582670 Algorithms for Bioinformatics

Lecture 4: Dynamic Programming and Sequence Alignment

18.9.2014

Adapted from slides by Alexandru Tomescu, Leena Salmela and Veli Mäkinen

Sequence similarity

- Genome rearrangement problem assumed we know for each gene in species A its counterpart in species B (if exists).
  - Orthologous genes: same ancestor in evolution
  - Paralogous genes: gene duplication
  - Homolog = Ortholog or paralog
- Often sequence similarity is the only way to predict whether two genes are homologs
  - Very unlikely that same (long) sequences have evolved independently from different ancestors
  - ... except horizontal gene transfer

Sequence similarity vs. distance

- Let A and B be two strings (sequences) from alphabet Σ
- Many different ways to define *similarity* or *distance* of A and B
- Recall Hamming distance $d_H(A, B)$
  - Only defined when $|A| = |B|$.
- What is the simplest measure to extend Hamming distance to different length strings?
  - For many purposes it is useful if the distance is a *metric*

Edit distance

- The most studied distance function extending Hamming distance is unit cost edit distance or Levenshtein distance.
- $d_L(A, B)$ is the minimum amount of single symbol *insertions, deletions and substitutions* required to convert A into B.
- For example, when $A = \text{"tukholma"}$ and $B = \text{"stockholm"}$ we have $d_L(A, B) = 4$:
  - insert s, substitute u $\rightarrow$ o, insert c, delete a
  - ... or insert s, insert o, substitute u $\rightarrow$ c, delete a
  - ... or is there a better sequence of edits?

-	t	u	-	k	h	o	l	m	a
s	t	o	c	k	h	o	l	m	-

Dynamic Programming

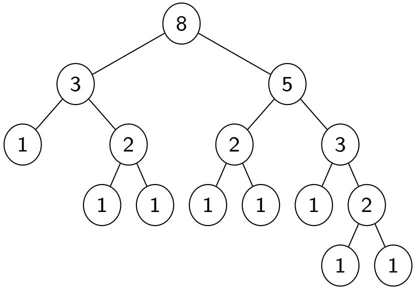
- Some problems can be broken into smaller subproblems so that the solution to the problem can be constructed from the solutions of the subproblems.
- This often leads to several instances of the same subproblem
- Dynamic programming is a technique to *organize the computation* and *save the solutions* of the subproblems so that they only need to be solved once.
- We will use dynamic programming to compute edit distance.

Example: Computing Fibonacci numbers

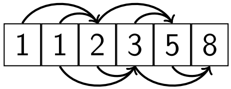
- Remember Fibonacci numbers:

$$F(n) = \begin{cases} 1 & \text{if } n = 1 \text{ or } n = 2 \\ F(n-2) + F(n-1) & \text{otherwise} \end{cases}$$

- The recursion to compute $F(n)$ contains many identical subproblems:



- We can avoid solving the same subproblem several times by saving the results in an array:



Example: Computing Fibonacci numbers

- Remember Fibonacci numbers:

$$F(n) = \begin{cases} 1 & \text{if } n = 1 \text{ or } n = 2 \\ F(n-2) + F(n-1) & \text{otherwise} \end{cases}$$

- The recursion to compute $F(n)$ contains many identical subproblems:

```
$F(n)$:
1: **if** $n = 1$ or $n = 2$ **then**
2:     **return** 1
3: **else**
4:     **return** $F(n-2) + F(n-1)$
```

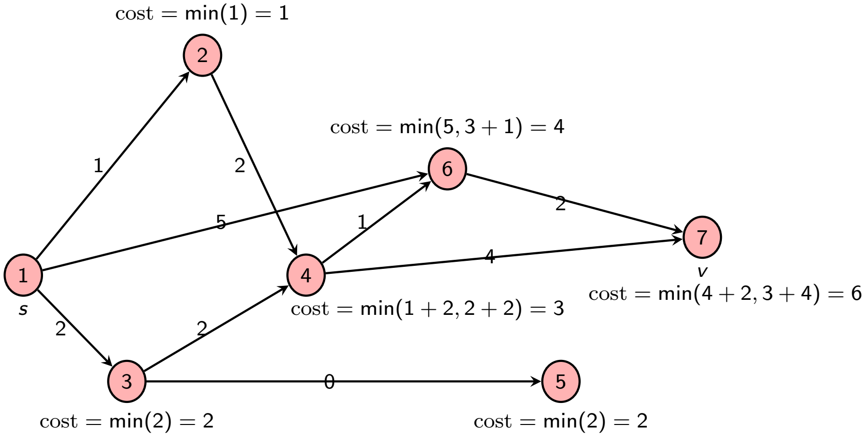
- We can avoid solving the same subproblem several times by saving the results in an array:

```
$F(n)$:
1: $f_1 \leftarrow 1$
2: $f_2 \leftarrow 1$
3: **for** $i \leftarrow 3$ to $n$ **do**
4:     $f_i \leftarrow f_{i-2} + f_{i-1}$
5: **return** $f_n$
```

Example: Shortest path in a DAG

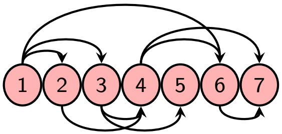
DAG=directed acyclic graph

Lightest path from s to v?



$\mathrm{cost} = \min(1) = 1$
$\mathrm{cost} = \min(5, 3+1) = 4$
$\mathrm{cost} = \min(1+2, 2+2) = 3$
$\mathrm{cost} = \min(4+2, 3+4) = 6$
$\mathrm{cost} = \min(2) = 2$
$\mathrm{cost} = \min(2) = 2$
1
2
3
4
5
6
7
s
v
1
2
5
1
2
4
2
2
0

Topological sort



Edit distance

- Consider an optimal listing of edits to convert the prefix $a_1 a_2 \ldots a_i$ of A into prefix $b_1 b_2 \ldots b_j$ of B
- Let the corresponding edit distance be $d_L(a_1 a_2 \ldots a_i, b_1 b_2 \ldots b_j)$
- If $a_i = b_j$, we know that $d_L(a_1 a_2 \ldots a_i, b_1 b_2 \ldots b_j) = d_L(a_1 a_2 \ldots a_{i-1}, b_1 b_2 \ldots b_{j-1})$
- Otherwise either a_i is substituted by b_j, or a_i is deleted, or b_j is inserted in the optimal list of edits
- Hence we have

$$d_L(a_1 a_2 \ldots a_i, b_1 b_2 \ldots b_j) = \\ \min \begin{cases} d_L(a_1 a_2 \ldots a_{i-1}, b_1 b_2 \ldots b_{j-1}) + (\text{if } a_i = b_j \text{ then } 0 \text{ else } 1) \\ d_L(a_1 a_2 \ldots a_{i-1}, b_1 b_2 \ldots b_j) + 1 \\ d_L(a_1 a_2 \ldots a_i, b_1 b_2 \ldots b_{j-1}) + 1 \end{cases}$$

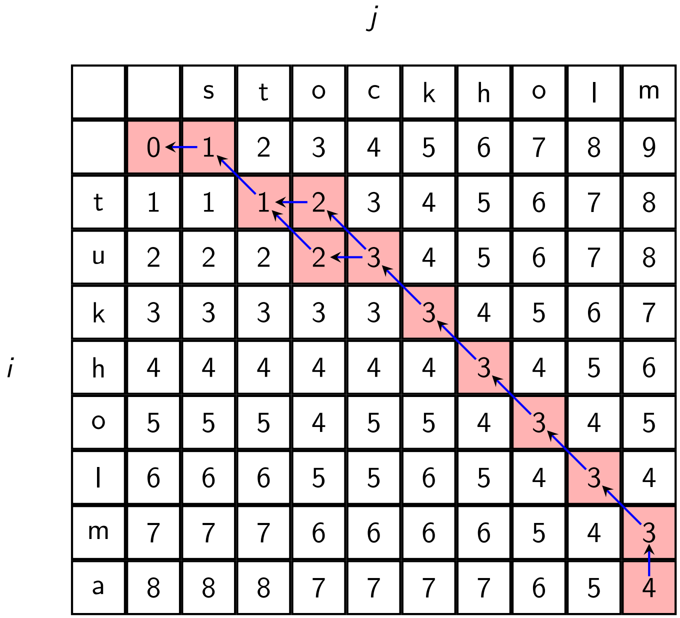
Edit distance matrix $D[i,j]$

- Let $D[i,j]$ denote $d_L(a_1a_2\dots a_i, b_1b_2\dots b_j)$.
- Obviously $D[0,j] = j$ and $D[i,0] = i$ because the other prefix is of length 0
- Induction from previous slide gives:

$$D[i,j] = \min \begin{cases} D[i-1,j-1] + (\text{if } a_i = b_j \text{ then } 0 \text{ else } 1) \\ D[i-1,j] + 1 \\ D[i,j-1] + 1 \end{cases}$$

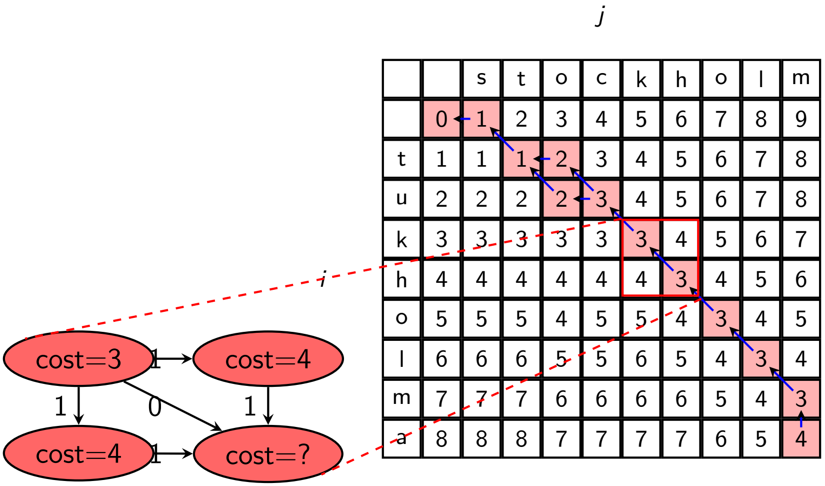
- Matrix D can be computed in many evaluation orders:
  - $D[i-1,j-1]$, $D[i-1,j]$, and $D[i,j-1]$ must be available when computing $D[i,j]$
  - E.g. compute D row-by-row, column-by-column...
- Running time to compute $D[m,n]$ is $O(mn)$

Edit distance: example



		s	t	o	c	k	h	o	l	m
	0	1	2	3	4	5	6	7	8	9
t	1	1	1	2	3	4	5	6	7	8
u	2	2	2	2	3	4	5	6	7	8
k	3	3	3	3	3	3	4	5	6	7
h	4	4	4	4	4	4	3	4	5	6
o	5	5	5	4	5	5	4	3	4	5
l	6	6	6	5	5	6	5	4	3	4
m	7	7	7	6	6	6	6	5	4	3
a	8	8	8	7	7	7	7	6	5	4

Edit distance matrix as a DAG



		s	t	o	c	k	h	o	l	m
	0	1	2	3	4	5	6	7	8	9
t	1	1	1	2	3	4	5	6	7	8
u	2	2	2	2	3	4	5	6	7	8
k	3	3	3	3	3	3	4	5	6	7
h	4	4	4	4	4	4	3	4	5	6
o	5	5	5	4	5	5	4	3	4	5
l	6	6	6	5	5	6	5	4	3	4
m	7	7	7	6	6	6	6	5	4	3
a	8	8	8	7	7	7	7	6	5	4

$$\mathrm{cost} = \min(3+0, 4+1, 4+1) = 3$$

Finding optimal alignments

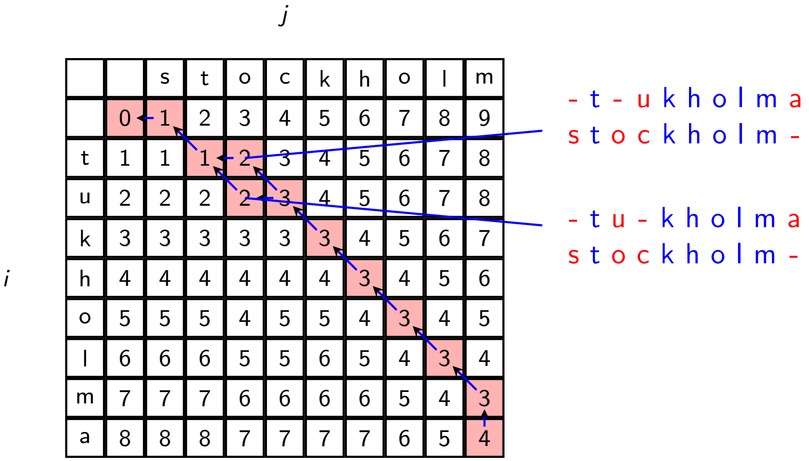
One alignment:

- Store pointer to each cell telling from which cell the minimum was obtained.
- Follow the pointers from (m, n) to $(0, 0)$.
- Reverse the list.

All alignments:

- Backtrack from (m, n) to $(0, 0)$ by checking at each cell (i, j) on the path whether the value $D[i, j]$ could have been obtained from cell $(i, j - 1)$, $(i - 1, j - 1)$, or $(i - 1, j)$.
- Explore all directions.
  - All three directions possible.
  - Exponential number of optimal paths in the worst case.

Edit distance: example



		s	t	o	c	k	h	o	l	m
	0	1	2	3	4	5	6	7	8	9
t	1	1	1	2	3	4	5	6	7	8
u	2	2	2	2	3	4	5	6	7	8
k	3	3	3	3	3	3	4	5	6	7
h	4	4	4	4	4	4	3	4	5	6
o	5	5	5	4	5	5	4	3	4	5
l	6	6	6	5	5	6	5	4	3	4
m	7	7	7	6	6	6	6	5	4	3
a	8	8	8	7	7	7	7	6	5	4

```
- t - u k h o l m a
s t o c k h o l m -
```

```
- t u - k h o l m a
s t o c k h o l m -
```

Searching homologs with edit distance?

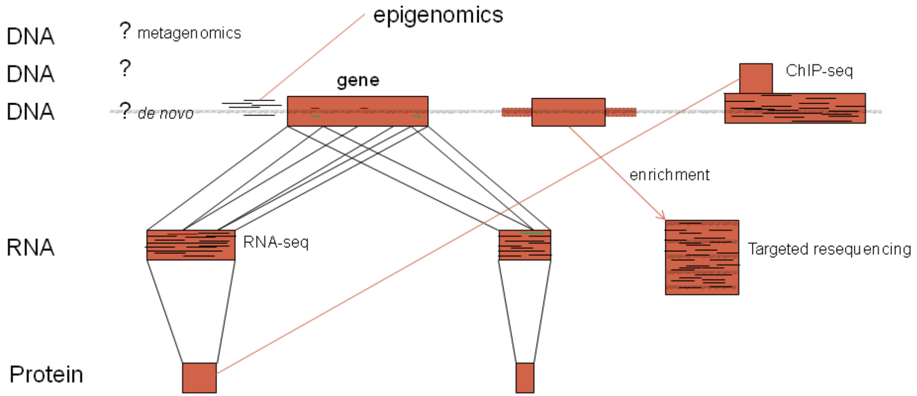
- Take DNA sequences A and B of two genes suspected to be homologs.
- Edit distance of A and B can be *huge* even if A and B are true homologs:
  - One reason is *silent mutations* that alter DNA sequence so that the codons still encode the same amino acids
  - In principle, A and B can differ in almost every third nucleotide.
- Better to compare protein sequences.
  - Some substitutions are more likely than the others...
  - Lot of tuning needed to use proper weight for operations

Better models $\Longrightarrow$ 582483 Biological Sequence Analysis (4cr), period III

Edit distance and NGS

- High-throughput next-generation sequencing (NGS) has raised again the issue of using edit distance.
- Short DNA *reads* (50-1000 bp) a.k.a. *patterns* are measured from e.g. cells of a patient.
- The reads are aligned against the reference genome
  - Typically only SNPs and measurement errors need to be taken into account.
  - The occurrence of the reads in the reference genome can be determined by finding the substring of the genome whose edit distance (or Hamming distance) to the reads is minimum.
  - Approximate string matching problem.

NGS: RNA-seq, ChIP-seq, (targeted) resequencing, *de novo* sequencing, metagenomics, ...



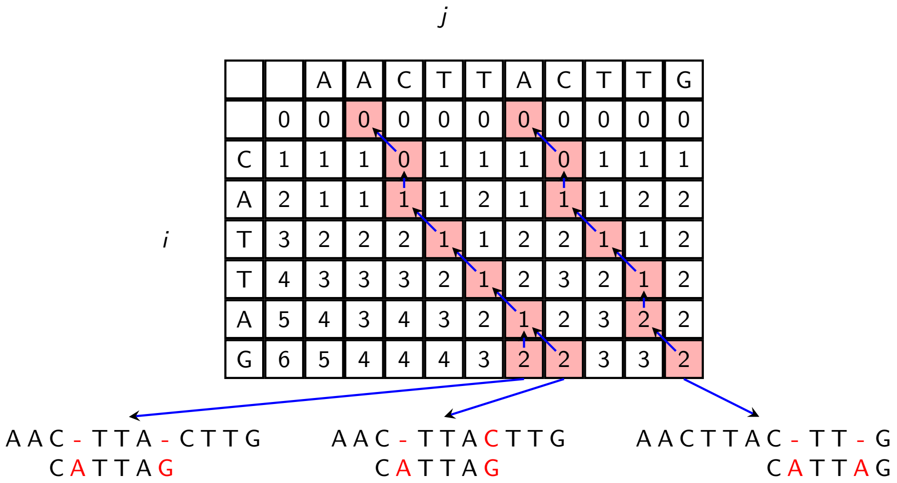
Approximate string matching with Hamming distance d_H

- k-mismatches problem: Search all occurrences O of pattern $P[1, m]$ in text $T[1, n]$ such that P differs in at most k positions from the occurrence substring.
  - More formally: $j \in O$ is a k-mismatch occurrence position of P in T if $d_H(P, T[j, j + m - 1]) \leq k$
- Naive algorithm:
  - Compare P against each $T[j, j + m - 1]$ but skip as soon as $k + 1$ mismatches are encountered.
  - Worst case time $O(mn)$ but expected time $O(kn)$.

Approximate string matching with edit distance d_L

- *k*-errors problem is the approximate string matching problem with edit distance:
  - More formally: $j \in O$ is a k-errors occurrence with (end)position j of P in T if and only if $d_L(P, T[j', j]) \leq k$ for some j'.
- Can be solved with the "zero the first row trick":
  - $D[0, j] = 0$ for all j.
  - Otherwise the computation is identical to edit distance computation using matrix D.
  - $D[i, j]$ then equals the minimum number of edits to convert $P[1, i]$ into *some suffix of* $T[1, j]$.
  - If $D[m, j] \leq k$ then P can be converted to some substring $T[j', j]$ with at most k edit operations.

Approximate string matching: example



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

```
A A C - T T A - C T T G
    C A T T A G
```

```
A A C - T T A C T T G
    C A T T A G
```

```
A A C T T A C - T T - G
            C A T T A G
```

NGS and approximate string matching 1/3

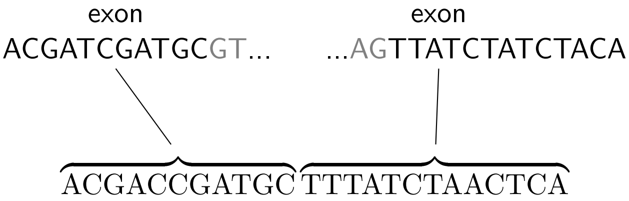
- Aligning reads from ChIP-seq and targeted sequences works using basic approximate string matching.
- Tens of millions of reads, speed is an issue.
- Reference genome can be preprocessed to speed up search.
- Suffix tree like techniques work but...
  - Suffix tree of human genome takes 50-200 GB!
  - More space-efficient index structures have been developed (e.g. based on *Burrows-Wheeler transform* that drop the space to $\sim$ 3 GB).

Faster algorithms $\Longrightarrow$ 58093 String Processing Algorithms (5 cr), period II

Space-efficient indexes $\Longrightarrow$ 582487 Data Compression Techniques (4 cr), period III

NGS atlas and approximate string matching 2/3

- Reads from RNA-seq need more advanced alignment:
  - Read can span two exons



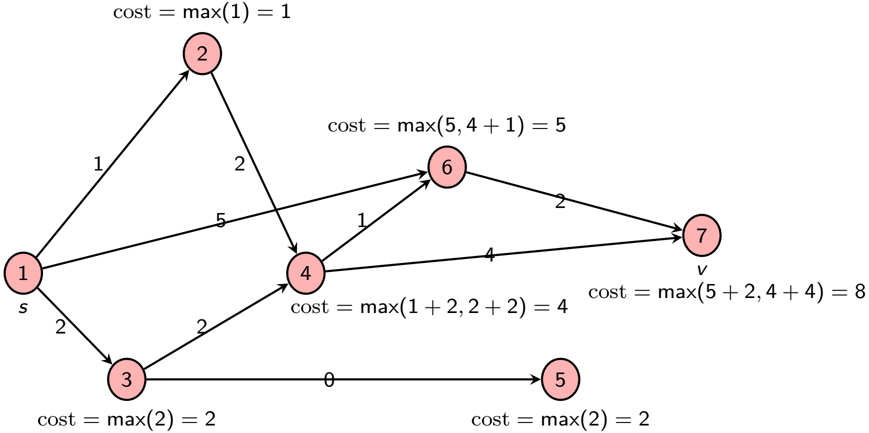
```
A C G A T C G A T G C T T T A T C T A T C T A C A
A C G A C C G A T G C T T T A T C T A A C T - C A
```

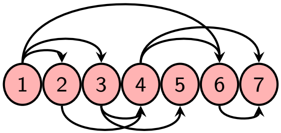
NGS and approximate string matching 3/3

- *de novo* sequenceing and metagenomics are much harder since there is no reference genome.
- Shortest approximate superstring (exercise 2.4)
- How to modify edit distance computations for overlaps?
    - Next week's exercise

Variations: Heaviest path in a DAG

Heaviest path from s to v?



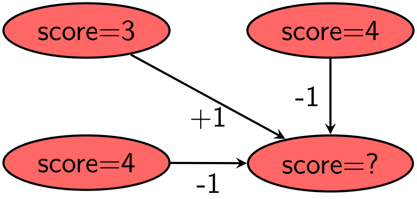


Topological sort

Heaviest paths in sequence alignment

- Consider the DAG of edit distance matrix.
- Turn minimization into maximization.
- Give *score* $\delta(a_i, b_j)$ for diagonal edges.
- Give *score* $\delta(a_i, -)$ for vertical edges.
- Give *score* $\delta(-, b_j)$ for horizontal edges.
- Longest path in the DAG corresponds to the global alignment with highest score
- Typically $\delta(a_i, b_j) = 1$ if $a_i = b_j$ and otherwise $\delta(a_i, b_j) = -\mu$
- Typically $\delta(a_i, -) = \delta(-, b_j) = -\sigma$

Global alignment DAG and recurrence



$$\text{score} = \max(3+1, 4-1, 4-1) = 4$$

$$S[i,j] = \max \begin{cases} S[i-1,j-1] + \delta(a_i, b_j) \\ S[i-1,j] + \delta(a_i, -) \\ S[i,j-1] + \delta(-, b_j) \end{cases}$$

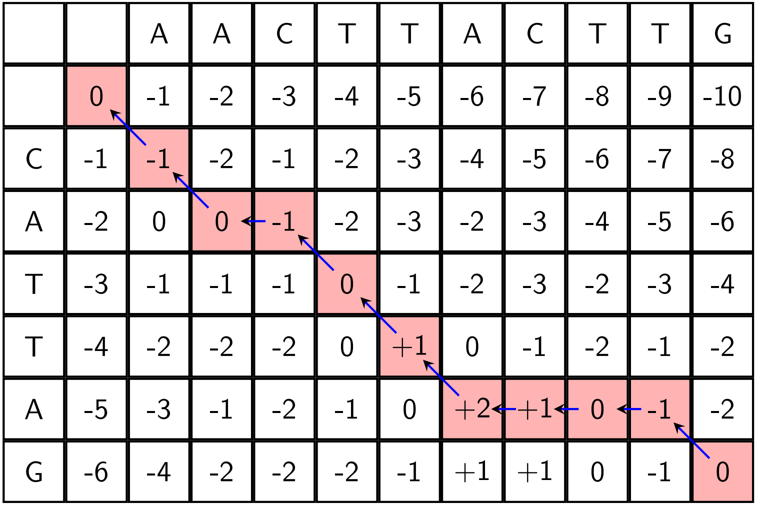
Global alignment: Example

$\delta(a_i, b_j) = 1$, if $a_i = b_j$
$\delta(a_i, b_j) = -1$, otherwise

$\delta(a_i, -) = \delta(-, b_j) = -1$

j

i

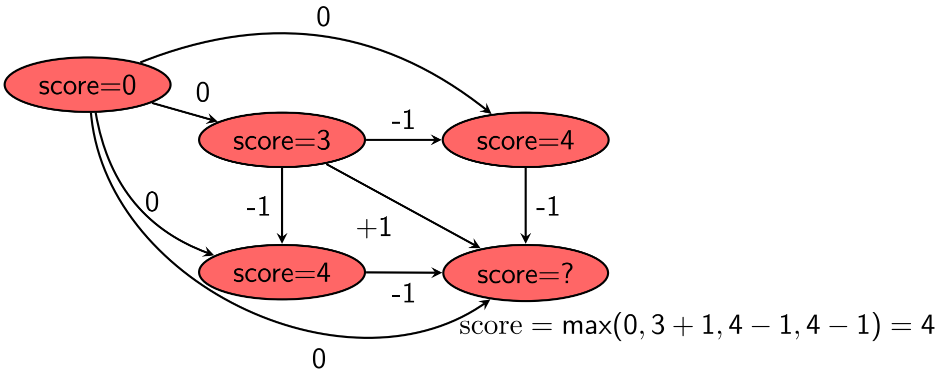


		A	A	C	T	T	A	C	T	T	G
	0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
C	-1	-1	-2	-1	-2	-3	-4	-5	-6	-7	-8
A	-2	0	0	-1	-2	-3	-2	-3	-4	-5	-6
T	-3	-1	-1	-1	0	-1	-2	-3	-2	-3	-4
T	-4	-2	-2	-2	0	+1	0	-1	-2	-1	-2
A	-5	-3	-1	-2	-1	0	+2	+1	0	-1	-2
G	-6	-4	-2	-2	-2	-1	+1	+1	0	-1	0

Heaviest *local* paths in sequence alignment

- How to find heaviest subpaths (local path)?
- Define that the empty path has score 0.
- It is enough to search for subpaths (local paths) with weight greater than 0.
- No heaviest path can have a prefix with negative score
- Add an edge with score 0 from the first node to all other nodes.

Local alignment DAG and recurrence



$$S[i,j] = \max \begin{cases} 0 \\ S[i-1,j-1] + \delta(a_i, b_j) \\ S[i-1,j] + \delta(a_i, -) \\ S[i,j-1] + \delta(-, b_j) \end{cases}$$

Local alignment: Example

$\delta(a_i, b_j) = 1$, if $a_i = b_j$
$\delta(a_i, b_j) = -1$, otherwise

$\delta(a_i, -) = \delta(-, b_j) = -1$

j

i

		A	A	C	T	T	A	C	T	T	G
	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	1	0	0	0	1	0	0	0
A	0	1	1	0	0	0	1	0	0	0	0
T	0	0	0	0	1	1	0	0	1	1	0
T	0	0	0	0	1	2	1	0	1	2	1
A	0	1	1	0	0	1	3	2	1	1	1
G	0	0	0	0	0	0	2	2	1	0	2

Longest common subsequence

- Global alignment with
  - $\delta(a_i, b_j) = 1$ when $a_i = b_j$ and otherwise $\delta(a_i, b_j) = -\infty$
  - $\delta(a_i, -) = \delta(-, b_j) = 0$

  gives the length of the longest common subsequence C of A and B:
  - Longest sequence C that can be obtained by deleting 0 or more symbols from A and also by deleting 0 or more symbols from B.

    AACGCATACGG ACGACTGATCG

    ACGCTACG
  - Connection: $d_{\mathrm{ID}}(A, B) = m + n - 2 \cdot |\mathrm{LCS}(A, B)|$, where $d_{\mathrm{ID}}(A, B)$ is the edit distance with substitution cost ∞

Outline

Study Group 1: Random assignment on lecture

- Read pages 42–45 from Sung: Algorithms in Bioinformatics: A Practical Introduction, CRC Press 2010
  - General gap penalty model
  - Affine gap penalty model
  - Copies distributed at the lecture
- In the study group
  - Explain the idea of each of the tables in the recurrence for the affine gap model: V, G, F, and E.
  - What is the best global alignment of CGAGAT and CAT using the affine gap model? Use cost +4 for a match, -2 for mismatch, -3 for gap opening, -1 for gap extension. What is the score of the alignment?

Study Group 2: Random assignment on lecture

- Read pages 203–207 from Jones and Pevzner.
  - Gene prediction by spliced alignment:
  - Application/extension of heaviest path on a DAG
  - Copies distributed at the lecture
- At study group, explain the idea visually and explain how the recurrences are derived. What is the running time of the algorithm?

Study Group 3: Those without a handout on lecture

- Read the following article before coming to the study group:

  Sear R. Eddy: How do RNA folding algorithms work? *Nature Biotechnology* **22**, 1457 - 1458 (2004).

  `http://www.nature.com/nbt/journal/v22/n11/abs/nbt1104-1457.html`

  - RNA secondary structure prediction.
  - Basic dynamic programming formulation.
- At study group, give an example of RNA secondary structure, how the recurrence is derived for its computation, and how the recurrence is evaluated.