COMP251: Dynamic programming (1)

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen et al., 2002) & (Kleinberg & Tardos, 2005)
Announces

Midterm:
• Mean 70%, Median 72%, Best 94%.

• Available for review (with solution) during my office hours.

Office hours:
• Today: Moved to 3pm to 4pm.

Assignment 3:
• Deadline postponed to March 22.
• Assignment 4 will be released on Monday.
Algorithms paradigms

• **Greedy:**
  - Build up a solution incrementally.
  - Iteratively decompose and reduce the size of the problem.
  - Top-down approach.

• **Dynamic programming:**
  - Solve all possible sub-problems.
  - Assemble them to build up solutions to larger problems.
  - Bottom-up approach.
INTRODUCTION
Activity-selection Problem

• **Input:** Set $S$ of $n$ activities, $a_1, a_2, ..., a_n$.
  - $s_i =$ start time of activity $i$.
  - $f_i =$ finish time of activity $i$.

• **Output:** Subset $A$ of maximum number of compatible activities.
  - 2 activities are compatible, if their intervals do not overlap.

Example:

Activities in each line are compatible.
Activity-selection Problem

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>$f_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Activities sorted by finishing time.

Activities:
- $a_1$ with $s_1 = 0$ and $f_1 = 2$
- $a_2$ with $s_2 = 1$ and $f_2 = 3$
- $a_3$ with $s_3 = 2$ and $f_3 = 5$
- $a_4$ with $s_4 = 4$ and $f_4 = 6$
- $a_5$ with $s_5 = 5$ and $f_5 = 9$
- $a_6$ with $s_6 = 6$ and $f_6 = 9$
- $a_7$ with $s_7 = 8$ and $f_7 = 10$
Activity-selection Problem

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>$f_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Activities sorted by finishing time.
Activity-selection Problem

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>$f_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Activities sorted by finishing time.
Activity-selection Problem

<table>
<thead>
<tr>
<th>i</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_i$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>$f_i$</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Activities sorted by finishing time.
Optimal sub-structure

• Let $S_{ij}$ = subset of activities in $S$ that start after $a_i$ finishes and finish before $a_j$ starts.

$$S_{ij} = \left\{ a_k \in S : \forall i, j \quad f_i \leq s_k < f_k \leq s_j \right\}$$

• $A_{ij}$ = optimal solution to $S_{ij}$

• $A_{ij} = A_{ik} \cup \{ a_k \} \cup A_{kj}$
# Greedy choice

<table>
<thead>
<tr>
<th># subproblems in optimal solution</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td># choices to consider</td>
<td>j-i-1</td>
</tr>
</tbody>
</table>

We can solve the problem $S_{ij}$ top-down:

- Consider all $a_m \in S_{ij}$
- Solve $S_{im}$ and $S_{mj}$
- Pick the best $m$ such that $A_{im} = A_{im} U \{ a_k \} U A_{im}$

$$A_{ij} = A_{ik} U \{ a_k \} U A_{kj}$$
Greedy choice

Theorem:
Let $S_{ij} \neq \emptyset$, and let $a_m$ be the activity in $S_{ij}$ with the earliest finish time: $f_m = \min \{ f_k : a_k \in S_{ij} \}$. Then:

1. $a_m$ is used in some maximum-size subset of mutually compatible activities of $S_{ij}$.
2. $S_{im} = \emptyset$, so that choosing $a_m$ leaves $S_{mj}$ as the only nonempty subproblem.
# Greedy choice

<table>
<thead>
<tr>
<th></th>
<th>Before theorem</th>
<th>After theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td># subproblems in</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>optimal solution</td>
<td></td>
<td></td>
</tr>
<tr>
<td># choices to consider</td>
<td>j-i-1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ A_{ij} = A_{ik} \cup \{ a_k \} \cup A_{kj} \]

\[ A_{ij} = \{ a_m \} \cup A_{mj} \]

We can now solve the problem \( S_{ij} \) top-down:

- Choose \( a_m \subseteq S_{ij} \) with the earliest finish time (greedy choice).
- Solve \( S_{mj} \).
Challenges

• Greedy choice is not always available.
• How to solve problem that have optimal substructures?
WEIGHTED INTERVAL SCHEDULING
Weighted interval scheduling

- **Input:** Set $S$ of $n$ activities, $a_1, a_2, \ldots, a_n$.
  - $s_i =$ start time of activity $i$.
  - $f_i =$ finish time of activity $i$.
  - $w_i =$ weight of activity $i$
- **Output:** find maximum weight subset of mutually compatible activities.
  - 2 activities are compatible, if their intervals do not overlap.

Example:
Application of the greedy algorithm

For $W=9$:

- The algorithm successfully packs the items.

For $W=3$:

- The algorithm fails to pack all items within the weight limit.

The diagrams illustrate the placement of items with their respective weights.
Discussion

• **Optimal substructure:** ✓
  - \( A_{ij} = \text{optimal solution to } S_{ij} \)
  - \( A_{ij} = A_{ik} \cup \{ a_k \} \cup A_{kj} \)

• **Greedy Choice:** ×
  - Select the activity with earliest finish time.
**Data structure**

**Notation:** All activities are sorted by finishing time $f_1 \leq f_2 \leq ... \leq f_n$

**Definition:** $p(j) =$ largest index $i < j$ such that activity/job $i$ is compatible with activity/job $j$.

**Examples:** $p(6)=4$, $p(5)=2$, $p(4)=2$, $p(2)=0$. 

---

![Diagram showing activities and their finishing times](image-url)
Binary Choice

**Notation:** \( \text{OPT}(j) = \text{value of the optimal solution to the problem} \)

\( = \text{max total weight of compatible activities} \ 1 \ldots j \)

**Case 1:** OPT selects activity \( j \)
- Add weight \( w_j \)
- Cannot use incompatible activities
- Must include optimal solution on remaining compatible activities \{ 1, 2, ..., \( p(j) \) \}.

**Case 2:** OPT does not select activity \( j \)
Must include optimal solution on others activities \{ 1, 2, ..., j-1 \}.

\[
\text{OPT}(j) = \begin{cases} 
0 & \text{if } j = 0 \\
\max\{w_j + \text{OPT}(p(j)), \text{OPT}(j - 1)\} & \text{Otherwise}
\end{cases}
\]
Recursive call

Input: n, s[1..n], f[1..n], v[1..n]

Sort jobs by finish time so that f[1] ≤ f[2] ≤ ... ≤ f[n].

Compute p[1], p[2], ..., p[n].

Compute-Opt(j)
if j = 0
    return 0.
else
    return max(v[j] + Compute-Opt(p[j]), Compute-Opt(j-1)).
Brute Force Approach

Observation: OPT(j) is calculated multiple times...

Case 1

Case 2

OPT(6)

w_6 + OPT(4)

w_6 + w_4 + OPT(2)

w_6 + w_4 + w_2

w_6 + w_3 + OPT(1)

w_6 + OPT(1)

w_6 + OPT(2)

w_5 + w_2

OPT(5)

w_5 + OPT(2)

w_5 + OPT(1)

OPT(4)

OPT(j) is calculated multiple times...
Memoization

Memoization: Cache results of each subproblem; lookup as needed.

Input: n, s[1..n], f[1..n], v[1..n]
Sort jobs by finish time so that f[1]≤f[2]≤ ... ≤f[n].
Compute p[1], p[2], ..., p[n].

for j = 1 to n
    M[j] ← empty.
M[0] ← 0.

M-Compute-Opt(j)
if M[j] is empty
    M[j] ← max(v[j]+M-Compute-Opt(p[j]),
        M-Compute-Opt(j–1)).
return M[j].
Claim. Memoized version of algorithm takes $O(n \log n)$ time.

- Sort by finish time: $O(n \log n)$.
- Computing $p(\cdot)$: $O(n \log n)$ via sorting by start time.

- **M-COMPUTE-OPT(j):** each invocation takes $O(1)$ time and either
  - (i) returns an existing value $M[j]$
  - (ii) fills in one new entry $M[j]$ and makes two recursive calls

- Progress measure $\Phi = \#$ nonempty entries of $M[\cdot]$.
  - Initially $\Phi = 0$, throughout $\Phi \leq n$.
  - (ii) increases $\Phi$ by 1 $\Rightarrow$ at most $2n$ recursive calls.

- Overall running time of **M-COMPUTE-OPT(n)** is $O(n)$.  ■

Remark. $O(n)$ if jobs are presorted by start and finish times.
DYNAMIC PROGRAMMING
Observation: When we compute $M[j]$, we only need values $M[k]$ for $k<j$.

**BOTTOM-UP** $(n; s_1, \ldots, s_n; f_1, \ldots, f_n; v_1, \ldots, v_n)$

Sort jobs by finish time so that $f_1 \leq f_2 \leq \ldots \leq f_n$. Compute $p(1), p(2), \ldots, p(n)$.

$M[0] \leftarrow 0$

for $j = 1$ TO $n$

\[
M[j] \leftarrow \max \{ v_j + M[p(j)], \ M[j-1] \}
\]

**Main Idea of Dynamic Programming:** Solve the sub-problems in an order that makes sure when you need an answer, it's already been computed.
Finding a solution

Dyn. Prog. algorithm computes optimal value.
Q: How to find solution itself?
A: Bactrack!

```
Find-Solution(j)
if j = 0
    return ∅.
else if (v[j] + M[p[j]] > M[j–1])
    return { j } ∪ Find-Solution(p[j])
else
    return Find-Solution(j–1).
```

Analysis. # of recursive calls ≤ n ⇒ O(n).
Example: Computing solution

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Computing solution

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$V_j+M[p(j)]$</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Computing solution

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Computing solution

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Computing solution

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>-</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Computing solution

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Reconstruction

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$V_j+M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
### Example: Reconstruction

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.
Example: Reconstruction

<table>
<thead>
<tr>
<th>activity</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>predecessor</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Best weight M</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>$V_j + M[p(j)]$</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$M[j-1]$</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>9</td>
</tr>
</tbody>
</table>

(1) Activities sorted by finishing time. (2) Weight equal to the length of activity.