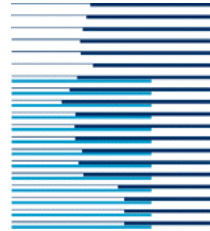


Approximation Algorithms

Monaldo Mastrolilli
IDSIA, Lugano (Switzerland)



Optimization Problems

Instance: I

Length of description: $|I|=n$

Goal: minimize objective function $\Rightarrow \text{Opt}(I)$

α -Approximation algorithm

$A(I)$ = solution value

- $A(I) \leq \alpha \text{Opt}(I)$ for all instances
- running time polynomial in $|I|$

α = worst-case approximation ratio
 $\alpha \geq 1$ Good: α close to 1

PTAS: Polynomial Time Approximation Scheme

Family $\{A_\varepsilon\}_{\varepsilon>0}$ of $(1+\varepsilon)$ -approximation algorithms

- running time polynomial in $|I|$
- ok: exponential in $1/\varepsilon$: e.g. $O(|I|^{1/\varepsilon})$

FPTAS: Fully PTAS

running time also polynomial in $1/\varepsilon$: e.g. $O(|I|/\varepsilon^3)$

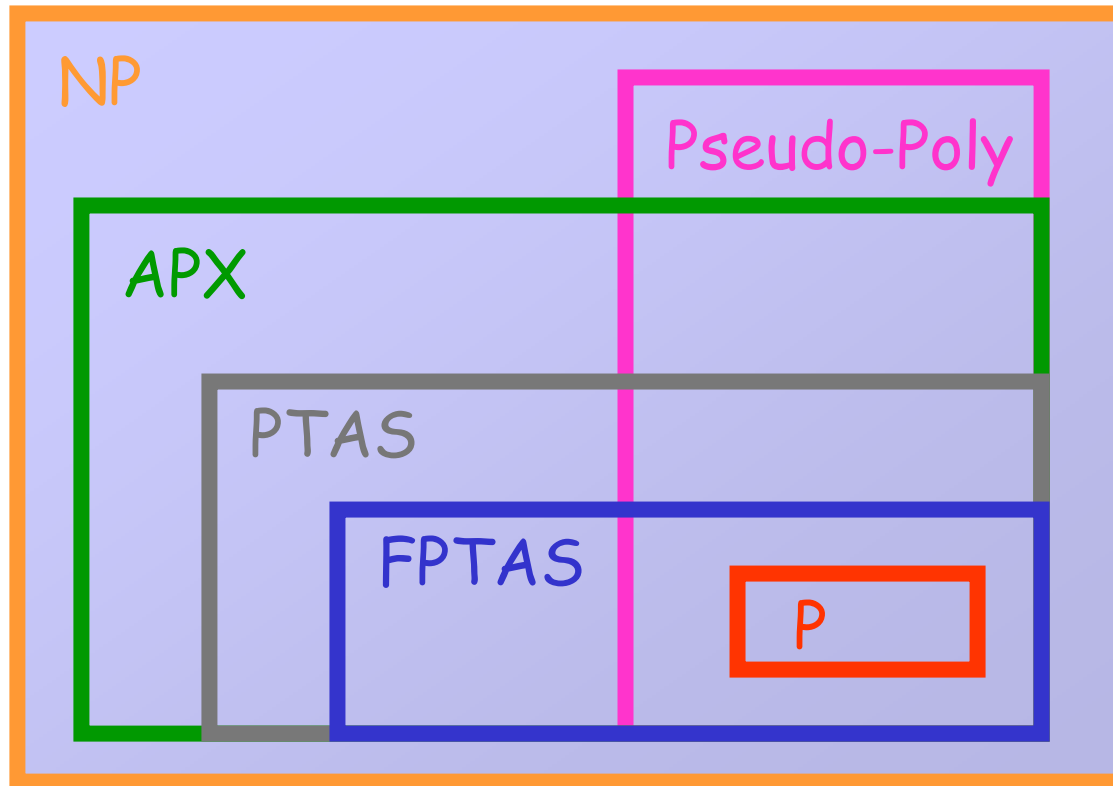
Strongly and Weakly NP-hard

- If a problem is NP-hard even if the input is encoded in unary, then it is called strongly NP-hard = NP-hard in the strong sense = unary NP-hard
- If a problem is polynomially solvable under a unary encoding, then it is solvable in pseudo-polynomial time.

weak sense

strong
sense

Complexity Classes Relationships



Approximation Algorithms

Non-constant worst-case ratio

- Graph coloring $O(n^{1/2-\epsilon})$
- Total flow time $O(n^{1/2})$
- Set covering $O(\log n)$

Constant worst-case ratio

- TSP with triangle-inequalities $3/2$
- Max Sat 1.2987

PTAS

- Bin packing

FPTAS

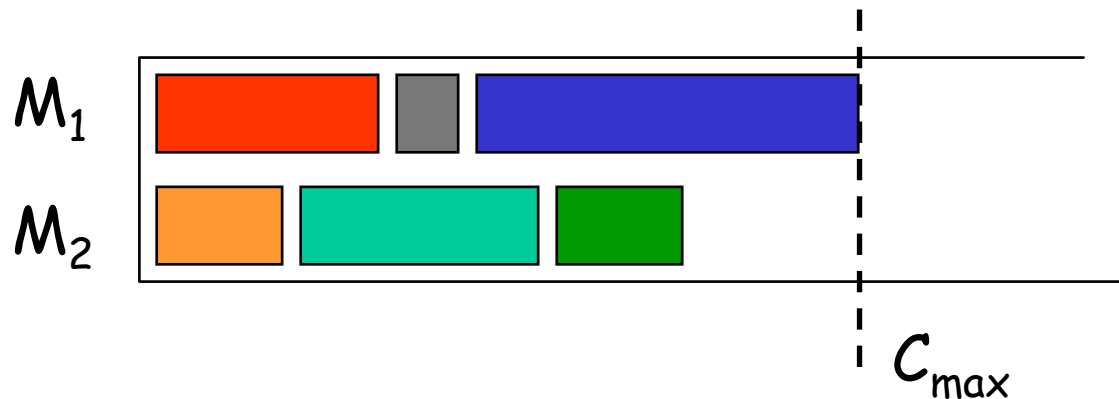
- Makespan on 2 machines

The first Approximation Algorithm (Graham '66)

$P||C_{\max}$

Makespan minimization on m identical machines (strongly NP-hard)

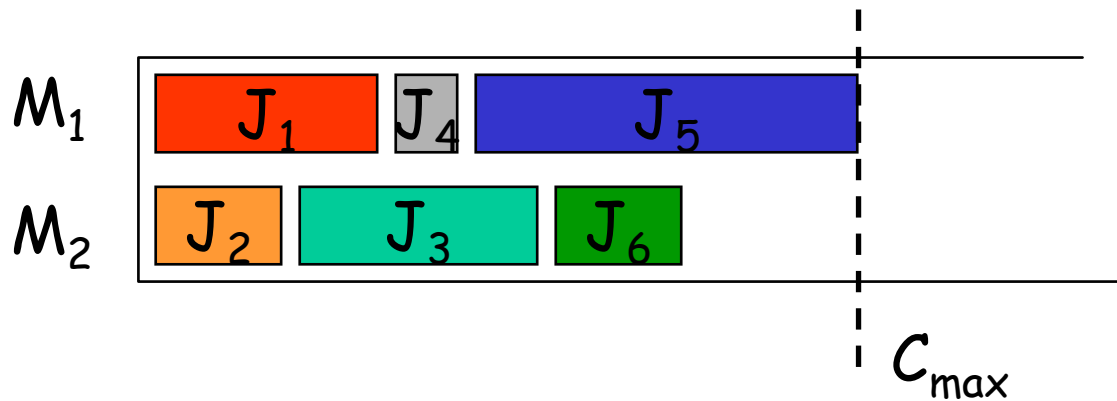
I: m identical machines
 n jobs with lengths p_1, p_2, \dots, p_n



List-Scheduling (LS)

LS: schedule jobs in any given order to the first available (i.e. idle) machine

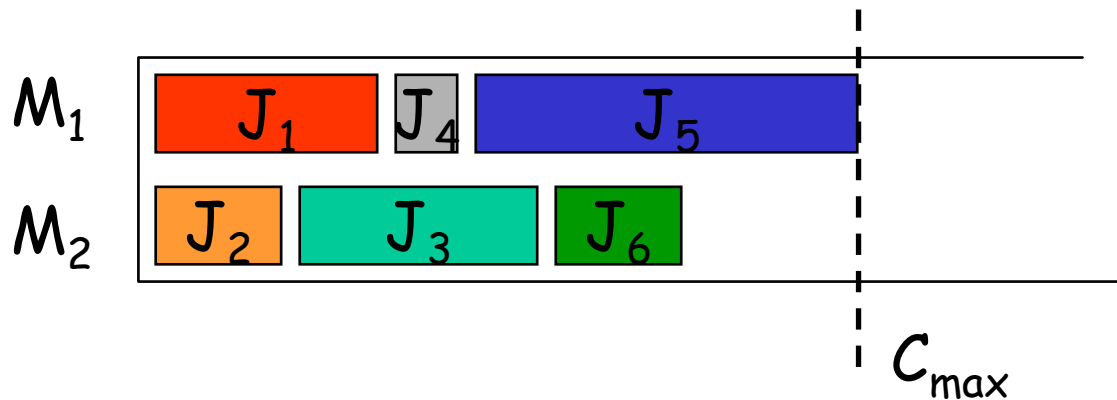
List: $J_1, J_2, J_3, J_4, J_5, J_6$



LS: Analysis

$$\text{Opt} \geq \text{LB} = \max \left\{ \frac{1}{m} \sum p_j; \max p_j \right\}$$

s_f : starting time of the job that completes last
 $C_{\max}^{\text{LS}} = s_f + p_f$
 E_i : completion time of machine M_i

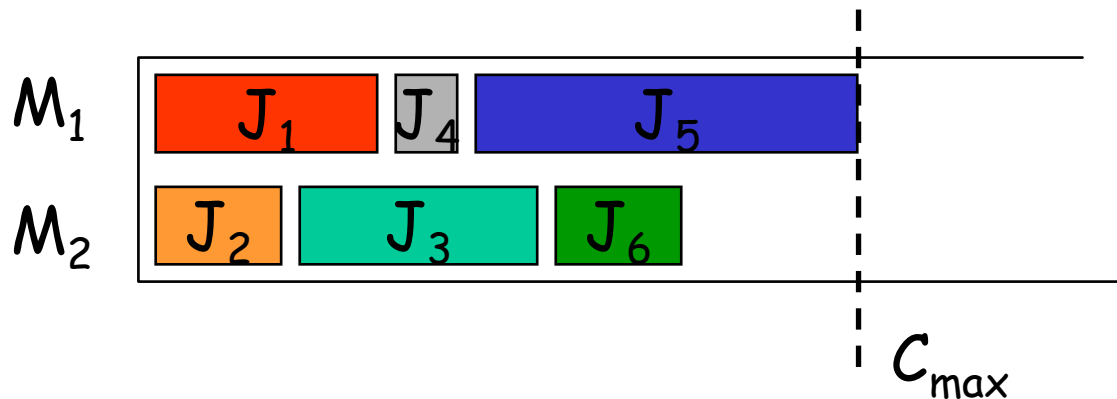


LS: Analysis

$$\left\{ \begin{array}{l} S_f \leq E_i \\ S_f = E_1 - p_f \end{array} \right. \xrightarrow{\text{sum}} mS_f \leq \sum_{i=1}^m E_i - p_f$$

$$S_f \leq \frac{1}{m} \left(\sum_{i=1}^m E_i - p_f \right) = \frac{1}{m} \left(\sum p_j - p_f \right)$$

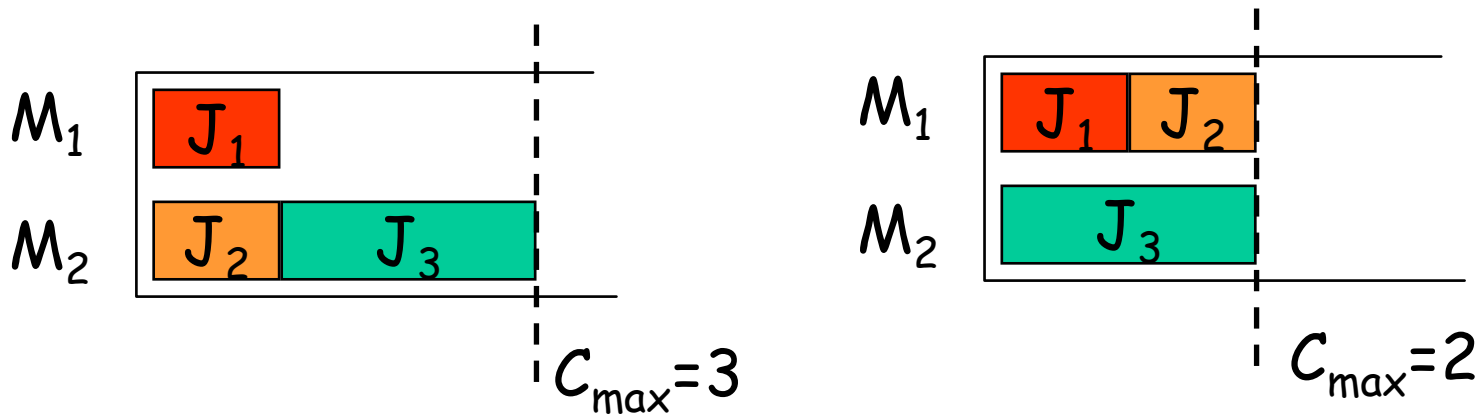
$$C_{\max}^{\text{LS}} \leq \frac{1}{m} \sum p_j + p_f \left(1 - \frac{1}{m} \right) \leq \underline{\text{Opt} \left(2 - \frac{1}{m} \right)}$$



LS: Analysis

Th: LS is a $(2-1/m)$ -approximation algorithm.

The approximation ratio is tight.
Example: $p_1 = p_2 = 1$ and $p_3 = 2$



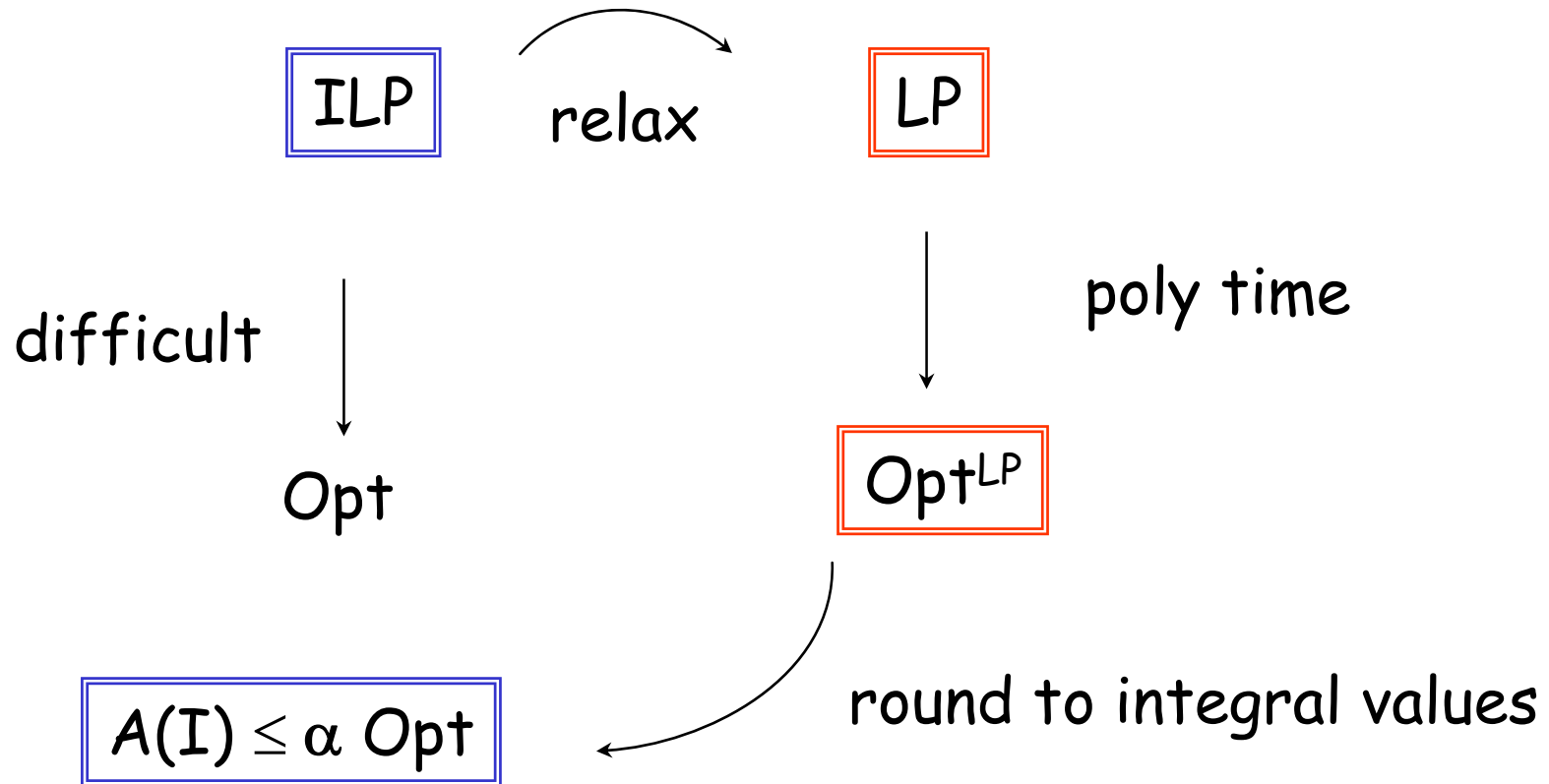
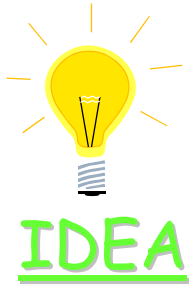
Worst vs. Average Case

$$\left. \begin{array}{l} C_{\max}^{\text{LS}} \leq \frac{1}{m} \sum p_j + p_f \left(1 - \frac{1}{m}\right) \\ \text{Opt} \geq \frac{1}{m} \sum p_j \end{array} \right\} \Rightarrow \frac{C_{\max}^{\text{LS}}}{\text{Opt}} \geq 1 + \frac{p_f(m-1)}{\sum p_j}$$

Assumption: processing times are independently and uniformly distributed in $[0..b]$

$$\Pr \left[\frac{C_{\max}^{\text{LS}}}{\text{Opt}} \geq 1 + \frac{4(m-1)}{n} \right] \leq \Pr \left[\sum p_j \leq \frac{4}{n} \right] \leq e^{-n/8}$$

Linear Programming based approximation algorithms



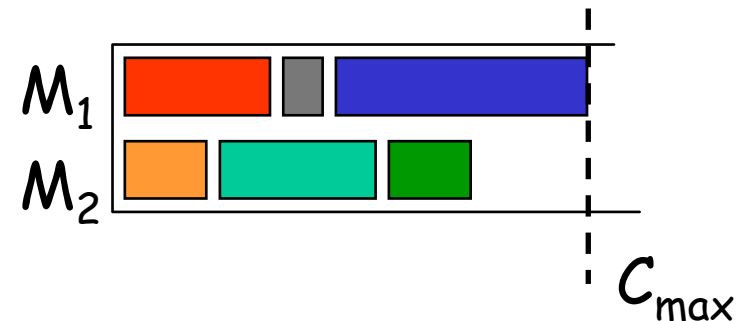
Example: $R2||C_{\max}$

$R2||C_{\max}$: Makespan minimization on 2 unrelated machines (weakly NP-hard)

I: 2 unrelated machines n jobs

Job j has length

- p_{1j} on machine M_1
- p_{2j} on machine M_2



Integer Linear Program (ILP)

Minimize C_{\max}

subject to $x_{1j} + x_{2j} = 1, \quad j = 1, \dots, n$

$$\sum_{j=1}^n p_{1j} x_{1j} \leq C_{\max}$$
$$\sum_{j=1}^n p_{2j} x_{2j} \leq C_{\max}$$
$$x_{1j}, x_{2j} \in \{0,1\} \quad j = 1, \dots, n$$

LP relaxation

$$\begin{aligned} &\text{Minimize} && C_{\max} \\ &\text{subject to} && \mathbf{x}_{1j} + \mathbf{x}_{2j} = 1, && j = 1, \dots, n \\ & && \sum_{j=1}^n \mathbf{p}_{1j} \mathbf{x}_{1j} \leq C_{\max} \\ & && \sum_{j=1}^n \mathbf{p}_{2j} \mathbf{x}_{2j} \leq C_{\max} \\ & && \mathbf{x}_{1j}, \mathbf{x}_{2j} \geq 0 && j = 1, \dots, n \end{aligned}$$

Number of fractional jobs

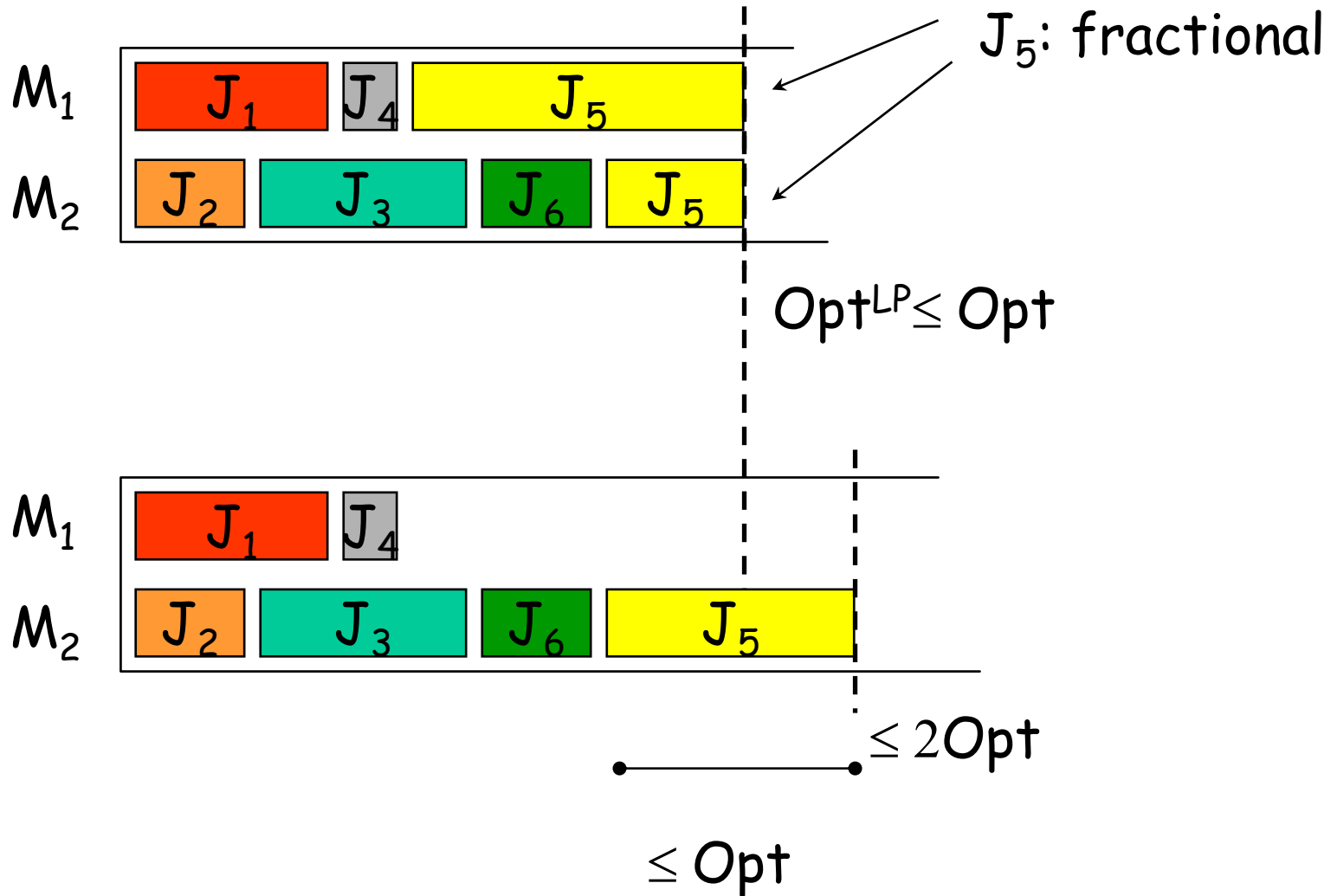
Known: a basic optimal LP solution has the property that the number of variables that get positive values is at most the number of rows in the constraint matrix $\Rightarrow n+2$

Since C_{\max} is always positive at most $n+1$ of the x_{ij} variables are positive

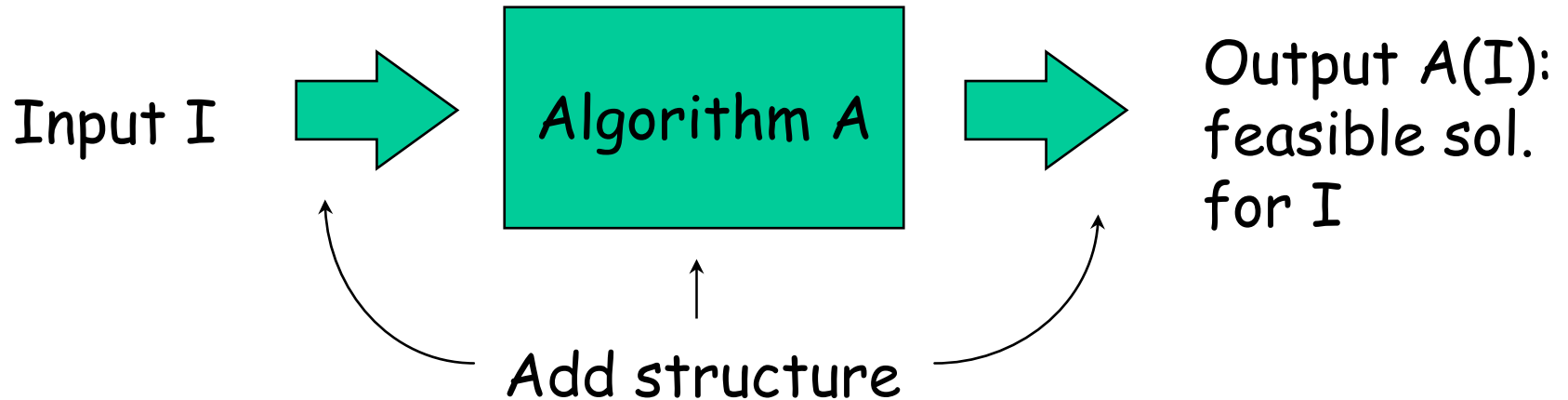
Every job has at least one positive variable associated with it.

At most **1** job has been split onto two machines !!

Rounding



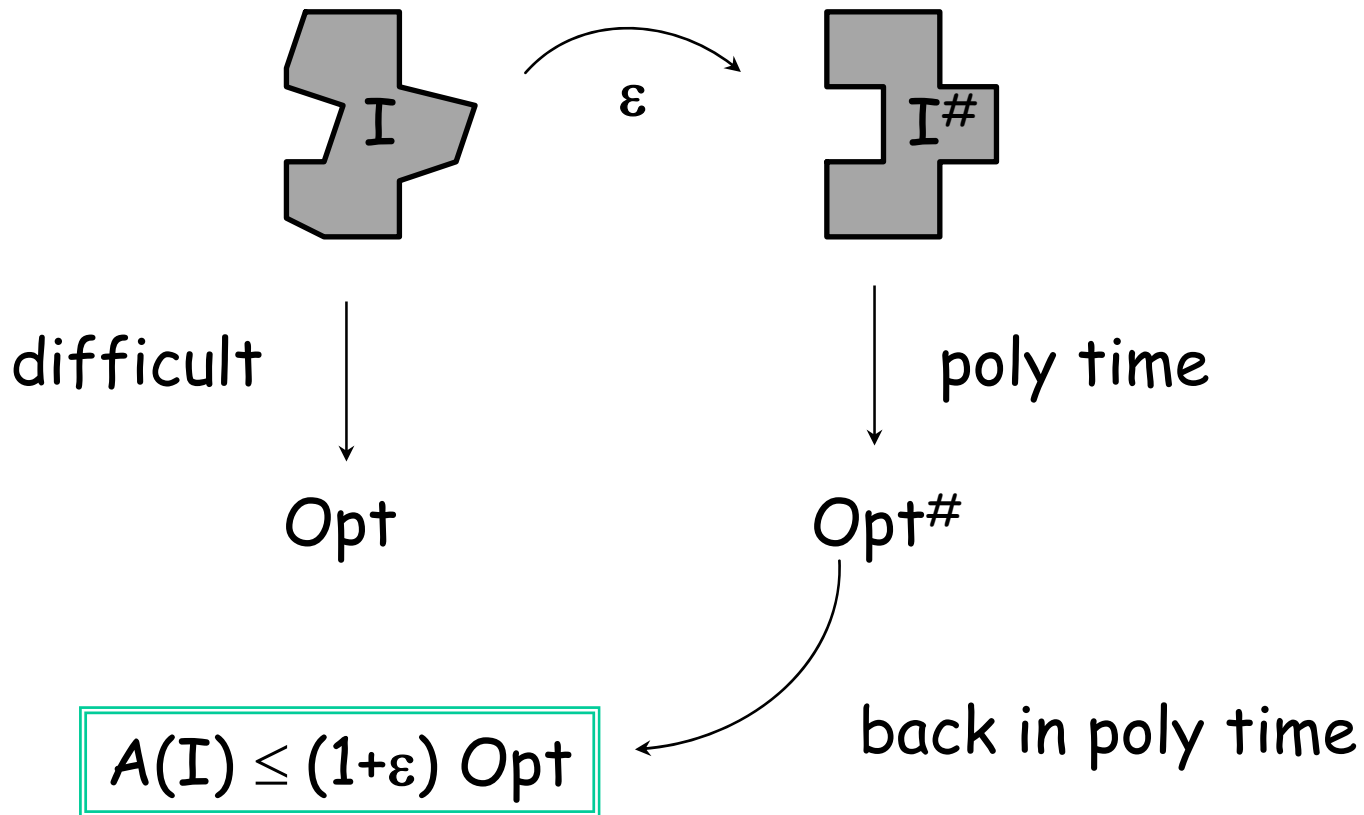
How to get a PTAS



IDEA: add more structure (depending on ε)
as $\varepsilon \rightarrow 0$, additional structure $\rightarrow 0$

Compare: as $\varepsilon \rightarrow 0$, $\frac{4 + 2\varepsilon}{3 + \varepsilon} \rightarrow \frac{4}{3}$

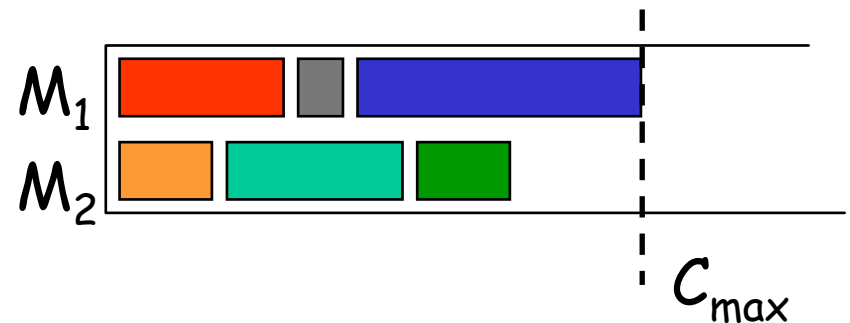
Structuring the Input



Example: $P2||C_{\max}$

$P2||C_{\max}$ Makespan minimization on 2 identical machines
(weakly NP-hard)

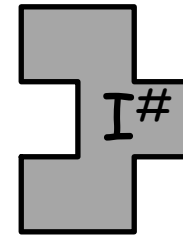
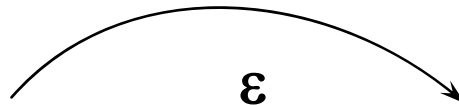
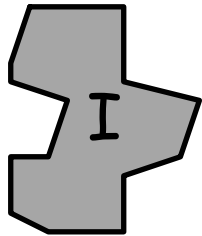
I: 2 identical machines
n jobs with lengths
 p_1, p_2, \dots, p_n



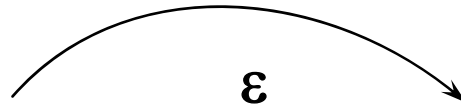
$$LB = \max \left\{ \frac{1}{2} \sum p_j; \max p_j \right\}$$

$$LB \leq Opt \leq 2 \cdot LB$$

How to round the input

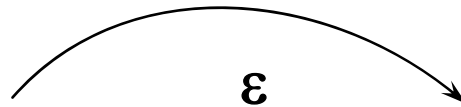


$p_j > \epsilon LB$
"big"



$p_j^\# := p_j$

$p_j \leq \epsilon LB$
"small"



$\left\lfloor \frac{S}{\epsilon LB} \right\rfloor$ jobs of length ϵLB

$S := \sum_{\text{small}} p_j$

Rounded Instance $I^\#$

Recall that $\sum p_j^\# \leq 2LB$

How many big jobs ?

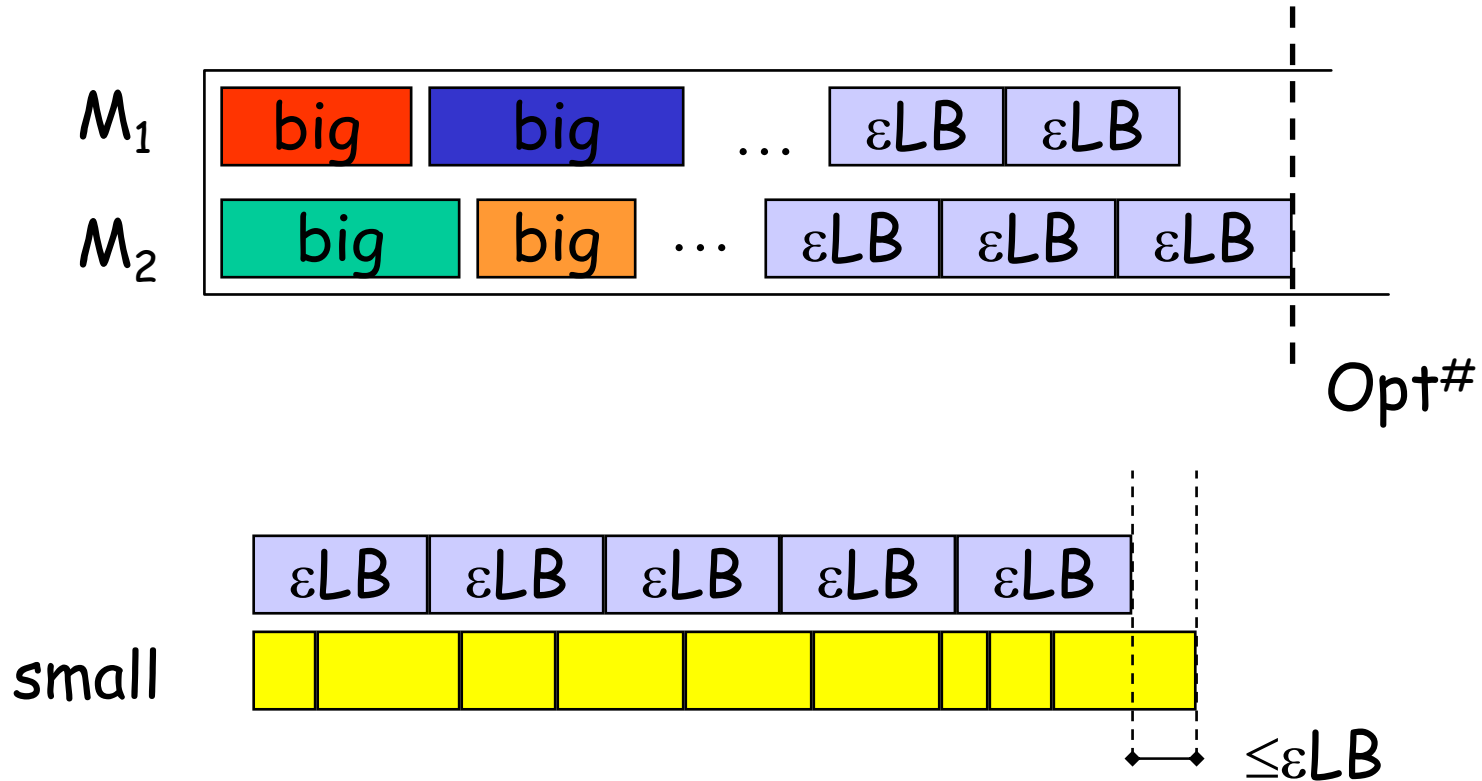
• a big job has $p_j^\# > \varepsilon LB \implies \#(\text{big}) \leq 2/\varepsilon$

How many small jobs ?

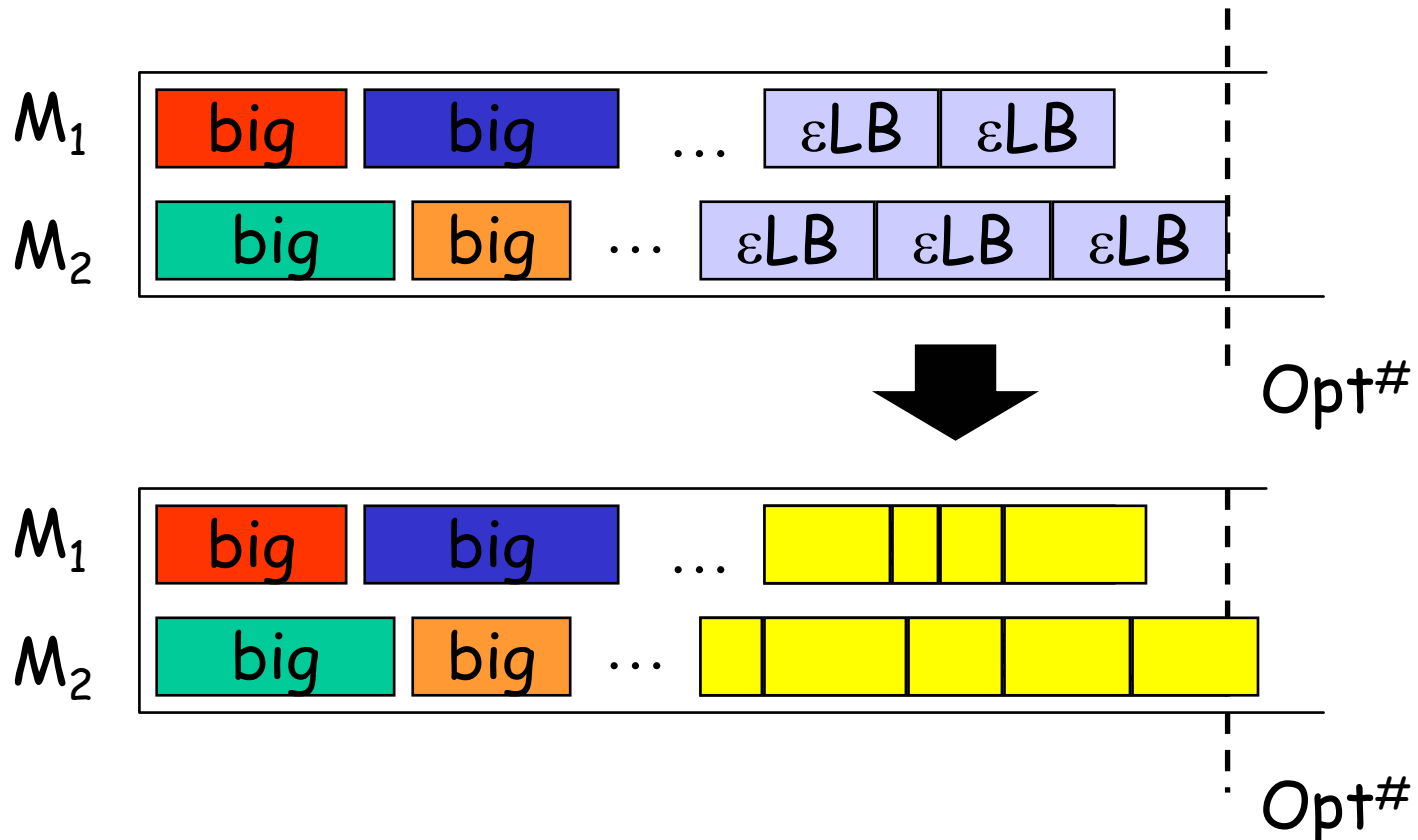
A small job has $p_j^\# = \varepsilon LB \implies \#(\text{small}) \leq 2/\varepsilon$

The rounded instance has a constant(ε) number of jobs.
 \implies optimal solution in constant time!!

Back to a feasible solution




Back to a feasible solution (ctd)



$$C_{\max} \leq Opt^\# + \epsilon LB \leq (1+\epsilon) Opt^\#$$

How much error is introduced?

$$\begin{array}{l|l} C_{\max} \leq \text{Opt}^\# + \varepsilon LB & \\ \text{Opt}^\# \leq \text{Opt} + \varepsilon LB & \end{array} \quad \Rightarrow \quad \begin{array}{l} C_{\max} \leq \text{Opt} + 2\varepsilon LB \\ \leq (1+2\varepsilon) \text{Opt}^\# \end{array}$$

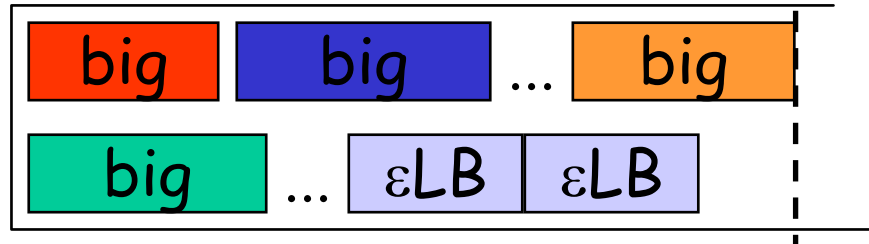

?

Opt vs. Opt[#]

(Case 1)

Opt[#] ≤ Opt

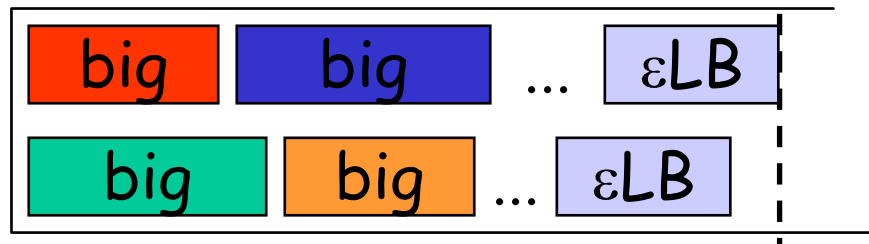
M₁



(Case 2)

M₁

M₂



$$\text{Opt}^{\#} \leq C_{\max}^{\text{LS}} \leq \frac{1}{m} \sum p_j^{\#} + p_j^{\#} \left(1 - \frac{1}{m}\right) \leq \text{Opt} + \varepsilon \text{LB}$$

Structuring the execution of an algorithm



IDEA: take an exact but slow algorithm A and interact with it while it is working. **Clean-up** part of its memory. As a result the algorithm becomes **faster** (less data to process) and generates **incorrect output**.

IDEAL CASE: the time complexity becomes polynomial and the incorrect output is a good approximation

Compare: tabu search vs. branch & bound

Example: $R2||C_{\max}$

Exact Algorithm (Dynamic Program)

Encode a partial schedule for the first k jobs by a vector (state) $[\alpha, \beta]$

- α = total processing time on M_1
- β = total processing time on M_2

Assign job one by one and update the states appropriately

Dynamic Program

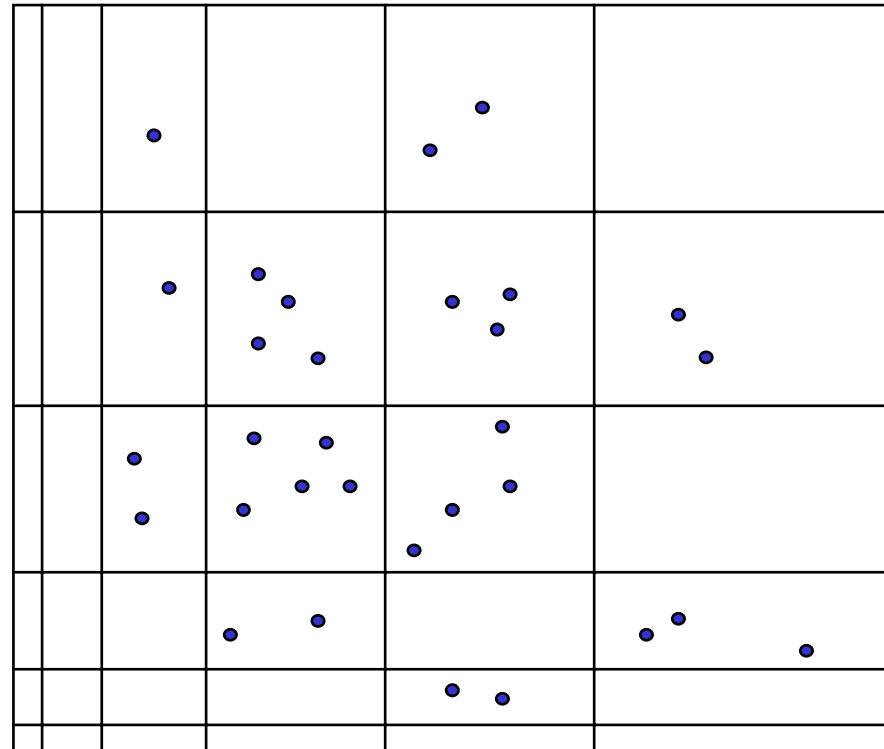
```
S0 := { [0,0] }  
For k:=1 to n do  
    Sk := {  
        For all [α, β] in Sk-1 do  
            add the two states  
                [α + p1k, β] and  
                [α, β + p2k] to Sk  
        End For  
    End For  
  
End For  
  
Output min{ max{α, β} | [α, β] in Sn }
```

Dynamic Program

```
S0 := { [0,0] }  
For k:=1 to n do  
    Sk := {  
        For all [α, β] in Sk-1 do  
            add the two states  
                [α + p1k, β] and  
                [α, β + p2k] to Sk  
        End For  
        CLEAN UP Sk  
    End For  
  
Output min{ max{α, β} | [α, β] in Sn }
```

State Space S_k

Δ^i Δ^{i+1} Δ^{i+2} Δ^{i+3} Δ^{i+4}



$P := \sum_j \min\{p_{1j}, p_{2j}\}$
 State space in
 $P \times P$ box

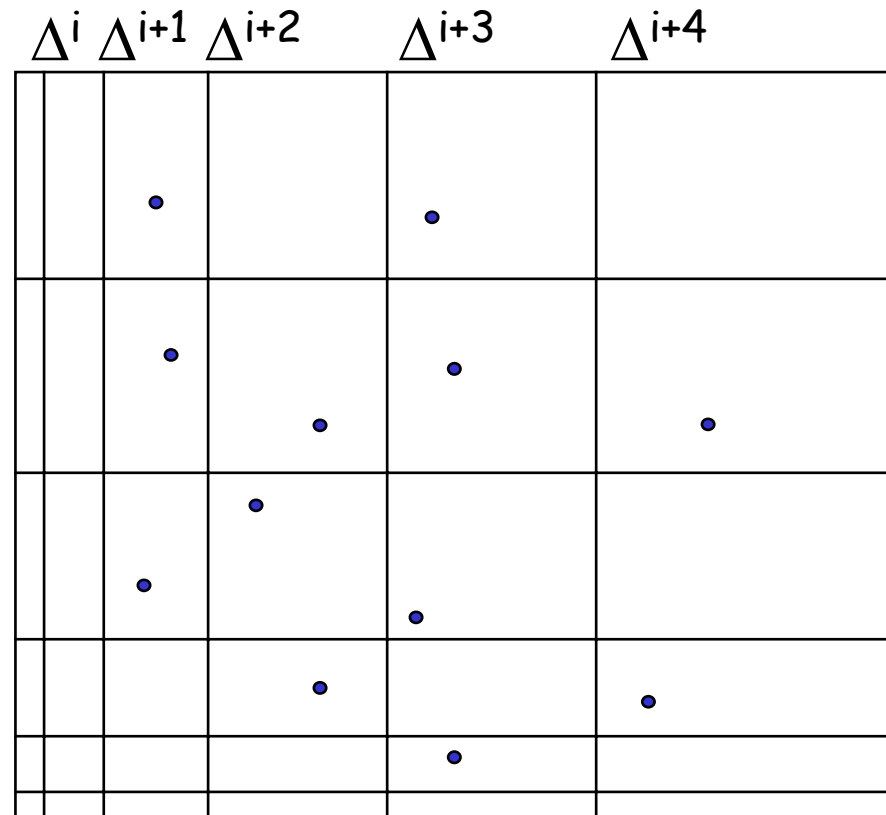
 $\Delta := (1+\varepsilon)^{1/n}$
 $\Delta, \Delta^2, \Delta^3, \dots$

Clean up S_k

$P := \sum_j \min\{p_{1j}, p_{2j}\}$
 State space in
 $P \times P$ box

$$\Delta := (1+\varepsilon)^{1/n}$$

$\Delta, \Delta^2, \Delta^3, \dots$



Running Time

The running time is n times the number of boxes.

The number of boxes is at most $(b+1)^2$ where

$$\Delta^b = P$$

Therefore

$$b = \log(P) / \log(\Delta)$$

Since $\Delta = (1+\varepsilon)^{1/n}$ then

$$b \leq n \log(P) / \log(1+\varepsilon) \leq n \log(P) (1+\varepsilon) / \varepsilon$$

How much error is introduced by "clean up"?

Clean up "replaces" state $[\alpha, \beta]$ by another state $[\alpha', \beta']$ in the same box:

$$\alpha/\Delta \leq \alpha' \leq \alpha\Delta$$

$$\beta/\Delta \leq \beta' \leq \beta\Delta$$

Therefore \implies error $\sim \Delta$

There are n "clean up", and each clean up has error $\Delta \implies$ Overall error $\leq \Delta^n = 1 + \epsilon$

In-Approximability Techniques

How do we disprove the existence of a PTAS?



MAIN IDEA:

Prove that the existence of a polynomial time approximation algorithm for problem X with worst case guarantee better than α

implies

the existence of an exact polynomial time algorithm for an NP-hard problem Y

implies

$P=NP$

Main Techniques

FPTAS: Strong NP-hardness \implies no FPTAS

PTAS: Gap-technique \implies no PTAS

(MAX-SNP-hardness \implies no PTAS)

Disproof of an FPTAS

$P||C_{\max}$ Makespan minimization on m identical machines

I: m identical machines
 n jobs with lengths p_1, p_2, \dots, p_n

- Assume p_1, p_2, \dots, p_n are integers encoded in unary (e.g., $p_1=3$ write: $p_1=111$)
- Known: $P||C_{\max}$ is NP-hard even under unary encoding
- $P := \sum p_j$ polynomial in size of input
- $\text{Opt} \leq P$ and Opt is an integer: $[0, 1, 2, \dots, P]$

Disproof of an FPTAS (ctd)

Suppose an FPTAS for $P||C_{\max}$ with time complexity,

$$\text{say, } O\left(\left(\frac{1}{\varepsilon}\right)^a \cdot n^b \cdot m^c \cdot P^d\right)$$

- Choose $\varepsilon = \frac{1}{2P}$ and call FPTAS.
- Running Time: $O(n^b \cdot m^c \cdot P^{d+a})$ polynomial.

$$A(I) \leq (1+\varepsilon)\text{Opt} = (1 + 1/2P) \text{Opt} \leq \text{Opt} + 1/2$$

$$\implies A(I) < \text{Opt} + 1 \quad \implies A(I) = \text{Opt}$$

$$\implies P = \text{NP} !!$$

The Gap Technique

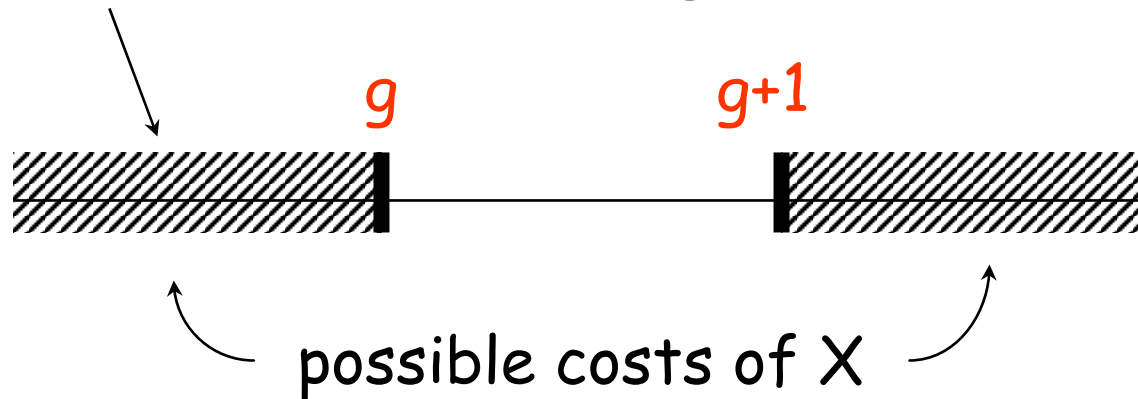


IDEA:

Let X be a minimization with integral objective function values (costs). Let g be a fixed integer. Assume that the problem of deciding whether an instance of X has a feasible solution with cost at most g is NP-hard. Then X does not have an approximation algorithm with ratio $\alpha < (g + 1)/g$, unless $P=NP$.

The Gap Technique (ctd)

hard to decide if the cost is $\leq g$



Assume for any instance I it exists an α -approximation algorithm A with $\alpha < (g + 1)/g$.

- If $\text{Opt}(I) \leq g$ then $A(I) \leq \alpha g < g + 1 \implies \text{YES}$
- If $\text{Opt}(I) \geq g + 1$ then $A(I) \geq g + 1 \implies \text{NO}$

Example

Problem

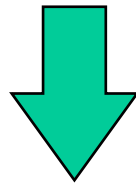
(bin packing or scheduling with deadline)

Input: n jobs with lengths p_1, p_2, \dots, p_n and a hard deadline d

Goal: find the minimum number m of machines on which all jobs can be completed before their deadline d .

Example (ctd)

THEOREM. Deciding if $m \leq 2$ is NP-hard.
PROOF: (use partition).



Gap Technique

For the Bin packing problem **no α -approximation** algorithm is possible with **$\alpha < 3/2$** (= no PTAS!), unless $P=NP$.

References

Books and Tutorials:

- Ausiello et al. (1999), *Complexity and Approximation*, Springer.
- D.S. Hochbaum (eds), (1995), *Approximation Algorithms for NP-hard Problems*, PWS.
- R. Motwani and P. Raghavan (1995), *Randomized Algorithms*, Cambridge.
- P. Schuurman and G. Woeginger (2001), *Approximation Schemes - A Tutorial*.

Conferences:

STOC, FOCS, SODA, ESA, ICALP, IPCO, STACS, SWAT, WADS,
FSTTCS, APPROX, RANDOM, ISAAC, FCT, MFCS, COCOON, LATIN,...