Design and Analysis of Algorithms (XX)

Polynomial Time Approximation Scheme

Guoqiang Li
School of Software

SHANGHAI JIAO TONG UNIVERSITY

Polynomial Time Approximation Scheme

Approximation Scheme

Let Π be an $\mathbf{NP}$-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 $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

Approximation Scheme

Let Π be an $\mathbf{NP}$-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 $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

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

PTAS and FPTAS

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

PTAS and FPTAS

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

If we require that the running time of $\mathcal{A}$ be bounded by a polynomial in the size of instance I and $1/\epsilon$, then $\mathcal{A}$ will be said to be a fully polynomial-time approximation scheme, abbreviated FPTAS.

Knapsack

Knapsack: Problem Statement

KNAPSACK

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

An Example

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

Knapsack size: B

Greedy is Bad

Greedy is Bad

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.

Greedy is Bad

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.

Greedy is Bad

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.

$$100/1,\ (100 * B - 1)/B$$

Some Concepts and Notations

Some Concepts and Notations

For any optimization problem Π, an instance consists of objects, such as sets or graphs, and numbers, such as cost, profit, size, etc.

Some Concepts and Notations

For any optimization problem Π, an instance consists of objects, such as sets or graphs, and numbers, such as cost, profit, size, etc.

We assume that all numbers occurring in a problem instance I are written in binary.

Some Concepts and Notations

For any optimization problem Π, an instance consists of objects, such as sets or graphs, and numbers, such as cost, profit, size, etc.

We assume 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.

Some Concepts and Notations

For any optimization problem Π, an instance consists of objects, such as sets or graphs, and numbers, such as cost, profit, size, etc.

We assume 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.

Some Concepts and Notations

For any optimization problem Π, an instance consists of objects, such as sets or graphs, and numbers, such as cost, profit, size, etc.

We assume 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.

Pseudo-Polynomial Time Algorithm

Pseudo-Polynomial Time Algorithm

An algorithm for problem Π is said to be efficient if its running time on instance I is bounded by a polynomial in $|I|$.

Pseudo-Polynomial Time Algorithm

An algorithm for problem Π is said to be efficient if its running time on instance I is bounded by a polynomial in $|I|$.

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.

Dynamic Programming

Dynamic Programming

Knapsack with Repetition

$$K(w) = \max_{a_i : \text{size}(a_i) \leq w} \{K(w - \text{size}(a_i)) + \text{profit}(a_i)\}$$

The running time is $O(n \cdot B)$.

Dynamic Programming

Knapsack with Repetition

$$K(w) = \max_{a_i : \text{size}(a_i) \le w} \{K(w - \text{size}(a_i)) + \text{profit}(a_i)\}$$

The running time is $O(n \cdot B)$.

Knapsack without Repetition

$$K(w, j) = \max\{K(w - \text{size}(a_j), j - 1) + \text{profit}(a_j), K(w, j - 1)\}$$

The running time is $O(n \cdot B)$.

Dynamic Programming

Dynamic Programming

Let P be the profit of the most profitable object, i.e.,

$$P = \max_{a \in S} \text{profit}(a)$$

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 upper bound on the profit that can be achieved by any solution.

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 upper bound 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.

Dynamic Programming

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

$A(1,p)$ is known for every $p \in \{1, \ldots, nP\}$.

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p)=\infty$ if no such set exists).

$A(1,p)$ is known for every $p \in \{1,\ldots,nP\}$.

The following recurrence helps compute all values $A(i,p)$ in $O(n^2P)$ time:

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

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

$A(1,p)$ is known for every $p \in \{1,\ldots,nP\}$.

The following recurrence helps compute all values $A(i,p)$ in $O(n^2P)$ time:

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

The maximum profit achievable by objects of total size bounded by B is $\max\{p \mid A(n,p) \le B\}$.

An Example

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

Knapsack size: B

An Example

	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	∞	∞	∞	∞

An FPTAS for Knapsack

An FPTAS for Knapsack

If the profits of objects were small numbers, say, bounded by a polynomial in n, then the algorithm would be a regular polynomial time algorithm, since its running time would be bounded by a polynomial in $|I|$.

An FPTAS for Knapsack

If the profits of objects were small numbers, say, bounded by a polynomial in n, then the algorithm would be a regular polynomial time algorithm, since its running time would be bounded by a polynomial in $|I|$.

In FPTAS we will ignore a certain number of least significant bits of profits of objects (depending on ϵ), so that the modified profits can be viewed as numbers bounded by a polynomial in n and $1/\epsilon$.

An FPTAS for Knapsack

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let

$$K = \frac{\epsilon P}{n}$$

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let

$$K = \frac{\epsilon P}{n}$$

2. For each object a_i, define

$$\mathrm{profit}'(a_i) = \lfloor \frac{\mathrm{profit}(a_i)}{K} \rfloor$$

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let

$$K = \frac{\epsilon P}{n}$$

2. For each object a_i, define

$$\text{profit}'(a_i) = \lfloor \frac{\text{profit}(a_i)}{K} \rfloor$$

3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set, say S'.

An FPTAS for Knapsack

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

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\operatorname{profit}(A) \geq (1-\epsilon) \cdot \mathrm{OPT}.$$

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\mathrm{profit}(A) \geq (1-\epsilon)\cdot \mathrm{OPT}.$$

Proof.

Analysis

Lemma

Let A denote the set output by the algorithm. Then

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

Proof.

Let O denote the optimal set.

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\mathrm{profit}(A) \geq (1-\epsilon)\cdot \mathrm{OPT}.$$

Proof.

Let O denote the optimal set.

For any object a,

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\mathrm{profit}(A) \geq (1-\epsilon)\cdot \mathrm{OPT}.$$

Proof.

Let O denote the optimal set.

For any object a,

- because of rounding down, $K \cdot \mathrm{profit}'(a)$ can be smaller than $\mathrm{profit}(a)$,

Analysis

Lemma

Let A denote the set output by the algorithm. Then

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

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. Say, $\text{profit}(a) - K \cdot \text{profit}'(a) \leq K$

Analysis

Lemma

Let A denote the set output by the algorithm. Then

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

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. Say, $\text{profit}(a) - K \cdot \text{profit}'(a) \leq K$

Therefore,

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

Analysis

Lemma

Let A denote the set output by the algorithm. Then

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

Proof.

Analysis

Lemma

Let A denote the set output by the algorithm. Then

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

Proof.

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

Analysis

Lemma

Let A denote the set output by the algorithm. Then

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

Proof.

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

Therefore,

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

Analysis

Analysis

By previous Lemma, the solution found is within $(1 - \epsilon)$ factor of OPT. Since the running time of the algorithm is

$$O(n^2 \lfloor \frac{P}{K} \rfloor) = O(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$$

which is polynomial in n and $1/\epsilon$, thus it is a FPTAS for knapsack.

Bin Packing

Bin Packing: Problem Statement

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.

An 2-approximation Algorithm

First-Fit Algorithm

An 2-approximation Algorithm

First-Fit Algorithm

- Consider items in arbitrary order.

An 2-approximation Algorithm

First-Fit Algorithm

- Consider items in arbitrary order.
- In the i-th step, it has a list of partially packed bins, say $B_1, \ldots, B_k$.

An 2-approximation Algorithm

First-Fit Algorithm

- Consider items in arbitrary order.
- In the i-th 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.

An 2-approximation Algorithm

First-Fit Algorithm

- Consider items in arbitrary order.
- In the i-th 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.

Analysis

Analysis

If the algorithm uses m bins, then at least $m-1$ bins are more than half full.

Analysis

If the algorithm uses m bins, then at least $m-1$ bins are more than half full.

Therefore,

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

Analysis

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, $m-1 < 2\cdot\mathrm{OPT}$, i.e., $m \le 2\cdot\mathrm{OPT}$.

A Hardness Result

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

A Hardness Result

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

Proof.

A Hardness Result

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

Proof.

If there were such an algorithm, then the NPC 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 $1/2 \sum_i a_i$.

A Hardness Result

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

Proof.

If there were such an algorithm, then the NPC 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 $1/2 \sum_i a_i$.

The answer to this question is “yes” iff the n items can be packed in 2 bins of size $1/2 \sum_i a_i$.

A Hardness Result

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

Proof.

If there were such an algorithm, then the NPC 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 $1/2 \sum_i a_i$.

The answer to this question is “yes” iff the n items can be packed in 2 bins of size $1/2 \sum_i a_i$.

If the answer is “yes” the $3/2 - \epsilon$ factor algorithm will have to give an optimal packing.

APTAS

APTAS

An asymptotic polynomial-time approximation scheme (APTAS) is a family of algorithm $\{A_\epsilon\}$ along with a constant c where there is an algorithm A_ϵ for each $\epsilon > 0$ such that A_ϵ returns a solution of value at most $(1+\epsilon)\mathrm{OPT} + c$ for minimization problems.

An APTAS for Bin-Packing

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

An APTAS for Bin-Packing

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

We will introduce the algorithm in three steps.

Instances with Large Items

Lemma

Let $\epsilon > 0$ 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.

Instances with Large Items

Proof.

Instances with Large Items

Proof.

The number of items in a bin is bounded by $\lfloor 1/\epsilon \rfloor$. Denote this by M. Therefore, the number of different bin types is bounded by

$$R = \binom{M+K}{M}$$

which is a large constant.

Instances with Large Items

Proof.

The number of items in a bin is bounded by $\lfloor 1/\epsilon \rfloor$. Denote this by M. Therefore, the number of different bin types is bounded by

$$R = \binom{M+K}{M}$$

which is a large constant.

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.

Instances with Large Items

Proof.

The number of items in a bin is bounded by $\lfloor 1/\epsilon \rfloor$. Denote this by M. Therefore, the number of different bin types is bounded by

$$R = \binom{M+K}{M}$$

which is a large constant.

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.

Enumerating them and picking the best packing gives the optimal answer.

k Composition of M

$$x_1 + x_2 + \ldots + x_k = M$$

k Composition of M

$$x_1 + x_2 + \ldots + x_k = M$$

- k composition of M: $x_i \geq 1$

k Composition of M

$$x_1 + x_2 + \ldots + x_k = M$$

- k composition of M: $x_i \geq 1$

$$\binom{M-1}{k-1}$$

k Composition of M

$$x_1 + x_2 + \ldots + x_k = M$$

- k composition of M: $x_i \geq 1$

$$\binom{M-1}{k-1}$$

- weak k composition of M: $x_i \geq 0$

k Composition of M

$$x_1 + x_2 + \ldots + x_k = M$$

- k composition of M: $x_i \geq 1$

$$\binom{M-1}{k-1}$$

- weak k composition of M: $x_i \geq 0$

$$\binom{M+k-1}{k-1}$$

Removing the Restriction of K

Lemma

Let $\epsilon > 0$ 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 $(1 + \epsilon)$.

Removing the Restriction of K

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

Removing the Restriction of K

Removing the Restriction of K

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.

Removing the Restriction of K

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.

Then we can find an optimal packing for J, this will also be a valid packing for the original item size.

Removing the Restriction of K

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.

Then we can find an optimal packing for J, this will also be a valid packing for the original item size.

We will show that

$$\mathrm{OPT}(J) \leq (1+\epsilon)\mathrm{OPT}(I)$$

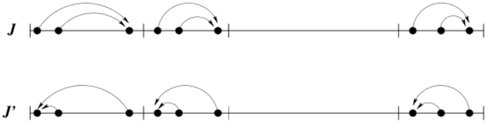
Removing the Restriction of K

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.

Then we can find an optimal packing for J, this will also be a valid packing for the original item size.

We will show that

$$\mathrm{OPT}(J) \leq (1+\epsilon)\mathrm{OPT}(I)$$



Proof

Proof

Let us construct another instance, say J' , by rounding down the size of each item to that of the smallest item in its group.

Proof

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 $\text{OPT}(J') \le \text{OPT}(I)$.

Proof

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. Therefore,

Proof

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. Therefore,

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

Proof

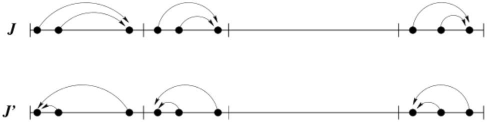
Let us construct another instance, say J', by rounding down the size of each item to that of the smallest item in its group.

Clearly $\mathrm{OPT}(J') \le \mathrm{OPT}(I)$.

The crucial observation is that a packing for instance J yields a packing for all but the largest Q items of instance J. Therefore,

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

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



The Algorithm

Now we present the APTAS algorithm for Bin-Packing.

The Algorithm

Now we present the APTAS algorithm for Bin-Packing.

- Let I denote the given instance, and I' denote the instance obtained by discarding items of size $< \epsilon$ from I.

The Algorithm

Now we present the APTAS algorithm for Bin-Packing.

- Let I denote the given instance, and I' denote the instance obtained by discarding items of size $< \epsilon$ from I.
- By previous lemma, we can find a packing for I' using at most $(1+\epsilon)\mathrm{OPT}(I')$ bins.

The Algorithm

Now we present the APTAS algorithm for Bin-Packing.

- Let I denote the given instance, and I' denote the instance obtained by discarding items of size $< \epsilon$ from I.
- By previous lemma, we can find a packing for I' using at most $(1+\epsilon)\mathrm{OPT}(I')$ bins.
- Next, we start packing the small items (of size $< \epsilon$) 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.

Analysis

Analysis

If no additional bins are needed, then we have a packing in $(1+\epsilon)\mathrm{OPT}(I') \le (1+\epsilon)\mathrm{OPT}(I)$ bins.

Analysis

If no additional bins are needed, then we have a packing in $(1+\epsilon)\mathrm{OPT}(I') \le (1+\epsilon)\mathrm{OPT}(I)$ 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 $1-\epsilon$.

Analysis

If no additional bins are needed, then we have a packing in $(1+\epsilon)\mathrm{OPT}(I') \leq (1+\epsilon)\mathrm{OPT}(I)$ 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 $1-\epsilon$.

Therefore, the sum of the item sizes in I is at least $(M-1)(1-\epsilon)$. Since this is a lower bound on OPT, we get

Analysis

Analysis

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

Analysis

$$M \le \frac{\mathrm{OPT}}{(1-\epsilon)} + 1 \le (1+2\epsilon)\mathrm{OPT} + 1$$

where we have used the assumption that $\epsilon \le 1/2$.

Analysis

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

where we have used the assumption that $\epsilon \leq 1/2$.

Hence, for each value of ϵ, $0 < \epsilon \leq 1/2$, we have a polynomial time algorithm achieving a guarantee of $(1+2\epsilon)\mathrm{OPT} + 1$.

Summary of Algorithm

Summary of Algorithm

Algorithm A_ϵ is summarized below.

Summary of Algorithm

Algorithm A_ϵ is summarized below.

Algorithm

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

Referred Materials

Referred Materials

Content of this lecture comes from Chapter 8 and 9 in [Vaz04], and Section 3.3 in [WS11].

Suggest to read Chapter 10 in [Vaz04] and Chapter 3 in [WS11].