Intractable Problems

Part Three

Announcements

- Problem Set Six due right now.
  - Due Wednesday with a late day.
- Final project distributed at the end of lecture; details later today.

Please evaluate this course on Axess.

Your feedback really makes a difference.

Outline for Today

- **Pseudopolynomial Time**
  - A quick clarification from last time.
- **Another Algorithm for 0/1 Knapsack**
  - A totally different approach to knapsack.
- **FPTAS**
  - Extremely efficient approximation algorithms.

The 0/1 Knapsack Problem

The 0/1 Knapsack Problem

$500

45g

$200

20g

$900

53g

$1,500

25g

$400

35g

The 0/1 Knapsack Problem

$500

45g

$200

20g

$900

53g

$1,500

25g

$400

35g

The 0/1 Knapsack Problem

$500

45g

$200

20g

$900

53g

$1,500

25g

$400

35g

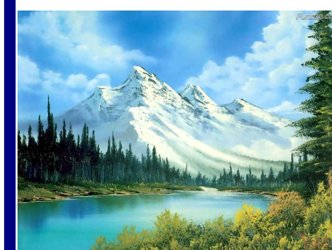
The 0/1 Knapsack Problem

$500

45g

$200

20g



$900

53g

$1,500

25g

$400

35g

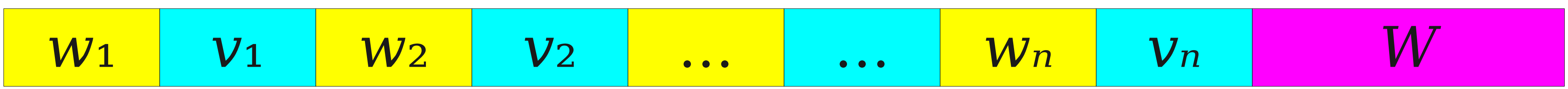
The 0/1 Knapsack Problem

- You are given a list of n items with weights $w_1, \ldots, w_n$ and values $v_1, \ldots, v_n$.
- You have a bag (knapsack) that can carry W total weight.
- Weights are assumed to be integers.
- **Question:** What is the maximum value of items that you can fit into the knapsack?
- This problem is known to be **NP**-hard.

From Last Time

- There is a DP algorithm that runs in time $O(nW)$, where n is the total number of items and W is the knapsack capacity.
- **Claim:** This is not a polynomial-time algorithm.
  - Rationale: The number W takes $\Theta(\log W)$ bits to write out, so the runtime is *exponential* in the number of bits of W.
- **Question:** Why is it polynomial in n?

Input Structure



Pseudopolynomial Time

- It takes $\Omega(n)$ bits to write out a list of n items, so an algorithm that works with n items and has runtime $O(n^k)$ runs in polynomial time.
- It takes $\Theta(\log n)$ bits to write out the number n, so an algorithm that takes in the number n and has runtime $O(n^k)$ runs in exponential time.

A Different Approach to 0/1 Knapsack

Parameterized Complexity

- Recall: a problem is ***fixed-parameter tractable*** if there is an algorithm for it with runtime $O(f(k) \cdot p(n))$ for some function $f(k)$ and polynomial $p(n)$.
- We can pick many different parameters for the same problem and get different algorithms.
- Useful: Depending on which parameters are fixed and can vary, different algorithms can be appropriate.

A Different Algorithm

- Our current algorithm asked the following question:

  What is the maximum value that fits in *X* space given just the first *k* items?

- Here is a different way to think about the problem:

  What is the minimum space needed to make *X* value with the first *k* items?

- Can solve 0/1 knapsack by answering this question for all possible profits and finding the highest value that can fit into the knapsack.

A Recurrence Relation

- Let OPT(k, X) be the minimum space necessary to store exactly X value with the first k items. (and ∞ if it's not possible to do so)
- **Claim:** OPT(k, X) satisfies this recurrence:

$$\mathrm{OPT}(k, X) = \begin{cases} 0 & \text{if } k=0 \text{ and } X=0 \\ \infty & \text{if } k=0 \text{ and } X>0 \\ OPT(k-1, X) & \text{if } v_k > X \\ min \begin{Bmatrix} OPT(k-1, X), \\ w_k + OPT(k-1, X - v_k) \end{Bmatrix} & \text{otherwise} \end{cases}$$

- Let V denote the maximum possible value obtainable ($V = v_1 + v_2 + \ldots + v_n$).

$$\mathrm{OPT}(k,X)=\begin{cases} 0 & \text{if } k=0 \text{ and } X=0 \\ \infty & \text{if } k=0 \text{ and } X>0 \\ OPT(k-1,X) & \text{if } v_k>X \\ min\begin{Bmatrix} OPT(k-1,X), \\ w_k+OPT(k-1,X-v_k) \end{Bmatrix} & \text{otherwise} \end{cases}$$

Let DP be an $(n + 1) \times (V + 1)$ table.

Set DP[0][0] = 0.

For X = 1 to V: Set DP[0][X] = ∞

For k = 1 to n, for X = 1 to V:

If $v_k > X$, set DP[k][X] = DP[k – 1][X].

Else, set DP[k][X] = min {
DP[k – 1][X], w_k + DP[k – 1][X – v_k].
}

For X = V to 0: if DP[n][X] ≤ W, return X.

Comparing Algorithms

- Brute-force algorithm: $O(2^n n)$
- First DP algorithm: $O(nW)$.
- This DP algorithm: **$O(nV)$**.
- Can use first DP algorithm if capacity is fixed and n will grow large.
- Can use second DP algorithm if total value is fixed and n will grow large.

An Interesting Observation

Approximation Schemes

- Let P be an optimization problem. Let X^* be the value of the optimal answer for P.
- Let A be an algorithm parameterized over two quantities:
  - The input to the problem.
  - An *accuracy* parameter $\varepsilon \in (0, 1]$.
- A is called an **approximation scheme** iff it produces a feasible answer X to P satisfying

$$(1 - \varepsilon)X^* \leq X$$

Our Algorithm

- Choose some integer k in terms of ε (we'll discuss how later on.)
- Let $v'_i = \lfloor v_i / k \rfloor$ for all v_i.
- Use the value-based DP algorithm to find the value of the optimal solution for the problem instance using values v'_i and the same weights as before.
- Return k times this value.

Our Algorithm

- Choose $\boldsymbol{k = \varepsilon v_{\max} / n}$.

- Let $v'_i = \lfloor v_i / k \rfloor$ for all v_i.
- Use the value-based DP algorithm to find the value of the optimal solution for the problem instance using values v'_i and the same weights as before.
- Return k times this value.

The Math, Part I

- For any feasible solution S to the original problem, let $c(S)$ denote the value of the items in S using the original values and $c'(S)$ denote the value of the items in S using the reduced values.
- Let S^* be the optimal solution to the original problem and S'^* be the optimal solution to the reduced values.
- Note: Optimal solution to the original problem is $c(S^*)$, and our approximation returns $kc'(S'^*)$.

The Math, Part II

- We want to bound the difference of the optimal solution and our estimate, which is given by $c(S^*) - kc'(S'^*)$.
- First, note that $c'(S'^*) \geq c'(S^*)$.
  - Rationale: S'^* is the optimal solution to the reduced problem, so its value in the reduced problem is at least the value of *any* solution in the reduced problem, including S^*.
- Therefore:

$$c(S^*) - kc'(S'^*) \leq c(S^*) - kc'(S^*)$$

The Math, Part III

- What is $c(S^*) - kc'(S^*)$?
- Note that

$$
\begin{aligned}
c(S^*) - kc'(S^*) &= \sum_{i \in S^*} v_i - k \sum_{i \in S^*} \lfloor \frac{v_i}{k} \rfloor \\
&= \sum_{i \in S^*} \left(v_i - k \lfloor \frac{v_i}{k} \rfloor \right) \\
&< \sum_{i \in S^*} k \\
&= nk
\end{aligned}
$$

- So $c(S^*) - kc'(S'^*) \leq nk$

The Math, Part IV

- For notational simplicity, let $X^* = c(S^*)$ and let $X = kc'(S'^*)$. This means that X^* is the optimal solution and X is our solution.
- From before, $X^* - X \leq nk$, so $X^* - nk \leq X$.
- Goal: Choose k so that $(1 - \varepsilon)X^* \leq X$.
- Note: If $nk \leq \varepsilon X^*$, then

$$(1 - \varepsilon)X^* = X^* - \varepsilon X^* \leq X^* - nk \leq X$$

The Math, Part V

- If $kn \leq \varepsilon X^*$, then $(1 - \varepsilon)X^* \leq X$ and we are done.
- So choose k so that $k \leq \varepsilon X^* / n$.
- Let v_{max} be the value of the highest-value item that fits into the knapsack.
- Then $X^* \geq v_{max}$. Set k = $\boldsymbol{\varepsilon v_{max} / n}$ to get

$$k = \varepsilon v_{max} / n \leq \varepsilon X^* / n$$

as required.

The Runtime

- For any k, the runtime is $O(nV / k)$.
- Since $k = \varepsilon v_{max} / n$, the runtime is $O(n^2V / \varepsilon v_{\mathrm{max}})$.
- Note that $V \leq nv_{max}$, so the runtime is **$O(n^3 / \varepsilon)$**.
- A **fully polynomial-time approxiximation scheme** (or **FPTAS**) is an approximation scheme whose runtime is a polynomial in the input size and $1 / \varepsilon$.
- *This is about as good as it gets if **P ≠ NP**!*

Why This Matters

- Some (but not all) **NP**-hard problems can be approximated using FPTAS's.
- Even if $\mathbf{P} \neq \mathbf{NP}$, can still approximate the answer to arbitrary precision in polynomial time.
- If you can settle for an approximate solution, you can often find very fast polynomial-time algorithms.

Dealing with Intractability

- To review:
  - If you need an exact answer, you can often do better than brute-forcing the answer.
  - If you need an exact answer, you can often find parameterized algorithms that are efficient for your setup.
  - If you can settle for an approximate answer, you can sometimes find efficient approximation algorithms.
- ***Intractable problems are not always as scary as they might seem!***

Next Time

- Where to Go From Here
- Further Topics in Algorithms
- Additional Courses in Algorithms
- ***Final Thoughts!***

The Final Project

The Final Project

- Choose and complete ***two*** of the three problems.
  - Please only submit answers to two problems; you're welcome to do all three, but we will only grade two.
- Each problem combines two of the techniques from the course, so solving two problems demonstrates an understanding of four techniques from the course.
- ***Please work independently***. Collaboration is not allowed on this project.
- ***Please do not use outside sources***. Refer to the handout for more details.
- Course staff can answer clarifying questions about the problems, but we will not offer as much help as normal.

Good Luck!