Approximation algorithms for the knapsack problem (2/2)

Outline

Performance guarantees
- The worst case performance ratio of GR
- The worst case performance ratio of GR^e
- The worst case performance ratio of H^2
- The worst case performance ratio of H^ℓ
- Approximation schemes
- The worst case performance ratio of the Basic-FPTAS

Absolute performance guarantee

Typical mistakes

C++ implementation

In summary

Where we were

- In the first part we described five algorithms for the 0/1 knapsack problem (KP), with the items sorted by non-increasing efficiency and the split item s:
  - the greedy algorithm GR and the Ext-Greedy algorithm GR^e;
  - the H^2 and H^ℓ algorithms, which guess the items of largest profit of an optimal solution;
  - the Basic-FPTAS algorithm FP^ε, which rounds down the profits.

- In this part we prove how far from the optimum each of them can be, in the worst case.

Performance guarantees

Introduction: Performance guarantees

- We now determine the worst case performance ratio of the algorithms of the previous sections
- For each algorithm: an upper bound on the ratio for every instance, and a family of instances that shows the bound is tight
- At the end we obtain two approximation schemes, a PTAS and an FPTAS

The performance ratio

- Recall that, the KP being a maximization problem, the performance ratio of an algorithm $\mathscr{A}$ on an instance I is

$$\frac{z(I)}{\mathscr{A}(I)} \geq 1.$$

- $r(\mathscr{A})$ is its supremum over all the instances: the **worst case performance ratio** of $\mathscr{A}$.

The worst case performance ratio of *GR*

Proposition 1

There are KP instances I such that:

$$\frac{z(I)}{GR(I)} \to +\infty \qquad \text{as } c \to +\infty.$$

The proof builds a family of instances with two items.

Proof (1/2)

Fix an integer $\vartheta > 2$ and consider the KP instance I_ϑ with knapsack capacity $c = \vartheta$ and the items set $\mathscr{I} = \{1, 2\}$, where the item profits and weights are

$$p_1 = 2, \quad w_1 = 1, \qquad \text{and} \qquad p_2 = \vartheta, \quad w_2 = \vartheta.$$

An optimal solution packs item 2, and therefore

$$z(I_\vartheta) = \vartheta.$$

Proof (2/2)

Since

$$\frac{p_1}{w_1} = 2 > \frac{p_2}{w_2} = \frac{\vartheta}{\vartheta} = 1,$$

GR packs item 1 first, leaving available capacity $\vartheta - 1$, so item 2 cannot be added. Thus

$$GR(I_\vartheta) = 2.$$

Hence,

$$\lim_{\vartheta \to +\infty} \frac{z(I_\vartheta)}{GR(I_\vartheta)} = \lim_{\vartheta \to +\infty} \frac{\vartheta}{2} = +\infty.$$

□

GR has no constant guarantee

- The proposition shows that there is no constant $\alpha \in [1, +\infty)$ such that
$$\frac{z(I)}{GR(I)} \leq \alpha \qquad \text{for all instances } I.$$
- Equivalently, the ratio $\frac{z(I)}{GR(I)}$ can be made arbitrarily large by a suitable choice of the instance.
- In other words, there is no constant $\alpha \in [1, +\infty)$ such that *GR* is an α-approximation algorithm, and the worst-case performance ratio of *GR* is unbounded:
$$r(GR) = +\infty.$$

The key drawback of *GR*

- A key drawback of *GR* is that it may fail to select the item with maximum profit (even when that single item is optimal).
- In particular, by prioritizing efficiency p_i/w_i, *GR* can fill the knapsack with high-efficiency items and leave insufficient available capacity to include the maximum-profit item.

The worst case performance ratio of GR^e

Proposition 2

For every KP instance I, we have:

$$\frac{z(I)}{GR^e(I)} \leq 2.$$

The proof compares $z(I)$ with the LP relaxation, written with the split item,
and $GR^e(I)$ with its two bounds.

Proof (1/2)

Assume items are sorted by non-increasing efficiency. Let s be the split item and rc be the residual capacity. We have

$$\underbrace{\sum_{i=1}^{s-1} p_i \; + \; p_s \frac{rc}{w_s}}_{=z\left(\mathrm{LP_{KP}}\right)} \; \leq \; \sum_{i=1}^{s-1} p_i \; + \; p_s \qquad \text{since} \qquad rc < w_s,$$

where $z\left(\mathrm{LP_{KP}}\right)$ is the optimal solution value of the LP relaxation of the natural ILP formulation of the KP.

Accordingly,

$$z(I) \leq z\left(\mathrm{LP_{KP}}\right) \leq \sum_{i=1}^{s-1} p_i + p_s \leq \sum_{i=1}^{s-1} p_i + p_{\max},$$

since by definition of $p_{\max}$ we have $p_{\max} \geq p_s$.

Proof (2/2)

Moreover,

$$GR^e(I) \geq GR(I) \geq \sum_{i=1}^{s-1} p_i \quad \text{and} \quad GR^e(I) \geq p_{\max}.$$

Summing these two inequalities yields

$$2\, GR^e(I) = GR^e(I) + GR^e(I) \;\geq\; \sum_{i=1}^{s-1} p_i + p_{\max}.$$

Therefore,

$$z(I) \leq \sum_{i=1}^{s-1} p_i + p_{\max} \;\leq\; 2\, GR^e(I).$$

Dividing by $GR^e(I) > 0$ yields

$$\frac{z(I)}{GR^e(I)} \leq 2.$$

The guarantee 2 is tight

Proposition 3

There are KP instances I such that:

$$\frac{z(I)}{GR^e(I)} \to 2 \quad \text{as} \quad c \to +\infty.$$

The proof builds a family of instances with three items.

Proof (1/2)

Fix an integer $\vartheta > 2$ and consider the KP instance I_ϑ with capacity $c = 2\vartheta$ and 3 items:

$$p_1 = 2,\ w_1 = 1, \qquad p_2 = \vartheta,\ w_2 = \vartheta, \qquad p_3 = \vartheta,\ w_3 = \vartheta.$$

Items 2 and 3 fit together into the knapsack since $w_2 + w_3 = 2\vartheta = c$; hence an optimal solution packs items 2 and 3 and

$$z(I_\vartheta) = 2\vartheta.$$

Moreover,

$$\frac{p_1}{w_1} = 2 > \frac{p_2}{w_2} = \frac{p_3}{w_3} = 1,$$

so *GR* packs item 1 first.

Proof (2/2)

After packing item 1, the available capacity is $2\vartheta - 1$, hence exactly one of items 2 and 3 can be added. Therefore GR returns a feasible solution of value $\vartheta + 2$.

Since $p_{\max} = \max\{p_1, p_2, p_3\} = \vartheta$, we have $\vartheta + 2 > p_{\max}$, and thus

$$GR^e(I_\vartheta) = \vartheta + 2.$$

Consequently,

$$\lim_{\vartheta \to +\infty} \frac{z(I_\vartheta)}{GR^e(I_\vartheta)} = \lim_{\vartheta \to +\infty} \frac{2\vartheta}{\vartheta + 2} = \lim_{\vartheta \to +\infty} \frac{2}{1 + \frac{2}{\vartheta}} = 2.$$

□

GR^e is a 2-approximation algorithm

- From the two propositions above it follows that the worst case performance ratio of GR^e is two, i.e., we have

$$r(GR^e) = 2$$

and GR^e is a 2-approximation algorithm.

- A key strength of GR^e is that it mitigates the main weakness of GR: even if the greedy packing misses a very profitable item because of the efficiency ordering, GR^e explicitly compares the greedy solution with the best single-item feasible solution and returns the better one, thereby guaranteeing a constant-factor approximation algorithm.

The limit of GR^e

- However, the guarantee $r(GR^e) = 2$ is tight: there are instances in which GR^e is misled by efficiency and, after selecting an early high-efficiency item, it no longer has enough capacity to pack together the two items with the largest profits (which form the optimal solution), yielding a ratio arbitrarily close to 2.

Towards better algorithms

- Getting a solution which is in the worst-case only half as good as the optimal solution does not sound very impressive.
- Hence, it is worthwhile trying to find a better approximation algorithm, possibly at the cost of an increased running time.
- In the previous worst-case example GR^e made the "mistake" of excluding item 3, which had a very high profit value. To avoid such a mistake we need more advanced heuristic algorithms.

The worst case performance ratio of H^2

Proposition 4

For every KP instance I, we have:

$$\frac{z(I)}{H^2(I)} \leq \frac{4}{3}.$$

The proof looks at the pair of items of largest profit of an optimal solution,
and splits into two cases.

Proof (1/6)

If $z(I)$ is generated by a single item, we have $H^2(I) = z(I)$.

Otherwise consider a feasible pair $(\tilde{i}, \tilde{k}) \in \mathscr{P}$ which contributes the highest profit in an optimal solution.

- The residual instance is $I(\tilde{i}, \tilde{k})$ and the residual set of items is $\mathscr{I}(\tilde{i}, \tilde{k})$.
- The H^2 algorithm considers this pair of items during its execution and applies GR^e on the residual instance $I(\tilde{i}, \tilde{k})$.

The pair contains the two items of largest profit of an optimal solution, so every other item of that solution has profit at most $\min\{p_{\tilde{i}}, p_{\tilde{k}}\}$ and fits into the residual capacity: it belongs to $\mathscr{I}(\tilde{i}, \tilde{k})$.

Proof (2/6)

The optimal solution value can therefore be written as:

$$z(I) = z\big(I(\tilde{i}, \tilde{k})\big) + p_{\tilde{i}} + p_{\tilde{k}} \qquad (*)$$

where $z\big(I(\tilde{i}, \tilde{k})\big)$ is the optimal solution value of the residual instance $I(\tilde{i}, \tilde{k})$.

From the operations of H^2, we have:

$$H^2(I) \quad \geq \quad p_{\tilde{i}} + p_{\tilde{k}} + GR^e\big(I(\tilde{i}, \tilde{k})\big) \qquad (**)$$

and since GR^e is a 2-approximation algorithm, i.e., $GR^e(I) \geq \frac{1}{2}\, z(I)$, we have:

$$GR^e\big(I(\tilde{i}, \tilde{k})\big) \quad \geq \quad \frac{1}{2}\, z\big(I(\tilde{i}, \tilde{k})\big)$$

Proof (3/6)

Accordingly:

$$H^2(I) \geq p_{\tilde{i}} + p_{\tilde{k}} + \frac{1}{2} z\big(I(\tilde{i}, \tilde{k})\big)$$

Moreover, due to $(*)$, we have:

$$H^2(I) \geq p_{\tilde{i}} + p_{\tilde{k}} + \frac{1}{2}\big(z(I) - p_{\tilde{i}} - p_{\tilde{k}}\big) = \frac{1}{2}\big(z(I) + p_{\tilde{i}} + p_{\tilde{k}}\big)$$

We consider two possible cases:

$$i) \;\; p_{\tilde{i}} + p_{\tilde{k}} \geq \frac{1}{2} z(I) \qquad \text{and} \qquad ii) \;\; p_{\tilde{i}} + p_{\tilde{k}} < \frac{1}{2} z(I)$$

Proof (4/6)

Case i):

$$p_{\tilde{i}} + p_{\tilde{k}} \geq \frac{1}{2} z(I)$$

We have in this case:

$$H^2(I) \geq \frac{1}{2}\left(z(I) + p_{\tilde{i}} + p_{\tilde{k}}\right)$$

$$\geq \frac{1}{2}\left(z(I) + \frac{1}{2} z(I)\right) = \frac{3}{4} z(I)$$

and accordingly

$$H^2(I) \geq \frac{3}{4} z(I)$$

Proof (5/6)

Case *ii*):

$$p_{\tilde{i}} + p_{\tilde{k}} < \frac{1}{2} z(I)$$

Since their profits sum to less than $\frac{1}{2} z(I)$, at least one of the two items $\tilde{i}$, $\tilde{k}$ has a profit smaller than $\frac{1}{4} z(I)$.

From $\mathrm{LP}_{\mathrm{KP}}$ of the residual instance $I(\tilde{i}, \tilde{k})$, we have:

$$\begin{aligned} z\big(I(\tilde{i}, \tilde{k})\big) &\leq z\big(\mathrm{LP}_{\mathrm{KP}}\big) \\ &\leq GR^e\big(I(\tilde{i}, \tilde{k})\big) + p_s \\ &\leq GR^e\big(I(\tilde{i}, \tilde{k})\big) + \frac{1}{4} z(I) \end{aligned}$$

where s is the split item of $I(\tilde{i}, \tilde{k})$ and $p_s < \frac{1}{4} z(I)$ since all its items have a profit at most $\min\{p_{\tilde{i}}, p_{\tilde{k}}\} < \frac{1}{4} z(I)$ (by construction of the residual instance).

Proof (6/6)

If all the items of the residual instance fit into the residual capacity there is no split item: then GR^e packs all of them, $z\big(I(\tilde{i},\tilde{k})\big) = GR^e\big(I(\tilde{i},\tilde{k})\big)$, and the inequalities hold with $p_s := 0$.
We have:

$$z(I) = p_{\tilde{i}} + p_{\tilde{k}} + z\big(I(\tilde{i},\tilde{k})\big)$$

$$\leq \underbrace{p_{\tilde{i}} + p_{\tilde{k}} + GR^e\big(I(\tilde{i},\tilde{k})\big)}_{\leq H^2(I),\ \text{by } (**)} + \frac{1}{4}\, z(I)$$

$$\leq H^2(I) + \frac{1}{4}\, z(I)$$

and accordingly $H^2(I) \geq \frac{3}{4}\, z(I)$.

In both cases $H^2(I) \geq \frac{3}{4}\, z(I)$, that is, dividing by $H^2(I) > 0$,

$$\frac{z(I)}{H^2(I)} \leq \frac{4}{3}.$$

The guarantee $\frac{4}{3}$ is tight

Proposition 5

There are KP instances I_M such that:

$$\frac{z(I_M)}{H^2(I_M)} \to \frac{4}{3} \qquad \text{as} \qquad M \to +\infty.$$

The proof builds a family of instances with one small item and four big items.

Proof (1/2)

Let us consider the KP instance I_M with capacity $c = 4\,M$ and 5 items:

$$p_1 = 2, \quad w_1 = 1, \quad \text{and}$$

$$p_2 = p_3 = p_4 = p_5 = M, \quad w_2 = w_3 = w_4 = w_5 = M,$$

where $M \in \mathbb{Z}_{\geq 1}, M > 2$ is an integer number.

An optimal solution packs the four big items, since $w_2 + w_3 + w_4 + w_5 = 4\,M = c$, hence

$$z(I_M) = 4\,M.$$

The efficiencies are

$$\frac{p_1}{w_1} = 2 \quad > \quad \frac{p_2}{w_2} = \frac{p_3}{w_3} = \frac{p_4}{w_4} = \frac{p_5}{w_5} = \frac{M}{M} = 1$$

so item 1 comes first in the efficiency order.

Proof (2/2)

We consider the two possible kinds of pair examined by H^2:

1. a pair of two big items: the residual capacity is $2\,M$ and the residual instance keeps item 1 and the two remaining big items, so GR^e packs item 1 and one big item, for a total value of $3\,M + 2$;
2. a pair consisting of item 1 and one big item: the residual instance is empty, because no item has profit at most $\min\{2, M\} = 2$, for a total value of $M + 2$.

The first kind of pair gives the larger value, hence $H^2(I_M) = 3\,M + 2$. Consequently,

$$\lim_{M\to\infty} \frac{z(I_M)}{H^2(I_M)} = \lim_{M\to\infty} \frac{4\,M}{3\,M+2} = \lim_{M\to\infty} \frac{4}{3+\frac{2}{M}} = \frac{4}{3}.$$

□

H^2 is a $\frac{4}{3}$-approximation algorithm

- From the two propositions above it follows that the worst case performance ratio of H^2 is four thirds, i.e., we have

$$r(H^2) = \frac{4}{3}$$

and H^2 is a $\frac{4}{3}$-approximation algorithm.

- Guessing two items instead of one brings the ratio down from the 2 of GR^e to $\frac{4}{3}$.

The worst case performance ratio of H^ℓ

Proposition 6

For every KP instance I, we have:

$$\frac{z(I)}{H^\ell(I)} \leq \frac{\ell+2}{\ell+1}.$$

The proof follows the one for H^2, with the set of the ℓ items of largest profit of an optimal solution in place of the pair.

Proof (1/4)

If an optimal solution contains at most $\ell - 1$ items, the algorithm finds it among the sets of $\widehat{T}(\ell)$, and $H^{\ell}(I) = z(I)$.

Otherwise, let $\tilde{L} \in T(\ell)$ be the set of the ℓ items of largest profit of an optimal solution. The algorithm considers $\tilde{L}$ during its execution and applies GR^{e} to the residual instance $I(\tilde{L})$. All the other items of the optimal solution belong to $\mathscr{I}(\tilde{L})$, hence

$$z(I) = \sum_{i \in \tilde{L}} p_i + z\big(I(\tilde{L})\big), \qquad (*)$$

where $z\big(I(\tilde{L})\big)$ is the optimal value of the residual instance.

Proof (2/4)

From the operations of H^ℓ, and since GR^e is a 2-approximation algorithm,

$$H^\ell(I) \geq \sum_{i \in \tilde{L}} p_i + GR^e\big(I(\tilde{L})\big)$$

$$\geq \sum_{i \in \tilde{L}} p_i + \frac{1}{2} z\big(I(\tilde{L})\big) = \frac{1}{2}\Big(z(I) + \sum_{i \in \tilde{L}} p_i\Big),$$

where the last equality follows from $(*)$. We consider two cases.

Case i): $\sum_{i \in \tilde{L}} p_i \geq \frac{\ell}{\ell+2} z(I)$. Then

$$H^\ell(I) \geq \frac{1}{2}\left(z(I) + \frac{\ell}{\ell+2} z(I)\right) = \frac{\ell+1}{\ell+2} z(I).$$

Proof (3/4)

Case *ii*): $\sum_{i \in \tilde{L}} p_i < \frac{\ell}{\ell+2} z(I)$.

- Then at least one item of $\tilde{L}$ has profit smaller than $\frac{1}{\ell+2} z(I)$, and so do all the items of the residual instance, whose profits are at most $\min\{p_i : i \in \tilde{L}\}$.
- Let s be the split item of $I(\tilde{L})$ (if there is none, GR^e packs all the items of $I(\tilde{L})$, and the inequalities below hold with $p_s := 0$).

The LP relaxation of the residual instance gives

$$\begin{aligned} z\big(I(\tilde{L})\big) \;\leq\; z\big(\mathrm{LP}_{\mathrm{KP}}\big) \;&\leq\; GR^e\big(I(\tilde{L})\big) + p_s \\ &<\; GR^e\big(I(\tilde{L})\big) + \frac{1}{\ell+2} z(I), \end{aligned}$$

Proof (4/4)

and therefore, by $(*)$,

$$z(I) < \sum_{i \in \tilde{L}} p_i + GR^e\left(I(\tilde{L})\right) + \frac{1}{\ell+2} z(I)$$

$$\leq H^\ell(I) + \frac{1}{\ell+2} z(I),$$

that is, $H^\ell(I) > \frac{\ell+1}{\ell+2} z(I)$.

In both cases $H^\ell(I) \geq \frac{\ell+1}{\ell+2} z(I)$, and dividing by $H^\ell(I) > 0$ gives the statement. □

The guarantee $\frac{\ell+2}{\ell+1}$ is tight

Proposition 7

There are KP instances I_M such that:

$$\frac{z(I_M)}{H^{\ell}(I_M)} \rightarrow \frac{\ell+2}{\ell+1} \qquad \text{as} \qquad M \rightarrow +\infty.$$

The proof generalizes the instance used for H^2.

Proof (1/2)

Fix an integer $M > 2$ and consider the KP instance I_M with capacity $c = (\ell + 2)\, M$ and $n = \ell + 3$ items, where

$$p_1 = 2, \quad w_1 = 1, \quad \text{and} \quad p_j = w_j = M \quad \text{for} \quad j \in \{2, 3, \ldots, n\}.$$

- An optimal solution packs the $\ell + 2$ items of weight M, which fill the knapsack, hence $z(I_M) = (\ell + 2)\, M$.
- Item 1 comes first in the efficiency order, since $\frac{p_1}{w_1} = 2 > 1$.

Proof (2/2)

The sets L examined by H^ℓ are of two kinds:

1. L made of ℓ items of profit M: the residual capacity is $2M$, the residual instance keeps item 1 and the two remaining items of profit M, and GR^e packs item 1 and one of them, for a total value of $\ell M + M + 2 = (\ell + 1) M + 2$;
2. L containing item 1 and $\ell - 1$ items of profit M: the residual instance is empty, since no item has profit at most $\min\{2, M\} = 2$, for a total value of $(\ell - 1) M + 2$.

The sets of at most $\ell - 1$ items give at most $(\ell - 1) M + 2$ as well. Hence $H^\ell(I_M) = (\ell + 1) M + 2$ and

$$\lim_{M \to \infty} \frac{z(I_M)}{H^\ell(I_M)} = \lim_{M \to \infty} \frac{(\ell + 2) M}{(\ell + 1) M + 2} = \lim_{M \to \infty} \frac{\ell + 2}{\ell + 1 + \frac{2}{M}} = \frac{\ell + 2}{\ell + 1}.$$

For $\ell = 2$ this is the instance used for H^2. □

H^ℓ is a $\frac{\ell+2}{\ell+1}$-approximation algorithm

- From the two propositions above it follows that the worst case performance ratio of H^ℓ is

$$r(H^\ell) = \frac{\ell + 2}{\ell + 1}$$

and H^ℓ is a $\frac{\ell+2}{\ell+1}$-approximation algorithm.

- For $\ell = 2$ we find again $r(H^2) = \frac{4}{3}$, and the ratio tends to 1 as ℓ grows.

Approximation schemes: the definitions

We recall the definitions of the chapter «Principles of approximation algorithms».

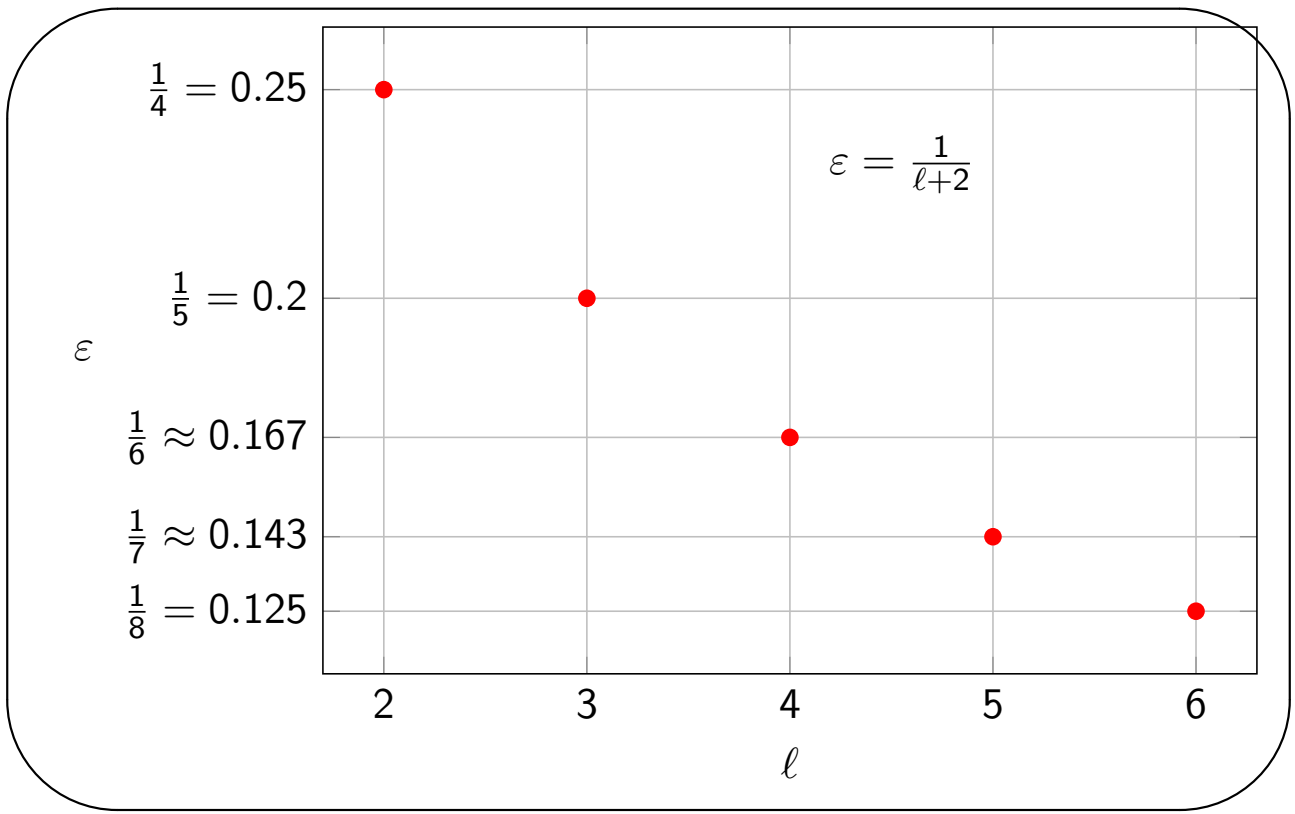
- The **relative error** of an algorithm $\mathscr{A}$ on a KP instance I is $\dfrac{z(I) - \mathscr{A}(I)}{z(I)}$.
- An **approximation scheme** is a family of algorithms $\{\mathscr{A}_\varepsilon\}_{\varepsilon \in (0,1)}$ such that, for every ε and every instance I, the relative error of $\mathscr{A}_\varepsilon$ is at most ε.
- For a maximization problem this means that $\mathscr{A}_\varepsilon$ is a $\frac{1}{1-\varepsilon}$-approximation algorithm.
- The scheme is a **PTAS** if, for every fixed ε, the running time of $\mathscr{A}_\varepsilon$ is polynomial in the size of the instance.
- It is an **FPTAS** if the running time is polynomial both in the size of the instance and in the accuracy $\frac{1}{\varepsilon}$.

The relative error of H^ℓ

By the guarantee of H^ℓ, $H^\ell(I) \geq \frac{\ell+1}{\ell+2}\, z(I)$, hence the relative error of H^ℓ is at most

$$1 - \frac{\ell+1}{\ell+2} = \frac{1}{\ell+2},$$

which tends to 0 as ℓ grows:



The H^ε algorithm

- The relation can be read the other way round: given the relative error ε to be attained, it is enough to choose ℓ with $\frac{1}{\ell+2} \leq \varepsilon$, i.e., $\ell \geq \frac{1}{\varepsilon} - 2$.

- By choosing

$$\ell(\varepsilon) = \min\left\{\max\left\{\left\lceil\frac{1}{\varepsilon}\right\rceil - 2,\ 2\right\},\ n\right\}$$

  we obtain the H^ε algorithm, i.e., H^ℓ with $\ell = \ell(\varepsilon)$, whose relative error is at most ε for every $\varepsilon \in (0, 1)$.

- For $\varepsilon \geq \frac{1}{4}$ we have $\ell(\varepsilon) = 2$, and H^ε is the H^2 algorithm.

- The size ℓ of the guessed sets cannot exceed n, and with $\ell = n$ the algorithm enumerates all the feasible solutions and is exact.

H^ε is a PTAS

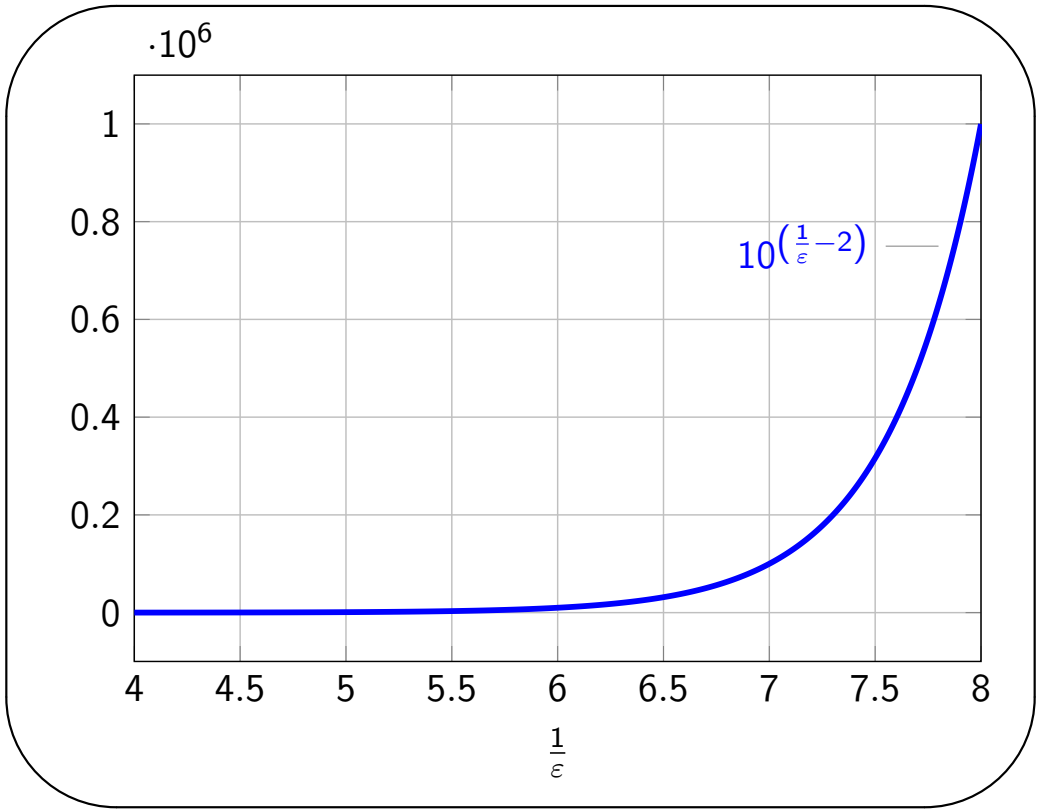
Observation 1

The family $\{H^\varepsilon\}_{\varepsilon\in(0,1)}$ is a PTAS for the 0/1 knapsack problem.

- For every fixed ε, $\ell(\varepsilon)$ is a constant, and by the observation on the running time of H^ℓ the running time $O\left(n^{\ell(\varepsilon)+1}\right)$ is polynomial in the size n of the instance.
- However, H^ε is **not** an FPTAS: the exponent $\ell(\varepsilon) = \left\lceil \frac{1}{\varepsilon} \right\rceil - 2$ grows with the accuracy $\frac{1}{\varepsilon}$, so the running time is not polynomial in $\frac{1}{\varepsilon}$.

The running time of H^{ε}

The plot shows, for $n = 10$ items and $4 \leq \frac{1}{\varepsilon} \leq 8$, the number $n^{\frac{1}{\varepsilon}-2}$, a lower bound on $n^{\ell(\varepsilon)}$ (the cap $\ell(\varepsilon) \leq n$ is not active, since $\frac{1}{\varepsilon} \leq 12$):



H^{ε} in practice

- The H^{ε} algorithm is a straightforward method and quite easy to implement.
- However, its running time explodes as the required accuracy grows.
- Even for a moderate relative error $\varepsilon = \frac{1}{10}$, i.e., a deviation of 10 percent from the optimal value, we have $\ell(\varepsilon) = 8$ and a running time of $O(n^9)$, or still $O(n^8)$ with the improved implementation.

The worst case performance ratio of the Basic-FPTAS

Proposition 8

For every KP instance I and every $\varepsilon \in (0,1)$, we have:

$$\frac{z(I)}{FP^{\varepsilon}(I)} \leq \frac{1}{1-\varepsilon}.$$

The proof compares the solution returned with an optimal solution, through the scaled profits.

Proof (1/2)

Let $\tilde{S}$ be an optimal solution of I and let S be the solution returned by FP^{ε}, i.e., an optimal solution of the scaled instance I'. Since I and I' have the same feasible solutions, $\tilde{S}$ is feasible for I' as well, and by optimality of S for I',

$$\sum_{j \in S} p'_j \geq \sum_{j \in \tilde{S}} p'_j.$$

By definition of the scaled profits, $K\, p'_j \leq p_j$ and $K\, p'_j > p_j - K$ for every item j. Therefore

$$FP^{\varepsilon}(I) = \sum_{j \in S} p_j \geq K \sum_{j \in S} p'_j \geq K \sum_{j \in \tilde{S}} p'_j$$

$$> \sum_{j \in \tilde{S}} (p_j - K) = z(I) - |\tilde{S}|\, K.$$

Proof (2/2)

Since an optimal solution contains at most n items and $K = \varepsilon\, GR^e(I)/n$,

$$|\tilde{S}|\, K \;\leq\; n\, K \;=\; \varepsilon\, GR^e(I) \;\leq\; \varepsilon\, z(I),$$

where the last inequality holds because $GR^e(I)$ is the value of a feasible solution.

Combining the two displays,

$$FP^{\varepsilon}(I) \;\geq\; z(I) - \varepsilon\, z(I) \;=\; (1-\varepsilon)\, z(I),$$

and dividing by $FP^{\varepsilon}(I) > 0$ gives the statement.

On the instance of the execution of FP^{ε} in the first part, with $\varepsilon = \frac{1}{2}$, the ratio is $\frac{47}{38} \approx 1.24$, well below the bound $\frac{1}{1-\varepsilon} = 2$. □

The running time of FP^{ε}

Observation 2

The FP^{ε} algorithm can be computed in $O\left(n^2/\varepsilon\right)$ time.

The proof bounds the optimal value of the scaled instance, which fixes the size of the table of the dynamic program.

Proof (1/2)

Let $\tilde{q}$ be the optimal value of the scaled instance I'. Every feasible solution of I' is feasible for I, hence

$$\tilde{q} \leq \frac{z(I)}{K} \leq \frac{2\,GR^e(I)}{K} = \frac{2\,GR^e(I)\,n}{\varepsilon\,GR^e(I)} = \frac{2n}{\varepsilon},$$

where $z(I) \leq 2\,GR^e(I)$ is exactly the relative performance guarantee of GR^e.

Since $\tilde{q}$ is an integer, $\tilde{q} \leq Q = \lfloor 2n/\varepsilon \rfloor$, and truncating the table of the dynamic program at Q does not change the result.

Proof (2/2)

- The table has $n+1$ rows and $Q+1$ columns, so the dynamic program runs in $O(n\,Q) = O(n^2/\varepsilon)$ time, and the scaling and the call to GR^e take $O(n)$ and $O(n \log n)$ time.
- Without the truncation, the table would reach the total scaled profit $P' = \sum_j p'_j$.
- Since $p_j \leq GR^e(I)$ gives $p'_j \leq \frac{n}{\varepsilon}$, we only have $P' \leq \frac{n^2}{\varepsilon}$, which can be reached, and the running time would be $O(n\,P') = O(n^3/\varepsilon)$.

□

FP^{ε} is an FPTAS

- From the last proposition and the last observation it follows that the family $\{FP^{\varepsilon}\}_{\varepsilon \in (0,1)}$ is an **approximation scheme** whose running time is polynomial both in the size n of the instance and in the accuracy $\frac{1}{\varepsilon}$, i.e., an **FPTAS** for the 0/1 knapsack problem.

- The guarantee of GR^e is used twice, and in two different roles:

  - $GR^e(I) \leq z(I)$ fixes the scaling factor;
  - $z(I) \leq 2\, GR^e(I)$ bounds the size of the dynamic programming table.

- A better heuristic in place of GR^e would shorten the table, not improve the guarantee.

Summary (1/2)

Worst case performance ratios:

- $r(GR) = +\infty$: no constant guarantee
- $r(GR^e) = 2$, proved with the LP relaxation and the split item, tight
- $r(H^2) = \frac{4}{3}$ and $r(H^\ell) = \frac{\ell+2}{\ell+1}$, tight, by two cases on the profit of the guessed items

Tightness:

- a small item of efficiency 2 and big items of efficiency 1 mislead every algorithm of the chapter

Summary (2/2)

Approximation schemes:

- $\{H^\varepsilon\}$, with $\ell(\varepsilon) = \min\{\max\{\lceil \frac{1}{\varepsilon} \rceil - 2, 2\}, n\}$, is a PTAS, but not an FPTAS: its running time $O(n^{\ell(\varepsilon)+1})$ is not polynomial in $\frac{1}{\varepsilon}$
- $\{FP^\varepsilon\}$ guarantees $\frac{1}{1-\varepsilon}$ in time $O(n^2/\varepsilon)$: it is an FPTAS

Absolute performance guarantee

Introduction: Absolute performance guarantee

- So far the guarantees were relative: a ratio between $z(I)$ and $\mathscr{A}(I)$
- An absolute guarantee would bound the difference $z(I) - \mathscr{A}(I)$ by a constant
- For the KP there is only a negative result: such an algorithm would be an exact algorithm

Only a negative result

- Unfortunately, there is only a negative result concerning absolute performance guarantees for the 0/1 knapsack problem.
- The following theorem says that if an approximation algorithm would have an absolute performance guarantee k for some fixed k, then this algorithm could be used to solve any instance of a 0/1 knapsack problem to optimality.
- Let $f(n, c)$ be a function describing the running time of an algorithm for the KP with n items and capacity c.

Theorem 1

Let $\mathscr{A}$ be an approximation algorithm for the KP with absolute performance guarantee k and running time in $O\big(f(n,c)\big)$ for any instance with n items and capacity c. Then $\mathscr{A}$ also computes the optimal solution of any such instance in $O(f(n,c))$ time.

The proof scales the profits by $k+1$:
an error of at most k then becomes an error smaller than 1.

Proof (1/3)

Consider an approximation algorithm $\mathscr{A}$ for the KP with absolute performance guarantee k and let I be given by

$$p_1, \ldots, p_n, \quad w_1, \ldots, w_n, \quad c.$$

We construct the new instance I' with

$$p'_1, \ldots, p'_n, \quad w'_1, \ldots, w'_n, \quad c'$$

such that

$$p'_j = (k+1)\, p_j, \quad w'_j = w_j, \quad \forall j \in \mathscr{I}, \quad \text{and} \quad c' = c.$$

- Both instances I and I' have exactly the same set of feasible solutions.
- Since the objective function value of a solution of I' is exactly $k+1$ times as large as the objective function value of the same solution for I, the two instances also have the same optimal solutions.

Proof (2/3)

Applying algorithm $\mathscr{A}$ to the instance I' yields a solution of value $\mathscr{A}(I')$ with

$$z(I') - \mathscr{A}(I') \leq k$$

using the absolute performance guarantee of $\mathscr{A}$.

Interpreting the output of $\mathscr{A}$ as a solution for I yields a feasible solution of value $\mathscr{A}(I)$, and

$$\mathscr{A}(I') = (k+1)\,\mathscr{A}(I) \qquad \text{and} \qquad z(I') = (k+1)\,z(I)$$

Inserting in the above inequality yields

$$(k+1)\,z(I) - (k+1)\,\mathscr{A}(I) \leq k \iff z(I) - \mathscr{A}(I) \leq \frac{k}{k+1}$$

Proof (3/3)

But since all profit values are integer, also $z(I) - \mathscr{A}(I)$ must be integer and hence this inequality implies

$$z(I) - \mathscr{A}(I) \leq \left\lfloor \frac{k}{k+1} \right\rfloor \Longrightarrow z(I) - \mathscr{A}(I) \leq 0 \Longrightarrow z(I) = \mathscr{A}(I)$$

Since the approximate solution of I' was computed in $O(f(n, c))$ running time an optimal solution of I could be constructed within the same time. □

What the theorem says

- This theorem indicates that an algorithm with fixed absolute performance guarantee for the KP would have the same unpleasant running time as an optimal algorithm.

It is very unlikely that an exact algorithm for KP with a running time polynomial in n exists.

- Hence, also approximation algorithms with fixed absolute performance guarantee are bound to have a pseudo-polynomial running time which is unpleasantly dependent on the magnitude of the input values.

Summary

Absolute performance guarantee:

- multiplying the profits by $k+1$ does not change the feasible and the optimal solutions
- an error of at most k on the scaled instance is an error of at most $\frac{k}{k+1} < 1$ on the original one, hence 0, since the profits are integer

Consequence:

- an algorithm with a fixed absolute guarantee for the KP solves it exactly, in the same running time

Typical mistakes

Introduction: Typical mistakes

- We collect the mistakes that are made most often with the algorithms of this chapter
- Check them before running an algorithm by hand, or before stating its guarantee

Observation 3: typical mistakes (1/2)

- ***Stopping the greedy algorithm at the split item.*** *GR skips the split item s but keeps scanning, and it may select items that follow s (as in the first execution of GR).*
- ***Confusing i_m and s.*** *In GR^e the item i_m is the item of maximum profit $p_{\max}$, not the split item s: the two coincide only by chance.*
- ***Forgetting to sort by efficiency.*** *All the algorithms of this chapter assume the items sorted by non-increasing $\frac{p_j}{w_j}$; the split item and the greedy solution depend on the order.*

Observation: typical mistakes (2/2)

- ***Calling H^ℓ an FPTAS.*** *Its running time $O(n^{\ell+1})$ has the exponent $\ell(\varepsilon) = \lceil \frac{1}{\varepsilon} \rceil - 2$ growing with $\frac{1}{\varepsilon}$: H^ε is a PTAS, not an FPTAS.*
- ***Scaling the weights.*** *The Basic-FPTAS algorithm rounds down the profits only; the weights and the capacity are untouched, so that the feasible solutions are the same.*
- ***Counting the whole table of the dynamic program.*** *The bound $O(n^2/\varepsilon)$ holds because the table is truncated at $\lfloor 2n/\varepsilon \rfloor$; the total scaled profit can be as large as $\frac{n^2}{\varepsilon}$.*

Summary

When running the algorithms:

- sort the items by efficiency; GR keeps scanning after the split item; i_m is not s

When stating the guarantees:

- H^{ε} is a PTAS, not an FPTAS
- the Basic-FPTAS scales the profits, never the weights, and truncates the table at $\lfloor 2n/\varepsilon \rfloor$

C++ implementation

Introduction: C++ implementation

- The algorithms of the chapter are implemented in C++, in the folder `KP_HEUR_CODE/src`
- We look at short excerpts of the codes, and we follow step by step the code of GR and of GR^e on two instances of the chapter
- In this section we use 0-based indexing for the items. The algorithmic logic is unchanged.

The greedy algorithm in C++

An excerpt of `GreedyKP`, the C++ implementation of the **greedy algorithm**:

```
for (int i = 0; i < n; ++i)
{
    x[i] = 0; // item i is initially not selected
}
int rc = c; // residual capacity

for (int i = 0; i < n; ++i)
{
    if (w[i] <= rc)
    {
        x[i] = 1; // select item i
        rc = rc - w[i]; // update residual capacity
    }
}
```

Execution of *GR* in C++ (1/3)

This is the step-by-step execution of the C++ implementation of *GR* on the KP instance of the split item example ($c = 9$, $n = 7$).

Initialization

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
	-	-	-	-	-	-	-

$rc = 9$

Iteration 1 – Item 0 is selected

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
	1	-	-	-	-	-	-

$rc = 7$

Iteration 2 – Item 1 is selected

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$	$i = 4$	$i = 5$	$i = 6$
	1	1	-	-	-	-	-

$rc = 4$

Execution of *GR* in C++ (2/3)

Iteration 3 – Item 2 is not selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	-	-	-	-

$rc = 4$

Iteration 4 – Item 3 is not selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	-	-	-

$rc = 4$

Iteration 5 – Item 4 is not selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	-	-

$rc = 4$

Execution of *GR* in C++ (3/3)

Iteration 6 – Item 5 is not selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	0	-

$rc = 4$

Iteration 7 – Item 6 is selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	0	1

$rc = 0$

The value computed is

$$z = 14$$

Execution of *GR* in C++ (1/2)

This is the step-by-step execution of the C++ implementation of *GR* on the KP instance with $c = 10$ and 4 items.

Initialization

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
	-	-	-	-

$rc = 10$

Iteration 1 – Item 0 is selected

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
	1	-	-	-

$rc = 8$

Iteration 2 – Item 1 is selected

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
	1	1	-	-

$rc = 5$

Execution of *GR* in C++ (2/2)

Iteration 3 – Item 2 is not selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	1	0	-

$rc = 5$

Iteration 4 – Item 3 is selected

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$
	1	1	0	1

$rc = 0$

The value computed is

$$z = 18$$

The extended greedy algorithm in C++

An excerpt of `ExtendedGreedyKP`, the C++ implementation of the **extended greedy algorithm**: after the call to `GreedyKP`, the best single item.

```
int im = 0; // best single item
int pm = p[0];
for (int i = 1; i < n; ++i)
{
    if (p[i] > pm)
    {
        im = i; // update best single item
        pm = p[i]; // update its profit
    }
}
```

Execution of GR^e in C++

This is the step-by-step execution of the C++ implementation of GR^e on the KP instance of the split item example. The solution computed by GR is

$x[i]$	$i=0$	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$	$i=6$
	1	1	0	0	0	0	1

- The value computed is $z = 14$.
- The maximum profit is $p_{\max} = 9$, and the item of maximum profit is (0-based indexing) $i_m = 3$.
- Since $14 \geq 9$, GR^e keeps the greedy solution.

Execution of GR^e in C++

This is the step-by-step execution of the C++ implementation of GR^e on the KP instance with $c = 10$ and 4 items. The solution computed by GR is

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
	1	1	0	1

- The value computed is $z = 18$; the maximum profit is $p_{\max} = 19$, and the item of maximum profit is (0-based indexing) $i_m = 2$.
- Since $18 < 19$, GR^e replaces the greedy solution with the best single-item feasible solution:

$x[i]$	$i = 0$	$i = 1$	$i = 2$	$i = 3$
	0	0	1	0

The updated value is $z = 19$.

The H^2 algorithm in C++

An excerpt of `H2KP`: it calls `ExtendedGreedyKP` on the residual instance of every feasible pair, exactly as the pseudo-code does, and it keeps the best solution found.

```
1  if (w[i] + w[k] > c) continue; // (i,k) is not in P
2  int cbar = c - w[i] - w[k]; // residual capacity
3  int pmin = (p[i] < p[k] ? p[i] : p[k]);
4  int m = 0; // residual set I(i,k)
5  for (int j = 0; j < n; ++j)
6      if (j != i && j != k && p[j] <= pmin
7          && w[j] <= cbar)
8      { idx[m] = j; pr[m] = p[j]; wr[m] = w[j]; m++; }
9  int zres = (m > 0) ?
10     ExtendedGreedyKP(m, cbar, wr, pr, xr) : 0;
11 int lbik = p[i] + p[k] + zres;
```

The H^2 algorithm in C++: remarks

- On the KP instance of the H^2 example, with $n = 7$ items and capacity $c = 12$, the function returns $lb = 20$ with $x = (1, 1, 0, 1, 0, 0, 0)$, i.e., the solution $S = \{1, 2, 4\}$ already found in that example, which is optimal for that instance.

- The two nested loops over i and k perform $O(n^2)$ iterations, and each iteration builds the residual instance and runs `ExtendedGreedyKP` in $O(n)$ time: this is the $O(n^3)$ running time of the proposition on the running time of H^2.

The H^{ℓ} algorithm and the DP by profits in C++

- `HlKP` implements the H^{ℓ} **algorithm**. The function `NextSubset` enumerates the subsets of exactly r items in lexicographic order; the preprocessing examines the subsets of at most $\ell - 1$ items, and each subset of exactly ℓ items is completed by `ExtendedGreedyKP` on its residual instance, as in the pseudo-code.

- `DPProfits` implements the **dynamic programming by profits**, the second DP algorithm of the chapter «DP for the 0/1 knapsack problem» of the module *Dynamic Programming*, used by the Basic-FPTAS algorithm.
  - With the 0-based indexing of the code, the cell `u[i][q]` stores the minimum total weight of a subset of the first i items of total profit exactly q, and an optimal solution is reconstructed backwards.
  - The parameter `qmax` truncates the table: only the profits $q \leq \min\{P, \texttt{qmax}\}$ are computed, where P is the total profit.

The Basic-FPTAS algorithm in C++

An excerpt of `BasicFPTAS`. The accuracy ε is passed as a fraction $\frac{\texttt{eps_num}}{\texttt{eps_den}}$, so that the scaled profits $\left\lfloor \frac{p_j}{K} \right\rfloor = \left\lfloor \frac{p_j \, n \, \texttt{eps_den}}{\texttt{eps_num} \, GR^e(I)} \right\rfloor$ are computed in exact integer arithmetic, without rounding errors.

```
1 int lb = ExtendedGreedyKP(n, c, w, p, x);
2 int* ps = new int[n]; // scaled profits p'
3 for (int j = 0; j < n; ++j)
4     ps[j] = (int)(((long long)p[j] * n * eps_den)
5                   / ((long long)eps_num * lb));
6 int qmax = (int)((2LL * n * eps_den) / eps_num);
7 DPProfits(n, c, w, ps, qmax, x); // scaled instance
```

The table of `DPProfits` is truncated at $\lfloor 2n/\varepsilon \rfloor$.

Summary

The codes:

- `GreedyKP`, `ExtendedGreedyKP`, `H2KP`, `H1KP`, `DPProfits` and `BasicFPTAS` follow the pseudo-codes, with 0-based indexing
- the step-by-step executions of GR and GR^e give the values 14 and 18, and GR^e gives 19 on the second instance

Tests:

- all the functions are in the folder `KP_HEUR_CODE/src`; the command `make test` compiles them and runs every example of this chapter and every instance of the exercises, checking the value found against the expected one

In summary

Introduction: In summary

- We collect the key ideas of the chapter on the approximation algorithms for the KP
- The dashboard gathers the bounds found so far on the instance of the split item example

Summary (1/2)

- GR has no constant guarantee, GR^e (the better of GR and the best single item) has $r(GR^e) = 2$, H^2 has $r(H^2) = \frac{4}{3}$ and H^ℓ has $r(H^\ell) = \frac{\ell+2}{\ell+1}$.
- $\{H^\varepsilon\}$ is a PTAS with running time $O(n^{\ell+1})$; the Basic-FPTAS algorithm rounds down the profits, solves the scaled instance exactly by dynamic programming, and is an FPTAS: it guarantees $\frac{1}{1-\varepsilon}$ in time $O(n^2/\varepsilon)$.
- No algorithm with a fixed absolute performance guarantee exists, unless it solves the KP exactly.

Summary (2/2)

Dashboard of the KP instance of the split item example ($c = 9$, $p = (6, 5, 8, 9, 6, 7, 3)$, $w = (2, 3, 6, 7, 5, 9, 4)$). The bounds on $z(I)$ found so far:

Source	Lower bound	Upper bound	From
Dantzig bound	–	16	$\lfloor \frac{49}{3} \rfloor$, with $s = 3$ and $rc = 4$
This chapter, GR and GR^e	14	–	$S = \{1, 2, 7\}$
This chapter, H^2 and H^3	15	–	$S = \{1, 4\}$
This chapter, FP^ε with $\varepsilon = \frac{1}{2}$	15	–	$K = 1$, the scaled instance is the original one
Best bounds	15	16	$\mathrm{UB} - \mathrm{LB} = 1$; the optimal value is 15