Approximation Algorithms (ADM III)
3- Rounding Data and Dynamic Programming

Guillaume Sagnol

Technische Universität Berlin

Outline

Knapsack Problem

Given: n items $I = \{1, \ldots, n\}$, values $v_i \in \mathbb{Z}_{>0}$ and sizes $s_i \in \mathbb{Z}_{>0}$, $i \in I$;
knapsack of size $B \in \mathbb{Z}_{>0}$ (assume w.l.o.g. $s_i \leq B$, for all $i \in I$)

Task: find subset of items $S \subseteq I$ with $\sum_{i \in S} s_i \leq B$ maximizing $\sum_{i \in S} v_i$.

KS.png

Knapsack Problem

Given: n items $I = \{1, \ldots, n\}$, values $v_i \in \mathbb{Z}_{>0}$ and sizes $s_i \in \mathbb{Z}_{>0}$, $i \in I$;
knapsack of size $B \in \mathbb{Z}_{>0}$ (assume w.l.o.g. $s_i \leq B$, for all $i \in I$)

Task: find subset of items $S \subseteq I$ with $\sum_{i \in S} s_i \leq B$ maximizing $\sum_{i \in S} v_i$.

KS.png

Remarks

- The Knapsack Problem is *NP*-hard (reduction from Partition).
- It can be solved in pseudo-polynomial time by dynamic

Dynamic Program for Knapsack Problem (1/3)

- Denote by $J(i,b)$ the maximum value that can be packed in a knapsack of capacity $b \leq B$, using only a subset of items $T \subseteq \{1,\ldots,i\}$:

$$J(i,b) = \max\Big\{v(T) : s(T) \leq b,\ T \subseteq \{1,\ldots,i\}\Big\}.$$

Dynamic Program for Knapsack Problem (1/3)

- Denote by $J(i,b)$ the maximum value that can be packed in a knapsack of capacity $b \leq B$, using only a subset of items $T \subseteq \{1,\ldots,i\}$:

$$J(i,b) = \max\Big\{v(T) : s(T) \leq b,\ T \subseteq \{1,\ldots,i\}\Big\}.$$

- The values of $J(i,b)$ can be computed recursively:

$$J(i,b) = \begin{cases} 0 & \text{if } i = 0; \\ J(i-1,b) & \text{if } s_i > b; \\ \max(J(i-1,b), J(i-1,b-s_i)+v_i) & \text{otherwise.} \end{cases}$$

Dynamic Program for Knapsack Problem (1/3)

- Denote by $J(i, b)$ the maximum value that can be packed in a knapsack of capacity $b \leq B$, using only a subset of items $T \subseteq \{1, \ldots, i\}$:

$$J(i, b) = \max \Big\{ v(T) : s(T) \leq b,\ T \subseteq \{1, \ldots, i\} \Big\}.$$

- The values of $J(i, b)$ can be computed recursively:

$$J(i, b) = \begin{cases} 0 & \text{if } i = 0; \\ J(i-1, b) & \text{if } s_i > b; \\ \max(J(i-1, b), J(i-1, b-s_i) + v_i) & \text{otherwise.} \end{cases}$$

- Optimal solution $\texttt{OPT} = J(n, B)$ can be computed by “filling” the table. Hence, the complexity of this algorithm is $O(nB)$.

Dynamic Program for Knapsack Problem (1/3)

- Denote by $J(i,b)$ the maximum value that can be packed in a knapsack of capacity $b \leq B$, using only a subset of items $T \subseteq \{1,\ldots,i\}$:

$$J(i,b) = \max \Big\{ v(T) : s(T) \leq b,\ T \subseteq \{1,\ldots,i\} \Big\}.$$

- The values of $J(i,b)$ can be computed recursively:

$$J(i,b) = \begin{cases} 0 & \text{if } i = 0; \\ J(i-1,b) & \text{if } s_i > b; \\ \max(J(i-1,b), J(i-1,b-s_i)+v_i) & \text{otherwise.} \end{cases}$$

- Optimal solution $\texttt{OPT} = J(n,B)$ can be computed by “filling” the table. Hence, the complexity of this algorithm is $O(nB)$.

- However, this is not a polytime algorithm, as the input can be described with only $\langle B \rangle := \log_2 B$ bits.

Dynamic Program for Knapsack Problem (2/3)

- To construct a polynomial time approximation algorithm, we need another dynamic program, that enumerates the different values $w \leq V = \sum_{i=1}^{n} v_i$ that a knapsack can achieve.

Dynamic Program for Knapsack Problem (2/3)

- To construct a polynomial time approximation algorithm, we need another dynamic program, that enumerates the different values $w \leq V = \sum_{i=1}^{n} v_i$ that a knapsack can achieve.
- Let $G(i, w)$ denote the minimum capacity of a knapsack that can hold a value at least w, using only a subset of items $T \subseteq \{1, \ldots, i\}$:

$$G(i, w) = \min \Big\{ s(T) : v(T) \geq w,\ T \subseteq \{1, \ldots, i\} \Big\}.$$

Dynamic Program for Knapsack Problem (2/3)

- To construct a polynomial time approximation algorithm, we need another dynamic program, that enumerates the different values $w \leq V = \sum_{i=1}^n v_i$ that a knapsack can achieve.
- Let $G(i, w)$ denote the minimum capacity of a knapsack that can hold a value at least w, using only a subset of items $T \subseteq \{1, \ldots, i\}$:

$$G(i, w) = \min \Big\{ s(T) : v(T) \geq w,\ T \subseteq \{1, \ldots, i\} \Big\}.$$

- As before, there is a recursive formula to compute the $G(i, w)$'s, and `OPT` is the largest w such that $G(n, w) \leq B$. The time complexity of this algorithm is $O(nV)$.

Dynamic Program for Knapsack Problem (2/3)

- To construct a polynomial time approximation algorithm, we need another dynamic program, that enumerates the different values $w \leq V = \sum_{i=1}^{n} v_i$ that a knapsack can achieve.
- Let $G(i, w)$ denote the minimum capacity of a knapsack that can hold a value at least w, using only a subset of items $T \subseteq \{1, \ldots, i\}$:

$$G(i, w) = \min \Big\{ s(T) : v(T) \geq w,\ T \subseteq \{1, \ldots, i\} \Big\}.$$

- As before, there is a recursive formula to compute the $G(i, w)$'s, and `OPT` is the largest w such that $G(n, w) \leq B$. The time complexity of this algorithm is $O(nV)$.
- In fact, we can make the best of both worlds and get a DP of complexity $O(n \min(B, V))$, by using a notion of dominance.

Dynamic Program for Knapsack Problem (2/3)

- To construct a polynomial time approximation algorithm, we need another dynamic program, that enumerates the different values $w \leq V = \sum_{i=1}^{n} v_i$ that a knapsack can achieve.
- Let $G(i, w)$ denote the minimum capacity of a knapsack that can hold a value at least w, using only a subset of items $T \subseteq \{1, \ldots, i\}$:

$$G(i, w) = \min \Big\{ s(T) : v(T) \geq w,\ T \subseteq \{1, \ldots, i\} \Big\}.$$

- As before, there is a recursive formula to compute the $G(i, w)$'s, and `OPT` is the largest w such that $G(n, w) \leq B$. The time complexity of this algorithm is $O(nV)$.
- In fact, we can make the best of both worlds and get a DP of complexity $O(n \min(B, V))$, by using a notion of dominance.
- We identify a subset of items T with a pair $(t, w) = (s(T), v(T))$. We say that a pair (t_1, w_1) is dominated by (t_2, w_2) if $t_1 \geq t_2$ and $w_1 \leq w_2$, and $(t_1, w_1) \neq (t_2, w_2)$.

Dynamic Program for Knapsack Problem (3/3)

For $j = 0, 1, \ldots, n$ let $A(j)$ denote the set of feasible non-dominated pairs given by all subsets $S \subseteq \{1, \ldots, j\}$.

Dynamic Program for Knapsack Problem (3/3)

For $j = 0, 1, \ldots, n$ let $A(j)$ denote the set of feasible non-dominated pairs given by all subsets $S \subseteq \{1, \ldots, j\}$.

$A(0) := \{(0, 0)\}$;
for $j = 1, \ldots, n$ let $A(j) := A(j-1)$;
    for each $(t, w) \in A(j-1)$
        if $t + s_j \leq B$ then add $(t + s_j, w + v_j)$ to $A(j)$;
    remove dominated pairs from $A(j)$;
return $\max\{w : (t, w) \in A(n)\}$;

Dynamic Program for Knapsack Problem (3/3)

For $j = 0, 1, \ldots, n$ let $A(j)$ denote the set of feasible non-dominated pairs given by all subsets $S \subseteq \{1, \ldots, j\}$.

$A(0) := \{(0, 0)\}$;
for $j = 1, \ldots, n$ let $A(j) := A(j-1)$;
    for each $(t, w) \in A(j-1)$
        if $t + s_j \leq B$ then add $(t + s_j, w + v_j)$ to $A(j)$;
    remove dominated pairs from $A(j)$;
return $\max\{w : (t, w) \in A(n)\}$;

Theorem 3.1

The dynamic program correctly computes the optimal value of the knapsack problem in $O(n \cdot \min\{B, V\})$ time where $V := \sum_{i=1}^{n} v_i$.

Dynamic Program for Knapsack Problem (3/3)

For $j = 0, 1, \ldots, n$ let $A(j)$ denote the set of feasible non-dominated pairs given by all subsets $S \subseteq \{1, \ldots, j\}$.

1. $A(0) := \{(0, 0)\}$;
2. for $j = 1, \ldots, n$ let $A(j) := A(j-1)$;
3.   for each $(t, w) \in A(j-1)$
4.   if $t + s_j \leq B$ then add $(t + s_j, w + v_j)$ to $A(j)$;
5.   remove dominated pairs from $A(j)$;
6. return $\max\{w : (t, w) \in A(n)\}$;

Theorem 3.1

The dynamic program correctly computes the optimal value of the knapsack problem in $O(n \cdot \min\{B, V\})$ time where $V := \sum_{i=1}^{n} v_i$.

Remark: An optimal subset $S \subseteq I$ can be obtained by backtracking.

FPTAS for the Knapsack Problem

Definition 3.2 (FPTAS)

A fully polynomial-time approximation scheme is a PTAS $(A_\varepsilon)_{\varepsilon>0}$ such that the running time of A_ε is bounded by a polynomial in $1/\varepsilon$.

FPTAS for the Knapsack Problem

Definition 3.2 (FPTAS)

A fully polynomial-time approximation scheme is a PTAS $(A_\varepsilon)_{\varepsilon>0}$ such that the running time of A_ε is bounded by a polynomial in $1/\varepsilon$.

FPTAS for Knapsack Problem

1. let $M := \max_{i \in I} v_i$; $\quad \mu := \varepsilon \cdot M/n$; $\quad v_i' := \lfloor v_i/\mu \rfloor$ for $i \in I$;
2. solve knapsack instance with values v_i' by dynamic programming;

FPTAS for the Knapsack Problem

Definition 3.2 (FPTAS)

A fully polynomial-time approximation scheme is a PTAS $(A_\varepsilon)_{\varepsilon>0}$ such that the running time of A_ε is bounded by a polynomial in $1/\varepsilon$.

FPTAS for Knapsack Problem

1. let $M := \max_{i \in I} v_i$; $\mu := \varepsilon \cdot M/n$; $v_i' := \lfloor v_i/\mu \rfloor$ for $i \in I$;
2. solve knapsack instance with values v_i' by dynamic programming;

Theorem 3.3

The algorithm above is a fully polynomial-time approximation scheme for the Knapsack Problem.

FPTAS for the Knapsack Problem

Definition 3.2 (FPTAS)

A fully polynomial-time approximation scheme is a PTAS $(A_\varepsilon)_{\varepsilon>0}$ such that the running time of A_ε is bounded by a polynomial in $1/\varepsilon$.

FPTAS for Knapsack Problem

1. let $M := \max_{i \in I} v_i;\quad \mu := \varepsilon \cdot M/n;\quad v_i' := \lfloor v_i/\mu \rfloor$ for $i \in I$;
2. solve knapsack instance with values v_i' by dynamic programming;

Theorem 3.3

The algorithm above is a fully polynomial-time approximation scheme for the Knapsack Problem.

Proof:… □

Outline

Scheduling Jobs on Identical Parallel Machines

Given: n jobs $j = 1, \dots, n$ with processing times $p_j \geq 0$, $j = 1, \dots, n$, and m identical parallel machines.

Task: Process each job j nonpreemptively for p_j time units on one of the m machines. Minimize the makespan.

Scheduling Jobs on Identical Parallel Machines

Given: n jobs $j = 1, \ldots, n$ with processing times $p_j \geq 0$, $j = 1, \ldots, n$, and m identical parallel machines.

Task: Process each job j nonpreemptively for p_j time units on one of the m machines. Minimize the makespan.

In other words, we consider $Pm||C_{\max}$ (or $P||C_{\max}$ if m is part of the input)

Scheduling Jobs on Identical Parallel Machines

Given: n jobs $j = 1, \ldots, n$ with processing times $p_j \geq 0$, $j = 1, \ldots, n$, and m identical parallel machines.

Task: Process each job j nonpreemptively for p_j time units on one of the m machines. Minimize the makespan.

In other words, we consider $Pm||C_{\max}$ (or $P||C_{\max}$ if m is part of the input)

Remember:

- List scheduling in arbitrary order is $\left(2 - \dfrac{1}{m}\right)$-approximation algorithm.
- List scheduling in LPT order is a $\left(\dfrac{4}{3} - \dfrac{1}{3m}\right)$-approximation algorithm.
- The analysis of both results relies on the fact that

$$C_{\max} \leq \left(1 - \tfrac{1}{m}\right) p_\ell + \tfrac{1}{m} \sum_{j=1}^{n} p_j \leq \left(1 - \tfrac{1}{m}\right) p_\ell + C^*_{\max}$$

  where ℓ is a job with maximal completion time $C_\ell = C_{\max}$.

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

Partition into short and long jobs:

- A job ℓ is called short if $p_\ell \leq \frac{\varepsilon}{m} \sum_j p_j$; otherwise, job ℓ is long.

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

Partition into short and long jobs:

- A job ℓ is called short if $p_\ell \leq \frac{\varepsilon}{m} \sum_j p_j$; otherwise, job ℓ is long.

Notice: There are at most $\lfloor m/\varepsilon \rfloor$ long jobs and this number is constant.

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

Partition into short and long jobs:

- A job ℓ is called short if $p_\ell \leq \frac{\varepsilon}{m} \sum_j p_j$; otherwise, job ℓ is long.

Notice: There are at most $\lfloor m/\varepsilon \rfloor$ long jobs and this number is constant.

Algorithm A_ε

1. enumerate all schedules of long jobs; choose one with min makespan;

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

Partition into short and long jobs:

- A job ℓ is called short if $p_\ell \leq \frac{\varepsilon}{m} \sum_j p_j$; otherwise, job ℓ is long.

Notice: There are at most $\lfloor m/\varepsilon \rfloor$ long jobs and this number is constant.

Algorithm A_ε

1. enumerate all schedules of long jobs; choose one with min makespan;
2. extend this schedule by using list scheduling for short jobs;

PTAS for Constant Number of Machines

Let the number of machines m be constant and $\varepsilon > 0$ fixed.

Partition into short and long jobs:

- A job ℓ is called short if $p_\ell \leq \frac{\varepsilon}{m} \sum_j p_j$; otherwise, job ℓ is long.

Notice: There are at most $\lfloor m/\varepsilon \rfloor$ long jobs and this number is constant.

Algorithm A_ε

1. enumerate all schedules of long jobs; choose one with min makespan;
2. extend this schedule by using list scheduling for short jobs;

Theorem 3.4

Algorithm A_ε runs in polynomial time for the problem $Pm||C_{\max}$ and produces a schedule of makespan at most $(1+\varepsilon) \cdot C^*_{\max}$.

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

PTAS for Arbitrary Number of Machines ($P||C_{max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.
- For each long job j let $\bar{p}_j := \lfloor p_j / (\varepsilon^2 T) \rfloor \cdot \varepsilon^2 T$.

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.
- For each long job j let $\bar{p}_j := \lfloor p_j / (\varepsilon^2 T) \rfloor \cdot \varepsilon^2 T$.
- Notice that there are at most $\lfloor 1/\varepsilon^2 \rfloor$ different rounded job sizes $\bar{p}_j$.

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.
- For each long job j let $\bar{p}_j := \lfloor p_j/(\varepsilon^2 T) \rfloor \cdot \varepsilon^2 T$.
- Notice that there are at most $\lfloor 1/\varepsilon^2 \rfloor$ different rounded job sizes $\bar{p}_j$.

Algorithm B_ε

1. find schedule for all long jobs with rounded sizes $\bar{p}_j$ of makespan $\leq T$; if no such schedule exists, then return "T too small";

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.
- For each long job j let $\bar{p}_j := \lfloor p_j / (\varepsilon^2 T) \rfloor \cdot \varepsilon^2 T$.
- Notice that there are at most $\lfloor 1/\varepsilon^2 \rfloor$ different rounded job sizes $\bar{p}_j$.

Algorithm B_ε

1. find schedule for all long jobs with rounded sizes $\bar{p}_j$ of makespan $\leq T$; if no such schedule exists, then return "T too small";
2. interpret as a schedule for the long jobs with original sizes p_j;

PTAS for Arbitrary Number of Machines ($P||C_{\max}$)

Now, m is no longer constant but part of the input. Let $\varepsilon > 0$ be fixed.

Main ideas:

- An approximate schedule for the long jobs suffices.
- Round long jobs such that there are constantly many different sizes.

Let T be a target length for the schedule ($T \geq \max_j p_j$, $T \geq \frac{1}{m} \sum_j p_j$).

Short and long jobs:

- A job j is called short if $p_j \leq \varepsilon T$; otherwise, job j is long.
- For each long job j let $\bar{p}_j := \lfloor p_j/(\varepsilon^2 T) \rfloor \cdot \varepsilon^2 T$.
- Notice that there are at most $\lfloor 1/\varepsilon^2 \rfloor$ different rounded job sizes $\bar{p}_j$.

Algorithm B_ε

1. find schedule for all long jobs with rounded sizes $\bar{p}_j$ of makespan $\leq T$; if no such schedule exists, then return "T too small";
2. interpret as a schedule for the long jobs with original sizes p_j;
3. extend this schedule by using list scheduling for short jobs;

Analysis and Results

Theorem 3.5

For a given problem instance and a target length T, Algorithm B_ε either correctly decides that there is no schedule of length $\leq T$ or it finds a schedule of length $\leq (1+\varepsilon)\cdot T$.

Analysis and Results

Theorem 3.5

For a given problem instance and a target length T, Algorithm B_ε either correctly decides that there is no schedule of length $\leq T$ or it finds a schedule of length $\leq (1+\varepsilon) \cdot T$.

Proof:… $\square$

Analysis and Results

Theorem 3.5

For a given problem instance and a target length T, Algorithm B_ε either correctly decides that there is no schedule of length $\leq T$ or it finds a schedule of length $\leq (1+\varepsilon)\cdot T$.

Proof:... □

Remarks:

- According to Theorem 3.5 Algorithm B_ε is a $(1+\varepsilon)$-approximate decision procedure.

Analysis and Results

Theorem 3.5

For a given problem instance and a target length T, Algorithm B_ε either correctly decides that there is no schedule of length $\leq T$ or it finds a schedule of length $\leq (1+\varepsilon) \cdot T$.

Proof:... □

Remarks:

- According to Theorem 3.5 Algorithm B_ε is a $(1+\varepsilon)$-approximate decision procedure.
- Together with a binary search framework for the optimal makespan $T = C^*_{\text{max}}$, we get a polynomial-time approximation scheme (PTAS).

Analysis and Results

Theorem 3.5

For a given problem instance and a target length T, Algorithm B_ε either correctly decides that there is no schedule of length $\leq T$ or it finds a schedule of length $\leq (1+\varepsilon) \cdot T$.

Proof:… □

Remarks:

- According to Theorem 3.5 Algorithm B_ε is a $(1+\varepsilon)$-approximate decision procedure.
- Together with a binary search framework for the optimal makespan $T = C^*_{\max}$, we get a polynomial-time approximation scheme (PTAS).

Theorem 3.6

There is a polynomial-time approximation scheme for $P||C_{\max}$.

Existence of an FPTAS

We state the next theorems without proof:

Theorem 3.7

There is a fully polynomial-time approximation scheme for the problem of minimizing the makespan on constantly many identical parallel machines: $Pm||C_{\max}$.

Existence of an FPTAS

We state the next theorems without proof:

Theorem 3.7

There is a fully polynomial-time approximation scheme for the problem of minimizing the makespan on constantly many identical parallel machines: $Pm||C_{\max}$.

Theorem 3.8

If the number of machines is part of the input, i.e.for the problem $P||C_{\max}$, there is no FPTAS, unless $P = NP$.

Existence of an FPTAS

We state the next theorems without proof:

Theorem 3.7

There is a fully polynomial-time approximation scheme for the problem of minimizing the makespan on constantly many identical parallel machines: $Pm||C_{\max}$.

Theorem 3.8

If the number of machines is part of the input, i.e.for the problem $P||C_{\max}$, there is no FPTAS, unless $P = NP$.

Remark: More generally, a strongly NP-hard optimization problem whose objective function values are integral and polynomially bounded in the numbers occurring in the input does not have an FPTAS, unless $P = NP$.

Outline

Bin-Packing Problem

Given: n items with positive sizes $a_1, \ldots, a_n \leq 1$.

Task: Pack the items into a minimal number of unit-size bins.

Bin-Packing Problem

Given: n items with positive sizes $a_1, \ldots, a_n \leq 1$.

Task: Pack the items into a minimal number of unit-size bins.

Theorem 3.9

Unless $P = NP$, there is no α-approximation algorithm for the Bin-Packing Problem for any $\alpha < 3/2$.

Bin-Packing Problem

Given: n items with positive sizes $a_1, \dots, a_n \leq 1$.

Task: Pack the items into a minimal number of unit-size bins.

Theorem 3.9

Unless $P = NP$, there is no α-approximation algorithm for the Bin-Packing Problem for any $\alpha < 3/2$.

Proof: Reduce the Partition Problem. □

Bin-Packing Problem

Given: n items with positive sizes $a_1, \ldots, a_n \leq 1$.

Task: Pack the items into a minimal number of unit-size bins.

Theorem 3.9

Unless $P = NP$, there is no α-approximation algorithm for the Bin-Packing Problem for any $\alpha < 3/2$.

Proof: Reduce the Partition Problem. □

Algorithm Next-Fit

- consider items in arbitrary order; start to pack them into the first bin;
- whenever next item does not fit into the current bin, open a new bin;

Bin-Packing Problem

Given: n items with positive sizes $a_1, \ldots, a_n \leq 1$.

Task: Pack the items into a minimal number of unit-size bins.

Theorem 3.9

Unless $P = NP$, there is no α-approximation algorithm for the Bin-Packing Problem for any $\alpha < 3/2$.

Proof: Reduce the Partition Problem. □

Algorithm Next-Fit

- consider items in arbitrary order; start to pack them into the first bin;
- whenever next item does not fit into the current bin, open a new bin;

Theorem 3.10

Algorithm Next-Fit runs in $O(n)$ time and uses at most $2 \cdot \mathtt{OPT} - 1$ bins.

First-Fit Heuristics for Bin-Packing

Algorithm First-Fit

- consider items in arbitrary order; open one bin;
- pack the next item into the first open bin in which it fits;
- if the item does not fit into any open bin, open a new bin;

First-Fit Heuristics for Bin-Packing

Algorithm First-Fit

- consider items in arbitrary order; open one bin;
- pack the next item into the first open bin in which it fits;
- if the item does not fit into any open bin, open a new bin;

Theorem 3.11 (Dósa, Sgall 2013)

Algorithm First-Fit runs in polynomial time; it uses at most $\lfloor \frac{17}{10} \mathtt{OPT} \rfloor$ bins.

First-Fit Heuristics for Bin-Packing

Algorithm First-Fit

- consider items in arbitrary order; open one bin;
- pack the next item into the first open bin in which it fits;
- if the item does not fit into any open bin, open a new bin;

Theorem 3.11 (Dósa, Sgall 2013)

Algorithm First-Fit runs in polynomial time; it uses at most $\lfloor \frac{17}{10} \mathtt{OPT} \rfloor$ bins.

First-Fit Heuristics for Bin-Packing

Algorithm First-Fit

- consider items in arbitrary order; open one bin;
- pack the next item into the first open bin in which it fits;
- if the item does not fit into any open bin, open a new bin;

Theorem 3.11 (Dósa, Sgall 2013)

Algorithm First-Fit runs in polynomial time; it uses at most $\left\lfloor \frac{17}{10} \mathtt{OPT} \right\rfloor$ bins.

Algorithm First-Fit-Decreasing

- Same as `First-Fit`, but consider items in order of decreasing size

First-Fit Heuristics for Bin-Packing

Algorithm First-Fit

- consider items in arbitrary order; open one bin;
- pack the next item into the first open bin in which it fits;
- if the item does not fit into any open bin, open a new bin;

Theorem 3.11 (Dósa, Sgall 2013)

Algorithm First-Fit runs in polynomial time; it uses at most $\lfloor \frac{17}{10} \mathtt{OPT} \rfloor$ bins.

Algorithm First-Fit-Decreasing

- Same as `First-Fit`, but consider items in order of decreasing size

Theorem 3.12 (Dósa 2007)

Algorithm First-Fit-Decreasing uses at most $\frac{11}{9}\mathtt{OPT} + \frac{2}{3}$ bins.

Towards an Asymptotic PTAS for Bin-Packing

Definition 3.13

An asymptotic polynomial-time approximation scheme (APTAS) is a family of polynomial-time algorithms $(A_\varepsilon)_{\varepsilon>0}$ along with a constant c such that A_ε returns a solution of value at most $(1+\varepsilon)\texttt{OPT} + c$.

Towards an Asymptotic PTAS for Bin-Packing

Definition 3.13

An asymptotic polynomial-time approximation scheme (APTAS) is a family of polynomial-time algorithms $(A_\varepsilon)_{\varepsilon>0}$ along with a constant c such that A_ε returns a solution of value at most $(1+\varepsilon)\texttt{OPT} + c$.

Lemma 3.14

Any packing of all items of size $\geq \gamma$ into ℓ bins can be greedily extended to a packing of all items into at most $\max\left\{\ell, \frac{1}{1-\gamma}\mathsf{SIZE}(I) + 1\right\}$ bins.

Towards an Asymptotic PTAS for Bin-Packing

Definition 3.13

An asymptotic polynomial-time approximation scheme (APTAS) is a family of polynomial-time algorithms $(A_\varepsilon)_{\varepsilon>0}$ along with a constant c such that A_ε returns a solution of value at most $(1+\varepsilon)\mathtt{OPT} + c$.

Lemma 3.14

Any packing of all items of size $\geq \gamma$ into ℓ bins can be greedily extended to a packing of all items into at most $\max\{\ell, \frac{1}{1-\gamma}\mathsf{SIZE}(I)+1\}$ bins.

Proof:… □

Towards an Asymptotic PTAS for Bin-Packing

Definition 3.13

An asymptotic polynomial-time approximation scheme (APTAS) is a family of polynomial-time algorithms $(A_\varepsilon)_{\varepsilon>0}$ along with a constant c such that A_ε returns a solution of value at most $(1+\varepsilon)\texttt{OPT}+c$.

Lemma 3.14

Any packing of all items of size $\geq \gamma$ into ℓ bins can be greedily extended to a packing of all items into at most $\max\left\{\ell, \frac{1}{1-\gamma}\mathrm{SIZE}(I)+1\right\}$ bins.

Proof:... □

Remarks:

- For $\gamma=\varepsilon/2$, the lemma yields a packing of all items into at most $\max\left\{\ell,(1+\varepsilon)\texttt{OPT}+1\right\}$ bins.
- In the following we can thus restrict to items of size at least $\varepsilon/2$.

Linear Grouping Scheme

For given instance I and parameter $k \in \mathbb{Z}_{>0}$, define a new instance I':

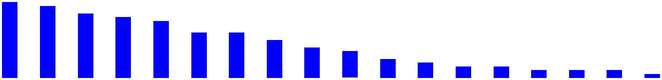
1. sort the items of instance I such that $a_1 \geq a_2 \geq \cdots \geq a_n$;
2. instance I' has $n - k$ items of size $a'_i := a_{\lceil i/k \rceil \cdot k+1}$, $i = 1, \ldots, n-k$;

Linear Grouping Scheme

For given instance I and parameter $k \in \mathbb{Z}_{>0}$, define a new instance I':

1. sort the items of instance I such that $a_1 \geq a_2 \geq \cdots \geq a_n$;
2. instance I' has $n-k$ items of size $a'_i := a_{\lceil i/k \rceil \cdot k+1}$, $i = 1, \ldots, n-k$;

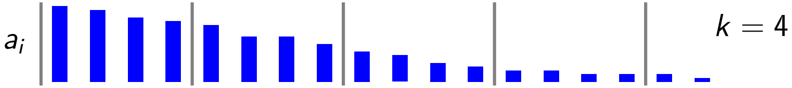
a_i



Linear Grouping Scheme

For given instance I and parameter $k \in \mathbb{Z}_{>0}$, define a new instance I':

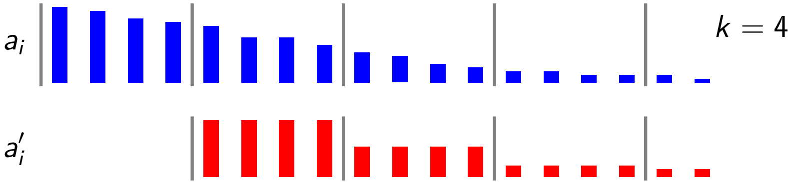
1. sort the items of instance I such that $a_1 \geq a_2 \geq \cdots \geq a_n$;
2. instance I' has $n-k$ items of size $a'_i := a_{\lceil i/k \rceil \cdot k+1}$, $i = 1, \ldots, n-k$;



Linear Grouping Scheme

For given instance I and parameter $k \in \mathbb{Z}_{>0}$, define a new instance I':

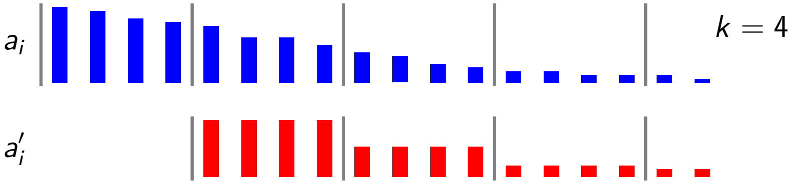
1. sort the items of instance I such that $a_1 \geq a_2 \geq \cdots \geq a_n$;
2. instance I' has $n-k$ items of size $a'_i := a_{\lceil i/k \rceil \cdot k+1}$, $i = 1, \ldots, n-k$;



Linear Grouping Scheme

For given instance I and parameter $k \in \mathbb{Z}_{>0}$, define a new instance I':

1. sort the items of instance I such that $a_1 \geq a_2 \geq \cdots \geq a_n$;
2. instance I' has $n-k$ items of size $a'_i := a_{\lceil i/k \rceil \cdot k+1}$, $i = 1, \ldots, n-k$;



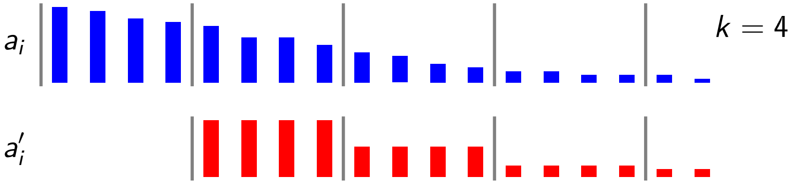
Remarks

- Instance I' has at most $\lfloor n/k \rfloor$ distinct item sizes.
- It holds that $a_{i+k} \leq a'_i \leq a_i$ for $i = 1, \ldots, n-k$.

Linear Grouping Scheme

For given instance I and parameter $k \in \mathbb{Z}_{>0}$, define a new instance I':

1. sort the items of instance I such that $a_1 \geq a_2 \geq \cdots \geq a_n$;
2. instance I' has $n-k$ items of size $a'_i := a_{\lceil i/k \rceil \cdot k+1}$, $i = 1, \ldots, n-k$;



Remarks

- Instance I' has at most $\lfloor n/k \rfloor$ distinct item sizes.
- It holds that $a_{i+k} \leq a'_i \leq a_i$ for $i = 1, \ldots, n-k$.

Lemma 3.15

Any packing of I' can be easily turned into a packing of I with at most k additional bins. Moreover, $\texttt{OPT}(I') \leq \texttt{OPT}(I) \leq \texttt{OPT}(I') + k$.

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\text{SIZE}(I) \geq \varepsilon n/2$.

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\mathrm{SIZE}(I) \geq \varepsilon n/2$.
- W.l.o.g.: $\varepsilon \cdot \mathrm{SIZE}(I) \geq 1$ (otherwise, there are at most $2/\varepsilon^2$ items…).

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\text{SIZE}(I) \geq \varepsilon n/2$.
- W.l.o.g.: $\varepsilon \cdot \text{SIZE}(I) \geq 1$ (otherwise, there are at most $2/\varepsilon^2$ items...).
- Set $k := \lfloor \varepsilon \cdot \text{SIZE}(I) \rfloor$ and apply the linear grouping scheme.

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\mathrm{SIZE}(I) \geq \varepsilon n/2$.
- W.l.o.g.: $\varepsilon \cdot \mathrm{SIZE}(I) \geq 1$ (otherwise, there are at most $2/\varepsilon^2$ items...).
- Set $k := \lfloor \varepsilon \cdot \mathrm{SIZE}(I) \rfloor$ and apply the linear grouping scheme.
- Resulting instance I' has at most $n/k \leq 4/\varepsilon^2$ distinct item sizes.

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\text{SIZE}(I) \geq \varepsilon n/2$.
- W.l.o.g.: $\varepsilon \cdot \text{SIZE}(I) \geq 1$ (otherwise, there are at most $2/\varepsilon^2$ items...).
- Set $k := \lfloor \varepsilon \cdot \text{SIZE}(I) \rfloor$ and apply the linear grouping scheme.
- Resulting instance I' has at most $n/k \leq 4/\varepsilon^2$ distinct item sizes.
- Thus, instance I' can be solved optimally in polynomial time.

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\text{SIZE}(I) \geq \varepsilon n/2$.
- W.l.o.g.: $\varepsilon \cdot \text{SIZE}(I) \geq 1$ (otherwise, there are at most $2/\varepsilon^2$ items…).
- Set $k := \lfloor \varepsilon \cdot \text{SIZE}(I) \rfloor$ and apply the linear grouping scheme.
- Resulting instance I' has at most $n/k \leq 4/\varepsilon^2$ distinct item sizes.
- Thus, instance I' can be solved optimally in polynomial time.

Theorem 3.16

For any $\varepsilon > 0$, there is a polynomial-time algorithm for the Bin-Packing Problem that computes a solution with at most $(1+\varepsilon) \cdot \texttt{OPT} + 1$ bins.

APTAS for Bin-Packing

Ingredients:

- All items have size at least $\varepsilon/2$ such that $\text{SIZE}(I) \geq \varepsilon n/2$.
- W.l.o.g.: $\varepsilon \cdot \text{SIZE}(I) \geq 1$ (otherwise, there are at most $2/\varepsilon^2$ items...).
- Set $k := \lfloor \varepsilon \cdot \text{SIZE}(I) \rfloor$ and apply the linear grouping scheme.
- Resulting instance I' has at most $n/k \leq 4/\varepsilon^2$ distinct item sizes.
- Thus, instance I' can be solved optimally in polynomial time.

Theorem 3.16

For any $\varepsilon > 0$, there is a polynomial-time algorithm for the Bin-Packing Problem that computes a solution with at most $(1+\varepsilon) \cdot \texttt{OPT} + 1$ bins.

Proof:... □