COMP251: Dynamic programming (3)

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Kleinberg & Tardos, 2005)
PAIRWISE SEQUENCE ALIGNMENT
Pairwise Sequence Alignment

**Match:** letters are identical  
**Substitution:** letters are different

**Insertion:** a letter of $b$ is mapped to the empty character  
**Deletion:** a letter of $a$ is mapped to the empty character

Each operation has a cost $\implies$ find alignment with optimal score.

Example:  
$a = ABBCEE$, $b = BBCCDE$

```
  A B B - C E E
  | | | : |
- B B C C D E
```

- **deletion**
- **insertion**
- **substitution**
- **match**
Needleman-Wunch Algorithm

for $i=0$ to $m$ do
    $d(i,0)=i\cdot\delta(-,-)$
for $j=0$ to $n$ do
    $d(0,j)=j\cdot\delta(-,-)$

for $i=1$ to $m$ do
    for $j=1$ to $n$ do
        $d(i,j) = \min(d(i-1,j)+\delta(a_i,-),$
        $d(i-1,j-1)+\delta(a_i,b_j),$ $d(i,j-1)+\delta(-,b_j))$

return $d(m,n)$
Example

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[
\delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\]

<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>d</td>
<td></td>
<td>-</td>
<td>A</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ d[i,j] = \text{optimal alignment score of } a_1 \ldots a_i \text{ with } b_1 \ldots b_j \]
Example

\( a = \text{ATTG} \quad b = \text{CT} \)

\( \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise} 
\end{cases} \)

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

- match/substitution: \( d(0,0) + \delta(A, C) = 0 + (+1) = +1 \)
Example

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise} 
\end{cases} \]

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

- match/substitution: \( d(0,0) + \delta(A, C) = 0 + (+1) = +1 \)
- insertion: \( d(1,0) + \delta(-, C) = 1 + (+1) = +2 \)
Example

\( a = \text{ATTG} \ b = \text{CT} \)

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases} \]

\[
\begin{array}{c|c|c|c|c|c}
 & - & A & T & T & G \\
\hline
- & 0 & 1 & 2 & 3 & 4 \\
\hline
C & 1 & ? & & & \\
\hline
T & 2 & & & & \\
\end{array}
\]

- match/substitution: \( d(0,0) + \delta(A, C) = 0 + (+1) = +1 \)
- insertion: \( d(1,0) + \delta(\_, C) = +1 + (+1) = +2 \)
- deletion: \( d(0,1) + \delta(A, \_) = +1 + (+1) = +2 \)
Example

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases} \]

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

- match/substitution: \( d(0,0) + \delta(A,C) = 0 + (+1) = +1 \)
- insertion: \( d(1,0) + \delta(-,C) = +1 + (+1) = +2 \)
- deletion: \( d(0,1) + \delta(A,-) = +1 + (+1) = +2 \)
Example

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases} \]

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

\[a = \text{ATTG} \quad b = \text{CT}\]

\[
\delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\]

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

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise} 
\end{cases} \]

<table>
<thead>
<tr>
<th></th>
<th>-</th>
<th>A</th>
<th>T</th>
<th>T</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>T</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Example**

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise} 
\end{cases} \]

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

\( a = \text{ATTG} \quad b = \text{CT} \)

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases} \]

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

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[
\delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\]

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

\[ a=\text{ATTG} \quad b=\text{CT} \]

\[
\delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\]

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

How to retrieve the optimal alignment?
• Each move is associated to one edit operation
  • Vertical = insertion
  • Diagonal = match/substitution
  • Horizontal = deletion
• We use one of these 3 move to fill a cell of the array
• From the bottom-right corner (i.e. $d(m,n)$), find the move that has been used to determine the value of this cell.
• Apply this principle recursively.
### Example

\( a = \text{ATTG} \quad b = \text{CT} \)

\[ \delta(x, y) = \begin{cases} 
  0 & \text{if } x = y \\
  1 & \text{otherwise}
\end{cases} \]

<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>d</td>
<td>-</td>
<td>A</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Example

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\ 
1 & \text{otherwise} 
\end{cases} \]

\[
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & 4 \\
\hline
0 & - & 0 & 1 & 2 & 3 \\
1 & C & 1 & 1 & 2 & 3 \Rightarrow 4 \\
2 & T & 2 & 2 & 1 & 2 \\
\end{array}
\]

\[
\begin{pmatrix}
\text{ATT} \\
\text{C}
\end{pmatrix}
\begin{pmatrix}
\text{G} \\
\text{T}
\end{pmatrix}
\]

\[ d[3,1] + \delta(G,T) = 3 + 1 = 4 \quad \times \]
Example

a=ATTG  b=CT

\[
\delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\]

<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>d</td>
<td>-</td>
<td>A</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>1</td>
<td>C</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

\[
\left(\begin{array}{c}
\text{ATT} \\
\text{C}
\end{array}\right)
\left(\begin{array}{c}
\text{G} \\
\text{T}
\end{array}\right)
= \left(\begin{array}{c}
\text{ATTG} \\
\text{CGT}
\end{array}\right)
\]

\[d[4,1] + \delta(-,T) = 4 + 1 = 5 \quad \times\]
Example

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise} 
\end{cases} \]

<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>d</td>
<td></td>
<td>A</td>
<td>T</td>
<td>G</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
(\text{ATT}_C) & (\text{G}_T) = \text{(ATTG)} (\text{CT}) \\
\delta(G, -) & = 2 + 1 = 3 \\
d[3,2] + \delta(G,-) & = 2 + 1 = 3
\end{align*}
\]
Example

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases} \]

\[
a=\text{ATTG} \quad b=\text{CT}
\]

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

\[G\]

\[-\]
Example

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise} 
\end{cases} \]

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

TG
T-
Example

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[ \delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases} \]

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

\[ \text{TTG} \quad \text{--T--} \]
Example

\[ a = \text{ATTG} \quad b = \text{CT} \]

\[
\delta(x, y) = \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\]

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

\[
\text{ATTG} \\
\text{C-T-}
\]
Example

\[
\begin{align*}
a = \text{ATTG} & \quad b = \text{CT} \\
\delta(x, y) &= \begin{cases} 
0 & \text{if } x = y \\
1 & \text{otherwise}
\end{cases}
\end{align*}
\]

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

\[
\begin{align*}
\text{ATTG} & \quad \text{ATTG} & \quad \text{ATTG} \\
\text{C–T–} & \quad \text{CT–} & \quad \text{–CT–}
\end{align*}
\]
Analysis

**Theorem:** The dynamic programming algorithm computes the edit distance (and optimal alignment) of two strings of length \( m \) and \( n \) in \( \Theta(mn) \) time and \( \Theta(mn) \) space.

**Proof:**
- Algorithm computes edits distance.
- Can trace back to extract an optimal alignment.

**Q.** Can we avoid using quadratic space?
- **A.** Easy to compute optimal value in \( \Theta(mn) \) time and \( \Theta(m+n) \) space.
  - Compute \( \text{OPT}(i,\bullet) \) from \( \text{OPT}(i-1,\bullet) \).
  - But, no longer easy to recover optimal alignment itself.
• Different cost functions, For instance:

\[ \delta(x, y) = \begin{cases} 
1 & \text{if } x = y \\
-1 & \text{otherwise} 
\end{cases} \]

Cost of alignment is being maximized.

• Variants of optimal pairwise alignment algorithm:
  - Ignore trailing gaps (Smith & Waterman, 1981)

• Optimal alignment not practical for multiple sequences.
SINGLE SOURCE SHORTEST PATHS
Modeling as graphs

Input:
• Directed graph $G = (V, E)$
• Weight function $w : E \rightarrow \mathbb{R}$

**Weight of path** $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$= \sum_{k=1}^{n} w(v_{k-1}, v_k)$$

= sum of edge weights on path $p$.

**Shortest-path weight** $u$ to $v$:

$$\delta(u, v) = \begin{cases} 
\min \left\{ w(p) : u \xrightarrow{p} v \right\} & \text{if there exists a path } u \xrightarrow{p} v. \\
\infty & \text{otherwise.}
\end{cases}$$

Shortest path $u$ to $v$ is any path $p$ such that $w(p) = \delta(u, v)$.

Generalization of breadth-first search to weighted graphs.
Dijkstra’s algorithm

• No negative-weight edges.
• Weighted version of BFS:
  • Instead of a FIFO queue, uses a priority queue.
  • Keys are shortest-path weights ($d[v]$).
• Greedy choice: At each step we choose the light edge.

How to deal with negative weight edges?
• Allow re-insertion in queue? $\Rightarrow$ Exponential running time...
• Add constant to each edge?

![Diagram of Dijkstra's algorithm with negative weight edges](attachment:image.png)
Bellman-Ford Algorithm

- Allows negative-weight edges.
- Computes $d[v]$ and $\pi[v]$ for all $v \in V$.
- Returns TRUE if no negative-weight cycles reachable from $s$, FALSE otherwise.

If Bellman-Ford has not converged after $V(G) - 1$ iterations, then there cannot be a shortest path tree, so there must be a negative weight cycle.
Bellman-Ford Algorithm

• Can have negative-weight edges.
• Will “detect” **reachable** negative-weight cycles.

```
Initialize(G, s);
for i := 1 to |V[G]| – 1 do
  for each (u, v) in E[G] do
    Relax(u, v, w)
  for each (u, v) in E[G] do
    if d[v] > d[u] + w(u, v) then
      return false
return true
```

Time Complexity is O(VE).
Example

Graph with nodes S (0) and X (¥), and edges with weights 5, -2, and -4, and a connection to Y with weight 7.
Example

Iteration 1

\[
\begin{align*}
    s & \rightarrow 0 : 5 \\
    0 & \rightarrow \infty : -2 \\
    \infty & \rightarrow y : 7
\end{align*}
\]
Example

Iteration 1

\begin{align*}
S & \rightarrow 0 \quad \text{with cost 5} \\
0 & \rightarrow X \quad \text{with cost -2} \\
X & \rightarrow \text{-4} \\
\text{-4} & \rightarrow Y \quad \text{with cost 7}
\end{align*}
Example

Iteration 1

Diagram:
- Node S connected to node 0 with a weight of 5.
- Node S connected to node -4 with a weight of -4.
- Node 0 connected to node X with a weight of -2.
- Node 5 connected to node Y with a weight of 7.
Example

Iteration 1

Graph:
- Node S connected to node 0 with weight 5.
- Node S connected to node -4 with weight -4.
- Node 0 connected to node 3 with weight -2.
- Node 3 connected to node Y with weight 7.

Node labels:
- S
- 0
- 3
- -4
- Y
Example

Iteration 2

- Graph with nodes S, 0, X, -4, and Y.
- Edges with weights 5, -2, and -4.
- Node X connected to node 0 with weight 5.
- Node -4 connected to node X with weight 7.
Example

Iteration 2

- Iteration 2
- Graph with nodes S, 0, 3, -4, X, and Y
- Edges with labels: S to 0 (5), 0 to 3 (-2), 3 to -4 (-4), and 3 to Y (7)
- Red arrow from S to 0
Example

Iteration 2

Graph:
- Node S at (0, 0) with an edge to node X labeled -2.
- Node X at (3, 0) with an edge to node -4 labeled 7.
- Node -4 at (0, -4) with an edge to node 0 labeled -4.
- Edge from S to X labeled 5.

Table:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Edge Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>S</td>
<td>X</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td>S</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>-4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>X</td>
<td>5</td>
</tr>
</tbody>
</table>
Example

Iteration 2

Diagram:
- Node S connected to node 0 with weight 5.
- Node 0 connected to node X with weight -2.
- Node X connected to node -4 with weight 7.
- Node 0 connected to node -4 with weight -4.
Example 2

Diagram:

- Two nodes labeled S and X, connected by edges labeled 5 and -4.
- Node X connected to a node labeled Y with an edge labeled 7.
- Edges are directed from S to X and S to Y.
Example 2

Iteration 1

\[ s \rightarrow 0 : 5 \]
\[ 0 \rightarrow \infty : -4 \]
\[ \infty \rightarrow x : -4 \]
\[ x \rightarrow 5 : 7 \]
Example 2

Iteration 1

Diagram:
- Two nodes: S (source) and X (sink).
- Edge from S to X with weight -4.
- Edge from X to Y with weight 7.
- Edge from S to Y with weight 5.

Values:
- S: 0
- X: 5
- Y: -4
Example 2

Iteration 1

Graph:
- Node S connected to node 0 with weight 5.
- Node 0 connected to node 5 with weight -4.
- Node 5 connected to node -4 with weight 7.
- Node -4 connected to node y with weight -4.

Edges:
- S to 0: weight 5
- 0 to 5: weight -4
- 5 to -4: weight 7
- -4 to y: weight -4
Example 2

Iteration 1

Graph:

- Nodes: S, 0, 3, -4, Y
- Edges: S → 0 (5), 0 → 3 (-4), 3 → Y (7)

Initial State: S

Next State: 0 (Iteration 1)
Example 2

Iteration 2

Graph with nodes 0, 3, and 4, with edges labeled 5, -4, and 7.
Example 2

Iteration 2

\[ \begin{align*}
S & \rightarrow 0 \quad \text{cost} = 5 \\
F & \leftarrow 0 \quad \text{cost} = -4 \\
& X \rightarrow 3 \quad \text{cost} = -4 \\
& Y \rightarrow -4 \quad \text{cost} = 7
\end{align*} \]
Example 2

Iteration 2

- Iteration 2
- Node S with label -1 connected to node X with weight 5
- Node X with label 3 connected to node Y with weight 7
- Node Y with label -4 connected to node S with weight -4
Example 2

Iteration 2

-1  ->  3
  |    ^  5
  |    |   -2
  v    v
-4  --  X
      |    -4
      v
Y

s ↔ x

Iteration 2
Example 2

Check

\[ d[y] > d[s] + w(s, y) \implies \text{FALSE} \]
Another Look at Bellman-Ford

**Note:** This is essentially dynamic programming.

Let $d(i, j) =$ cost of the shortest path from $s$ to $i$ that is at most $j$ hops.

$$d(i, j) = \begin{cases} 
0 & \text{if } i = s \land j = 0 \\
\infty & \text{if } i \neq s \land j = 0 \\
\min\{d(k, j-1) + w(k, i): i \in \text{Adj}(k)\} \cup \{d(i, j-1)\} & \text{if } j > 0 
\end{cases}$$
KNAPSACK PROBLEM
Knapsack problem

- Given \( n \) objects and a "knapsack."
- Item \( i \) weighs \( w_i > 0 \) and has value \( v_i > 0 \).
- Knapsack has capacity of \( W \).
- Goal: fill knapsack so as to maximize total value.

Ex. \( \{ 1, 2, 5 \} \) has value 35.
Ex. \( \{ 3, 4 \} \) has value 40.
Ex. \( \{ 3, 5 \} \) has value 46 (but exceeds weight limit).

\[
\begin{array}{ccc}
 i & v_i & w_i \\
1 & 1 & 1 \\
2 & 6 & 2 \\
3 & 18 & 5 \\
4 & 22 & 6 \\
5 & 28 & 7 \\
\end{array}
\]

Knapsack instance 
(weight limit \( W = 11 \))

Greedy by value. Repeatedly add item with maximum \( v_i \).
Greedy by weight. Repeatedly add item with minimum \( w_i \).
Greedy by ratio. Repeatedly add item with maximum ratio \( v_i / w_i \).

Observation. None of greedy algorithms is optimal.
False start...

Def. $OPT(i) = \text{max profit subset of items } 1, \ldots, i$.

Case 1. $OPT$ does not select item $i$.
   • $OPT$ selects best of $\{1, 2, \ldots, i-1\}$.
   
   \[\text{optimal substructure property (proof via exchange argument)}\]

Case 2. $OPT$ selects item $i$.
   • Selecting item $i$ does not immediately imply that we will have to reject other items.
   • Without knowing what other items were selected before $i$, we don't even know if we have enough room for $i$.

Conclusion. Need more subproblems!
Def. $OPT(i, w) =$ max profit subset of items 1, ..., $i$ with weight limit $w$.

Case 1. $OPT$ does not select item $i$.
- $OPT$ selects best of $\{1, 2, ..., i-1\}$ using weight limit $w$.

Case 2. $OPT$ selects item $i$.
- New weight limit = $w - w_i$.
- $OPT$ selects best of $\{1, 2, ..., i-1\}$ using this new weight limit.

$$OPT(i, w) = \begin{cases} 
0 & \text{if } i = 0 \\
OPT(i-1, w) & \text{if } w_i > w \\
\max \{ OPT(i-1, w), \ v_i + OPT(i-1, w-w_i) \} & \text{otherwise}
\end{cases}$$
Dynamic programming algorithm

**KNAPSACK** \((n, W, w_1, \ldots, w_n, v_1, \ldots, v_n)\)

\[
\text{FOR } w = 0 \text { TO } W \\
\quad M[0, w] \leftarrow 0.
\]

\[
\text{FOR } i = 1 \text { TO } n \\
\quad \text{FOR } w = 1 \text { TO } W \\
\quad \quad \text{IF } (w_i > w) \quad M[i, w] \leftarrow M[i-1, w]. \\
\quad \quad \text{ELSE} \quad M[i, w] \leftarrow \max \{ M[i-1, w], \, v_i + M[i-1, w-w_i] \}.
\]

\[
\text{RETURN } M[n, W].
\]
Example

<table>
<thead>
<tr>
<th>i</th>
<th>$v_i$</th>
<th>$w_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>22</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>28</td>
<td>7</td>
</tr>
</tbody>
</table>

$OPT(i, w) = \begin{cases} 
0 & \text{if } i = 0 \\
OPT(i-1, w) & \text{if } w_i > w \\
\max\{OPT(i-1, w), v_i + OPT(i-1, w-w_i)\} & \text{otherwise}
\end{cases}$

<table>
<thead>
<tr>
<th>weight limit $w$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>{}</td>
</tr>
<tr>
<td>{1}</td>
</tr>
<tr>
<td>{1, 2}</td>
</tr>
<tr>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>{1, 2, 3, 4}</td>
</tr>
<tr>
<td>{1, 2, 3, 4, 5}</td>
</tr>
</tbody>
</table>

$OPT(i, w) = \max\text{ profit subset of items } 1, \ldots, i \text{ with weight limit } w$. 
Theorem. There exists an algorithm to solve the knapsack problem with $n$ items and maximum weight $W$ in $\Theta(nW)$ time and $\Theta(nW)$ space.

Pf.

- Takes $O(1)$ time per table entry.
- There are $\Theta(nW)$ table entries.
- After computing optimal values, can trace back to find solution: take item $i$ in $OPT(i, w)$ iff $M[i, w] < M[i-1, w]$.

Remarks.

- Not polynomial in input size!  

- Decision version of knapsack problem is NP-COMPLETE.  [CHAPTER 8]

- There exists a poly-time algorithm that produces a feasible solution that has value within 1% of optimum.  [SECTION 11.8]