Dynamic Programming: Edit Distance

Bioinformatics: Issues and Algorithms
CSE 308-408 • Fall 2007 • Lecture 10
Outline

• Setting the Stage
• DNA Sequence Comparison: First Successes
• The Change Problem
• The Manhattan Tourist Problem
• The Longest Common Subsequence Problem
• Sequence Alignment
• The Edit Distance Problem

http://www.bioalgorithms.info
Sequence comparison and alignment

Consider:

**AGTAGCATC**

versus

AGTAGCATC

**AGTGCACC**

versus

GACACGATT

That the two DNA fragments on the left somehow seem more similar than the two on the right could be significant.

Question: how can we measure sequence similarity?
**Motivation**

Why is this important?

- Given a new DNA sequence, one of the first things a biologist will want to do is search databases of known sequences to see if anyone has recorded something similar. (As we've seen, genetic sequences are long and the databases are enormous, so efficiency will be an issue.)

- Sequence similarity can provide clues about function.

- Similarity can provide clues about evolutionary relationships.

- Many other problems from computational biology incorporate some notion of sequence similarity as a basic premise.
### Sequence concepts

A *sequence* is a linear string of symbols over a finite alphabet.  
(Note: *string* and *sequence* are often used synonymously.)

Some basic sequence concepts:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>number of symbols in ( s ) (written (</td>
</tr>
<tr>
<td>empty string</td>
<td>sequence of length 0 (written ( \varepsilon )).</td>
</tr>
<tr>
<td>subsequence</td>
<td>sequence that can be obtained from ( s ) by removing some symbols (( \text{ACG} ) is a subsequence of ( \text{TATCTG} )).</td>
</tr>
<tr>
<td>supersequence</td>
<td>if ( t ) is a subsequence of ( s ), then ( s ) is a supersequence of ( t ).</td>
</tr>
</tbody>
</table>
More basic sequence concepts:

**substring** sequence of consecutive symbols appearing in $s$ (ACG is not a substring of TATCTG, but TCT is).

(Observation: every substring is a subsequence of the string in question, but not vice versa.)

**superstring** if $t$ is a substring of $s$, then $s$ is a superstring of $t$.

And lastly:

**prefix** substring of s of the form $s[1..j]$ where $0 \leq j \leq |s|$ (when $j = 0$, prefix is the empty string).

**suffix** substring of s of the form $s[i..|s|]$ where $1 \leq i \leq |s| + 1$ (when $i = |s| + 1$, suffix is the empty string).

**AT** is a prefix (and substring and subsequence) of **ATCCAG**.

**AG** is a suffix (and substring and subsequence) of **ATCCAG**.
Sequences

The concept of a sequence is extremely broad. In this course, we are concerned with genetic sequences. However, there are other important kinds of sequence data:

- ASCII text,
- speech,
- handwriting (pen-strokes).

Likewise, the same algorithmic techniques turn up again and again, often under different names:

- approximate string matching,
- edit (or evolutionary) distance,
- dynamic time warping.
Sequence comparison: a false start

An obvious idea that comes to mind is to line up each symbol and count the number that don't match:

![Sequence comparison diagram]

As we know, this is Hamming distance and forms the basis for most error correcting codes, as well as the motif-finding problem we saw earlier.

But it doesn't work for the kinds of sequences we care about:

![Sequence comparison diagram]

Just one missing symbol at the start of the second sequence leads to a large distance.
Sequence manipulation at the genetic level

Genomes aren't static ...

... sequence comparison must account for this.

http://www.accessexcellence.org/AB/GG/nhgri_PDFs/deletion.pdf
http://www.accessexcellence.org/AB/GG/nhgri_PDFs/insertion.pdf
In addition, errors can arise during the sequencing process:

"...the error rate is generally less than 1% over the first 650 bases and then rises significantly over the remaining sequence."

http://genome.med.harvard.edu/dnaseq.html

A hard-to-read gel (arrow marks location where bands of similar intensity appear in two different lanes):

Sequence alignment

As we have seen, the two sequences we wish to compare may have different lengths. As a result, we need to allow for deletions and insertions.

The notion of an *alignment* helps us visualize this:

```
<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>G</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>G</td>
<td>C</td>
</tr>
</tbody>
</table>
```

Alignment permits us to incorporate “spaces” (represented by a dash) in one or both sequences to make them the same length.
Sequence alignment

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>G</th>
<th>C</th>
<th>—</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>C</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>—</td>
<td>—</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>C</td>
</tr>
</tbody>
</table>

This alignment has 6 mismatches. Is it the best possible?

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
<th>G</th>
<th>C</th>
<th>G</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>—</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>—</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>C</td>
<td>G</td>
<td>T</td>
<td>—</td>
<td>C</td>
<td>—</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>C</td>
<td>T</td>
<td>G</td>
<td>C</td>
</tr>
</tbody>
</table>

No – this one has 5! How can we find the best alignment?
First successes:

- Finding sequence similarities with genes of known function is a common approach to infer function of a newly-sequenced gene.
- In 1984, Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene.

http://www.bioalgorithms.info
Cystic Fibrosis

- Cystic fibrosis (CF) is chronic and often fatal genetic disease of body's mucus glands (abnormally high level of mucus). Primarily affects respiratory systems in children.
- Mucus is slimy material that coats many epithelial surfaces and is secreted into fluids such as saliva.

Inheritance of Cystic Fibrosis:
- In early 1980's, biologists hypothesized that CF is an autosomal recessive disorder caused by mutations in a gene that remained unknown until 1989.
- Heterozygous carriers are asymptomatic.
- Must be homozygously recessive in this gene in order to be diagnosed with CF.

http://www.bioalgorithms.info
Cystic Fibrosis: finding the gene

http://www.bioalgorithms.info
Finding Similarities between Cystic Fibrosis Gene and ATP binding proteins:

- ATP binding proteins are present on cell membrane and act as transport channel.
- In 1989, biologists found similarity between cystic fibrosis gene and ATP binding proteins.
- A plausible function for cystic fibrosis gene, given that CF involves sweet secretion with abnormally high sodium level.

http://www.bioalgorithms.info
Cystic Fibrosis: mutation analysis

If a high percentage of cystic fibrosis (CF) patients have a certain mutation in the gene, while unaffected patients don’t, that could be an indicator of a mutation that is related to CF.

Indeed, a certain mutation was found in 70% of CF patients; this is convincing evidence of a predominant genetic diagnostics marker for CF.

http://www.bioalgorithms.info
Cystic Fibrosis and the CFTR protein

- CFTR (Cystic Fibrosis Transmembrane conductance Regulator) protein acts in cell membrane of epithelial cells that secrete mucus.
- These cells line airways of nose, lungs, stomach wall, etc.
- CFTR protein (1480 amino acids) regulates a chloride ion channel: adjusts “wateriness” of fluids secreted by cell.
- CF sufferers are missing a single amino acid in their CFTR.
- Mucus ends up being too thick, affecting many organs.

http://www.bioalgorithms.info
Bring in the Bioinformaticians

• Gene similarities between two genes with known and unknown function alert biologists to some possibilities.
• Computing a similarity score between two genes tells how likely it is that they have similar functions.
• Dynamic programming is a technique for revealing similarities between genes.
• The Change Problem is a good problem to introduce idea of dynamic programming.

http://www.bioalgorithms.info
The Change Problem

You hand cashier $20 for a $19.23 bill. Your change could be:

3 quarters + 2 pennies
77 pennies

etc.

The Change Problem.
Convert some amount of money $M$ into given denominations, using fewest possible number of coins.

**Input:** An amount of money $M$, and an array of $d$ denominations, $c = (c_1, c_2, \ldots, c_d)$, with $(c_1 > c_2 > \ldots > c_d)$.

**Output:** A list of $d$ integers, $i_1, i_2, \ldots, i_d$, such that

$$c_1i_1 + c_2i_2 + \ldots + c_d i_d = M$$

and $i_1 + i_2 + \ldots + i_d$ is minimal.
The Change Problem: an example

Given denominations 1, 3, and 5, what is minimum number of coins needed to make change for a given value?

<table>
<thead>
<tr>
<th>Value</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>Min # of coins</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Only one coin needed to make change for values 1, 3, and 5.

http://www.bioalgorithms.info
The Change Problem: an example

Given denominations 1, 3, and 5, what is minimum number of coins needed to make change for a given value?

<table>
<thead>
<tr>
<th>Value</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>Min # of coins</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

However, two coins needed to make change for values 2, 4, 6, 8, and 10.

http://www.bioalgorithms.info
The Change Problem: an example

Given denominations 1, 3, and 5, what is minimum number of coins needed to make change for a given value?

![Table showing minimum number of coins needed for values 1 to 10.]

<table>
<thead>
<tr>
<th>Value</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>Min # of coins</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Lastly, three coins needed to make change for values 7 and 9.

http://www.bioalgorithms.info
The Change Problem: recurrence

This example is expressed by the following recurrence relation:

\[ \text{minNumCoins}(M) = \min \text{ of} \]

\[
\begin{align*}
\text{minNumCoins}(M-1) + 1 \\
\text{minNumCoins}(M-3) + 1 \\
\text{minNumCoins}(M-5) + 1 \\
\end{align*}
\]

Given denominations \(c_1, c_2, \ldots, c_d\), the recurrence relation is:

\[ \text{minNumCoins}(M) = \min \text{ of} \]

\[
\begin{align*}
\text{minNumCoins}(M-c_1) + 1 \\
\text{minNumCoins}(M-c_2) + 1 \\
\vdots \\
\text{minNumCoins}(M-c_d) + 1 \\
\end{align*}
\]

http://www.bioalgorithms.info
RecursiveChange\((M, c, d)\)

\[
\text{if } M = 0 \\
\quad \text{return } 0 \\
\]

\[
\text{bestNumCoins } \leftarrow \text{infinity} \\
\]

\[
\text{for } i \leftarrow 1 \text{ to } d \\
\quad \text{if } M \geq c_i \\
\quad \quad \text{numCoins } \leftarrow \text{RecursiveChange}(M - c_i, c, d) \\
\quad \quad \text{if } \text{numCoins} + 1 < \text{bestNumCoins} \\
\quad \quad \quad \text{bestNumCoins } \leftarrow \text{numCoins} + 1 \\
\]

\[
\text{return } \text{bestNumCoins} \\
\]
The Change Problem: a recursive algorithm

The RecursiveChange algorithm works, but it is not efficient:

- It recalculates optimal coin combination for a given amount of money repeatedly.
- E.g., if $M = 77$ and $c = (1,3,7)$, then optimal combination of coins for 70 cents is computed on 9 different occasions!
The Change Problem: we can do better

Rather than re-compute a given value more than once:

- Save results of each computation for 0 to $M$.
- This way, we can do a reference call to find an already computed value, instead of re-computing each time.
- Running time is $M \times d$, where $M$ is the value of money and $d$ is the number of denominations.

http://www.bioalgorithms.info
DPChange($M, c, d$)

$bestNumCoins_0 \leftarrow 0$

for $m \leftarrow 1$ to $M$

$bestNumCoins_m \leftarrow \infty$

for $i \leftarrow 1$ to $d$

if $m \geq c_i$

if $bestNumCoins_{m-c_i} + 1 < bestNumCoins_m$

$bestNumCoins_m \leftarrow bestNumCoins_{m-c_i} + 1$

return $bestNumCoins_M$

http://www.bioalgorithms.info
DPChange example

c = (1, 3, 7)
M = 9
Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.
The Manhattan Tourist Problem (MTP)

Imagine seeking a path (from source to sink) to travel (only eastward and southward) with the most number of attractions (*) in the Manhattan grid.

http://www.bioalgorithms.info
The Manhattan Tourist Problem (MTP)

The Manhattan Tourist Problem.
Find the longest path in a weighted grid.

Input: A weighted grid $G$ with two distinct vertices, one labeled “source” and the other labeled “sink.”
Output: A longest path in $G$ from “source” to “sink.”

http://www.bioalgorithms.info
**MTP: an example**

```
+---+---+---+---+---+---+---+---+---+
| 0 | 3 | 5 | 9 | 0 | 4 | 3 | 1 | 4 |
+---+---+---+---+---+---+---+---+---+
| 1 | 0 | 2 | 4 | 4 | 2 | 2 | 1 | 1 |
+---+---+---+---+---+---+---+---+---+
| 0 | 1 | 7 | 3 | 5 | 2 | 3 | 2 | 3 |
+---+---+---+---+---+---+---+---+---+
| 2 | 5 | 2 | 15 | 4 | 2 | 19 | 4 | 4 |
+---+---+---+---+---+---+---+---+---+
| 3 | 6 | 3 | 0 | 3 | 0 | 20 | 3 | 3 |
+---+---+---+---+---+---+---+---+---+
| 4 | 7 | 8 | 5 | 5 | 2 | 23 | 2 | 2 |
+---+---+---+---+---+---+---+---+---+
```

The diagram represents a network flow problem. The nodes are labeled with their coordinates (i, j), and the edges are labeled with weights. The source is at (0, 0) and the sink is at (4, 4). The goal is to find the maximum flow from the source to the sink.
MTP: greedy is not optimal

source

sink

promising start, but leads to bad choices!

http://www.bioalgorithms.info
MTP: a simple recursive program

MT(n, m)
    if n = 0 and m = 0
        return 0
    if n > 0
        x ← MT(n-1, m) + length of edge from (n-1, m) to (n, m)
    else
        x ← 0
    if m > 0
        y ← MT(n, m-1) + length of edge from (n, m-1) to (n, m)
    else
        y ← 0
    return max(x, y)

What's wrong with this approach?

http://www.bioalgorithms.info
MTP: Dynamic Programming

- Calculate optimal path score for each vertex in graph.
- Each vertex’s score is maximum of prior vertices' scores plus weight of respective edges in between.

http://www.bioalgorithms.info
MTP: Dynamic Programming (continued)

http://www.bioalgorithms.info
MTP: Dynamic Programming (continued)

![Dynamic Programming Diagram](http://www.bioalgorithms.info)

- $S_{3,0} = 8$
- $S_{1,2} = 13$
- $S_{2,1} = 9$
Greedy algorithm fails!
MTP: Dynamic Programming (continued)

Done!

\[ S_{3,3} = 16 \]
Computing score for a point \((i, j)\) by recurrence relation:

\[
s_{i,j} = \max \begin{cases} 
    s_{i-1,j} + \text{weight of the edge between } (i-1, j) \text{ and } (i, j) \\
    s_{i,j-1} + \text{weight of the edge between } (i, j-1) \text{ and } (i, j)
\end{cases}
\]

The running time is \(n \times m\) for a \(n\)-by-\(m\) grid, where \(n\) is the number of rows and \(m\) is the number of columns.

http://www.bioalgorithms.info
But Manhattan is not a perfect grid ...

What about diagonals?

Score at point B is given by:

\[ s_B = \max \left\{ \begin{array}{l}
    s_{A1} + \text{weight of the edge } (A_1, B) \\
    s_{A2} + \text{weight of the edge } (A_2, B) \\
    s_{A3} + \text{weight of the edge } (A_3, B)
\end{array} \right\} \]

http://www.bioalgorithms.info
But Manhattan is not a perfect grid ...

Computing score $s_x$ for point $x$ is given by recurrence relation:

$$s_x = \max \left\{ s_y + \text{weight of edge } (y, x) \mid y \in \text{Predecessors}(x) \right\}$$

Predecessors($x$) = set of vertices that have edges leading to $x$

Running time for a graph $G(V, E)$ is $O(E)$ since each edge is evaluated once

Note: $V$ is set of all vertices and $E$ is set of all edges.

http://www.bioalgorithms.info
Alignment: two-row representation

Given 2 DNA sequences $v$ and $w$:

$v : \quad A \quad T \quad C \quad T \quad G \quad A \quad T \quad \quad m = 7$

$w : \quad T \quad G \quad C \quad A \quad T \quad A \quad \quad n = 6$

Alignment: 2 * $k$ matrix ( $k > m, n$ )

<table>
<thead>
<tr>
<th>letters of $v$</th>
<th>$A$</th>
<th>$T$</th>
<th>$-$</th>
<th>$C$</th>
<th>$-$</th>
<th>$T$</th>
<th>$G$</th>
<th>$A$</th>
<th>$T$</th>
</tr>
</thead>
<tbody>
<tr>
<td>letters of $w$</td>
<td>$-$</td>
<td>$T$</td>
<td>$G$</td>
<td>$C$</td>
<td>$A$</td>
<td>$T$</td>
<td>$-$</td>
<td>$A$</td>
<td>$-$</td>
</tr>
</tbody>
</table>

4 matches 2 insertions 3 deletions
Aligning DNA sequences

\[ v = ATCTGATG \quad n = 8 \]
\[ w = TGCATAC \quad m = 7 \]

4 matches
1 mismatches
3 deletions
2 insertions

http://www.bioalgorithms.info
The Longest Common Subsequence Problem (LCS)

The Longest Common Subsequence Problem.
Find the longest common subsequence of two sequences.

**Input:** Two sequences:

\[ v = v_1 v_2 \ldots v_m \quad \text{and} \quad w = w_1 w_2 \ldots w_n \]

**Output:** A sequence of positions in

\[ v : 1 \leq i_1 < i_2 < \ldots < i_t \leq m \]

and a sequence of positions in

\[ w : 1 \leq j_1 < j_2 < \ldots < j_t \leq n \]

such that symbol \( i_k \) of \( v \) matches \( j_k \) of \( w \) and \( t \) is maximal.
### LCS Example

**Elements of \( v \):**

- AT - C -
- T C T G A T C

**Elements of \( w \):**

- T G C A T - A - C

**Coordinates:**

<table>
<thead>
<tr>
<th>( i ) coords</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j ) coords</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

\[(0,0) \rightarrow (1,0) \rightarrow (2,1) \rightarrow (2,2) \rightarrow (3,3) \rightarrow (3,4) \rightarrow (4,5) \rightarrow (5,5) \rightarrow (6,6) \rightarrow (7,6) \rightarrow (8,7)\]

**Matches shown in red:**

Positions in \( v \): 2 < 3 < 4 < 6 < 8

Positions in \( w \): 1 < 3 < 5 < 6 < 7

Every common subsequence is a path in 2-D grid.

[Link to Bioalgorithms.info](http://www.bioalgorithms.info)
LCS Dynamic Programming

Find LCS of two strings.

**Input:** a weighted graph $G$ with two distinct vertices, one labeled “source” and one labeled “sink.”

**Output:** a longest path in $G$ from “source” to “sink.”

Solve using an MTP graph with diagonals replaced by +1 edges.

http://www.bioalgorithms.info
LCS Problem as Manhattan Tourist Problem

http://www.bioalgorithms.info
MTP Graph for LCS Problem

http://www.bioalgorithms.info
LCS Problem: find path with maximum diagonal edges.

http://www.bioalgorithms.info
Computing LCS

Let $v_i = \text{prefix of } v \text{ of length } i : \ v_1 \ldots v_i$
and $w_j = \text{prefix of } w \text{ of length } j : \ w_1 \ldots w_j$

The length of LCS($v_i$, $w_j$) is computed by:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} \\ s_{i,j-1} \\ s_{i-1,j-1} + 1 \text{ if } v_i = w_j \end{cases}$$
LCS Alignments

Every path in grid corresponds to an alignment:

\[
\begin{array}{cccccc}
V & W & A & T & C & G \\
0 & 0 & 1 & 2 & 3 & 4 \\
1 & & & & & \\
2 & & & & & \\
3 & & & & & \\
4 & & & & & \\
\end{array}
\]

\[
\begin{align*}
V &= AT \quad GT \\
W &= ATCG \quad - \\
\end{align*}
\]

http://www.bioalgorithms.info
Edit distance

In 1966, Levenshtein introduced *edit distance* between two strings as minimum number of elementary operations (insertions, deletions, and substitutions) needed to transform one string into the other.

\[ d(v, w) = \text{MIN number of elementary operations needed to transform } v \rightarrow w. \]

Keep in mind that LCS computation uses a MAX.
The Edit Distance Problem.

Given two sequences, find the optimal series of deletions, insertions, and substitutions to transform one into the other.

**Input:** Two sequences:

\[ v = v_1 v_2 \ldots v_m \quad \text{and} \quad w = w_1 w_2 \ldots w_n \]

**Output:** An optimal series of basic editing operations:

\[ e_1, e_2, \ldots, e_t \]

such then, when applied to one of the sequences, say \( v \), it is transformed into the other sequence, \( w \).

Here “optimal” can mean any of a number of things, including “fewest” or “lowest- / highest-cost.”
Edit distance vs. Hamming distance

Hamming distance always compares $i^{th}$ letter of $v$ with $i^{th}$ letter of $w$

\[ v = \text{ATATATAT} \]
\[ w = \text{TATATATA} \]

Hamming distance: \[ d(v, w) = 8 \]
Computing Hamming distance is trivial task.

Edit distance may compare $i^{th}$ letter of $v$ with $j^{th}$ letter of $w$

\[ v = \text{ATATATAT} \]
\[ w = \text{TATATATA} \]

Just one shift
Make it all line up

Edit distance: \[ d(v, w) = 2 \]
Computing edit distance is non-trivial task.

http://www.bioalgorithms.info
Edit distance vs. Hamming distance

Hamming distance always compares $i^{th}$ letter of $v$ with $i^{th}$ letter of $w$

$v = \text{ATATATAT}$
$w = \text{TATATATATA}$

Just one shift
Make it all line up

Hamming distance:
$d(v, w) = 8$

Edit distance may compare $i^{th}$ letter of $v$ with $j^{th}$ letter of $w$

$v = \text{-ATATATAT}$
$w = \text{TATATATA-}$

Edit distance:
$d(v, w) = 2$

One insertion and one deletion.

http://www.bioalgorithms.info
Edit distance example

TGCATAT \rightarrow ATCCGAT in 5 steps:

TGCATAT \rightarrow (delete last T)
TGCATA \rightarrow (delete last A)
TGCAT \rightarrow (insert A at front)
ATGCAT \rightarrow (substitute C for 3\textsuperscript{rd} G)
ATCCAT \rightarrow (insert G before last A)
ATCCGAT \rightarrow (Done)

http://www.bioalgorithms.info
Edit distance example

TGCATAT → ATCCGAT in 5 steps:

1. TGCATAT → (delete last T)
2. TGCATA → (delete last A)
3. TGCAT → (insert A at front)
4. ATGCAT → (substitute C for 3rd G)
5. ATCCAT → (insert G before last A)
6. ATCCGAT → (Done)

What is the edit distance? 5?

http://www.bioalgorithms.info
Edit distance example

TGCATAT → ATCCGAT in 4 steps:

TGCATAT → (insert A at front)
ATGCATAT → (delete 8th T)
ATGCAATA → (substitute G for 5th A)
ATGCGTA → (substitute C for 3rd G)
ATCCGAT  → (Done)

http://www.bioalgorithms.info
TGCATAT → ATCCGAT in 4 steps:

TGCATAT → (insert A at front)
ATGCATAT → (delete 8th T)
ATGCATA → (substitute G for 5th A)
ATGCCTA → (substitute C for 3rd G)
ATCCGAT  (Done)
Can it be done in 3 steps???
Alignment grid

Every alignment path is from source to sink.
Alignment as a path in edit graph

Corresponding path:

(0,0), (1,1), (2,2), (2,3), (3,4), (4,5), (5,5), (6,6), (7,6), (7,7)
Alignment as a path in edit graph

\(v\) and \(w\) represent indels in \(v\) and \(w\) with score 0.

\(\downarrow\) represent matches with score 1.

The score of the alignment path is 5.
Alignment as a path in edit graph

Every path in edit graph corresponds to an alignment:

http://www.bioalgorithms.info
Alignment as a path in edit graph

Old Alignment

\begin{align*}
v &= \text{AT\_GTTAT}\_ \\
w &= \text{ATCGT\_A\_C} \\
\text{0122345677} & \quad \text{0123455667}
\end{align*}

New Alignment

\begin{align*}
v &= \text{AT\_GTTAT}\_ \\
w &= \text{ATCG\_TA\_C} \\
\text{0122345677} & \quad \text{0123445667}
\end{align*}

http://www.bioalgorithms.info
Alignment as a path in edit graph

\[ v = \text{AT}_\text{G}_\text{T}_\text{TAT}_\text{A} \]
\[ w = \text{ATCGT}_\text{A}_\text{A}_\text{C} \]

(0,0), (1,1), (2,2), (2,3), (3,4), (4,5), (5,5), (6,6), (7,6), (7,7)

http://www.bioalgorithms.info
We can't afford to enumerate all possible alignments looking for the best one – that would be an exponential search.

Fortunately we don't have to. The optimal alignment can be found using dynamic programming, which we've just seen.

Dynamic programming is based on premise of computing solutions to smaller subproblems first and then using these to solve successively larger problems until we have our answer.

(Dynamic programming was invented by Richard Bellman in the 1950's. Its application to sequence comparison came later, in the 1970's.)

http://fens.sabanciuniv.edu/msie/operations_research_50_years/anniversary/or50/1526-5463-2002-50-01-0048.pdf
First some ground-rules.

- **Legal:**
  - **deletion:** 
  - **insertion:** 
  - **match:** 
  - **mismatch:**

- **Not legal:**

http://www.bioalgorithms.info
## Sequence comparison: the basic algorithm

Given two sequences $v$ and $w$, consider what's required to compute optimal alignment for prefixes $v[1..i]$ and $w[1..j]$. Based on our rules for alignments, there are three possible cases:

<table>
<thead>
<tr>
<th>Case</th>
<th>Alignment for $v[1..i-1]$ and $w[1..j-1]$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>$v[i]$</td>
<td>deletion</td>
</tr>
<tr>
<td>II</td>
<td>$w[j]$</td>
<td>insertion</td>
</tr>
<tr>
<td>III</td>
<td>$v[i]$ or $w[j]$</td>
<td>substitution or match</td>
</tr>
</tbody>
</table>

So, assuming we've already computed solutions for all shorter prefixes, we can compute the alignment for $v[1..i]$ and $w[1..j]$. 

---

---
Sequence comparison: the basic algorithm

Conceptually, this might look something like this:

\[
\text{optimal alignment at } v[1..i] \text{ and } w[1..j] = \max \]

Here we assume that deletions, insertions, and mismatches have negative costs, while matches have positive costs.
This computation can be viewed as building a 2-D matrix:

<table>
<thead>
<tr>
<th></th>
<th>string w</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>cost of inserting w</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>string v</th>
<th>cost of deleting v</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td></td>
</tr>
</tbody>
</table>

\[ d[i,j] = \max \begin{cases} 
  d[i-1,j] + \text{indel} \\
  d[i,j-1] + \text{indel} \\
  d[i-1,j-1] + \text{sub}(v[i], w[j]) 
\end{cases} \]

where \text{indel} is cost of a deletion or insertion, and \text{sub} is cost of a match/mismatch involving symbols \(v[i]\) and \(w[j]\).
Sequence comparison: the basic algorithm

Stated more generally, say that our two sequences are:


Then:

\[ d[0,0] = 0 \]
\[ d[i,0] = d[i-1,0] + c_{del}(v[i]) \quad 1 \leq i \leq m \]
\[ d[0,j] = d[0,j-1] + c_{ins}(w[j]) \quad 1 \leq j \leq n \]

And:

\[ d[i,j] = \max \left\{ \begin{array}{l} d[i-1,j] + c_{del}(v[i]) \\
                          d[i,j-1] + c_{ins}(w[j]) \\
                          d[i-1,j-1] + c_{sub}(v[i], w[j]) \end{array} \right. \quad 1 \leq i \leq m, \ 1 \leq j \leq n \]

Where \( c_{del}, c_{ins}, \) and \( c_{sub} \) are the costs of a deletion, an insertion, and a substitution, respectively.
Sequence comparison: the basic algorithm

Example: say that $c_{\text{del}} = -1$, $c_{\text{ins}} = -1$, $c_{\text{sub}} = -1$ if mismatch and +1 if match

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>G</td>
<td>-3</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>T</td>
<td>-4</td>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>
Algorithm for computing edit distance

\[ \text{EditDistance1}(v, w) \]
\[
\text{for } i \leftarrow 1 \text{ to } n \\
\quad d_{i,0} \leftarrow d_{i-1,0} + c_{\text{del}}(v_i) \\
\text{for } j \leftarrow 1 \text{ to } m \\
\quad d_{0,j} \leftarrow d_{0,j-1} + c_{\text{ins}}(w_j) \\
\text{for } i \leftarrow 1 \text{ to } n \\
\quad \text{for } j \leftarrow 1 \text{ to } m \\
\quad \quad d_{i,j} \leftarrow \max \left[ d_{i-1,j} + c_{\text{del}}(v_i), d_{i,j-1} + c_{\text{ins}}(w_j), d_{i-1,j-1} + c_{\text{sub}}(v_i, w_j) \right] \\
\text{return } (d_{n,m}) \]

http://www.bioalgorithms.info
Sequence comparison: some observations

Computation can progress in a number of ways:


For sequences of length \( m \) and \( n \),

What is the computation time? \( O(mn) \)

How much memory (storage) is required? \( O(mn) \)
**Sequence comparison: getting an alignment**

We started with the notion of alignment. How do we get this? By keeping track of optimal decisions made during algorithm, ... and then tracing back optimal path.

(May not be unique).
Maintaining the traceback

\text{EditDistance1}(v, w)

\ldots

\textbf{for } i \leftarrow 1 \textbf{ to } n

\textbf{for } j \leftarrow 1 \textbf{ to } m

\quad d_{i,j} \leftarrow \max\left[ \begin{array}{l}
d_{i-1,j} + c_{\text{del}}(v_i) \\
d_{i,j-1} + c_{\text{ins}}(w_j) \\
d_{i-1,j-1} + c_{\text{sub}}(v_i, w_j)
\end{array} \right]

\text{if } d_{i,j} = d_{i-1,j} + c_{\text{del}}(v_i)

\quad b_{i,j} \leftarrow \text{true}

\text{if } d_{i,j} = d_{i,j-1} + c_{\text{ins}}(w_j)

\quad b_{i,j} \leftarrow \text{false}

\text{if } d_{i,j} = d_{i-1,j-1} + c_{\text{sub}}(v_i, w_j)

\quad b_{i,j} \leftarrow \text{false}

\text{return } (d_{n,m}, b)

\text{http://www.bioalgorithms.info}
More examples

Comparing **ACGT** vs. **CAT**:

![Diagram showing comparisons between ACGT and CAT sequences](image)
More examples

Comparing **ACGTAT** vs. **AGTTTG**:
More examples

Comparing **ACGTGCGCTGCTG** vs. **CGTCCTGCCTGC**:

Recall the trace we saw earlier:
Optimality of the algorithm

How do we know this algorithm finds an optimal alignment?

We won't dig into finer details now, but basically it hinges on:

- Bellman's *principle of optimality* for dynamic programming.
- The cost functions behaving properly (i.e., they must satisfy the *triangle inequality*).

This could be an interesting topic for a final paper ...
Wrap-up

Readings for next time:
• Continue reading IBA Chapter 6 (sequence comparison).

Remember:
• Come to class having done the readings.
• Check Blackboard regularly for updates.