An Approximation Scheme for the Knapsack Problem

CS 511

Iowa State University

December 8, 2008
The Knapsack Problem: Problem Definition

**Input:** Set of $n$ objects, where item $i$ has value $v_i > 0$ and weight $w_i > 0$; a knapsack that can carry weight up to $W$.

**Goal:** Fill knapsack so as to maximize total value.
The Knapsack Problem

Example

Suppose $W = 11$.

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
<th>Weight</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>

- $S_1 = \{1, 2, 5\} \Rightarrow w(S_1) = 10, v(S_1) = 35.$
- $S_2 = \{3, 4\} \Rightarrow w(S_2) = 11, v(S_2) = 40.$
Knapsack is NP-complete

Definition (Knapsack, Decision Version)

Given a finite set $X$, nonnegative weights $w_i$, nonnegative values $v_i$, a weight limit $W$, and a target value $V$, is there a subset $S \subseteq X$ such that:

$$\sum_{i \in S} w_i \leq W \quad \text{and} \quad \sum_{i \in S} v_i \geq V$$

Theorem

Knapsack is NP-complete.

Proof.

Reduction from Subset-Sum.
A Dynamic Programming Algorithm

Subproblems
For each $i$ and $v$, find the minimum weight of a subset of $\{1, \ldots, i\}$ that yields value exactly $v$.

Substructure

Case 1: Optimum solution for $\{1, \ldots, i\}$ does not contain item $i$.
- Optimum solution is the minimum weight of a subset of $\{1, \ldots, i-1\}$ that achieves exactly value $v$.

Case 2: Optimum solution for $\{1, \ldots, i\}$ contains item $i$.
- Item $i$ consumes weight $w_i$.
- Optimum solution is $w_i$ plus the minimum weight of a subset of $\{1, \ldots, i-1\}$ that achieves exactly value $v - v_i$. 
A Dynamic Programming Algorithm

Definition

\( \text{opt}(i, v) \) is the minimum weight of a subset of \( \{1, \ldots, i\} \) that yields value exactly \( v \).

Recurrence Relation

\[
\begin{align*}
\text{opt}(i, v) &= \begin{cases} 
0 & \text{if } v = 0 \\
\infty & \text{if } i = 0, \ v > 0 \\
\text{opt}(i - 1, v) & \text{if } v_i > v \\
\min\{\text{opt}(i - 1, v), \ w_i + \text{opt}(i - 1, v - v_i)\} & \text{otherwise}
\end{cases}
\end{align*}
\]
A Dynamic Programming Algorithm

Algorithm

1. For $0 \leq i \leq n$, $0 \leq v \leq n v_{\text{max}}$, compute $\text{opt}(i, v)$, where $v_{\text{max}} = \max_i v_i$.

2. Return $V^* = \max\{v : \text{opt}(n, v) \leq W\}$

Run time

- Dominated by Step 1:
  - $O(n^2 v_{\text{max}})$ values to compute, $O(1)$ time per value.

- Total: $O(n^2 v_{\text{max}})$.

- Not polynomial.

- However, run time is pseudopolynomial.
Knapsack Approximation Algorithm

**Algorithm**

**Input:** An instance \((\{w_i\}, \{v_i\}, W)\) of Knapsack, and a real number \(\epsilon > 0\) (the precision parameter).

1. Let \(\theta\), the scaling factor, be

   \[
   \theta = \frac{\epsilon v_{\text{max}}}{n}.
   \]

2. (Rounding) For \(i = 1, 2, \ldots, n\), let

   \[
   \hat{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil.
   \]

3. Run the dynamic programming algorithm using values \(\hat{v}_i\), original weights \(w_i\) and original knapsack size \(W\).

4. Return the set \(S\) of items found in step 2.
Knapsack Approximation Algorithm

Run time

- Dominated by step 3:

\[ O(n^2 \hat{v}_{\text{max}}) = O\left(n^2 \left\lceil \frac{v_{\text{max}}}{\theta} \right\rceil \right) = O\left(\frac{n^3}{\epsilon}\right) \]

- Polynomial for each fixed \( \epsilon \).
Knapsack Approximation Algorithm

Intuition

Let

\[ \hat{v}_i = \left\lfloor \frac{v_i}{\theta} \right\rfloor \quad \bar{v}_i = \left\lceil \frac{v_i}{\theta} \right\rceil \theta \]

- Optimal solution to problems with \( \hat{v}_i \) or \( \bar{v}_i \) are equivalent.
- \( \bar{v}_i \)'s are close to \( v_i \)'s, so optimal solution using \( \bar{v}_i \)'s is nearly optimal.
- \( \hat{v}_i \)'s are small and integral, so dynamic programming algorithm is fast.
Knapsack Approximation Algorithm: Analysis

**Theorem**

*If S is solution found by our algorithm and S\(^*\) is any other feasible solution then \((1 + \epsilon) \sum_{i \in S} v_i \geq \sum_{i \in S^*} v_i.**

**Proof.**

\[
\sum_{i \in S^*} v_i \leq \sum_{i \in S^*} \bar{v}_i, \quad \text{since we round up}
\]
\[
\leq \sum_{i \in S} \bar{v}_i, \quad \text{since rounded instance is solved optimally}
\]
\[
\leq \sum_{i \in S} (v_i + \theta), \quad \text{since we round up by at most } \theta
\]
\[
\leq \sum_{i \in S} v_i + n\theta, \quad \text{since } |S| \leq n
\]
\[
\leq (1 + \epsilon) \sum_{i \in S} v_i, \quad \text{since } n\theta = \epsilon v_{\max} \text{ and } v_{\max} \leq \sum_{i \in S} v_i
\]
Summary: For every fixed $\epsilon$, there exists a polynomial-time approximation algorithm for the knapsack problem.

- Running time is $O(n^3/\epsilon)$.

In fact, we have a family of approximation algorithms for knapsack, one per choice of $\epsilon$. That is, we have a polynomial-time approximation scheme (PTAS).

Even better: Our algorithms are polynomial in $n$ and $1/\epsilon$, so we have a fully polynomial-time approximation scheme (FPTAS).

- Wouldn’t have been the case if running time had been, say, $O(n^{1/\epsilon})$. 