Bin Packing

Given n items with sizes $s_1, \ldots, s_n$ where

$$1 > s_1 \geq \cdots \geq s_n > 0 .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

Theorem 5
There is no ρ-approximation for Bin Packing with $\rho < 3/2$ unless $P = NP$.

Bin Packing

Given n items with sizes $s_1, \ldots, s_n$ where

$$1 > s_1 \geq \cdots \geq s_n > 0 .$$

Pack items into a minimum number of bins where each bin can hold items of total size at most 1.

Theorem 5
There is no ρ-approximation for Bin Packing with $\rho < 3/2$ unless $P = NP$.

Bin Packing

Proof

▶ In the partition problem we are given positive integers
$b_1, \ldots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers
into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking
whether we can pack the resulting items into 2 bins or not.

▶ A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3
or more bins when 2 are optimal.

▶ Hence, such an algorithm can solve Partition.

Bin Packing

Proof

▶ In the partition problem we are given positive integers $b_1, \ldots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.

▶ A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.

▶ Hence, such an algorithm can solve Partition.

Bin Packing

Proof

▶ In the partition problem we are given positive integers $b_1, \ldots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.

▶ A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.

▶ Hence, such an algorithm can solve Partition.

Bin Packing

Proof

▶ In the partition problem we are given positive integers $b_1, \ldots, b_n$ with $B = \sum_i b_i$ even. Can we partition the integers into two sets S and T s.t.

$$\sum_{i \in S} b_i = \sum_{i \in T} b_i \quad ?$$

▶ We can solve this problem by setting $s_i := 2b_i/B$ and asking whether we can pack the resulting items into 2 bins or not.

▶ A ρ-approximation algorithm with $\rho < 3/2$ cannot output 3 or more bins when 2 are optimal.

▶ Hence, such an algorithm can solve Partition.

Bin Packing

Definition 6
An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1 + \epsilon)\mathrm{OPT} + c$ for minimization problems.

Bin Packing

Definition 6
An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1 + \epsilon)\mathrm{OPT} + c$ for minimization problems.

- ▶ Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.

- ▶ However, we will develop an APTAS for Bin Packing.

Bin Packing

Definition 6
An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithms $\{A_\epsilon\}$ along with a constant c such that A_ϵ returns a solution of value at most $(1 + \epsilon)\text{OPT} + c$ for minimization problems.

- ▶ Note that for Set Cover or for Knapsack it makes no sense to differentiate between the notion of a PTAS or an APTAS because of scaling.

- ▶ However, we will develop an APTAS for Bin Packing.

Bin Packing

Again we can differentiate between small and large items.

Lemma 7
Any packing of items of size at most γ into ℓ bins can be extended to a packing of all items into $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

Bin Packing

Again we can differentiate between small and large items.

Lemma 7

Any packing of items of size at most γ into ℓ bins can be extended to a packing of all items into $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

▶ If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.

▶ Hence, $r(1 - \gamma) \leq \text{SIZE}(I)$ where r is the number of nearly-full bins.

▶ This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 7

Any packing of items of size at most γ into ℓ bins can be extended to a packing of all items into $\max\{\ell, \frac{1}{1-\gamma}\mathrm{SIZE}(I) + 1\}$ bins, where $\mathrm{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.

- Hence, $r(1 - \gamma) \leq \mathrm{SIZE}(I)$ where r is the number of nearly-full bins.

- This gives the lemma.

Bin Packing

Again we can differentiate between small and large items.

Lemma 7
Any packing of items of size at most γ into ℓ bins can be extended to a packing of all items into $\max\{\ell, \frac{1}{1-\gamma}\text{SIZE}(I) + 1\}$ bins, where $\text{SIZE}(I) = \sum_i s_i$ is the sum of all item sizes.

- ▶ If after Greedy we use more than ℓ bins, all bins (apart from the last) must be full to at least $1 - \gamma$.

- ▶ Hence, $r(1 - \gamma) \leq \text{SIZE}(I)$ where r is the number of nearly-full bins.

- ▶ This gives the lemma.

Choose $y = \epsilon/2$. Then we either use ℓ bins or at most

$$\frac{1}{1 - \epsilon/2} \cdot \text{OPT} + 1 \leq (1 + \epsilon) \cdot \text{OPT} + 1$$

bins.

It remains to find an algorithm for the large items.

Bin Packing

Linear Grouping:

Generate an instance I' (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

Bin Packing

Linear Grouping:

Generate an instance I' (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

Bin Packing

Linear Grouping:
Generate an instance I' (for large items) as follows.

- Order large items according to size.
- Let the first k items belong to group 1; the following k items belong to group 2; etc.
- Delete items in the first group;
- Round items in the remaining groups to the size of the largest item in the group.

Bin Packing

Linear Grouping:

Generate an instance I' (for large items) as follows.

- ▶ Order large items according to size.
- ▶ Let the first k items belong to group 1; the following k items belong to group 2; etc.
- ▶ Delete items in the first group;
- ▶ Round items in the remaining groups to the size of the largest item in the group.

Lemma 8

$\text{OPT}(I') \le \text{OPT}(I) \le \text{OPT}(I') + k$

Lemma 8

$\text{OPT}(I') \le \text{OPT}(I) \le \text{OPT}(I') + k$

Proof 1:

▶ Any bin packing for I gives a bin packing for I' as follows.

▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;

▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;

▶ . . .

Lemma 8

$\mathrm{OPT}(I') \le \mathrm{OPT}(I) \le \mathrm{OPT}(I') + k$

Proof 1:

▶ Any bin packing for I gives a bin packing for I' as follows.

▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;

▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;

▶ . . .

Lemma 8
$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$

Proof 1:

- ▶ Any bin packing for I gives a bin packing for I' as follows.
- ▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;
- ▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;
- ▶ ...

Lemma 8

$\text{OPT}(I') \le \text{OPT}(I) \le \text{OPT}(I') + k$

Proof 1:

▶ Any bin packing for I gives a bin packing for I' as follows.

▶ Pack the items of group 2, where in the packing for I the items for group 1 have been packed;

▶ Pack the items of groups 3, where in the packing for I the items for group 2 have been packed;

▶ ...

Lemma 9

$\mathrm{OPT}(I') \leq \mathrm{OPT}(I) \leq \mathrm{OPT}(I') + k$

Proof 2:

- ▶ Any bin packing for I' gives a bin packing for I as follows.

- ▶ Pack the items of group 1 into k new bins;

- ▶ Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;

- ▶ . . .

Lemma 9
$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$

Proof 2:

- Any bin packing for I' gives a bin packing for I as follows.
- Pack the items of group 1 into k new bins;
- Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ...

Lemma 9

$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$

Proof 2:

- Any bin packing for I' gives a bin packing for I as follows.
- Pack the items of group 1 into k new bins;
- Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ...

Lemma 9

$\text{OPT}(I') \leq \text{OPT}(I) \leq \text{OPT}(I') + k$

Proof 2:

- ▶ Any bin packing for I' gives a bin packing for I as follows.
- ▶ Pack the items of group 1 into k new bins;
- ▶ Pack the items of groups 2, where in the packing for I' the items for group 2 have been packed;
- ▶ . . .

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1 + \epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1 + \epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon \mathrm{SIZE}(I) \leq (1 + \epsilon)\mathrm{OPT}(I)$$

- running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\mathrm{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \mathrm{SIZE}(I) \rfloor$.

Then $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes ($4/\epsilon^2$) and at most a constant number ($2/\epsilon$) can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

▸ cost (for large items) at most

$$\mathrm{OPT}(I') + k \leq \mathrm{OPT}(I) + \epsilon\mathrm{SIZE}(I) \leq (1 + \epsilon)\mathrm{OPT}(I)$$

▸ running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

▶ cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon)\text{OPT}(I)$$

▶ running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.

Assume that our instance does not contain pieces smaller than $\epsilon/2$. Then $\text{SIZE}(I) \geq \epsilon n/2$.

We set $k = \lfloor \epsilon \text{SIZE}(I) \rfloor$.

Then $n/k \leq 2n/\lfloor \epsilon^2 n/2 \rfloor \leq 4/\epsilon^2$ (here we used $\lfloor \alpha \rfloor \geq \alpha/2$ for $\alpha \geq 1$).

Hence, after grouping we have a constant number of piece sizes $(4/\epsilon^2)$ and at most a constant number $(2/\epsilon)$ can fit into any bin.

We can find an optimal packing for such instances by the previous Dynamic Programming approach.

- ► cost (for large items) at most

$$\text{OPT}(I') + k \leq \text{OPT}(I) + \epsilon \text{SIZE}(I) \leq (1 + \epsilon)\text{OPT}(I)$$

- ► running time $\mathcal{O}((\frac{2}{\epsilon}n)^{4/\epsilon^2})$.