# Lecture 7: Polynomial-time approximation schemes

When faced with an NP-hard problem one cannot expect to find a polynomial-time algorithm that always gives an optimal solution. Hence, one has to settle for an approximate solution. Of course one would prefer that the approximate solution is very close optimal, for example at most 5% worse. In other words, one would like to have an approximation ratio very close to 1. The approximation algorithms we have seen so far do not quite achieve this: for Load Balancing we gave an algorithm with approximation ratio 3/2, for Weighted Vertex Cover we gave an algorithm with approximation ratio 2, and for Weighted Set Cover the approximation ratio was even $O(\log n)$. Unfortunately it is not always possible to get a better approximation ratio: for some problems one can prove that it is not only NP-hard to solve the problem exactly, but that there is a constant $c > 1$ such that there is no polynomial-time $c$-approximation algorithm unless P=NP. Vertex Cover, for instance, cannot be approximated to within a factor 1.3606... unless P=NP, and for Set Cover one cannot obtain a better approximation factor than $\Theta(\log n)$.

Fortunately there are also problems where much better solutions are possible. In particular, some problems admit a so-called *polynomial-time approximation scheme*, or *PTAS* for short. Such an algorithm works as follows. Its input is, of course, an instance of the problem at hand, but in addition there is an input parameter $\varepsilon > 0$. The output of the algorithm is then a solution whose value is at most $(1 + \varepsilon) \cdot \text{OPT}$ (for a minimization problem) or at least $(1 - \varepsilon) \cdot \text{OPT}$ (for a maximization problem). The running time of the algorithm should be polynomial in $n$; its dependency on $\varepsilon$ can be exponential however. So the running time can be $O(2^{1/\varepsilon} n^2)$ for example, or $O(n^{1/\varepsilon})$, or $O(n^2/\varepsilon)$, etc. If the dependency on the parameter $1/\varepsilon$ is also polynomial then we speak of a *fully polynomial-time approximation scheme (FPTAS)*. In this lecture we give an example of an FPTAS.

## 7.1 Knapsack

The Knapsack problem is defined as follows. We are given a set $X = \{x_1, \ldots, x_n\}$ of $n$ items that each have a (positive) *weight* and a (positive) *profit*. The weight and profit of $x_i$ are denoted by $\mathit{weight}(x_i)$ and $\mathit{profit}(x_i)$, respectively. Moreover, we have a knapsack that can carry items of total weight $W$. For a subset $S \subset X$, define $\mathit{weight}(S) := \sum_{x \in S} \mathit{weight}(x)$ and $\mathit{profit}(S) := \sum_{x \in S} \mathit{profit}(x)$. The goal is now to select a subset of the items whose profit is maximized, under the condition that the total weight of the selected items is at most $W$. From now on, we will assume that $\mathit{weight}(x_i) \leqslant W$ for all $i$. (Items with $\mathit{weight}(x_i) > W$ can of course simply be ignored.)

**The case of integer profits.** We will first develop an algorithm for the case where all the profits are integers. Let $P := \mathit{profit}(X)$, that is, $P$ is the total profit of all items. The running time of our algorithm will depend on $n$ and $P$. Since $P$ can be arbitrarily large, the running time of our algorithm will not necessarily be polynomial in $n$. In the next section we will then show how to obtain an FPTAS for Knapsack that uses this algorithm as a subroutine.

Our algorithm for the case where all profits are integers is a dynamic-programming algorithm. For $1 \leqslant i \leqslant n$ and $0 \leqslant p \leqslant P$, define

$$A[i,p] = \min\{\mathit{weight}(S) : S \subset \{x_1, \ldots, x_i\} \text{ and } \mathit{profit}(S) = p\}.$$

In other words, $A[i,p]$ denotes the minimum possible weight of any subset $S$ of the first $i$ items such that $\mathit{profit}(S)$ is exactly $p$. When there is no subset $S \subset \{x_1,\ldots,x_i\}$ of profit exactly $p$ then we define $A[i,p] = \infty$. Note that KNAPSACK asks for a subset of weight at most $W$ with the maximum profit. This maximum profit is given by $\textsc{opt} := \max\{p : 0 \leqslant p \leqslant P \text{ and } A[n,p] \leqslant W\}$. This means that if we can compute all values $A[i,p]$ then we can compute $\textsc{opt}$. From the table $A$ we can then also compute a subset $S$ such that $\mathit{profit}(S) = \textsc{opt}$—see below for details. As is usual in dynamic programming, the values $A[i,p]$ are computed bottom-up by filling in a table. It will be convenient to extend the definition of $A[i,p]$ to include the case $i = 0$, as follows: $A[0,0] = 0$ and $A[0,p] = \infty$ for $p > 0$. Now we can give a recursive formula for $A[i,p]$.

**Lemma 7.1**

$$A[i,p] \;=\; \begin{cases} 0 & \textit{if } p = 0 \\ \infty & \textit{if } i = 0 \textit{ and } p > 0 \\ A[i-1,p] & \textit{if } i > 0 \textit{ and } 0 < p < \mathit{profit}(x_i) \\ \min(A[i-1,p], A[i-1,p-\mathit{profit}(x_i)] + \mathit{weight}(x_i)) & \textit{if } i > 0 \textit{ and } p \geqslant \mathit{profit}(x_i) \end{cases}$$

*Proof.* The first two cases are simply by definition. Now consider third and fourth case. Obviously the minimum weight of any subset of $\{x_1,\ldots,x_i\}$ of total profit $p$ is given by one of the following two possibilities:

- the minimum weight of any subset $S \subset \{x_1,\ldots,x_i\}$ with profit $p$ and $x_i \in S$, or
- the minimum weight of any subset $S \subset \{x_1,\ldots,x_i\}$ with profit $p$ and $x_i \notin S$.

In the former case, $\mathit{weight}(S)$ is equal to $\mathit{weight}(x_i)$ plus the minimum weight of any subset $S \subset \{x_1,\ldots,x_{i-1}\}$ with profit $p - \mathit{profit}(x_i)$, which is given by $A[i-1, p-\mathit{profit}(x_i)]$. (This is also correct when $A[i-1,p-\mathit{profit}(x_i)] = \infty$. In that case there is no subset $S \subset \{x_1,\ldots,x_{i-1}\}$ of profit $p - \mathit{profit}(x_i)$, so there is no subset $S \subset \{x_1,\ldots,x_i\}$ of profit $p$ that includes $x_i$.) In the latter case, $\mathit{weight}(S)$ is equal to the minimum weight of any subset $S \subset \{x_1,\ldots,x_{i-1}\}$ with profit $p$, which is $A[i-1,p]$. (Again, this is also correct when $A[i-1,p] = \infty$.) When $p < \mathit{profit}(x_i)$ the former possibility does not apply, which proves the lemma for the third case. Otherwise we have to take the best of the two possibilities, proving fourth case. □

Based on this lemma, we can immediately give a dynamic-programming algorithm.

```
Algorithm IntegerWeightKnapsack($X, W$)
1.  Let $A[0..n, 0..P]$ be an array, where $P = \sum_{i=1}^{n} profit(x_i)$.
2.  $A[0,0] \leftarrow 0$
3.  for $p \leftarrow 1$ to $P$
4.      do $A[0,p] \leftarrow \infty$
5.  for $i \leftarrow 1$ to $n$
6.      do for $p \leftarrow 1$ to $P$
7.             do if $profit(x_i) \leqslant p$
8.                   then $A[i,p] \leftarrow \min(A[i-1,p], weight(x_i) + A[i-1, p - profit(x_i)])$
9.                   else $A[i,p] \leftarrow A[i-1,p]$
10. OPT $\leftarrow \max\{p : 0 \leqslant p \leqslant P$ and $A[n,p] \leqslant W\}$
11. Using the table $A$, find a subset $S \subset X$ of profit OPT and total weight at most $W$.
12. return $S$
```

Finding an optimal subset $S$ in line 11 of the algorithm can be done by "walking back" in the table $A$, as is standard in dynamic-programming algorithms—see also the chapter on dynamic programming from [CLRS]. For completeness, we describe a subroutine *ReportSolution* that finds an optimal subset.

**Algorithm** *ReportSolution*$(X, A, \text{OPT})$
```
1.  $p \leftarrow \text{OPT}$; $S \leftarrow \emptyset$
2.  for $i \leftarrow n$ downto 1
3.      do if $\mathit{profit}(x_i) \leqslant p$
4.          then if $\mathit{weight}(x_i) + A[i-1, p - \mathit{profit}(x_i)] < A[i-1, p]$
5.              then $S \leftarrow S \cup \{x_i\}$; $p \leftarrow p - \mathit{profit}(x_i)$
6.  return $S$
```

It is easy to see that *IntegerWeightKnapsack*, including the subroutine *ReportSolution*, runs in $O(nP)$ time. We get the following theorem.

**Theorem 7.2** *Suppose all profits in a* Knapsack *instance are integers. Then the problem can be solved in $O(nP)$ time, where $P := \mathit{profit}(X)$ is the total profit of all items.*

**An FPTAS for** Knapsack. How can we use the result above to obtain an FPTAS for the general case, where the profits can be arbitrarily large and need not even be integers? For this we would need to scale the profits down so that they are not too large, and then round them so that they are integral. This scaling and rounding will introduce some "error" in the computations—that is, since we will not work with the exact profits anymore, we may erroneously believe that a certain subset is better than another subset. The goal is to do the scaling and rounding in such a way that this error is small so that the result will be close to optimal. To obtain a $(1-\varepsilon)$-approximation, we want the error to be at most $\varepsilon \cdot \text{OPT}$. This leads to the following idea.

Suppose we "round" every $\mathit{profit}(x_i)$ to the next larger multiple of $(\varepsilon/n)\cdot\text{OPT}$. Since there are no more than $n$ items in any subset $S$, such a rounding cannot incur an error of more than $\varepsilon \cdot \text{OPT}$ in the profit of $S$. The nice thing is that after this rounding the whole problem can be scaled, because every profit is now a multiple of $(\varepsilon/n)\cdot\text{OPT}$. This suggests to replace each $\mathit{profit}(x_i)$ by $p_i$, where $p_i$ is the smallest integer such that $\mathit{profit}(x_i) \leqslant p_i \cdot ((\varepsilon/n)\cdot\text{OPT})$. The value $p_i$ satisfying this condition is given by

$$p_i := \lceil \frac{\mathit{profit}(x_i)}{(\varepsilon/n)\cdot\text{OPT}} \rceil. \tag{1}$$

Okay, we have replaced the profits $\mathit{profit}(x_i)$ by integer profits $p_i$. How large can the integers $p_i$ be? Let $j$ be such that $x_j$ is an item of maximum profit, that is, $j$ is such that $\mathit{profit}(x_i) \leqslant \mathit{profit}(x_j)$ for all $1 \leqslant i \leqslant n$. Then we also have $p_i \leqslant p_j$ for all $i$. Obviously, $\text{OPT} \geqslant \mathit{profit}(p_j)$. Hence,

$$p_j \leqslant \lceil \frac{\mathit{profit}(x_j)}{(\varepsilon/n)\cdot \mathit{profit}(x_j)} \rceil = \lceil n/\varepsilon \rceil.$$

It seems we are in business: we have transformed the problem to a problem where all the profits are integers in a polynomial range, namely $1..\lceil n/\varepsilon \rceil$, in such a way that the error introduced by the transformation is not too large. There is one problem, however: we do not know OPT, so we cannot round the profits using (1). Therefore, instead of using OPT we use the lower bound $\text{LB} := \max_i \mathit{profit}(x_i)$, and instead of using the profits $p_i$ we use $\mathit{profit}^*(x_i)$ which is defined as

$$\mathit{profit}^*(x_i) := \lceil \frac{\mathit{profit}(x_i)}{(\varepsilon/n)\cdot \text{LB}} \rceil. \tag{2}$$

Note that the profits are still integers in the range $1..\lceil n/\varepsilon \rceil$. Our FPTAS now look as follows.

**Algorithm** *Knapsack-FPTAS*$(X, W, \varepsilon)$

```
1.  $\text{LB} \leftarrow \max_{1 \leqslant i \leqslant n} \mathit{profit}(x_i)$.
2.  For all $1 \leqslant i \leqslant n$, let $\mathit{profit}^*(x_i) \leftarrow \lceil \frac{\mathit{profit}(x_i)}{(\varepsilon/n)\cdot \text{LB}} \rceil$.
3.  Compute a subset $S^* \subset X$ of maximum profit and total weight at most $W$ using algorithm
    IntegerWeightKnapsack, using the new profits $\mathit{profit}^*(x_i)$ instead of $\mathit{profit}(x_i)$.
4.  return $S^*$
```

We conclude with the following theorem.

**Theorem 7.3** *Knapsack-FPTAS computes in $O(n^3/\varepsilon)$ time a subset $S^* \subset X$ of weight at most $W$ whose profit is at least $(1-\varepsilon)\cdot \text{OPT}$, where OPT is the maximum profit of any subset of weight at most $W$.*

*Proof.* To prove the running time, we observe that $\mathit{profit}^*(x_i) \leqslant \lceil n/\varepsilon \rceil$ for all $1 \leqslant i \leqslant n$. Hence, the total profit $\mathit{profit}^*(X)$ is at most $n \cdot \lceil n/\varepsilon \rceil$, so by Theorem 7.2 the algorithm runs in $O(n^3/\varepsilon)$ time.

To prove the approximation ratio, let $S_{\text{opt}}$ denote an optimal subset, that is, a subset of weight at most $W$ such that $\mathit{profit}(S_{\text{opt}}) = \text{OPT}$. Let $S^*$ denote the subset returned by the algorithm. Since we did not change the weights of the items, the subset $S^*$ has weight at most $W$. It remains to show that $\mathit{profit}(S^*) \geqslant (1-\varepsilon)\cdot \text{OPT}$.

Because $S^*$ is optimal for the new profits, we have $\mathit{profit}^*(S^*) \geqslant \mathit{profit}^*(S_{\text{opt}})$. Moreover

$$\frac{\mathit{profit}(x_i)}{(\varepsilon/n)\cdot \text{LB}} \leqslant \mathit{profit}^*(x_i) \leqslant \frac{\mathit{profit}(x_i)}{(\varepsilon/n)\cdot \text{LB}} + 1.$$

Hence, we have

$$\begin{aligned}
\text{OPT} &= \textstyle\sum_{x_i \in S_{\text{opt}}} \mathit{profit}(x_i) \\
&\leqslant \textstyle\sum_{x_i \in S_{\text{opt}}} ((\varepsilon/n)\cdot \text{LB}) \cdot \mathit{profit}^*(x_i) \\
&\leqslant ((\varepsilon/n)\cdot \text{LB}) \cdot \mathit{profit}^*(S_{\text{opt}}) \\
&\leqslant ((\varepsilon/n)\cdot \text{LB}) \cdot \mathit{profit}^*(S^*) \\
&\leqslant ((\varepsilon/n)\cdot \text{LB}) \cdot \textstyle\sum_{x_i \in S^*} \left( \frac{\mathit{profit}(x_i)}{(\varepsilon/n)\cdot \text{LB}} + 1 \right) \\
&= \mathit{profit}(S^*) + ((\varepsilon/n)\cdot \text{LB}) \cdot |S^*| \\
&\leqslant \mathit{profit}(S^*) + \varepsilon \cdot \text{LB} \\
&\leqslant \mathit{profit}(S^*) + \varepsilon \cdot \text{OPT}
\end{aligned}$$

It follows that $\mathit{profit}(S^*) \geqslant (1-\varepsilon)\cdot \text{OPT}$, as claimed. □