Problem

(1) Makespan Minimization on Identical Machines

Input:
- \( n \) jobs with processing times \( p_1, p_2, \ldots, p_n \)
- \( m \) machines.

Identical: Each job \( j \) takes same time on each machine.

If a set \( S \subseteq \{1, \ldots, n\} \) is scheduled on a machine, the total processing time is the sum of processing times.

Output: Allocation of jobs to machines.

Objective: Minimize makespan: \( \max_{\text{proc. time on any m/c}} \)

Formally, find a partition \( S_1, S_2, \ldots, S_m \)

\[ \{1, \ldots, n\} \]

\[ \text{of} \]

\[ \max_{\text{proc. time on any m/c}} \]
\[
\max \sum_{i=1}^{m} \sum_{j \in S_i} p_{ij}
\] is minimized.

**Graham Notation**

This problem is \( P \mid \mid C_{\text{max}} \)

- Identical machines
- Makespan
- No extra constraints.

For instance, we could have a machine dependent running time, \( p_{ij} \) - job \( j \) proc. time on machine \( i \), and we could’ve wanted to minimize the same obj. That prob is called \( R \mid \mid C_{\text{max}} \).

Today, we stick to \( P \mid \mid C_{\text{max}} \)

**Algorithm** (The “oldest” approx alg) ’1966 (Graham)

- Order the jobs in any order.
• Schedule job $j$ on the machine which has the least load so far.

Analysis

• Lower Bounds on OPT

\[
\begin{align*}
\max_p &:= \max_j p_j \\
\overline{p} &:= \frac{1}{m} \sum_{j=1}^{n} p_j
\end{align*}
\]

\[
L := \max (\max_p, \overline{p})
\]

\text{Obs: } \text{OPT} \geq L

We will use this easier lower bound to compare our algo. It may seem like we're taking a bit, but we have a better handle on it.

Back to analysis:

Let $i$ be the most loaded m/c. Consider the last job $j$ being added to $i$. Let $P$ be the load on m/c $i$ just before $j$ was added.

\[
- \text{ ALG } = P + p_j \leq P + L
\]
- \( P \leq \text{load on any other } m/c \) when \( j \) was added
  \[ \Rightarrow \quad P \leq \frac{1}{m} \sum_{i:j} p_{i,j} \leq L \]
  \[ \therefore \quad \text{ALG} \leq 2L \]

**Theorem:** The above algorithm is a factor 2 approx. alg. for \( P||C_{\text{max}} \).

A slightly “simpler” problem: \( P2||C_{\text{max}} \)

So, \( m = 2 \). Only 2 m/c.
The problem is still NP-hard.

(Do you see this?)

**Classification:**
- Fix any \( \varepsilon > 0 \).
- Job \( j \) is \textbf{big} if \( p_{i,j} > \varepsilon L \)
  \[ \text{small, o/w.} \]

**Partitioning Output Space:**
- \( B \) be the set of big jobs.
- Any schedule in particular partitions $B = B_1 \cup B_2$
- Let $(B_1^*, B_2^*)$ be the partition of $B$ in the optimal schedule.

*We can afford to guess this partition.*

Why? How many partitions are there? $|B|$

2

How large is $|B|$?

$2L \geq \sum_{j \in B} p_j \geq |B| \cdot 3L \Rightarrow |B| \leq \frac{2}{3}$

\[|B| \leq 2/3\]

- The # of partitions of $|B| \leq 2^{2/3} = O_\varepsilon(1)$

- By enumerating, we may assume we have guessed / known $(B_1^*, B_2^*)$

Note: in the future, whenever we say...
“Guess”, be aware how much time this guess takes. In this case, the guess is via enumeration and takes $2^{O(1/x)}$ time.

**Algorithm:**

1. Guess $(B_1^*, B_2^*)$ \(\text{//} 2^{O(1/x)} \text{ time via enum.} \)

2. Order small jobs in any order and schedule on the heaviest loaded m/c ... exactly like before.

**Analysis**

Two Cases.

1. The max-loaded m/c gets no small jobs. In that case,

   \[ \text{ALG} \leq \max (p(B_1^*), p(B_2^*)) \leq \text{OPT} \]

2. The max-loaded m/c gets at least one small job $j$. Now we can repeat the above argument ...

   \[ \text{ALG} \leq P + P_j \leq P + \epsilon L \]

   \[ \leq (1 + \epsilon) L \]
\[ \leq (1 + \varepsilon)^7 \leq (1 + \varepsilon) \text{OPT} \]

**Theorem**: For any \( \varepsilon > 0 \), the above algorithm is an \((1 + \varepsilon)\)-approximation running in time \( \text{poly}(n) \cdot 2^{O(1/\varepsilon)} \).

**Definitions**

**Polynomial Time Approximation Scheme (PTAS)**:

An alg. which takes an extra parameter \( \varepsilon > 0 \) as input and returns an \((1 + \varepsilon)\)-factor appx. in time which is a polynomial in \( n \) as long as \( \varepsilon \) is a constant.

Example running times: \( n^{1/\varepsilon} \), \( n^{2\cdot 1/\varepsilon} \), \( n^{1/\varepsilon} \cdot n/\varepsilon \)

**Fully Polynomial Time Approximation Scheme (FPTAS)**

These are PTASes but their
These are PTAS es, but their running times are poly \( (n, \frac{1}{\varepsilon}) \).

So, even if \( \varepsilon \approx \frac{1}{10} \), the scheme runs in polynomial time.

An FPTAS is the best you can hope for for an NP-hard problem.

So our above theorem can be succinctly stated as

Thm: \( \text{P2||C_{max}} \) has a PTAS

\[ P1||C_{max} \text{ with many machines} \]

1. \( \text{(Multiplicative Scaling)} \)

“Round up” all job sizes \( p_j \) to the smallest power of \((1 + \varepsilon)\) which is larger than \( p_j \).

- \( \text{opt}(X_{\text{old}}) \leq \text{opt}(X_{\text{new}}) \leq (1 + \varepsilon) \cdot \text{opt}(X_{\text{old}}) \)

- Given any soln. to \( X_{\text{new}} \), of makespan
(Guessing OPT) Till now we assumed we didn't know OPT... the number.
But we keep working with a 'guess'
and bump it by a factor \((1+\epsilon)\) if the final solution isn't within \(\leq (1+\epsilon)\)
times guess.

(Bucketing Big Jobs)
- Job \(j\) is big if \(p_j > \epsilon \cdot \text{OPT guess}\).
- \(B = \text{Set of big jobs}\).
- \(B_t \leq B\) with the same proc. time \(1 \leq t \leq N\).
- How big is \(N\)?
  \[ \log \left( \frac{1}{\epsilon} \right) \]
  \[ \approx \frac{1}{\epsilon} \log \frac{1}{\epsilon} \]
  \[ \log(1+\epsilon) \approx \epsilon \] if \(\epsilon\) is small...

(Profile of big jobs)
Every feasible schedule gives each machine a “profile”

\[ \overline{V} = (V_1, V_2, \ldots, V_N) \]

where \( V_t \in \mathbb{Z}^+ \)
denotes the \# of jobs of \( B_t \) is
given to this m/c.

Each big job \( \geq 3 \cdot \text{OPT} \) \( \Rightarrow \sum_{t=1}^{N} V_t \leq \frac{1}{3} \)

How many feasible profiles are there?

If \( \overline{V} \) is collection of all feas.
profiles,

\[ M = |\overline{V}| \leq \left( \frac{1}{3} \right)^N \approx 2^{\frac{1}{3} \log^2 \frac{1}{3}} \]

is a constant,

\( \text{super cubic} \)

(\text{Histogram of profiles})
Let's order the profiles of $W$ as $\bar{V}^{(1)}, \bar{V}^{(2)}, \ldots, \bar{V}^{(M)}$.

Any feasible schedule allocates each machine one of these profiles.

Let $h_i$ be the $\#$ of m/c's which pick the profile $\bar{V}^{(i)}$.

Here we use the "parallelness" of machines. It doesn't matter who chooses which of these $h_i$ profiles of type $\bar{V}^{(i)}$.

Consistency Conditions:

1. $\sum_{i=1}^{M} h_i = m$

2. $\sum_{i=1}^{M} h_i \bar{V}^{(i)} = |B_t| \quad \forall 1 \leq t \leq N$

# of big jobs of type $B_t$ alloc. among the m/c's.
(Guessing OPT's profile)

Let \((h^*_1, \ldots, h^*_M)\) be the histogram of \(OPT\). **Claim:** We can 'guess' \(h^*\) by enumeration.

How many consistent histograms are there?

\[
M^M \leq 2^{\frac{1}{\varepsilon} \log^2 \frac{1}{\varepsilon}} \text{ constant!}
\]

Once we guess \(OPT\)'s histogram, we give \(h_i^*\) arbitrary \(m/c\)'s the profile \(\mathcal{H}(i)\). Since its consistent, all big jobs are allocated.

The small jobs we allocate greedily. Just as in the \(P2||C_{\max}\) case, we get an \((1 + \varepsilon) \cdot OPT\) makespan.