

Lecture 4: The Knapsack Problem

1 The Knapsack Problem Defined

We are given a set

$$\begin{aligned} S &= a_1, a_2, \dots, a_{n-1}, a_n && \text{a collection of objects} \\ & s_1, s_2, \dots, s_{n-1}, s_n && \text{their sizes} \\ & p_1, p_2, \dots, p_{n-1}, p_n && \text{their profits} \end{aligned}$$

and a knapsack of capacity B .

Goal: Find a subset of objects whose total size is bounded by B and whose profit is maximized.

The Knapsack Problem is **NP-Hard**.

1.1 Running example

We will refer to this example throughout the remainder of the notes.

Objects	A	B	C	D	E
Sizes	7	2	9	3	1
Profits	3	2	3	1	2

Knapsack size: B

1.2 A First Guess at Approximation

For each i , consider

$$\frac{p_i}{s_i}$$

and reorder the objects s.t.

$$\frac{p_1}{s_1} > \frac{p_2}{s_2} > \dots > \frac{p_n}{s_n}$$

Referring back to the running example, this yields

Objects	A	B	C	D	E
Profits/Sizes	3/7	1	1/3	1/3	2
Order	a_3	a_2	a_4	a_5	a_1

Intuitively, this allows us to see the “profit density”, or profit per unit size.

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:

$$\begin{array}{ccc} 100/1 & & ((100 * B) - 1)/B \\ \uparrow & & \uparrow \\ \text{Chosen by Algorithm 1} & & \text{More profitable overall} \end{array}$$

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 $\{a_1, \dots, a_k\}$ 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 1.2.1 *Algorithm 2 always gives a profit at least $0.5 * OPT$, i.e. if the profit of the optimal solution is P^* , then the profit of the solution found by Algorithm 2 is $\geq \frac{P^*}{2}$.*

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



Figure 1: A knapsack filled in order of profit ratio.

Thus if I *increased the size of the knapsack* so that it fit exactly also the next most profitable (in ratio) 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 1.2.2 *The optimal profit $P^* \leq p_1 + p_2 + \dots + p_k + p_{k+1}$.*

Thus $\frac{P^*}{2} \leq \max((p_1 + p_2 + \dots + p_k), p_{k+1})$.

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 1.2.3 *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.

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:

$$I = \{ \langle a_i, p_i, s_i \rangle \text{ for } i = 1 \dots n \}$$

Let $I' = \{ \langle a'_i, p'_i, s'_i \rangle \}$ 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 same solution for I .

Let $M = A(I)$.

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

$$\begin{aligned} |A(I') - P^*(I')| &\leq k \\ \Rightarrow |M \cdot (k + 1) - P^*(I) \cdot (k + 1)| &\leq k \\ \Rightarrow |M - P^*(I)| &\leq \frac{k}{k + 1} \end{aligned}$$

However, $\frac{k}{k + 1} \leq 1$, and since our additive factor was integral

$$\Rightarrow |M - P^*(I)| \leq 0$$

Which implies that M is an optimal solution for I .

$\Rightarrow \Leftarrow$

Approximation Schemes

Definition 1.2.4 Let π be an optimization problem with objective function f_π and optimal solution S^* . We say that A is an **approximation scheme**

for π if on input (I, ϵ) where I is an instance of π and $\epsilon > 0$ is an error parameter, it outputs a solution S such that

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

Definition 1.2.5 *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 ϵ :

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

Definition 1.2.6 *A is a 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.

An FPTAS for Knapsack using Dynamic Programming

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

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

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

For each $i = 1, \dots, n$ and $p = 1, \dots, n \cdot P_{max}$, let $S_{i,p}$ denote a subset $a_1, \dots, a_i \subset S$ whose total profit is exactly p and whose total size is minimized.

Let $A(i, p)$ denote the size of set $S_{i,p}$ if it exists, or set to ∞ 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 $i = 1, \dots, n$ and $p = 1, \dots, n \cdot P_{max}$ in time $O(n^2 \cdot P_{max})$ using dynamic programming.

Notice: this is not a contradiction to the NP-Hardness of Knapsack, because P_{max} isn't necessarily bounded by a polynomial in n .

We will refer to the running example previously presented.

To compute $A(i, p)$:

$A(1, p)$ is known for every value of p .

Begin by initializing table so that $A(1, p_1) = S_1$ and $A(1, P_{i \neq 1}) = \infty$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	∞	∞	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2															
3															
4															
5															

Recurrence: $A(i + 1, p) = \min(A(i, p), A(i, p - p_{i+1}) + S_{i+1})$

Performing constant work to fill in each cell, it takes $O(n^2 \cdot P_{max})$ to fill the table.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	∞	∞	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	7	∞	9	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	2	7	∞	9	16	∞	18	∞	∞	∞	∞	∞	∞	∞
4	3	2	5	10	9	14	19	18	21	∞	∞	∞	∞	∞	∞
5	3	1	4	3	8	11	10	13	20	19	22	∞	∞	∞	∞

What to Do When Profits are Too Large

When P_{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 $k = \frac{\epsilon \cdot P_{max}}{n}$.
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 $\max(S_{max}, S')$ where S_{max} is the smallest object of profit P_{max} if $S_{max} \leq B$ otherwise 0.

Lemma 1.2.7 *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:

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

Since the algorithm also considers the most profitable element

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

Theorem 1.2.8 *Algorithm 3 is an FPTAS for Knapsack.*

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

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

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 1.2.9 *A problem Π is strongly NP-hard if every problem in MP can be polynomial-time reduced to Π so that all numbers can be written in unary.*

Theorem 1.2.10 *Let p be a polynomial and Π be a strongly NP-hard minimization problem s.t. the objective function f_Π is integer valued and on any instance I , the size of $OPT(I) < p(|I|)$. Then Π 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.