Optimality and 0-1 Knapsack

Lecture 14
CS 312

Project

• 5% discount for late work. Doubles each day (except Sunday).
  – 5% Saturday, 10% Monday, 20% Tuesday etc.
• Unless you’ve made prior arrangements due to a personal emergency
• Follow guidelines on web page, but don’t get wordy and fluffy.

Objectives

• Define optimality
• Describe why optimality is important for dynamic programming
• Use DP to solve 0-1 knapsack

Optimality

• An optimal solution to a problem contains optimal solutions to to subproblems.
• In an optimal sequence of choices, each subsequence must also be optimal.

Optimality in Driving

Shortest route from AF to Provo passes through Orem, so...

Optimality in Driving

Shortest route from AF to Orem follows the shortest route from AF to Provo.
Non-optimality with Resources

Suppose you drive from AF to Orem as fast as you can, then you have to get gas before you get to Provo.

Optimality in DP

• Optimal solution to any nontrivial problem is built from optimal solutions to subproblems.
  • Which subinstances should be solved?
    – If you know, greedy might work
    – If not, DP might be a good choice

DP for shortest route

Is the shortest path from AF to Saratoga Springs relevant? Don’t know, so solve all subproblems and use the relevant ones.

Precomputing vs. on-the-fly

• Like lazy vs. eager evaluation
• Precomputing
  – build table, then analyze results.
• On-the-fly
  – build table entries as you go

Question

• How does having a table of intermediate results help find the shortest path from AF to Provo?
• Answer: Reuse results for intermediate locations as you try different routes.

0-1 Knapsack

• Same deal, can’t divide objects into fractions
• No universally optimal selection function
• Solution idea:
  – try all combinations.
  – use a table to store intermediate values
0-1 Knapsack Table

<table>
<thead>
<tr>
<th>w1, v1</th>
<th>0 1 2 3 4 5 6 7 8 9 10 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>w2, v6</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>w4, v12</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>w5, v18</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>w6, v22</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>w7, v28</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11</td>
</tr>
</tbody>
</table>

Entry $i,j$ represents the max value you can get from objects 0 through $i$ at weight $j$.

Compute $i,j$ as:

$$\max (V[i-1,j], V[i-1, j-w] + v_i)$$

## Complexity:

$\Theta(nW)$ to build table and $\Theta(n+W)$ to get final answer

---

Reading the solution.

Start with $V[5,11]$ which means "use all objects 1-5 to get a solution with weight = 11.


Next, try $V[4,11]$ which means "use all objects 1-4 to get a solution with weight = 11.


Next, try $V[3,5]$ which means "use all objects 1-3 to get a solution with weight = 5.

What if you didn’t use a table?

- Try each combination and see which one gives the best answer.
  - \(O(2^n)\)

Homework

- 8.11 write the algorithm for filling in the table for 0-1 knapsack.
  - related to the project
  - \textit{coins} on page 264 might be useful.