8 Knapsack

In Chapter 1 we mentioned that some **NP**-hard optimization problems allow approximability to any required degree. In this chapter, we will formalize this notion and will show that the knapsack problem admits such an approximability.

Let Π be an **NP**-hard optimization problem with objective function f_Π. We will say that algorithm $\mathcal{A}$ is an *approximation scheme* for Π if on input (I, ε), where I is an instance of Π and $\varepsilon > 0$ is an error parameter, it outputs a solution s such that:

- $f_\Pi(I, s) \leq (1 + \varepsilon) \cdot \text{OPT}$ if Π is a minimization problem.
- $f_\Pi(I, s) \geq (1 - \varepsilon) \cdot \text{OPT}$ if Π is a maximization problem.

$\mathcal{A}$ will be said to be a *polynomial time approximation scheme*, abbreviated PTAS, if for each *fixed* $\varepsilon > 0$, its running time is bounded by a polynomial in the size of instance I.

The definition given above allows the running time of $\mathcal{A}$ to depend arbitrarily on ε. This is rectified in the following more stringent notion of approximability. If the previous definition is modified to require that the running time of $\mathcal{A}$ be bounded by a polynomial in the size of instance I and $1/\varepsilon$, then $\mathcal{A}$ will be said to be a *fully polynomial approximation scheme*, abbreviated FPTAS.

In a very technical sense, an FPTAS is the best one can hope for an **NP**-hard optimization problem, assuming **P** $\neq$ **NP**; see Section 8.3.1 for a short discussion on this issue. The knapsack problem admits an FPTAS.

Problem 8.1 (Knapsack) Given a set $S = \{a_1, \ldots, a_n\}$ of objects, with specified sizes and profits, $\text{size}(a_i) \in \mathbf{Z}^+$ and $\text{profit}(a_i) \in \mathbf{Z}^+$, and a "knapsack capacity" $B \in \mathbf{Z}^+$, find a subset of objects whose total size is bounded by B and total profit is maximized.

An obvious algorithm for this problem is to sort the objects by decreasing ratio of profit to size, and then greedily pick objects in this order. It is easy to see that as such this algorithm can be made to perform arbitrarily badly (Exercise 8.1).

8.1 A pseudo-polynomial time algorithm for knapsack

Before presenting an FPTAS for knapsack, we need one more concept. For any optimization problem Π, an instance consists of *objects*, such as sets or graphs, and *numbers*, such as cost, profit, size, etc. So far, we have assumed that all numbers occurring in a problem instance I are written in binary. The *size* of the instance, denoted $|I|$, was defined as the number of bits needed to write I under this assumption. Let us say that I_u will denote instance I with all numbers occurring in it written in unary. The *unary size* of instance I, denoted $|I_u|$, is defined as the number of bits needed to write I_u.

An algorithm for problem Π is said to be efficient if its running time on instance I is bounded by a polynomial in $|I|$. Let us consider the following weaker definition. An algorithm for problem Π whose running time on instance I is bounded by a polynomial in $|I_u|$ will be called a *pseudo-polynomial time algorithm.*

The knapsack problem, being **NP**-hard, does not admit a polynomial time algorithm; however, it does admit a pseudo-polynomial time algorithm. This fact is used critically in obtaining an FPTAS for it. All known pseudo-polynomial time algorithms for **NP**-hard problems are based on dynamic programming.

Let P be the profit of the most profitable object, i.e., $P = \max_{a \in S} \text{profit}(a)$. Then nP is a trivial upperbound on the profit that can be achieved by any solution. For each $i \in \{1, \ldots, n\}$ and $p \in \{1, \ldots, nP\}$, let $S_{i,p}$ denote a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimized. Let $A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists). Clearly $A(1,p)$ is known for every $p \in \{1, \ldots, nP\}$. The following recurrence helps compute all values $A(i,p)$ in $O(n^2P)$ time:

$$A(i+1,p) = \begin{cases} \min\{A(i,p),\ \text{size}(a_{i+1}) + A(i, p - \text{profit}(a_{i+1}))\} & \text{if } \text{profit}(a_{i+1}) < p \\ A(i+1,p) = A(i,p) & \text{otherwise} \end{cases}$$

The maximum profit achievable by objects of total size bounded by B is $\max\{p \mid A(n,p) \leq B\}$. We thus get a pseudo-polynomial algorithm for knapsack.

8.2 An FPTAS for knapsack

Notice that if the profits of objects were small numbers, i.e., they were bounded by a polynomial in n, then this would be a regular polynomial time algorithm, since its running time would be bounded by a polynomial in $|I|$. The key idea behind obtaining an FPTAS is to exploit precisely this fact: we will ignore a certain number of least significant bits of profits of objects (depending on the error parameter ε), so that the modified profits can be viewed as numbers bounded by a polynomial in n and $1/\varepsilon$. This will enable us to find a solution whose profit is at least $(1-\varepsilon)\cdot \text{OPT}$ in time bounded by a polynomial in n and $1/\varepsilon$.

Algorithm 8.2 (FPTAS for knapsack)

```
1. Given $\varepsilon > 0$, let $K = \frac{\varepsilon P}{n}$.
2. For each object $a_i$, define $\text{profit}'(a_i) = \left\lfloor \frac{\text{profit}(a_i)}{K} \right\rfloor$.
3. With these as profits of objects, using the dynamic programming
   algorithm, find the most profitable set, say $S'$.
4. Output $S'$.
```

Lemma 8.3 *Let A denote the set output by the algorithm. Then,*

$$\text{profit}(A) \geq (1-\varepsilon)\cdot \text{OPT}.$$

Proof: Let O denote the optimal set. For any object a, because of rounding down, $K\cdot \text{profit}'(a)$ can be smaller than $\text{profit}(a)$, but by not more than K. Therefore,

$$\text{profit}(O) - K\cdot \text{profit}'(O) \leq nK.$$

The dynamic programming step must return a set at least as good as O under the new profits. Therefore,

$$\begin{aligned}\text{profit}(S') \geq K\cdot \text{profit}'(O) &\geq \text{profit}(O) - nK = \text{OPT} - \varepsilon P \\ &\geq (1-\varepsilon)\cdot \text{OPT}\ ,\end{aligned}$$

where the last inequality follows from the observation that $\text{OPT} \geq P$. □

Theorem 8.4 *Algorithm 8.2 is a fully polynomial approximation scheme for knapsack.*

Proof: By Lemma 8.3, the solution found is within $(1-\varepsilon)$ factor of OPT. Since the running time of the algorithm is $O\left(n^2 \left\lfloor \frac{P}{K} \right\rfloor\right) = O\left(n^2 \left\lfloor \frac{n}{\varepsilon} \right\rfloor\right)$, which is polynomial in n and $1/\varepsilon$, the theorem follows. □

8.3 Strong NP-hardness and the existence of FPTAS's

In this section, we will prove in a formal sense that very few of the known **NP**-hard problems admit an FPTAS. First, here is a strengthening of the notion of **NP**-hardness in a similar sense in which a pseudo-polynomial algorithm is a weakening of the notion of an efficient algorithm. A problem Π is *strongly* **NP**-*hard* if every problem in **NP** can be polynomially reduced to Π in such a way that numbers in the reduced instance are always written in unary.

The restriction automatically forces the transducer to use polynomially bounded numbers only. Most known **NP**-hard problems are in fact strongly **NP**-hard; this includes all the problems in previous chapters for which approximation algorithms were obtained. A strongly **NP**-hard problem cannot have a pseudo-polynomial time algorithm, assuming $\mathbf{P} \neq \mathbf{NP}$ (Exercise 8.4). Thus, knapsack is not strongly **NP**-hard, assuming $\mathbf{P} \neq \mathbf{NP}$.

We will show below that under some very weak restrictions, any **NP**-hard problem admitting an FPTAS must admit a pseudo-polynomial time algorithm. Theorem 8.5 is proven for a minimization problem; a similar proof holds for a maximization problem.

Theorem 8.5 *Let p be a polynomial and Π be an* **NP**-*hard minimization problem such that the objective function f_Π is integer valued and on any instance I, $\mathrm{OPT}(I) < p(|I_u|)$. If Π admits an FPTAS, then it also admits a pseudo-polynomial time algorithm.*

Proof: Suppose there is an FPTAS for Π whose running time on instance I and error parameter ε is $q(|I|, 1/\varepsilon)$, where q is a polynomial.

On instance I, set the error parameter to $\varepsilon = 1/p(|I_u|)$, and run the FPTAS. Now, the solution produced will have objective function value less than or equal to:

$$(1+\varepsilon)\mathrm{OPT}(I) < \mathrm{OPT}(I) + \varepsilon p(|I_u|) = \mathrm{OPT}(I) + 1.$$

In fact, with this error parameter, the FPTAS will be forced to produce an optimal solution. The running time will be $q(|I|, p(|I_u|))$, i.e., polynomial in $|I_u|$. Therefore, we have obtained a pseudo-polynomial time algorithm for Π. □

The following corollary applies to most known **NP**-hard problems.

Corollary 8.6 *Let Π be an* **NP**-*hard optimization problem satisfying the restrictions of Theorem 8.5. If Π is strongly* **NP**-*hard, then Π does not admit an FPTAS, assuming* $\mathbf{P} \neq \mathbf{NP}$.

Proof: If Π admits an FPTAS, then it admits a pseudo-polynomial time algorithm by Theorem 8.5. But then it is not strongly **NP**-hard, assuming $\mathbf{P} \neq \mathbf{NP}$, leading to a contradiction. □

The stronger assumption that OPT $< p(|I|)$ in Theorem 8.5 would have enabled us to prove that there is a polynomial time algorithm for Π. However, this stronger assumption is less widely applicable. For instance, it is not satisfied by the minimum makespan problem, which we will study in Chapter 10.

8.3.1 Is an FPTAS the most desirable approximation algorithm?

The design of almost all known FPTAS's and PTAS's is based on the idea of trading accuracy for running time – the given problem instance is mapped to a coarser instance, depending on the error parameter ε, which is solved optimally by a dynamic programming approach. The latter ends up being an exhaustive search of polynomially many different possibilities (for instance, for knapsack, this involves computing $A(i, p)$ for all i and p). In most such algorithms, the running time is prohibitive even for reasonable n and ε. Further, if the algorithm had to resort to exhaustive search, does the problem really offer "footholds" to home in on a solution efficiently? Is an FPTAS or PTAS the best one can hope for for an **NP**-hard problem? Clearly, the issue is complex and there is no straightforward answer.

8.4 Exercises

8.1 Consider the greedy algorithm for the knapsack problem. Sort the objects by decreasing ratio of profit to size, and then greedily pick objects in this order. Show that this algorithm can be made to perform arbitrarily badly.

8.2 Consider the following modification to the algorithm given in Exercise 8.1. Let the sorted order of objects be $a_1, \ldots, a_n$. Find the lowest k such that the size of the first k objects exceeds B. Now, pick the more profitable of $\{a_1, \ldots, a_{k-1}\}$ and $\{a_k\}$ (we have assumed that the size of each object is at most B). Show that this algorithm achieves an approximation factor of 2.

8.3 (Bazgan, Santha, and Tuza [22]) Obtain an FPTAS for the following problem.

Problem 8.7 (Subset-sum ratio problem) Given n positive integers, $a_1 < \ldots < a_n$, find two disjoint nonempty subsets $S_1, S_2 \subseteq \{1, \ldots, n\}$ with $\sum_{i \in S_1} a_i \geq \sum_{i \in S_2} a_i$, such that the ratio

$$\frac{\sum_{i \in S_1} a_i}{\sum_{i \in S_2} a_i}$$

is minimized.

Hint: First, obtain a pseudo-polynomial time algorithm for this problem. Then, scale and round appropriately.

8.4 Show that a strongly **NP**-hard problem cannot have a pseudo-polynomial time algorithm, assuming $\mathbf{P} \neq \mathbf{NP}$.

8.5 Notes

Algorithm 8.2 is due to Ibarra and Kim [134]. Theorem 8.5 is due to Garey and Johnson [92].

9 Bin Packing

Consider the following problem.

Problem 9.1 (Bin packing) Given n items with sizes $a_1, \ldots, a_n \in (0, 1]$, find a packing in unit-sized bins that minimizes the number of bins used.

This problem finds many industrial applications. For instance, in the stock-cutting problem, bins correspond to a standard length of paper and items correspond to specified lengths that need to be cut.

It is easy to obtain a factor 2 approximation algorithm for this problem. For instance, let us consider the algorithm called First-Fit. This algorithm considers items in an arbitrary order. In the ith step, it has a list of partially packed bins, say $B_1, \ldots, B_k$. It attempts to put the next item, a_i, in one of these bins, in this order. If a_i does not fit into any of these bins, it opens a new bin B_{k+1}, and puts a_i in it. If the algorithm uses m bins, then at least $m-1$ bins are more than half full. Therefore,

$$\sum_{i=1}^{n} a_i > \frac{m-1}{2}.$$

Since the sum of the item sizes is a lower bound on OPT, $m - 1 < 2\text{OPT}$, i.e., $m \leq 2\text{OPT}$ (see Notes for a better analysis). On the negative side:

Theorem 9.2 *For any $\varepsilon > 0$, there is no approximation algorithm having a guarantee of $3/2 - \varepsilon$ for the bin packing problem, assuming $\mathbf{P} \neq \mathbf{NP}$.*

Proof: If there were such an algorithm, then we show how to solve the **NP**-hard problem of deciding if there is a way to partition n nonnegative numbers $a_1, \ldots, a_n$ into two sets, each adding up to $\frac{1}{2}\sum_i a_i$. Clearly, the answer to this question is 'yes' iff the n items can be packed in 2 bins of size $\frac{1}{2}\sum_i a_i$. If the answer is 'yes' the $3/2 - \varepsilon$ factor algorithm will have to give an optimal packing, and thereby solve the partitioning problem. $\square$

9.1 An asymptotic PTAS

Notice that the argument in Theorem 9.2 uses very special instances: those for which OPT is a small number, such as 2 or 3, even though the number of items is unbounded. What can we say about "typical" instances, those for which OPT increases with n?

Theorem 9.3 *For any ε, $0 < \varepsilon \leq 1/2$, there is an algorithm $\mathcal{A}_\varepsilon$ that runs in time polynomial in n and finds a packing using at most $(1 + 2\varepsilon)\mathrm{OPT} + 1$ bins.*

The sequence of algorithms, $\mathcal{A}_\varepsilon$, form an *asymptotic polynomial time approximation scheme* for bin packing, since for each $\varepsilon > 0$ $\exists N > 0$, and a polynomial time algorithm in this sequence, say $\mathcal{B}$, such that $\mathcal{B}$ has an approximation guarantee of $1 + \varepsilon$ for all instances having $\mathrm{OPT} \geq N$. However, Theorem 9.3 should not be considered a practical solution to the bin packing problem, since the running times of the algorithms $\mathcal{A}_\varepsilon$ are very high.

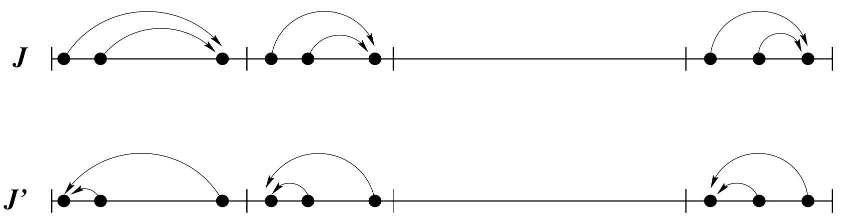
We will prove Theorem 9.3 in three steps.

Lemma 9.4 *Let $\varepsilon > 0$ be fixed, and let K be a fixed nonnegative integer. Consider the restriction of the bin packing problem to instances in which each item is of size at least ε and the number of distinct item sizes is K. There is a polynomial time algorithm that optimally solves this restricted problem.*

Proof: The number of items in a bin is bounded by $\lfloor 1/\varepsilon \rfloor$. Denote this by M. Therefore, the number of different bin types is bounded by $R = \binom{M+K}{M}$ (see Exercise 9.4), which is a (large!) constant. Clearly, the total number of bins used is at most n. Therefore, the number of possible feasible packings is bounded by $P = \binom{n+R}{R}$, which is polynomial in n (see Exercise 9.4). Enumerating them and picking the best packing gives the optimal answer. $\square$

Lemma 9.5 *Let $\varepsilon > 0$ be fixed. Consider the restriction of the bin packing problem to instances in which each item is of size at least ε. There is a polynomial time approximation algorithm that solves this restricted problem within a factor of $(1 + \varepsilon)$.*

Proof: Let I denote the given instance. Sort the n items by increasing size, and partition them into $K = \lceil 1/\varepsilon^2 \rceil$ groups each having at most $Q = \lfloor n\varepsilon^2 \rfloor$ items. Notice that two groups may contain items of the same size.



Construct instance J by rounding up the size of each item to the size of the largest item in its group. Instance J has at most K different item sizes.

Therefore, by Lemma 9.4, we can find an optimal packing for J. Clearly, this will also be a valid packing for the original item sizes. We show below that $\mathrm{OPT}(J) \leq (1+\varepsilon)\mathrm{OPT}(I)$, thereby proving the lemma.

The following clever argument accomplishes this. Let us construct another instance, say J', by rounding down the size of each item to that of the smallest item in its group. Clearly $\mathrm{OPT}(J') \leq \mathrm{OPT}(I)$. The crucial observation is that a packing for instance J' yields a packing for all but the largest Q items of instance J (Exercise 9.6 asks for a formal proof). Therefore,

$$\mathrm{OPT}(J) \leq \mathrm{OPT}(J') + Q \leq \mathrm{OPT}(I) + Q.$$

Since each item in I has size at least ε, $\mathrm{OPT}(I) \geq n\varepsilon$. Therefore, $Q = \lfloor n\varepsilon^2 \rfloor \leq \varepsilon\mathrm{OPT}$. Hence, $\mathrm{OPT}(J) \leq (1+\varepsilon)\mathrm{OPT}(I)$. □

Proof of Theorem 9.3: Let I denote the given instance, and I' denote the instance obtained by discarding items of size $< \varepsilon$ from I. By Lemma 9.5, we can find a packing for I' using at most $(1+\varepsilon)\mathrm{OPT}(I')$ bins. Next, we start packing the small items (of size $< \varepsilon$) in a First-Fit manner in the bins opened for packing I'. Additional bins are opened if an item does not fit into any of the already open bins.

If no additional bins are needed, then we have a packing in $(1+\varepsilon)\mathrm{OPT}(I') \leq (1+\varepsilon)\mathrm{OPT}(I)$ bins. In the second case, let M be the total number of bins used. Clearly, all but the last bin must be full to the extent of at least $1-\varepsilon$. Therefore, the sum of the item sizes in I is at least $(M-1)(1-\varepsilon)$. Since this is a lower bound on OPT, we get

$$M \leq \frac{\mathrm{OPT}}{(1-\varepsilon)} + 1 \leq (1+2\varepsilon)\mathrm{OPT} + 1,$$

where we have used the assumption that $\varepsilon \leq 1/2$. Hence, for each value of ε, $0 < \varepsilon \leq 1/2$, we have a polynomial time algorithm achieving a guarantee of $(1+2\varepsilon)\mathrm{OPT} + 1$. □

Algorithm $\mathcal{A}_\varepsilon$ is summarized below.

Algorithm 9.6 (Algorithm $\mathcal{A}_\varepsilon$ for bin packing)

1. Remove items of size $< \varepsilon$.
2. Round to obtain constant number of item sizes (Lemma 9.5).
3. Find optimal packing (Lemma 9.4).
4. Use this packing for original item sizes.
5. Pack items of size $< \varepsilon$ using First-Fit.

9.2 Exercises

9.1 Give an example on which First-Fit does at least as bad as $5/3 \cdot \text{OPT}$.

9.2 (Johnson [149]) Consider a more restricted algorithm than First-Fit, called Next-Fit, which tries to pack the next item only in the most recently started bin. If it does not fit, it is packed in a new bin. Show that this algorithm also achieves factor 2. Give a factor 2 tight example.

9.3 (C. Kenyon) Say that a bin packing algorithm is *monotonic* if the number of bins it uses for packing a subset of the items is at most the number of bins it uses for packing all n items. Show that whereas Next-Fit is monotonic, First-Fit is not.

9.4 Prove the bounds on R and P stated in Lemma 9.4.
Hint: Use the fact that the number of ways of throwing n identical balls into k distinct bins is $\binom{n+k-1}{n}$.

9.5 Consider an alternative way of establishing Lemma 9.5. All items having sizes in the interval $(\varepsilon(1+\varepsilon)^r, \varepsilon(1+\varepsilon)^{r+1}]$ are rounded up to $\min(\varepsilon(1+\varepsilon)^{r+1}, 1)$, for $r \geq 0$. Clearly, this yields a constant number of item sizes. Does the rest of the proof go through?
Hint: Consider the situation that there are lots of items of size $1/2$, and $1/2 \neq \varepsilon(1+\varepsilon)^r$ for any $r \geq 0$.

9.6 Prove the following statement made in Lemma 9.5, "A packing for instance J' yields a packing for all but the largest Q items of instance J."
Hint: Throw away the Q largest items of J and the Q smallest items of J', and establish a domination.

9.7 Use the fact that integer programming with a fixed number of variables is in **P** to give an alternative proof of Lemma 9.4. (Because of the exorbitant running time of the integer programming algorithm, this variant is also impractical.)

9.8 Show that if there is an algorithm for bin packing having a guarantee of $\text{OPT}(I) + \log^2(\text{OPT}(I))$, then there is a fully polynomial approximation scheme for this problem.

9.9 (C. Kenyon) Consider the following problem.

Problem 9.7 (Bin covering) Given n items with sizes $a_1, \ldots, a_n \in (0, 1]$, maximize the number of bins opened so that each bin has items summing to at least 1.

Give an asymptotic PTAS for this problem when restricted to instances in which item sizes are bounded below by c, for a fixed constant $c > 0$.
Hint: The main idea of Algorithm 9.6 applies to this problem as well.

9.3 Notes

The first nontrivial bin packing result, showing that First-Fit requires at most $(17/10)\mathrm{OPT}+3$ bins, was due to Ullman [248]. The asymptotic PTAS is due to Fernandez de la Vega and Lueker [86]. An improved algorithm, having a guarantee of $\mathrm{OPT}(I) + \log^2(\mathrm{OPT}(I))$ was given by Karmarkar and Karp [163]. For further results, see the survey of Coffman, Garey, and Johnson [50]. The result cited in Exercise 9.7, showing that integer programming with a fixed number of variables is in **P**, is due to Lenstra [185]. Bin packing has also been extensively studied in the on-line model. For these and other on-line algorithms see Borodin and El-Yaniv [31].