Lecture 5: The Knapsack Problem

1 The Knapsack Problem Defined

Suppose we are trying to burgle someone’s house. In this house we find the set

\[ S = a_1, a_2, \ldots, a_{n-1}, a_n \] a collection of objects

\[ s_1, s_2, \ldots, s_{n-1}, s_n \] their sizes

\[ p_1, p_2, \ldots, p_{n-1}, p_n \] their profits

that we would like to put in our knapsack of capacity \( B \).

Our goal is to find a subset of these objects, whose total size is bounded by \( B \) and whose profit is maximized. We can view this in terms of indicator variables, where

\[ x_i = \begin{cases} 
1 & \text{if } a_i \text{ is chosen} \\
0 & \text{otherwise}
\end{cases} \]

Then we need to find a setting of these \( x_i \)'s to maximize \( \sum p_i x_i \). This would be a very simple linear program, but we note that we have an integer constraint, since no fence will pay you one third of the price if you bring them one third of a piccolo. So the Knapsack Problem is an integer program that is \textbf{NP-Hard}. We would like to find a polynomial time approximation, so that we can leave the house we are burgling before the residents get home.

2 Running example

We will refer to this example throughout the remainder of the notes.
### A First Guess at Approximation

For each $i$, we consider the profit-to-size ratio $\frac{p_i}{s_i}$, and reorder the objects such that

$$\frac{p_1}{s_1} \geq \frac{p_2}{s_2} \geq \cdots \geq \frac{p_n}{s_n}$$

Referring back to the running example, this yields

<table>
<thead>
<tr>
<th>Objects</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sizes</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Profits</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Knapsack size: $B$

### Algorithm 1 (Greedy):

pick the first $k$ objects greedily in this order until the next object $a_{k+1}$ will not fit in the knapsack.

We can construct a simple example to show that this is not optimal:

$$\begin{align*}
\frac{100}{1} & \quad \frac{((100 \times B) - 1)}{B} \\
\uparrow & \quad \uparrow
\end{align*}$$

Chosen by Algorithm 1 More profitable overall

Note that by setting the profit appropriately, we can generalize this example to make the approximation ratio arbitrarily bad.

### Improving the Algorithm

We can modify the greedy algorithm to compensate for this issue:
Algorithm 2 (Smart Greedy): pick the more profitable of \( \{a_1, \ldots, a_k\} \) and \( \{a_{k+1}\} \).

This algorithm is still not optimal. Consider the example below:

\[
\begin{array}{ccc}
5/50 & 5/51 & 9/100 \\
\uparrow & \uparrow & \uparrow \\
\text{Chosen by Algorithm 1} & \text{Chosen by Algorithm 2} & \text{Most profitable overall}
\end{array}
\]

However, we are now going to be able to prove a bound on the approximation ratio.

**Claim 4.1** Algorithm 2 always gives a profit at least \( 0.5 \times OPT \), i.e. if the profit of the optimal solution is \( P^* \), then the profit of the solution found by Algorithm 2 is at least \( \frac{P^*}{2} \).

If we fill a knapsack to capacity, the most profit we can get is in order of profit density:

![Figure 1: A knapsack filled in order of profit ratio.](image)

Thus if we increased the size of the knapsack just enough to also allow space for the next most profitably-dense object, this would be an optimal profit for the slightly bigger knapsack! So it’s certainly an upper bound on the profit of the original knapsack, and so:

**Lemma 4.2** The optimal profit \( P^* \leq p_1 + p_2 + \cdots + p_k + p_{k+1} \).

Thus \( \frac{P^*}{2} \leq \max ((p_1 + p_2 + \cdots + p_k), p_{k+1}) \).
5 How close can we get to OPT?

We now show that no algorithm gets within an additive factor of OPT unless $P = NP$. In the next section, we then show that we can get arbitrarily close to optimal by a multiplicative factor.

**Theorem 5.1** If $P \neq NP$, no polynomial time approximation algorithm can solve Knapsack with value $P^* - k$ for any fixed constant $k$.

**Proof:** By contradiction.

Assume there exists a polynomial time algorithm $A$ with performance guarantee $k$ (an integer) for all instances of Knapsack. That is, for any Knapsack instance $I$, $A(I) = P^* - k$.

We show that $A$ can be used to construct a solution to Knapsack with value $P^*$ in polynomial time.

Suppose we have an instance of Knapsack:

$$I = \{(a_i, p_i, s_i)\} \text{ for } i = 1 \ldots n$$

Let $I' = \{(a'_i, p'_i, s'_i)\}$ where $a'_i = a_i$, $p'_i = p_i \cdot (k + 1)$ and $s'_i = s_i$.

Note: a feasible solution for $I$ corresponds to a feasible solution for $I'$, and the value of a solution for $I'$ is exactly $(k+1)$ times the value of the corresponding solution for $I$.

Let $M = A(I)$

If we run $A$ on $I'$, the solution satisfies the following:

$$|A(I') - P^*(I')| \leq k$$

$$|M \cdot (k + 1) - P^*(I) \cdot (k + 1)| \leq k$$

$$\Rightarrow |M - P^*(I)| \leq \frac{k}{k + 1}$$

Since $\frac{k}{k + 1} \leq 1$ and $M$ differs from $P^*(I)$ by an integral factor,

$$|M - P^*(I)| \leq 0$$
Which implies that $M$ is an optimal solution for $I$, which we demonstrated was impossible above.

$\Rightarrow \Leftarrow$

6 Approximation Schemes

Definition 6.1 Let $\pi$ be an optimization problem with objective function $f_\pi$ and optimal solution $S^*$. We say that $A$ is an approximation scheme for $\pi$ if on input $(I, \epsilon)$ where $I$ is an instance of $\pi$ and $\epsilon > 0$ is a fixed error parameter, it outputs a solution $S$ such that

\[
\begin{align*}
    f_\pi(I, S) &\leq (1 + \epsilon) \cdot S^* & \text{if } \pi \text{ is minimization problem} \\
    f_\pi(I, S) &\geq (1 - \epsilon) \cdot S^* & \text{if } \pi \text{ is maximization problem}
\end{align*}
\]

Definition 6.2 $A$ is a PTAS (Polynomial Time Approximation Scheme) if it is an approximation scheme where for each fixed $\epsilon > 0$, its running time is polynomial in the size of instance $I$.

Notice: “polynomial time” can depend arbitrarily badly on $\epsilon$:

\[n^3 \cdot 2^{2^{2^\frac{1}{\epsilon}}} \text{ or } n^3 2^{2^{\frac{1}{\epsilon}}}\]

Definition 6.3 $A$ is an FPTAS (Fully Polynomial Time Approximation Scheme) if in the previous scheme you require that the running time of $A$ be bounded by a polynomial in the size of $I$ and $1/\epsilon$.

Remark: If $P \neq NP$, an FPTAS is the best you can do for an NP-Hard problem.
7 An FPTAS for Knapsack using Dynamic Programming

We now show that Knapsack has an FPTAS, meaning that, at least in the approximate sense, it is an “easy” NP-hard problem.

Recall our running example

<table>
<thead>
<tr>
<th>Objects</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sizes</td>
<td>7</td>
<td>2</td>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Profits</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Knapsack size: B

Let $P_{max}$ be the profit of the most profitable object i.e. $P_{max} = \max_{i \in S}(p_i)$.

Trivially, $n \cdot P_{max} \geq P^*$.

Use the original arbitrary ordering. For each count $i = 1, \ldots, n$ and profit $p = 1, \ldots, n \cdot P_{max}$, let $S_{i,p}$ denote a subset of objects $a_1, \ldots, a_i$ whose total profit is exactly $p$, and whose total space is minimized.

Let $A(i, p)$ denote the space that set $S_{i,p}$ fills if it exists, or $\infty$ if no such set exists.

It is clear that

$$P^* = \max(p \mid A(n, p) \leq B)$$

We now show how to compute $A(i, p)$ for all $i = 1, \ldots, n$ and $p = 1, \ldots, n \cdot P_{max}$ in time $O(n^2 \cdot P_{max})$ using dynamic programming.

**Note:** This is not a contradiction to the NP-Hardness of Knapsack, because $P_{max}$ isn’t necessarily bounded by a polynomial in $n$ (although it would be if we required profits to be represented in unary).

To compute $A(i, p)$:

$A(1, p)$ is known for every value of $p$. Begin by initializing the table so that $A(1, p_1) = s_1$ and $A(1, p_{i \neq 1}) = \infty$. 
Using the recurrence relation: \( A(i + 1, p) = \min(A(i, p), A(i, p - p_{i+1}) + s_{i+1}) \),
it takes constant work to fill in each cell, thus costing \( O(n^2 \cdot P_{\text{max}}) \) to fill the table.

### 8 What to Do When Profits are Too Large

When \( P_{\text{max}} \) is bounded by a polynomial in the size of the input, then the above is a polynomial time algorithm for Knapsack. When it is not, we now present a \((1 - \epsilon)\)-approximation algorithm, by rounding the profit values.

**Algorithm 3 (FPTAS):**

1. Given an \( \epsilon > 0 \), let \( k = \frac{\epsilon \cdot P_{\text{max}}}{n} \).
2. For each \( a_i \), define \( p'_i = \left\lfloor \frac{p_i}{k} \right\rfloor \).
3. Let \( I' = (\langle a'_i, s'_i, p'_i \rangle) \) where \( a'_i = a_i, s'_i = s_i \), and \( p'_i \) as above.
4. Using the dynamic programming method from the previous section, find the most profitable set \( S' \).
5. Output $\max(S_{\text{max}}, S')$ where $S_{\text{max}}$ is the smallest object of profit $P_{\text{max}}$ if $S'_{\text{max}} \leq B$ otherwise 0.

**Lemma 8.1** Let $A$ denote the set output by Algorithm 3. Then $\text{Profit}(A) \geq (1 - \epsilon) \cdot P^*$. 

**Proof:** Let $O$ denote the set with profit $P^*$. For any $a_i$, $k \cdot p'_i$ can be smaller than $p_i$, but not by more than $k$.

$\Rightarrow \text{Profit}(O) - k \cdot \text{Profit}'(O) \leq n \cdot k$

Under the Profit' scheme, $\text{Profit}'(S')$ is OPTIMAL, i.e. $\text{Profit}'(S') \geq \text{Profit}'(Y)$ for all $Y$, and in particular $\text{Profit}'(S') \geq \text{Profit}'(O)$.

Thus:

$$\text{Profit}(S') \geq k \cdot \text{Profit}'(S') \geq k \cdot \text{Profit}'(O) \geq \text{Profit}(O) - n \cdot k = P^* - \epsilon \cdot P_{\text{max}}$$

Since the algorithm also considers the most profitable element

$$\text{Profit}(A) \geq \text{Profit}(P_{\text{max}}) \geq \text{Profit}(S') \geq P^* - \epsilon \cdot \text{Profit}(A)$$

**Theorem 8.2** Algorithm 3 is an FPTAS for Knapsack.

**Proof:** By lemma, the solution $P^*$ is within $(1 - \epsilon)$ of OPT.

The running time is $O\left(n^2 \left\lfloor \frac{P_{\text{max}}}{k} \right\rfloor \right) = O\left(n^2 \left\lfloor \frac{n}{\epsilon} \right\rfloor \right)$, which is polynomial in $n$ and $1/\epsilon$. 

8
9 Strong NP-Hardness

We now show that most problems are not as easy to approximate as knapsack, that is most problems do not admit an FPTAS.

Definition 9.1 A problem \( \pi \) is strongly NP-hard if every problem in NP can be polynomial-time reduced to \( \pi \) so that all numbers in the reduced instance can be written in unary.

Theorem 9.2 Let \( p \) be a polynomial and \( \pi \) be a strongly NP-hard minimization problem such that the objective function \( f_\pi \) is integer valued, and on any instance \( I \), the size of \( \text{OPT}(I) < p(|I|) \). Then \( \pi \) does not admit an FPTAS assuming \( P \neq NP \).

We don’t prove the theorem here even though the proof is not that hard. However, note that it implies that Knapsack is not strongly NP-hard.