Prelude to Sequence Alignment

**Content**

- General results in Combinatorial Pattern Matching / Stringology
  - Knuth-Morris-Pratt
  - Boyer-Moore
  - Suffix tree, Suffix array
  - Edit distance
  - Dynamic Programming
  - Approximate pattern search, k-mismatches, k-errors

- Solutions specific to Bioinformatics
  - Needleman-Wunsch (global alignment, score matrixes)
  - Smith-Waterman (local alignment)
  - FASTA, BLAST
  - ...
Classical results from Stringology

- The world of "text of length $n$, pattern of length $m$".
- Knuth-Morris-Pratt: $O(n)$ time exact pattern search.
- Boyer-Moore: $O(n/m)$ time exact pattern search on average.
- Powerful general tools: *Suffix tree* and *suffix array*
- Numerous theoretical results on approximate pattern matching

pattern CAC

text AGAT.....CAC....CAC...GTAT
- Suffix tree is a compressed keyword trie of all suffixes of a sequence.
- E.g. suffixes of sequence CATACT are CATACT, ATACT, TACT, ACT, CT, T.
  - suffix tree looks like:
Suffix tree

582606 Introduction to Bioinformatics, Autumn 2009

22.-24. Sept / 4
Suffix tree

A
2
4

C
T
T

C
T

A
= [2,2]

T
= [3,3]

A

C
T
A
T
A
T

= [3,6]

= [2,6]

= [6,6]

= [4,6]

C
A
T

= [5,6]

1
2
3
4
5
6

582606 Introduction to Bioinformatics, Autumn 2009  22.-24. Sept / 5
Exact search on suffix tree

pattern = C

C A T A C T
1 2 3 4 5 6
Backtracking on suffix tree

ACA, 1 mismatch (SNP)

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

582606 Introduction to Bioinformatics, Autumn 2009
Simple analysis task: LCSS

- Let LCSS(A,B) denote the longest common substring of two sequences A and B. E.g.:
  - LCSS(AGATCTATCT, CGCCTCTATG)=TCTAT.
- A good solution is to build suffix tree for the shorter sequence and make a descending suffix walk with the other sequence.
Read B left-to-right, always going down in the tree when possible. If the next symbol of B does not match any edge label on current position, take suffix link, and try again. (Suffix link in the root to itself emits a symbol). The node v encountered with largest string depth is the solution.
Another common tool: Generalized suffix tree

node info:
subtree size 47813871
sequence count 87

ACCTTA....ACCT#CACATT..CAT#TGTCGT...GTA#TCACCACC...C$
Generalized suffix tree application

...ACC...#...ACC...#...ACC...ACC..#..ACC..ACC...#...ACC...#...
...#....#...#...#...ACC...#...#...#...#...#...#..#..ACC..ACC...#......#...

node info:
subtree size 4398
blue sequences 12/15
red sequences 2/62
Properties of suffix tree

- Suffix tree has \( n \) leaves and at most \( n-1 \) internal nodes, where \( n \) is the total length of all sequences indexed.
- Each node requires constant number of integers (pointers to first child, sibling, parent, text range of incoming edge, statistics counters, etc.).
- Can be constructed in linear time (e.g. Ukkonen's online linear time construction).
- In practice: Huge overhead due to pointer structure:
  - Standard implementation of suffix tree for human genome requires over 200 GB memory!
Reducing space: suffix array

C A T A C T

1 2 3 4 5 6
Suffix array

- Many algorithms on suffix tree can be simulated using suffix array.
- For example, exact pattern search works using binary search on suffix array.
- Suffix array is the basis for a popular bioinformatics tool called *Mummer*.
- Suffix array can be constructed easily from suffix tree, but there are also direct linear time construction algorithms that take less space (e.g. Kärkkäinen & Sanders algorithm).
Approximate string matching

- **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 and only if $d_H(P, T[j,j+m-1]) \leq k$, where $d_H(A,B)$ is the Hamming distance of $A$ and $B$.
  - Theory: $O(kn)$ time algorithm is easy to achieve (using suffix trees and some advanced data structure techniques) and very sophisticated algorithms exist to solve the problem even faster.
  - Practice: naive algorithm or backtracking on suffix tree (slide 7) work well for small $k$. 

582606 Introduction to Bioinformatics, Autumn 2009  22.-24. Sept / 16
Approximate pattern matching: filtering

- Best practical algorithms for approximate string matching use **filtering**:
  - Sweep the text with a fast algorithm to detect possible candidate occurrence positions.
  - Check all candidates for real occurrences.
  - There are *noisy filters* (that may fail to find some candidates that are real occurrences) and *noiseless filters* (that are guaranteed to find all real occurrences).

- Simple noiseless filter for k-mismatch search:
  - Partition the pattern into $k+1$ pieces.
  - Take all exact occurrences of the pieces as candidates.
  - Check all candidates with naive algorithm.
Approximate string matching: filtering example

- Text  \( T = \text{CGAGCGATAGCTACCGT} \)
- Pattern \( P = \text{ACAG}, k=1 \)
- Partition \( P \) into e.g. \( P^1 = \text{AC}, P^2 = \text{AG} \)
- Search \( P^1 \) and \( P^2 \) in \( T: \text{CGAGCGATAGCTACCGT} \)
- Check the candidates: \( \text{CGAGCGATAGCTACCGT} \)
- Running time:
  - Build suffix tree of \( T: O(n) \) time.
  - Search \( P^1 \) and \( P^2 \) in suffix tree of \( T: O(m + \#candidates) \) time.
  - Checking \( O(\#candidates \times m) \) time.
- The challenge: \#candidates >> \#occurrences
  - Better filters than above exist (with smaller \#candidates)
Different versions of approximate pattern matching can be defined modifying the distance function $d(A,B)$.

The most studied distance function 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, on $A=$ "stockholm" and $B=$ "tukholma" we have $d_L(A,B)=4$:
  - delete s, substitute o->u, delete c, insert a
  - .. or delete s, delete o, substitute c->u, insert a
  - .. or is there better sequence of edits???

\[ s t o c k h o l m -
- t u - k h o l m a \]
Dynamic programming

- Way to compute edit distance optimally.
- General algorithm principle:
  - Can be seen as a variant of Dijkstra's shortest path algorithm.
- Abstract idea: Use induction to break the problem into smaller subproblems and suitable evaluation order so that subproblem solutions are available when needed.
- Concrete example, Fibonacci numbers:
  - \(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, \ldots\)
  - \(F(i) = F(i-2) + F(i-1)\) with \(F(0) = 0, F(1) = 1\)
  - The recursion to compute \(F(i)\) contains many identical subproblems.
Edit distance

- Let $A=a_1a_2...a_m$ and $B=b_1b_2...b_n$ be two strings.
- Consider an optimal listing of edits to convert the prefix $a_1a_2...a_i$ of $A$ into prefix $b_1b_2...b_j$ of $B$ corresponding to $d_L(a_1a_2...a_i,b_1b_2...b_j)$:
  - If $a_i=b_j$ we know that $d_L(a_1a_2...a_i,b_1b_2...b_j)=d_L(a_1a_2...a_{i-1},b_1b_2...b_{j-1})$
  - 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 $d_L(a_1a_2...a_i,b_1b_2...b_j)=\min(d_L(a_1a_2...a_{i-1},b_1b_2...b_{j-1})+(\text{if } a_i=b_j \text{ then } 0 \text{ else } 1), d_L(a_1a_2...a_i,b_1b_2...b_j)+1, d_L(a_1a_2...a_i,b_1b_2...b_{j-1})+1)$. 


Let $D[i,j]$ denote $d_L(a_1a_2...a_i,b_1b_2...b_j)$.

Obviously $D[0,j]=j$ and $D[i,0]=i$.

The induction from previous slide gives

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

Matrix $D$ can be computed row-by-row, column-by-column (or in many other evaluation orders) so that $D[i-1,j-1]$, $D[i-1,j]$, and $D[i,j-1]$ are available when computing $D[i,j]$.

Running time to compute $D[m,n]$ is $O(mn)$. 
Edit distance example

<table>
<thead>
<tr>
<th>s</th>
<th>t</th>
<th>o</th>
<th>c</th>
<th>k</th>
<th>h</th>
<th>o</th>
<th>l</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>t</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>u</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>k</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>h</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>o</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>l</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>m</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
- **k-errors problem** is the approximate string matching problem with edit distance:
  - More formally: \( j \in O \) is a k-errors occurrence (end)position 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 \( D \).
  - Intuition: \( D[i,j] \) 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.
Current applications

- Short-read sequencing (454, Solexa, SOLiD) has raised again the issue of doing fast k-mismatches and k-errors matching.
- Some popular software packages exploit the suffix tree backtracking idea (bowtie, bwa, SOAP2):
  - Instead of suffix tree, a compressed suffix array based on so-called Burrows-Wheeler transform is used as backbone of the search.
  - The index size for e.g. human genome can be kept in ~3 GB.
  - Compression does not affect the running time significantly.
More on general string processing techniques...

- Gusfield's book: *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*
- 58093-3 Merkkijonomenetelmät (String Processing Algorithms)
  - Lectured previously Autumn 2008.
  - Next time Autumn 2010 in English?
Sequence alignment

- *The biological problem*
- Global alignment
- Local alignment
- Multiple alignment
Background: comparative genomics

- Basic question in biology: *what properties are shared among organisms?*
- Genome sequencing allows comparison of organisms at DNA and protein levels
- Comparisons can be used to
  - Find evolutionary relationships between organisms
  - Identify functionally conserved sequences
  - Identify corresponding genes in human and model organisms: develop models for human diseases
Homologs

- Two genes (sequences in general) $g_B$ and $g_C$ evolved from the same ancestor gene $g_A$ are called *homologs*

- Homologs usually exhibit conserved functions

- Close evolutionary relationship => expect a high number of homologs

$g_A = \text{agtgtccgtaagtgcttc}$

$g_B = \text{agtgcgttaagttgtacgtc}$

$g_C = \text{ctgacttgttggtggttc}$
Sequence similarity

- We expect homologs to be "similar" to each other
- Intuitively, similarity of two sequences refers to the degree of match between corresponding positions in sequence

\[ \text{agtgcggtaaagttgtacgtc} \]
\[ \text{ctgactgttttgtggttcc} \]

- What about sequences that differ in length?
Similarity vs homology

- **Sequence similarity is not sequence homology**
  - If the two sequences $g_B$ and $g_C$ have accumulated enough mutations, the similarity between them is likely to be low

Homology is more difficult to detect over greater evolutionary distances.
Similarity vs homology (2)

- Sequence similarity can occur by chance
  - *Similarity does not imply homology*

- Consider comparing two short sequences against each other
We distinguish between two types of homology

- **Orthologs**: homologs from two different species, separated by a *speciation* event.
- **Paralogs**: homologs within a species, separated by a *gene duplication* event.

![Diagram showing orthologs and paralogs](image-url)
Orthologs and paralogs (2)

- Orthologs typically retain the original function.
- In paralogs, one copy is free to mutate and acquire new function (no selective pressure).

![Diagram showing orthologs and paralogs between three organisms: A, B, and C.](image)
Paralogy example: hemoglobin

- Hemoglobin is a protein complex which transports oxygen
- In humans, hemoglobin consists of four protein subunits and four non-protein heme groups

Sickle cell diseases are caused by mutations in hemoglobin genes

Hemoglobin A, 
www.rcsb.org/pdb/explore.do?structureId=1GZX

In adults, three types are normally present:
- Hemoglobin A: 2 alpha and 2 beta subunits
- Hemoglobin A2: 2 alpha and 2 delta subunits
- Hemoglobin F: 2 alpha and 2 gamma subunits

Each type of subunit (alpha, beta, gamma, delta) is encoded by a separate gene.

Hemoglobin A,
www.rcsb.org/pdb/explore.do?structureId=1GZX
Paralogy example: hemoglobin

- The subunit genes are paralogs of each other, i.e., they have a common ancestor gene
- Exercise: hemoglobin human paralogs in NCBI sequence databases
  - Find human hemoglobin alpha, beta, gamma and delta
  - Compare sequences

Hemoglobin A,
www.rcsb.org/pdb/explore.do?structureId=1GZX
Orthology example: insulin

- The genes coding for insulin in human (*Homo sapiens*) and mouse (*Mus musculus*) are orthologs:
  - They have a common ancestor gene in the ancestor species of human and mouse
  - Exercise: find insulin orthologs from human and mouse in NCBI sequence databases
Sequence alignment

- Alignment specifies which positions in two sequences match

```
acgtctag    acgtctag    acgtctag
||       ||      ||
actctag-   -actctag  ac-tctag
```

2 matches    5 matches    7 matches
5 mismatches 2 mismatches 0 mismatches
1 not aligned 1 not aligned 1 not aligned
Sequence alignment

- Maximum alignment length is the total length of the two sequences

```
acgtctag------  ------acgtctag

-------actctag  actctag--------
```

0 matches 0 matches
0 mismatches 0 mismatches
15 not aligned 15 not aligned
Mutations: Insertions, deletions and substitutions

- **Indel**: insertion or deletion of a base with respect to the ancestor sequence

- **Mismatch**: substitution (point mutation) of a single base

- Insertions and/or deletions are called *indels*
  - *We can’t tell whether the ancestor sequence had a base or not at indel position!*

\[\text{acgtctag} \quad \text{ac} \quad \text{ctctag}\]
Problems

- What sorts of alignments should be considered?
- How to score alignments?
- How to find optimal or good scoring alignments?
- How to evaluate the statistical significance of scores?

In this course, we discuss each of these problems briefly.
Sequence Alignment (chapter 6)

- The biological problem
- *Global alignment*
- Local alignment
- Multiple alignment
Global alignment

- Problem: find optimal scoring alignment between two sequences (Needleman & Wunsch 1970)
- Every position in both sequences is included in the alignment
- We give score for each position in alignment
  - Identity (match) +1
  - Substitution (mismatch) -\( \mu \)
  - Indel -\( \delta \)
- Total score: sum of position scores
**Scoring: Toy example**

- Consider two sequences with characters drawn from the English language alphabet: WHAT, WHY

\[
S(\text{WHAT}/\text{WH-Y}) = 1 + 1 - \delta - \mu
\]

\[
S(\text{WHAT}/-\text{WHY}) = -\delta - \mu - \mu - \mu
\]
Alignments can be represented in the following tabular form.

Each alignment corresponds to a path through the table.

<table>
<thead>
<tr>
<th></th>
<th>W</th>
<th>H</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Representing alignments and scores

WH-AT

WHY--

WHAT---

-----WHY
### Representing alignments and scores

**Global alignment score** $S_{3,4} = 2-\delta-\mu$

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>W</th>
<th>H</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>2 $2-\delta$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>$2-\delta-\mu$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**WHAT**

||
---|---|
**WH-Y**

582606 Introduction to Bioinformatics, Autumn 2009

22.-24. Sept / 48
Filling the alignment matrix

Consider the alignment process at shaded square.

**Case 1.** Align H against H (match)

**Case 2.** Align H in WHY against -- (indel) in WHAT

**Case 3.** Align H in WHAT against -- (indel) in WHY
Filling the alignment matrix (2)

Scoring the alternatives.

Case 1. \( S_{2,2} = S_{1,1} + s(2, 2) \)

Case 2. \( S_{2,2} = S_{1,2} - \delta \)

Case 3. \( S_{2,2} = S_{2,1} - \delta \)

\( s(i, j) = 1 \) for matching positions,
\( s(i, j) = -\mu \) for substitutions.

Choose the case (path) that yields the maximum score.

Keep track of path choices.
Global alignment: formal development

\[ A = a_1 a_2 a_3 \ldots a_m, \]
\[ B = b_1 b_2 b_3 \ldots b_n \]

\[ b_1 \ b_2 \ b_3 \ b_4 \ - \]
\[ - \ a_1 \ - \ a_2 \ a_3 \]

- Any alignment can be written as a unique path through the matrix
- Score for aligning A and B up to positions i and j:

\[ S_{i,j} = S(a_1 a_2 a_3 \ldots a_i, b_1 b_2 b_3 \ldots b_j) \]
Scoring partial alignments

- Alignment of $A = a_1 a_2 a_3 \ldots a_i$ with $B = b_1 b_2 b_3 \ldots b_j$ can be end in three possible ways
  - Case 1: $(a_1 a_2 \ldots a_{i-1}) a_i$
    $$(b_1 b_2 \ldots b_{j-1}) b_j$$
  - Case 2: $(a_1 a_2 \ldots a_{i-1}) a_i$
    $$(b_1 b_2 \ldots b_j) -$$
  - Case 3: $(a_1 a_2 \ldots a_i) -$  
    $$(b_1 b_2 \ldots b_{j-1}) b_j$$
Scoring alignments

- Scores for each case:
  - Case 1: \((a_1a_2\ldots a_{i-1}) a_i\) 
    
    \((b_1b_2\ldots b_{j-1}) b_j\)
  - Case 2: \((a_1a_2\ldots a_{i-1}) a_i\) 
    
    \((b_1b_2\ldots b_j) -\)
  - Case 3: \((a_1a_2\ldots a_i) -\) 
    
    \((b_1b_2\ldots b_{j-1}) b_j\)

\[s(a_i, b_j) = \begin{cases} 
+1 & \text{if } a_i = b_j \\
-\mu & \text{otherwise} 
\end{cases}\]

\[s(a_i, -) = s(-, b_j) = -\delta\]
Scoring alignments (2)

- First row and first column correspond to initial alignment against indels:
  \[ S(i, 0) = -i \delta \]
  \[ S(0, j) = -j \delta \]

- Optimal global alignment score
  \[ S(A, B) = S_{m,n} \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-(\delta)</td>
<td>-2(\delta)</td>
<td>-3(\delta)</td>
</tr>
<tr>
<td>1</td>
<td>(a_1)</td>
<td>-(\delta)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(a_2)</td>
<td>-2(\delta)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(a_3)</td>
<td>-3(\delta)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Algorithm for global alignment

Input sequences $A$, $B$, $m = |A|$, $n = |B|$

Set $S_{i,0} := -\delta i$ for all $i$
Set $S_{0,j} := -\delta j$ for all $j$

for $i := 1$ to $m$
  for $j := 1$ to $n$
    $S_{i,j} := \max\{S_{i-1,j} - \delta, S_{i-1,j-1} + s(a_i,b_j), S_{i,j-1} - \delta\}$
  end
end

Algorithm takes $O(mn)$ time
Global alignment: example

\[ \mu = 1 \]
\[ \delta = 2 \]

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-6</td>
<td>-8</td>
<td>-10</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Global alignment: example

\[
\begin{array}{c}
\mu = 1 \\
\delta = 2 \\
\end{array}
\]

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-6</td>
<td>-8</td>
<td>-10</td>
</tr>
<tr>
<td>A</td>
<td>-2</td>
<td>-1</td>
<td>-3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>-6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>-8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>-10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[\mu = 1\]
\[\delta = 2\]
Global alignment: example (2)

\[ \mu = 1 \]
\[ \delta = 2 \]

\[
\begin{array}{cccccccc}
& & & & & & & \\
& - & T & G & G & T & G & \\
- & 0 & -2 & -4 & -6 & -8 & -10 & \\
A & -2 & -1 & -3 & -5 & -7 & -9 & \\
T & -4 & -1 & -2 & -4 & -4 & -6 & \\
C & -6 & -3 & -2 & -3 & -5 & -5 & \\
G & -8 & -5 & -2 & -1 & -3 & -4 & \\
T & -10 & -7 & -4 & -3 & 0 & -2 & \\
\end{array}
\]
Sequence Alignment (chapter 6)

- The biological problem
- Global alignment
- *Local alignment*
- Multiple alignment
Local alignment: rationale

- Otherwise dissimilar proteins may have local regions of similarity
  -> Proteins may share a function

Human bone morphogenic protein receptor type II precursor (left) has a 300 aa region that resembles 291 aa region in TGF-β receptor (right).

The shared function here is protein kinase.
Local alignment: rationale

- Global alignment would be inadequate
- Problem: find the highest scoring *local* alignment between two sequences
- Previous algorithm with minor modifications solves this problem (Smith & Waterman 1981)
From global to local alignment

- Modifications to the global alignment algorithm
  - Look for the highest-scoring path \textbf{in} the alignment matrix (not necessarily through the matrix), or in other words:
  - Allow preceding and trailing indels without penalty
Scoring local alignments

Let I and J be intervals (substrings) of A and B, respectively:

\[ I \subset A \quad J \subset B \]

Best local alignment score:

\[ M(A, B) = \max \{ S(I, J) : I \subset A, J \subset B \} \]

where \( S(I, J) \) is the alignment score for substrings I and J.
Allowing preceding and trailing indels

- First row and column initialised to zero:
  \[ M_{i,0} = M_{0,j} = 0 \]

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>b₁</th>
<th>b₂</th>
<th>b₃</th>
<th>b₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>a₁</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a₂</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a₃</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

582606 Introduction to Bioinformatics, Autumn 2009
### Recursion for local alignment

- \( M_{i,j} = \max \{ \)
  - \( M_{i-1,j-1} + s(a_i, b_j), \)
  - \( M_{i-1,j} - \delta, \)
  - \( M_{i,j-1} - \delta, \)
  - 0
- \( \}

Allow alignment to start anywhere in sequences
### Finding best local alignment

- **Optimal score is the highest value in the matrix**

\[
M(A, B) = \max \{ S(I, J) : I \subseteq A, J \subseteq B \} = \max_{i,j} M_{i,j}
\]

- **Best local alignment can be found by backtracking from the highest value in M**

- **What is the best local alignment in this example?**

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>T</th>
<th>G</th>
<th>G</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

---

582606 Introduction to Bioinformatics, Autumn 2009

22.-24. Sept / 66
### Local alignment: example

The alignment is computed using the following recurrence:

$$M_{i,j} = \max \{$$
$$M_{i-1,j-1} + s(a_i, b_j),$$
$$M_{i-1,j} - \delta, \quad \quad M_{i,j-1} - \delta,$$
$$0\}$$

Where $s(a_i, b_j)$ is the score for matching characters $a_i$ and $b_j$, and $\delta$ is the score for an indel.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>G</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Scoring (for example)**

- **Match**: +2
- **Mismatch**: -1
- **Indel**: -2
Local alignment: example

\[
M_{i,j} = \max \{ \\
M_{i-1,j-1} + s(a_i, b_i), \\
M_{i-1,j} - \delta, \\
M_{i,j-1} - \delta, \\
0 \}
\]

<table>
<thead>
<tr>
<th></th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>G</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scoring (for example)
- Match: +2
- Mismatch: -1
- Indel: -2
Local alignment: example

Optimal local alignment:

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>G</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>G</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Scoring (for example)

- Match: +2
- Mismatch: -1
- Indel: -2
Multiple optimal alignments
Non-optimal, good-scoring alignments

<table>
<thead>
<tr>
<th>How can you find</th>
<th>-</th>
<th>G</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>C</th>
<th>A</th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Optimal</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>alignments if</td>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>more than one</td>
<td></td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>exist?</td>
<td></td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>2. Non-optimal,</td>
<td></td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>good-scoring</td>
<td></td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>alignments?</td>
<td></td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

How can you find
1. Optimal alignments if more than one exist?
2. Non-optimal, good-scoring alignments?