IV. Approximation Algorithms via Exact Algorithms

Thomas Sauerwald

Easter 2020

UNIVERSITY OF CAMBRIDGE

Outline

The Subset-Sum Problem

Parallel Machine Scheduling

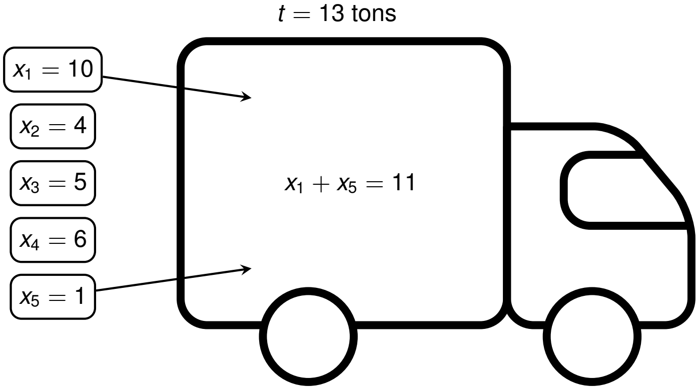
Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)

The Subset-Sum Problem

The Subset-Sum Problem

- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer t
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\, x_i \in S'} x_i \leq t$.

This problem is NP-hard

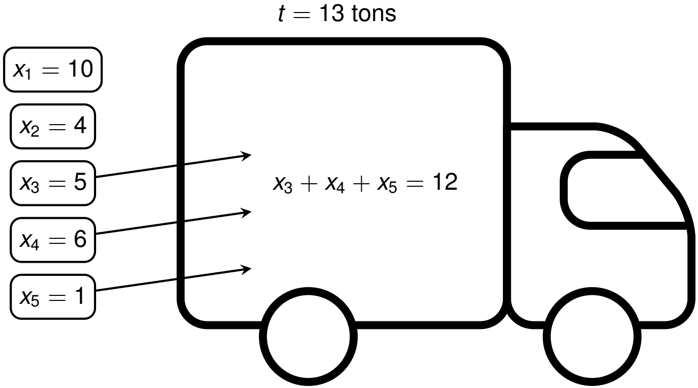


The Subset-Sum Problem

The Subset-Sum Problem

- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer t
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\, x_i \in S'} x_i \leq t$.

This problem is NP-hard



An Exact (Exponential-Time) Algorithm

Dynamic Progamming: Compute bottom-up all possible sums $\leq t$

implementable in time $O(|L_{i-1}|)$ (like Merge-Sort)

Returns the merged list (in sorted order and without duplicates)

$S + x := \{s + x : s \in S\}$

```
EXACT-SUBSET-SUM(S, t)
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i-1}, L_{i-1} + x_i)
    remove from L_i every element that is greater than t
return the largest element in L_n
```

Example:

- $S = \{1, 4, 5\}$, $t = 10$
- $L_0 = \langle 0 \rangle$
- $L_1 = \langle 0, 1 \rangle$
- $L_2 = \langle 0, 1, 4, 5 \rangle$
- $L_3 = \langle 0, 1, 4, 5, 6, 9, 10 \rangle$

An Exact (Exponential-Time) Algorithm

Dynamic Progamming: Compute bottom-up all possible sums $\leq t$

EXACT-SUBSET-SUM(S, t)

```
n = |S|
L_0 = ⟨0⟩
for i = 1 to n
    L_i = MERGE-LISTS(L_{i−1}, L_{i−1
    remove from L_i every element th
return the largest element in L_n
```

can be shown by induction on n

- **Correctness:** L_n contains all sums of $\{x_1, x_2, \ldots, x_n\}$
- **Runtime:** $O(2^1 + 2^2 + \cdots + 2^n) = O(2^n)$

There are 2^i subsets of $\{x_1, x_2, \ldots, x_i\}$.

Better runtime if t and/or $|L_i|$ are small.

Towards a FPTAS

Idea: Don't need to maintain two values in L which are close to each other.

Trimming a List

- Given a trimming parameter $0 < \delta < 1$
- Trimming L yields smaller sublist L' so that for every $y \in L$: $\exists z \in L'$:

$$\frac{y}{1+\delta} \leq z \leq y.$$

- $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$
- $\delta = 0.1$
- $L' = \langle 10, 12, 15, 20, 23, 29 \rangle$

```
TRIM(L, δ)
let m be the length of L
L' = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L'
        last = y_i
return L'
```

TRIM works in time $\Theta(m)$, if L is given in sorted order.

Illustration of the Trim Operation

```
TRIM(L, δ)
1  let m be the length of L
2  L′ = ⟨y_1⟩
3  last = y_1
4  for i = 2 to m
5      if y_i > last · (1 + δ)      // y_i ≥ last because L is sorted
6          append y_i onto the end of L′
7          last = y_i
8  return L′
```

$\delta = 0.1$

After the initialization (lines 1-3)

last

$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

i

$L' = \langle 10 \rangle$

Illustration of the Trim Operation

```
TRIM(L, δ)
let m be the length of L
L′ = ⟨y_1⟩
last = y_1
for i = 2 to m
    if y_i > last · (1 + δ)        // y_i ≥ last because L is sorted
        append y_i onto the end of L′
        last = y_i
return L′
```

$\delta = 0.1$

The returned list L'

last

$L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$

i

$L' = \langle 10, 12, 15, 20, 23, 29 \rangle$

The FPTAS

APPROX-SUBSET-SUM(S, t, ϵ)

$n = |S|$
$L_0 = \langle 0 \rangle$
for $i = 1$ **to** n
  $L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
  $L_i = \text{TRIM}(L_i, \epsilon/2n)$
  remove from L_i every element that is greater than t
let z^* be the largest value in L_n
return z^*

Repeated application of TRIM to make sure L_i's remain short.

EXACT-SUBSET-SUM(S, t)

$n = |S|$
$L_0 = \langle 0 \rangle$
for $i = 1$ **to** n
  $L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
  remove from L_i every element that is greater than t
return the largest element in L_n

- We must bound the inaccuracy introduced by repeated trimming
- We must show that the algorithm is polynomial time

Solution is a careful choice of δ!

Running through an Example (CLRS3)

APPROX-SUBSET-SUM(S, t, ϵ)

1 $n = |S|$
2 $L_0 = \langle 0 \rangle$
3 **for** $i = 1$ **to** n
4 $\quad L_i = \text{MERGE-LISTS}(L_{i-1}, L_{i-1} + x_i)$
5 $\quad L_i = \text{TRIM}(L_i, \epsilon/2n)$
6 $\quad$ remove from L_i every element that is greater than t
7 let z^* be the largest value in L_n
8 **return** z^*

- Input: $S = \langle 104, 102, 201, 101 \rangle$, $t = 308$, $\epsilon = 0.4$

$\Rightarrow$ Trimming parameter: $\delta = \epsilon/(2 \cdot n) = \epsilon/8 = 0.05$

- line 2: $L_0 = \langle 0 \rangle$
- line 4: $L_1 = \langle 0, 104 \rangle$
- line 5: $L_1 = \langle 0, 104 \rangle$
- line 6: $L_1 = \langle 0, 104 \rangle$
- line 4: $L_2 = \langle 0, 102, 104, 206 \rangle$
- line 5: $L_2 = \langle 0, 102, 206 \rangle$
- line 6: $L_2 = \langle 0, 102, 206 \rangle$
- line 4: $L_3 = \langle 0, 102, 201, 206, 303, 407 \rangle$
- line 5: $L_3 = \langle 0, 102, 201, 303, 407 \rangle$
- line 6: $L_3 = \langle 0, 102, 201, 303 \rangle$
- line 4: $L_4 = \langle 0, 101, 102, 201, 203, 302, 303, 404 \rangle$
- line 5: $L_4 = \langle 0, 101, 201, 302, 404 \rangle$
- line 6: $L_4 = \langle 0, 101, 201, 302 \rangle$

Returned solution $z^* = 302$, which is 2% within the optimum $307 = 104 + 102 + 101$

Reminder: Performance Ratios for Approximation Algorithms

Approximation Ratio

An algorithm for a problem has approximation ratio $\rho(n)$, if for any input of size n, the cost C of the returned solution and optimal cost C^* satisfy:

$$\max\left(\frac{C}{C^*}, \frac{C^*}{C}\right) \leq \rho(n).$$

For many problems: tradeoff between runtime and approximation ratio.

Approximation Schemes

An approximation scheme is an approximation algorithm, which given any input and $\epsilon > 0$, is a $(1+\epsilon)$-approximation algorithm.

- It is a polynomial-time approximation scheme (PTAS) if for any fixed $\epsilon > 0$, the runtime is polynomial in n. For example, $O(n^{2/\epsilon})$.
- It is a fully polynomial-time approximation scheme (FPTAS) if the runtime is polynomial in both $1/\epsilon$ and n. For example, $O((1/\epsilon)^2 \cdot n^3)$.

Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Approximation Ratio):

- Returned solution z^* is a valid solution ✓
- Let y^* denote an optimal solution
- For every possible sum $y \leq t$ of $x_1, \ldots, x_i$, there exists an element $z \in L'_i$ s.t.:

$$\frac{y}{(1+\epsilon/(2n))^i} \leq z \leq y \quad \overset{y=y^*, i=n}{\Rightarrow} \quad \frac{y^*}{(1+\epsilon/(2n))^n} \leq z \leq y^*$$

Can be shown by induction on i

$$\frac{y^*}{z} \leq \left(1 + \frac{\epsilon}{2n}\right)^n,$$

and now using the fact that $\left(1 + \frac{\epsilon/2}{n}\right)^n \overset{n\to\infty}{\longrightarrow} e^{\epsilon/2}$ yields

$$\frac{y^*}{z} \leq e^{\epsilon/2}$$

Taylor approximation of e

$$\leq 1 + \epsilon/2 + (\epsilon/2)^2 \leq 1 + \epsilon$$

Analysis of APPROX-SUBSET-SUM

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

Proof (Running Time):

- Strategy: Derive a bound on $|L_i|$ (running time is linear in $|L_i|$)
- After trimming, two successive elements z and z' satisfy $z'/z \geq 1 + \epsilon/(2n)$

$\Rightarrow$ Possible Values after trimming are 0, 1, and up to $\lfloor \log_{1+\epsilon/(2n)} t \rfloor$ additional values. Hence,

$$\log_{1+\epsilon/(2n)} t + 2 = \frac{\ln t}{\ln(1+\epsilon/(2n))} + 2$$

$$\leq \frac{2n(1+\epsilon/(2n)) \ln t}{\epsilon} + 2$$

$$< \frac{3n \ln t}{\epsilon} + 2.$$

For $x > -1$, $\ln(1+x) \geq \frac{x}{1+x}$

- This bound on $|L_i|$ is polynomial in the size of the input and in $1/\epsilon$. □

Need $\log(t)$ bits to represent t and n bits to represent S

Concluding Remarks

The Subset-Sum Problem

- Given: Set of positive integers $S = \{x_1, x_2, \ldots, x_n\}$ and positive integer t
- Goal: Find a subset $S' \subseteq S$ which maximizes $\sum_{i:\, x_i \in S'} x_i \leq t$.

Theorem 35.8

APPROX-SUBSET-SUM is a FPTAS for the subset-sum problem.

A more general problem than Subset-Sum

The Knapsack Problem

- Given: Items $i = 1, 2, \ldots, n$ with weights w_i and values v_i, and integer t
- Goal: Find a subset $S' \subseteq S$ which
  1. maximizes $\sum_{i \in S'} v_i$
  2. satisfies $\sum_{i \in S'} w_i \leq t$

Algorithm very similar to APPROX-SUBSET-SUM

Theorem

There is a FPTAS for the Knapsack problem.

Outline

The Subset-Sum Problem

Parallel Machine Scheduling

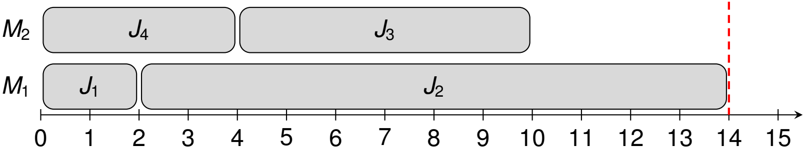
Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)

Parallel Machine Scheduling

Machine Scheduling Problem

- Given: n jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and m identical machines $M_1, M_2, \ldots, M_m$
- Goal: Schedule the jobs on the machines minimizing the makespan $C_{\max} = \max_{1 \leq j \leq n} C_j$, where C_k is the completion time of job J_k.

- J_1: $p_1 = 2$
- J_2: $p_2 = 12$
- J_3: $p_3 = 6$
- J_4: $p_4 = 4$



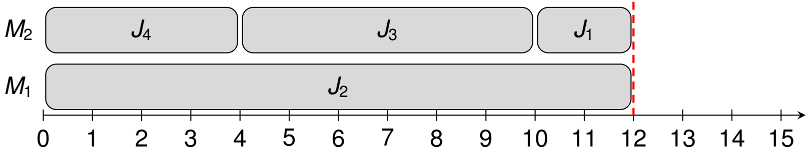
Parallel Machine Scheduling

Machine Scheduling Problem

- Given: n jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$, and m identical machines $M_1, M_2, \ldots, M_m$
- Goal: Schedule the jobs on the machines minimizing the makespan $C_{\max} = \max_{1 \leq j \leq n} C_j$, where C_k is the completion time of job J_k.

- J_1: $p_1 = 2$
- J_2: $p_2 = 12$
- J_3: $p_3 = 6$
- J_4: $p_4 = 4$

For the analysis, it will be convenient to denote by C_i the completion time of a machine i.

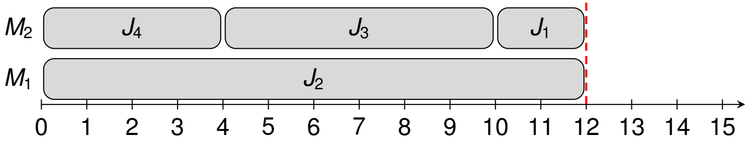


NP-Completeness of Parallel Machine Scheduling

Lemma

Parallel Machine Scheduling is NP-complete even if there are only two machines.

Proof Idea: Polynomial time reduction from NUMBER-PARTITIONING.



Equivalent to the following Online Algorithm [CLRS3]:
Whenever a machine is idle, schedule the next job on that machine.

LIST SCHEDULING$(J_1, J_2, \ldots, J_n, m)$

while there exists an unassigned job
    Schedule job on the machine with the least load

How good is this most basic Greedy Approach?

List Scheduling Analysis (Observations)

Ex 35-5 a.&b.

a. The optimal makespan is at least as large as the greatest processing time, that is,

$$C^*_{\max} \geq \max_{1 \leq k \leq n} p_k.$$

b. The optimal makespan is at least as large as the average machine load, that is,

$$C^*_{\max} \geq \frac{1}{m} \sum_{k=1}^{n} p_k.$$

Proof:

b. The total processing times of all n jobs equals $\sum_{k=1}^{n} p_k$

$\Rightarrow$ One machine must have a load of at least $\frac{1}{m} \cdot \sum_{k=1}^{n} p_k$ $\square$

List Scheduling Analysis (Final Step)

Ex 35-5 d. (Graham 1966)

For the schedule returned by the greedy algorithm it holds that

$$C_{\max} \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k.$$

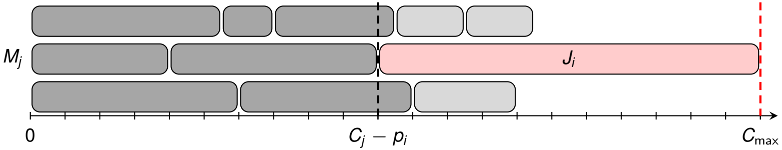
Hence list scheduling is a poly-time 2-approximation algorithm.

Proof:

- Let J_i be the last job scheduled on machine M_j with $C_{\max} = C_j$
- When J_i was scheduled to machine M_j, $C_j - p_i \leq C_k$ for all $1 \leq k \leq m$
- Averaging over k yields:

Using Ex 35-5 a. & b.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{m} C_k = \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq \frac{1}{m} \sum_{k=1}^{n} p_k + \max_{1 \leq k \leq n} p_k \leq 2 \cdot C^*_{\max}$$



Improving Greedy

The problem of the List-Scheduling Approach were the large jobs

Analysis can be shown to be almost tight. Is there a better algorithm?

```
LEAST PROCESSING TIME(J_1, J_2, ..., J_n, m)
Sort jobs decreasingly in their processing times
for i = 1 to m
    C_i = 0
    S_i = ∅
end for
for j = 1 to n
    i = argmin_{1≤k≤m} C_k
    S_i = S_i ∪ {j}, C_i = C_i + p_j
end for
return S_1, ..., S_m
```

Runtime:

- $O(n \log n)$ for sorting
- $O(n \log m)$ for extracting (and re-inserting) the minimum (use priority queue).

Analysis of Improved Greedy

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

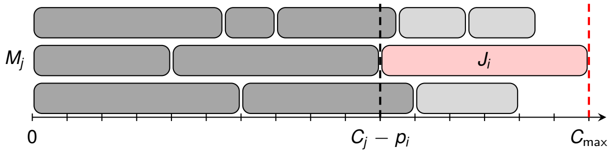
This can be shown to be tight (see next slide).

Proof (of approximation ratio 3/2).

- Observation 1: If there are at most m jobs, then the solution is optimal.
- Observation 2: If there are more than m jobs, then $C^*_{\max} \geq 2 \cdot p_{m+1}$.
- As in the analysis for list scheduling, we have

$$C_{\max} = C_j = (C_j - p_i) + p_i \leq C^*_{\max} + \frac{1}{2} C^*_{\max} = \frac{3}{2} C_{\max}. \qquad \square$$

This is for the case $i \geq m+1$ (otherwise, an even stronger inequality holds)



Tightness of the Bound for LPT

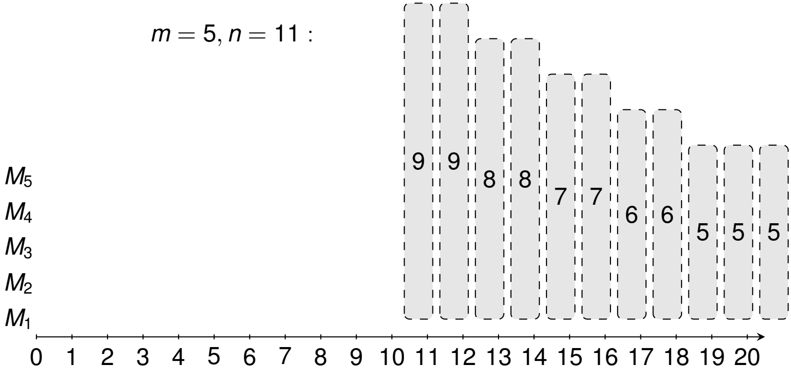
Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- m machines and $n = 2m + 1$ jobs:
- two of length $2m-1, 2m-2, \ldots, m$ and one extra job of length m

$m = 5, n = 11$:



Tightness of the Bound for LPT

Graham 1966

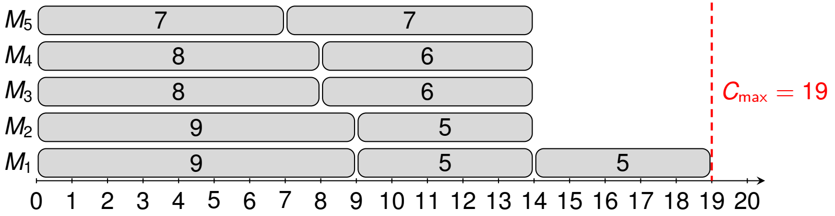
The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Proof of an instance which shows tightness:

- m machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length m

$m = 5, n = 11$:

LPT gives $C_{\max} = 19$



Tightness of the Bound for LPT

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

$\frac{19}{15} = \frac{20}{15} - \frac{1}{15}$

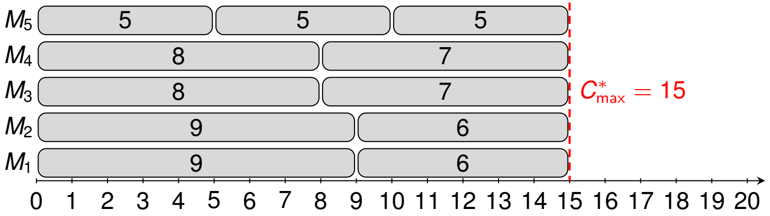
Proof of an instance which shows tightness:

- m machines and $n = 2m + 1$ jobs:
- two of length $2m - 1, 2m - 2, \ldots, m$ and one extra job of length m

$m = 5, n = 11$:

LPT gives $C_{\max} = 19$

Optimum is $C^*_{\max} = 15$



Conclusion

Graham 1966

List scheduling has an approximation ratio of 2.

Graham 1966

The LPT algorithm has an approximation ratio of $4/3 - 1/(3m)$.

Theorem (Hochbaum, Shmoys'87)

There exists a PTAS for Parallel Machine Scheduling which runs in time $O(n^{O(1/\epsilon^2)} \cdot \log P)$, where $P := \sum_{k=1}^{n} p_k$.

Can we find a FPTAS (for polynomially bounded processing times)?
No!

Because for sufficiently small approximation ratio $1 + \epsilon$, the computed solution has to be optimal, and Parallel Machine Scheduling is strongly NP-hard.

Exercise (easy): Run the LPT algorithm on three machines and jobs having processing times $\{3, 4, 4, 3, 5, 3, 5\}$. Which allocation do you get?

1. $[3, 3, 5], [4, 5], [4, 3]$
2. $[5, 3], [5, 4], [4, 3, 3]$
3. $[3, 3, 3], [5, 4], [5, 4]$

Outline

The Subset-Sum Problem

Parallel Machine Scheduling

Bonus Material: A PTAS for Parallel Machine Scheduling (non-examinable)

A PTAS for Parallel Machine Scheduling

Basic Idea: For $(1+\epsilon)$-approximation, don't have to work with exact p_k's.

SUBROUTINE$(J_1, J_2, \ldots, J_n, m, T)$

1: Either: **Return** a solution with $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$

2: Or: **Return** there is no solution with makespan $< T$

We will prove this on the next slides.

Key Lemma

SUBROUTINE can be implemented in time $n^{O(1/\epsilon^2)}$.

Theorem (Hochbaum, Shmoys'87)

There exists a PTAS for Parallel Machine Scheduling which runs in time $O(n^{O(1/\epsilon^2)} \cdot \log P)$, where $P := \sum_{k=1}^{n} p_k$.

polynomial in the size of the input

Since $0 \leq C^*_{\max} \leq P$ and $C^*_{\max}$ is integral, binary search terminates after $O(\log P)$ steps.

Proof (using Key Lemma):

PTAS$(J_1, J_2, \ldots, J_n, m)$

1: Do binary search to find smallest T s.t. $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$.

2: **Return** solution computed by SUBROUTINE$(J_1, J_2, \ldots, J_n, m, T)$

Implementation of Subroutine

SUBROUTINE$(J_1, J_2, \ldots, J_n, m, T)$

1: Either: **Return** a solution with $C_{\max} \leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\}$

2: Or: **Return** there is no solution with makespan $< T$

Observation

Divide jobs into two groups: $J_{\text{small}} = \{i \colon p_i \leq \epsilon \cdot T\}$ and $J_{\text{large}} = [n] \setminus J_{\text{small}}$. Given a solution for J_{large} only with makespan $(1+\epsilon) \cdot T$, then greedily placing J_{small} yields a solution with makespan $(1+\epsilon) \cdot \max\{T, C^*_{\max}\}$.

Proof:

- Let M_j be the machine with largest load
- If there are no jobs from J_{small}, then makespan is at most $(1+\epsilon) \cdot T$.
- Otherwise, let $i \in J_{\text{small}}$ be the last job added to M_j.

$$C_j - p_i \leq \frac{1}{m} \sum_{k=1}^{n} p_k \quad \Rightarrow \quad C_j \leq p_i + \frac{1}{m} \sum_{k=1}^{n} p_k$$

$$\leq \epsilon \cdot T + C^*_{\max}$$

$$\leq (1+\epsilon) \cdot \max\{T, C^*_{\max}\} \quad \square$$

the "well-known" formula

Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule J_{large} with makespan $(1+\epsilon)\cdot T$.

- Let b be the smallest integer with $1/b \leq \epsilon$. Define processing times $p'_i = \left\lceil \frac{p_j b^2}{T} \right\rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p'_i = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

Can assume there are no jobs with $p_j \geq T$!

- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

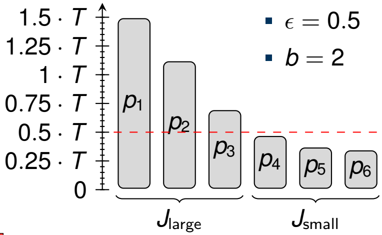
Assignments to one machine with makespan $\leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

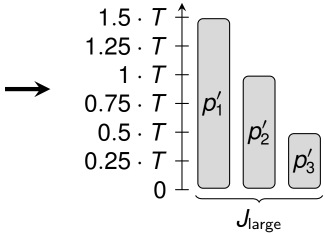
Assign some jobs to one machine, and then use as few machines as possible for the rest.

$$f(0,0,\ldots,0) = 0$$

$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2})\in\mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).$$



- $\epsilon = 0.5$
- $b = 2$



Proof of Key Lemma (non-examinable)

Use Dynamic Programming to schedule J_{large} with makespan $(1+\epsilon)\cdot T$.

- Let b be the smallest integer with $1/b \leq \epsilon$. Define processing times $p_i' = \lceil \frac{p_j b^2}{T} \rceil \cdot \frac{T}{b^2}$

$\Rightarrow$ Every $p_i' = \alpha \cdot \frac{T}{b^2}$ for $\alpha = b, b+1, \ldots, b^2$

- Let $\mathcal{C}$ be all $(s_b, s_{b+1}, \ldots, s_{b^2})$ with $\sum_{i=j}^{b^2} s_j \cdot j \cdot \frac{T}{b^2} \leq T$.

- Let $f(n_b, n_{b+1}, \ldots, n_{b^2})$ be the minimum number of machines required to schedule all jobs with makespan $\leq T$:

$$f(0,0,\ldots,0) = 0$$
$$f(n_b, n_{b+1}, \ldots, n_{b^2}) = 1 + \min_{(s_b, s_{b+1}, \ldots, s_{b^2}) \in \mathcal{C}} f(n_b - s_b, n_{b+1} - s_{b+1}, \ldots, n_{b^2} - s_{b^2}).$$

- Number of table entries is at most n^{b^2}, hence filling all entries takes $n^{O(b^2)}$
- If $f(n_b, n_{b+1}, \ldots, n_{b^2}) \leq m$ (for the jobs with p'), then return yes, otherwise no.
- As every machine is assigned at most b jobs ($p_i' \geq \frac{T}{b}$) and the makespan is $\leq T$,

$$C_{\max} \leq T + b \cdot \max_{i \in J_{\text{large}}} (p_i - p_i')$$
$$\leq T + b \cdot \frac{T}{b^2} \leq (1+\epsilon) \cdot T. \qquad \square$$