Dynamic Programming: Sequence alignment

CS 466
Saurabh Sinha
DNA Sequence Comparison: First Success Story

• Finding sequence similarities with genes of known function is a common approach to infer a newly sequenced gene’s function

• In 1984 Russell Doolittle and colleagues found similarities between cancer-causing gene and normal growth factor (PDGF) gene

• A normal growth gene switched on at the wrong time causes cancer!
Cystic Fibrosis

- **Cystic fibrosis** (CF) is a chronic and frequently fatal genetic disease of the body's mucus glands. CF primarily affects the respiratory systems in children.

- Search for the CF gene was narrowed to ~1 Mbp, and the region was sequenced.

- Scanned a database for matches to known genes. A segment in this region matched the gene for some ATP binding protein(s). These proteins are part of the ion transport channel, and CF involves sweat secretions with abnormal sodium content!
Role for Bioinformatics

• 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
Motivating Dynamic Programming
Dynamic programming example: Manhattan Tourist Problem

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

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

**Goal**: 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”
MTP: Greedy Algorithm Is Not Optimal

promising start, but leads to bad choices!
MTP: Simple Recursive Program

\[
\text{MT}(n,m) \\
\quad \text{if } n=0 \text{ or } m=0 \\
\quad \quad \text{return } MT(n,m) \\
\quad x \leftarrow MT(n-1,m)+ \\
\quad \quad \text{length of the edge from } (n-1,m) \text{ to } (n,m) \\
\quad y \leftarrow MT(n,m-1)+ \\
\quad \quad \text{length of the edge from } (n,m-1) \text{ to } (n,m) \\
\quad \text{return } \max\{x,y\}
\]

What’s wrong with this approach?
Here’s what’s wrong

• $M(n,m)$ needs $M(n, m-1)$ and $M(n-1, m)$
• Both of these need $M(n-1, m-1)$
• So $M(n-1, m-1)$ will be computed at least twice
• Dynamic programming: the same idea as this recursive algorithm, but keep all intermediate results in a table and reuse
MTP: Dynamic Programming

- Calculate optimal path score for each vertex in the graph.
- Each vertex’s score is the maximum of the prior vertices’ score plus the weight of the respective edge in between.
MTP: Dynamic Programming
(cont’d)

![Diagram of a dynamic programming problem with labels and values for specific states: S_{3,0} = 8, S_{1,2} = 13, S_{2,1} = 9, S_{3,0} = 8.](image)
MTP: Dynamic Programming
(cont’d)

source

i

j

S_{1,3} = 8
S_{2,2} = 12
S_{3,1} = 9

greedy alg. fails!
MTP: Dynamic Programming
(cont’d)

(source)

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

S_{3,3} = 16

Done!

(showing all back-traces)
MTP: Recurrence

Computing the score for a point \((i,j)\) by the recurrence relation:

\[
\begin{align*}
    s_{i,j} &= \max \left\{ s_{i-1,j} + \text{weight of the edge between } (i-1,j) \text{ and } (i,j), \\
    &\quad s_{i,j-1} + \text{weight of the edge between } (i,j-1) \text{ and } (i,j) \right\}
\end{align*}
\]

The running time is \(n \times m\) for a \(n\) by \(m\) grid

\((n = \# \text{ of rows}, \ m = \# \text{ of columns})\)
Manhattan Is Not A Perfect Grid

What about diagonals?

• The score at point B is given by:

\[ s_B = \max \left\{ s_{A1} + \text{weight of the edge } (A_1, B), s_{A2} + \text{weight of the edge } (A_2, B), s_{A3} + \text{weight of the edge } (A_3, B) \right\} \]
Manhattan Is Not A Perfect Grid (cont’d)

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

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

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

- The running time for a graph $G(V, E)$ ($V$ is the set of all vertices and $E$ is the set of all edges) is $O(E)$ since each edge is evaluated once
Traveling in the Grid

• By the time the vertex $x$ is analyzed, the values $s_y$ for all its predecessors $y$ should be computed – otherwise we are in trouble.

• We need to traverse the vertices in some order

• For a grid, can traverse vertices row by row, column by column, or diagonal by diagonal
Traversing the Manhattan Grid

• 3 different strategies:
  – a) Column by column
  – b) Row by row
  – c) Along diagonals
Traversing a DAG

• A numbering of vertices of the graph is called **topological ordering** of the DAG if every edge of the DAG connects a vertex with a smaller label to a vertex with a larger label

• How to obtain a topological ordering?
Alignment
Aligning DNA Sequences

Alignment: $2 \times k$ matrix ($k \geq m, n$)

$V = \text{ATCTGATG}$ \quad $n = 8$

$W = \text{TGCATAC}$ \quad $m = 7$

4 matches
1 mismatches
2 insertions
2 deletions

Alignment:

```
V  = ATCTGATG  n = 8
W  = TGCATAC  m = 7
```

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

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

Indels:

- Deletions:
  - At positions 1, 4, 6 of V
  - At positions 1, 2, 4, 6 of W

- Insertions:
  - At positions 2, 3 of V
  - At positions 2, 3 of W
Longest Common Subsequence (LCS) – Alignment without Mismatches

• Given two sequences
  \[\mathbf{v} = v_1 v_2 \ldots v_m \text{ and } \mathbf{w} = w_1 w_2 \ldots w_n\]

• The LCS of \(\mathbf{v}\) and \(\mathbf{w}\) is a sequence of positions in
  \[\mathbf{v}: 1 \leq i_1 < i_2 < \ldots < i_t \leq m\]
  and a sequence of positions in
  \[\mathbf{w}: 1 \leq j_1 < j_2 < \ldots < j_t \leq n\]

  such that \(i_t\)-th letter of \(\mathbf{v}\) equals to \(j_t\)-letter of \(\mathbf{w}\) and \(t\) is maximal
LCS: Example

<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>elements of v</td>
<td>A</td>
<td>T</td>
<td>--</td>
<td>C</td>
<td>--</td>
<td>T</td>
<td>G</td>
<td>A</td>
<td>T</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>elements 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>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

| j coords: | 0 | 0 | 1 | 2 | 3 | 4 | 5 | 5 | 6 | 6 | 7 |

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

$$s_{i,j} = \max \left\{ s_{i-1,j}, s_{i,j-1}, s_{i-1,j-1} + 1 \right\} \text{ if } v_i = w_j$$
LCS Problem as Manhattan Tourist Problem
Edit Graph for LCS Problem
Every path is a common subsequence.

Every diagonal edge adds an extra element to common subsequence

**LCS Problem:** Find a path with maximum number of diagonal edges
Backtracking

- $s_{i,j}$ allows us to compute the length of LCS for $v_i$ and $w_j$
- $s_{m,n}$ gives us the length of the LCS for $v$ and $w$
- How do we print the actual LCS?
- At each step, we chose an optimal decision $s_{i,j} = \max (...)$
- Record which of the choices was made in order to obtain this max
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 $\text{LCS}(v_i, w_j)$ is computed by:

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

1. PrintLCS(b,v,i,j)
2. if  $i = 0$ or $j = 0$
3.   return
4.   if $b_{i,j} = \text{""}$
5.      PrintLCS(b,v,i-1,j-1)
6.      print $v_i$
7.   else
8.      if $b_{i,j} = \text{ "\downarrow\"}$
9.         PrintLCS(b,v,i-1,j)
10.     else
11.    PrintLCS(b,v,i,j-1)
From LCS to Alignment

- The Longest Common Subsequence problem—the simplest form of sequence alignment—allows only insertions and deletions (no mismatches).
- In the LCS Problem, we scored 1 for matches and 0 for indels.
- Consider penalizing indels and mismatches with negative scores.
- Simplest **scoring scheme**:
  - $+1$ : match premium
  - $-\mu$ : mismatch penalty
  - $-\sigma$ : indel penalty
Simple Scoring

• When mismatches are penalized by $-\mu$, indels are penalized by $-\sigma$, and matches are rewarded with $+1$, the resulting score is:

$$\#\text{matches} - \mu(\#\text{mismatches}) - \sigma (\#\text{indels})$$
The Global Alignment Problem

Find the best alignment between two strings under a given scoring schema

**Input**: Strings \( v \) and \( w \) and a scoring schema

**Output**: Alignment of maximum score

\[
\begin{align*}
\downarrow \rightarrow &= -\sigma \\
\ &= 1 \text{ if match} \\
\ &= -\mu \text{ if mismatch}
\end{align*}
\]

\[
\begin{align*}
\alpha &= \begin{cases} 
    s_{i-1,j-1} + 1 & \text{if } v_i = w_j \\
    s_{i-1,j-1} - \mu & \text{if } v_i \neq w_j
    \end{cases}
\end{align*}
\]

\[
\begin{align*}
\gamma &= \begin{cases} 
    s_{i,j} - \sigma & \\
    s_{i-1,j} - \sigma & \\
    s_{i,j-1} - \sigma
    \end{cases}
\end{align*}
\]

\( \mu \): mismatch penalty

\( \sigma \): indel penalty
Scoring Matrices

To generalize scoring, consider a \((4+1) \times (4+1)\) scoring matrix \(\delta\).

In the case of an amino acid sequence alignment, the scoring matrix would be a \((20+1) \times (20+1)\) size. The addition of 1 is to include the score for comparison of a gap character “-”.

This will simplify the algorithm as follows:

\[
s_{i,j} = \max \left\{ \begin{array}{l} s_{i-1,j-1} + \delta(v_i, w_j) \\ s_{i-1,j} + \delta(v_i, -) \\ s_{i,j-1} + \delta(-, w_j) \end{array} \right. \]

Making a Scoring Matrix

• Scoring matrices are created based on biological evidence.
• Alignments can be thought of as two sequences that differ due to mutations.
• Some of these mutations have little effect on the protein’s function, therefore some penalties, $\delta(v_i, w_j)$, will be less harsh than others.
### Scoring Matrix: Example

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>R</th>
<th>N</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>R</td>
<td>-</td>
<td>7</td>
<td>-1</td>
<td>3</td>
</tr>
<tr>
<td>N</td>
<td>-</td>
<td>-</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>K</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
</tr>
</tbody>
</table>

- Notice that although R and K are different amino acids, they have a positive score.
- Why? They are both positively charged amino acids → will not greatly change function of protein.

```
AKRANR
KAAANK
```

\[-1 + (-1) + (-2) + 5 + 7 + 3 = 11\]
Conservation

• Amino acid changes that tend to preserve the physico-chemical properties of the original residue
  – Polar to polar
    • aspartate $\rightarrow$ glutamate
  – Nonpolar to nonpolar
    • alanine $\rightarrow$ valine
  – Similarly behaving residues
    • leucine to isoleucine
Local vs. Global Alignment

• The **Global Alignment Problem** tries to find the longest path between vertices \((0,0)\) and \((n,m)\) in the edit graph.

• The **Local Alignment Problem** tries to find the longest path among paths between *arbitrary vertices* \((i,j)\) and \((i', j')\) in the edit graph.

• In the edit graph with negatively-scored edges, Local Alignment may score higher than Global Alignment.
Local Alignment: Example

Compute a “mini” Global Alignment to get Local Alignment
Local Alignments: Why?

• Two genes in different species may be similar over short conserved regions and dissimilar over remaining regions.

• Example:
  – Homeobox genes have a short region called the *homeodomain* that is highly conserved between species.
  – A global alignment would not find the homeodomain because it would try to align the ENTIRE sequence
The Local Alignment Problem

• **Goal**: Find the best local alignment between two strings
• **Input**: Strings $v$, $w$ and scoring matrix $\delta$
• **Output**: Alignment of substrings of $v$ and $w$ whose alignment score is maximum among all possible alignment of all possible substrings
The Problem Is …

• Long run time $O(n^4)$:
  
  - In the grid of size $n \times n$ there are $\sim n^2$ vertices $(i,j)$ that may serve as a source.

  - For each such vertex computing alignments from $(i,j)$ to $(i',j')$ takes $O(n^2)$ time.

• This can be remedied by allowing every point to be the starting point
The Local Alignment Recurrence

• The largest value of $s_{i,j}$ over the whole edit graph is the score of the best local alignment.

• The recurrence:

$$s_{i,j} = \max \begin{cases} 
0 & \\
& s_{i-1,j-1} + \delta(v_i, w_j) \\
& s_{i-1,j} + \delta(v_i, -) \\
& s_{i,j-1} + \delta(-, w_j) 
\end{cases}$$

Notice there is only this change from the original recurrence of a Global Alignment.