0
C	1	2	1	2	3	1	0	0
G	0	0	1	1	1	2	0	0
T	0	0	0	0	0	0	1	0
ϕ	0	0	0	0	0	0	0	0

Highest score among the optimal alignment scores of the followings:

A. $r[5..6]:GT$ vs. $s[7..7]:T$
B. $r[5..6]:GT$ vs. $s[7..]:\emptyset$
C. $r[5..5]:G$ vs. $s[7..7]:T$
D. $r[5..5]:G$ vs. $s[7..]:\emptyset$
E. $r[5..]:\emptyset$ vs. $s[7..7]:T$
F. $r[5..]:\emptyset$ vs. $s[7..]:\emptyset$

Highest score among the optimal alignment scores of the followings:

G. $r[6..6]:T$ vs. $s[7..]:\emptyset$
H. $r[6..]:\emptyset$ vs. $s[7..]:\emptyset$

Highest score among the optimal alignment scores of the followings:

I. $r[5..6]:GT$ vs. $s[7..]:\emptyset$
J. $r[5..5]:G$ vs. $s[7..]:\emptyset$
K. $r[5..]:\emptyset$ vs. $s[7..]:\emptyset$

Highest score among the optimal alignment scores of the followings:

L. $r[6..6]:T$ vs. $s[7..7]:T$
M. $r[6..6]:T$ vs. $s[7..]:\emptyset$
N. $r[6..]:\emptyset$ vs. $s[7..7]:T$
O. $r[6..]:\emptyset$ vs. $s[7..]:\emptyset$

$A = \text{Max}\{G-1, I-1, L-1\}$
 $B = \text{Max}\{M-1\}$
 $C = \text{Max}\{H-1, J-1, N-1\}$
 $D = \text{Max}\{O-1\}$
 $E = \text{Max}\{K-1\}$
 $F = \text{Max}\{0\}$

$\text{Max}\{A, B, C, D, E, F\}$
 $= \text{Max}\{$
 $\text{Max}\{G-1, I-1, L-1\},$
 $\text{Max}\{M-1\},$
 $\text{Max}\{H-1, J-1, N-1\},$
 $\text{Max}\{O-1\},$
 $\text{Max}\{K-1\},$
 $\text{Max}\{0\}$
 $\}$
 $= \text{Max}\{$
 $\text{Max}\{G-1, H-1\},$
 $\text{Max}\{I-1, J-1, K-1\},$
 $\text{Max}\{L-1, M-1, N-1, O-1\},$
 $0\}$
 $= \text{Max}\{V[6,8]-1, V[5,8]-1,$
 $V[6,7]-1, 0\}$



- In general, after filling the dynamic programming table V , to find out the optimal **local** alignment:
 - Start from the **cell with the highest score**
 - If there are multiple of them, each will lead to a different set of optimal local alignments
 - Follow the red arrow backward
 - If there are multiple of them, each time follow one
 - Output the current alignment if the current cell has a score of 0**
 - Until reaching **a cell with no incoming red arrows (i.e., cannot go any further)**

Trace back example



- Alignment between $r=ACAGC$ and $s=ACTAG$:
 - Match: +1; mismatch: -1; indel: -2

$r \backslash s$	A	C	T	A	G	ϕ
A	2	0	0	1	0	0
C	0	1	1	0	0	0
A	1	0	0	2	0	0
G	0	0	0	0	1	0
C	0	1	0	0	0	0
ϕ	0	0	0	0	0	0

–Optimal alignments:

```
r  1 AC 2  1 AC AG 4  3 AG 4
s  1 AC 2  1 ACTAG 5  4 AG 5
```

Notes:

1. May have multiple starting points.
2. An optimal path may contain multiple 0's. Should output an alignment for each 0.



Global vs. local alignment

- *r*: ATGCGT
- *s*: ACGGCGT
- Match: +1; Otherwise: -1



Global alignment (Needleman-Wunsch)

<i>r</i> \ <i>s</i>	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	-2	-4	-6
T	1	2	3	3	1	-1	-3	-5
G	1	2	3	4	2	0	-2	-4
C	-1	0	1	2	3	1	-1	-3
G	-3	-2	-1	0	1	2	0	-2
T	-5	-4	-3	-2	-1	0	1	-1
ϕ	-7	-6	-5	-4	-3	-2	-1	0

Local alignment (Smith-Waterman)

<i>r</i> \ <i>s</i>	A	C	G	G	C	G	T	ϕ
A	3	2	2	2	0	0	0	0
T	1	2	3	3	1	0	1	0
G	1	2	3	4	2	1	0	0
C	1	2	1	2	3	1	0	0
G	0	0	1	1	1	2	0	0
T	0	0	0	0	0	0	1	0
ϕ	0	0	0	0	0	0	0	0



- Number of sequences
 - 2 sequences: Pairwise sequence alignment
 - >2 sequences: Multiple sequence alignment
- Which part to align
 - Whole sequences: Global alignment
 - Parts of sequences: Local alignment
- How to compute similarity
 - Substitution score
 - Gap penalty



- Can also be solved by dynamic programming
- Global alignment for three sequences [optional]:
 - $V(i, j, k) = \max \{$
 - $V(i+1, j+1, k+1) + \sigma(r[i], s[j], t[k]),$
 - $V(i+1, j+1, k) + \sigma(r[i], s[j], \text{'_'}),$
 - $V(i+1, j, k+1) + \sigma(r[i], \text{'_'}, t[k]),$
 - $V(i, j+1, k+1) + \sigma(\text{'_'}, s[j], t[k]),$
 - $V(i+1, j, k) + \sigma(r[i], \text{'_'}, \text{'_'}),$
 - $V(i, j+1, k) + \sigma(\text{'_'}, s[j], \text{'_'}),$
 - $V(i, j, k+1) + \sigma(\text{'_'}, \text{'_'}, t[k]) \}$
- Too many tables and each is too large. Quickly becoming infeasible
 - Need heuristics (to be discussed in next lecture)



Epilogue

Case Study, Summary and Further Readings



Image credit: New York Times, May 7, 2009

“In articles published online in The New England Journal of Medicine, virologists from the Centers for Disease Control and Prevention described those cases, most of them in young people in the Midwest who touched or were near pigs. All had a “triple reassortant” virus that combined human, swine and avian flu genes.”



The NEW ENGLAND JOURNAL of MEDICINE

[HOME](#)[ARTICLES & MULTIMEDIA ▾](#)[ISSUES ▾](#)[SPECIALTIES & TOPICS ▾](#)[FOR AUTHORS ▾](#)[CME >](#)

ORIGINAL ARTICLE

[A Correction Has Been Published >](#)

Emergence of a Novel Swine-Origin Influenza A (H1N1) Virus in Humans

Novel Swine-Origin Influenza A (H1N1) Virus Investigation Team

N Engl J Med 2009; 360:2605-2615 | [June 18, 2009](#) | DOI: 10.1056/NEJMoa0903810

Share: [f](#) [t](#) [g+](#) [in](#) [+](#)

[Abstract](#)[Article](#)[References](#)[Citing Articles \(938\)](#)

Triple-reassortant swine influenza viruses, which contain genes from human, swine, and avian influenza A viruses, have been identified in swine in the United States since 1998,^{1,2} and 12 cases of human infection with such viruses were identified in the United States from 2005 through 2009.³ On April 15 and April 17, 2009, the Centers for Disease Control and Prevention (CDC) identified two cases of human infection with a swine-origin influenza A (H1N1) virus (S-OIV) characterized by a unique combination of gene segments that had not been identified among human or swine influenza A viruses. As of May 5, 2009, cases of human infection with the same novel virus have also been identified in Mexico, Canada, and elsewhere. We report the first 642 confirmed cases of human infection with this virus in the United States.

Image credit: Novel Swine-Origin Influenza A (H1N1) Virus Investigation Team, The New England Journal of Medicine 360:2605-2615, (2009)





- Core of computational problems related to sequences: sequence alignment
- Problem components
 - Two or more sequences
 - Global or local alignment
 - Substitution matrix
 - Gap penalty model
- An optimal sequence alignment algorithm always returns the alignment(s) with the highest alignment score
- Impossible to find optimal alignment by brute-force enumeration due to exponential number of possible alignments
- Two key ideas behind dynamic programming:
 - Divide-and-conquer
 - Reusing the results of sub-problems



- Chapter 2 of *Algorithms in Bioinformatics: A Practical Introduction*
 - Algorithms that use less space and time
 - Methods for defining the scoring matrix (we will also study this topic later)
 - [Free slides](#) available
- Chapter 3 of *Algorithms in Bioinformatics: A Practical Introduction*
 - A data structure for fast searching of sub-sequences
 - [Free slides](#) available



- Li and Homer, A Survey of Sequence Alignment Algorithms for Next-Generation Sequencing. *Briefings in Bioinformatics* 11(5):473-483, (2010)
 - A review of some recent methods for performing optimal sequence alignments for high-throughput sequencing reads
 - Some of the methods mentioned in this paper are discussed in CSCI3220