Knapsack Problem

17 February 2021 08:03

Given 'n' items i with item i having profit $p_i \geq 0$ and size $s_i \geq 0$, and budget $B \geq 0$,

Choose a subset $X \subseteq [n]$ of items

s.t. $\sum_{i \in X} s_i \leq B$ [total size fits into the bag], and total profit $\sum_{i \in X} p_i$ is maximized

input bits = $\sum_{i \in [n]}$ bit complexity(s_i, p_i)

Polytime Algo ⇒ polynomial running time in # input bits.

GOAL for TODAY

PTAS (Polynomial-Time Approximation Scheme) for Knapsack.

Given any CONSTANT $\epsilon \geq 0$, the algorithm runs in time poly(inputsize). and computes a soln. with profit $\geq (1-\epsilon)$. OPTIMAL VALUE.

Ex: Algo can runtime $n^{1/\epsilon}$ is allowed

Ex: Algo can runtime n

Algo w/ runtime $\left(\frac{1}{\varepsilon}\right)^n$ is not allowed

Even better if Algo runs in time $\text{poly}(n, 1/\varepsilon)$.

FULLY POLYNOMIAL-TIME APPROXIMATION SCHEMES (FPTAS).

$\boxed{\text{Allows for a clean trade-off b/w solution Quality and Running Time}}$

No real limit on how good our solutions can be.

ALGORITHMS for KNAPSACK

① Greedy: Choose item w/ max $\frac{p_i}{s_i}$, insert it and repeat.

Can be problematic in some cases.

① In second round, chosen item can have size > remaining budget, so need to filter it out.

In fact, it's an issue in all the rounds.

A:

...

the routers.

Q: Can we be making a mis-step by filtering out these items in step 2 onwards

Ex: budget $= 100$

Item ①: $p_1 = 10$, $s_1 = 1$

Item ②: $p_2 = 999$, $s_2 = 100$

But algo is not too bad if we allow a slight violation of the constraint (let's say factor 2).

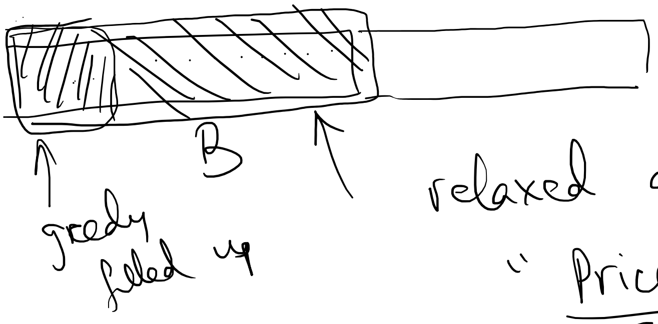
But in real-life some constraints are HARD, cannot be violated.

We'll devise an FPTAS based on dynamic Programming.

I think greedy algo will achive $\geq$ Optimal profit if it can use up to 2-times the budget.

" (but OPT - sol^n must respect the budget).

The previous algo was bad only b/c it had unused capacity - which it couldn't fill



relaxed greedy has as good "$\frac{\text{Price}}{\text{Size}}$" ratio as any item in OPT, for the first B size.

Try to formalize this proof

$\Rightarrow$ No item which OPT selects will be filtered out by our algorithm, for the first B size.

Back to dynamic Programming

Lets consider knapsack, but all profits p_i are integer valued and $0 \le p_1 \le p_2 \le p_3 \le \cdots \le p_n$.

Sizes & budget can be real-valued still.

Goal: Find DP for knapsack which runs in time

poly (n, P_n)

Let $V(i, p)$ denote the size of the minimum size subset of [i] items which achieves a profit of exactly p

first i items

In general, recursive form of

$$V(i,p) = \min\left(V(i-1, p),\ V(i-1, p-p_i) + s_i\right)$$

and base case:

$$V(1, p_1) = s_1$$

$$V(1, p) = \infty \quad \forall\ p \neq p_1 \text{ or } 0$$

$$V(1, 0) = 0$$

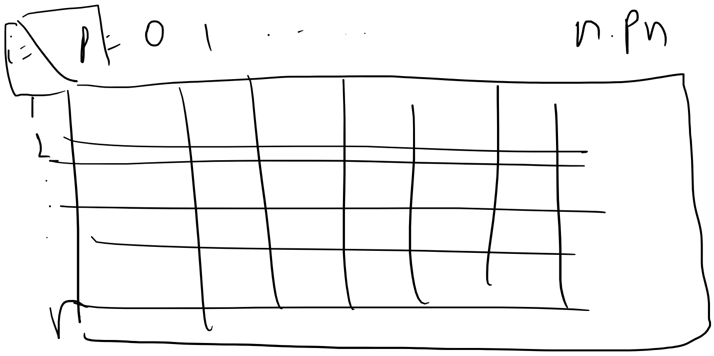
Eventually we want max P s.t.

$$V(n, P) \leq B$$

Clearly desired $P \leq nP_n$, Upper bound on max profit of OPT

Total run time $\propto$ # cells in DP table

$\propto \boxed{n^2 \cdot p_n}$.



"Add a ROUNDING / DISCTRETIZATION" step to reduce a general knapsack instance to this form of integer-valued instance

Example

$$p_1 = p_2$$

$$s_1 > s_2$$

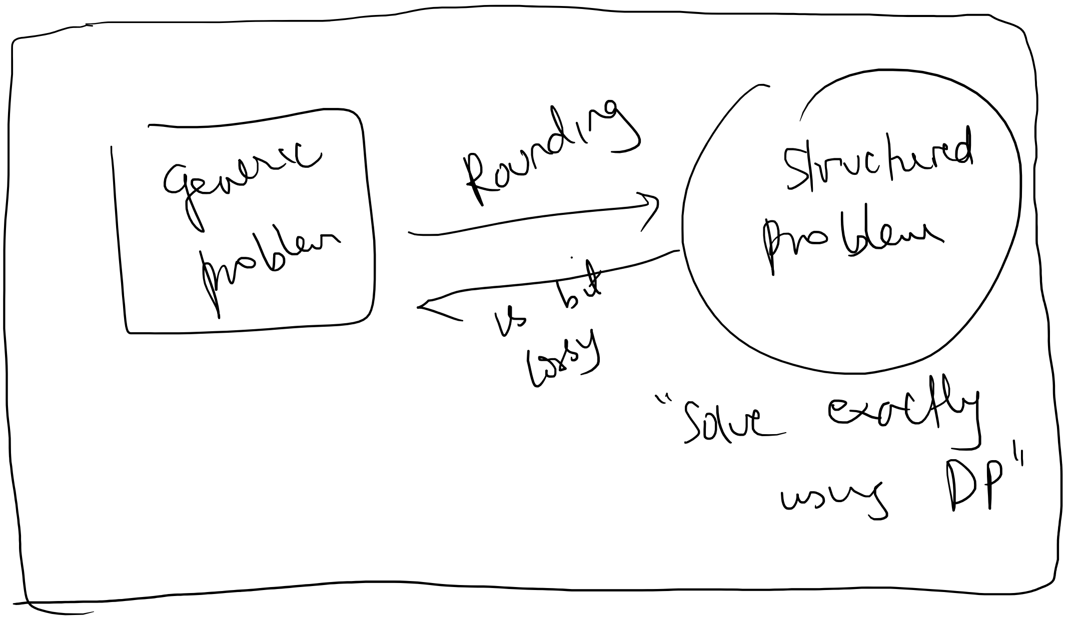
$$B = s_2$$

$$V(2, p_1) = \min\big(V(1, p_1),\ V(1,0) + s_2\big)$$

$$= \min(s_1, s_2)$$

$= \min(s_1, s_2)$

$= s_2$



From general instances to "discretized" instances

Given n items $0 \le p_1 \le \dots \le p_n$ (need not be integers) and sizes $s_1, s_2, \dots, s_n$, how do we convert this to a discrete instance on the profits?

Original instance I

Firstly, recall that $OPT \le n \cdot p_n$

Moreover, let's assume that all sizes $s_i \le B$ (else we can discard such items)

$\Rightarrow OPT \geq P_n$

Now, consider items which are profit $< \frac{\varepsilon \cdot P_n}{n}$

Idea: Even if all these items are aggregated, the total ~~profit they~~ εP_n $\leq \varepsilon \cdot OPT$

---

what if we "round down" all profits to the nearest multiple of $\frac{\varepsilon P_n}{n}$??

For each item i, set $\hat{P}_i$ to be "rounded-down" value.

Instance $\hat{I}$

Now, for any subset of items S, what is

$$0 \leq \sum_{i \in S} \left(P_i - \hat{P}_i \right) \leq \frac{\varepsilon \cdot P_n}{n} \cdot |S| \leq \varepsilon P_n \leq \varepsilon \cdot OPT$$

In particular,

the optimal value of new instance $\widehat{OPT}$

$\geq (1-\varepsilon)\, OPT$

Moreover, since we are only rounding down values, a good soln in new instance will be at least as good in original problem also.

In new instance, all profits are Integral multiples of $\frac{\varepsilon P_n}{n}$.

Let's focus on an "equivalent" instance $\bar{I}$ where item i has

profit $\bar{P}_i = \dfrac{\hat{P}_i}{\left(\frac{\varepsilon P_n}{n}\right)}$

$\Rightarrow$

$$\overline{OPT} = \frac{\widehat{OPT}}{\left(\frac{\varepsilon P_n}{n}\right)}$$

Obs ① $\bar{P}_i$ are integral for all i

Obs ② $\max \bar{P}_i \leq \dfrac{P_n\, n}{\varepsilon P_n} \leq \dfrac{n}{\varepsilon}$.

$\Rightarrow$ can use dynamic program to

Runtime of DP to solve $\bar{I}$
exactly solve the $\leq O\left(n^2 \cdot \frac{n}{\varepsilon}\right)$ instance $\bar{I}$

$= O(n^3/\varepsilon)$

Time { Polytime regardless of how large/small the values can be!

DP for $\bar{I}$ finds a sol'n with profit $\geq \overline{OPT} = \frac{\widehat{OPT}}{\varepsilon p_n/n}$

Same sol'n has profit $\geq \widehat{OPT}$ for $\hat{I}$

$\Rightarrow$ Same sol'n has profit $\geq \widehat{OPT}$ for I

$\geq (1-\varepsilon) \cdot OPT$ ∎

Point raised in class:
To compute residual instance $\hat{P}$,
we need to find

$\lfloor P \rfloor_{\frac{\varepsilon P_n}{n}}$

How much time does this operation take?

Can do some form of long division $\text{poly}(\log P_n)$ time.

Run Time of overall algo $= \text{poly}(n, \log P_n, \frac{1}{\varepsilon})$

FPTAS (fully poly-time Apx Scheme).

One could sometimes settle for

PTAS

(where dependence on $\frac{1}{\varepsilon}$ could be very bad)

Turns out there are some problems where neither FPTAS or PTAS is possible, if $P \neq NP$

Example

set cover

3SAT

HARDNESS OF APPROXIMATION

[Feige, Moshkovitz],

There exists constant $\varepsilon > 0$ s.t. if we have a poly-time $(1-\varepsilon)\ln n$ approximation algorithm for Set Cover, then we can solve 3SAT in poly-time.

[i.e. $P = NP$].

Similarly,

$\exists\ \varepsilon > 0$ st $\left(\frac{7}{8}+\varepsilon\right)$-approx for Max3SAT

$\Rightarrow$ we can solve 3SAT in poly time ($P = NP$)

PCP Theorem

Miracle, because it says exact 3SAT is no harder than $(7/8+\varepsilon)$-approximation

[Håstad, 2000]

$\rightarrow$ NP-Complete to do better

7/8 apx trivial in poly time

Similarly set cover

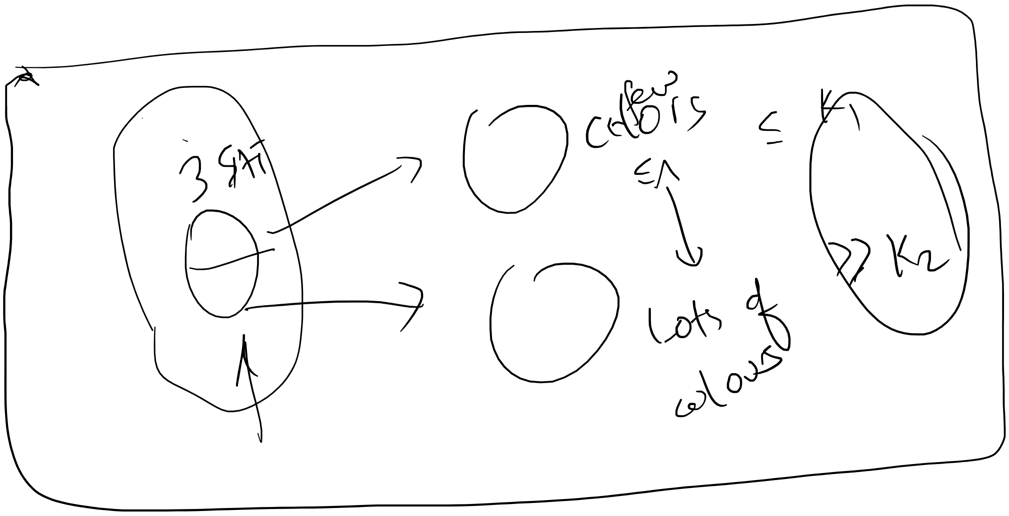
$(1-\varepsilon)\ln(n)$

NP-complete $\times$ $\longrightarrow$ ✓

NP complete ✗ — $\ln n$ approx (greedy) — ✓

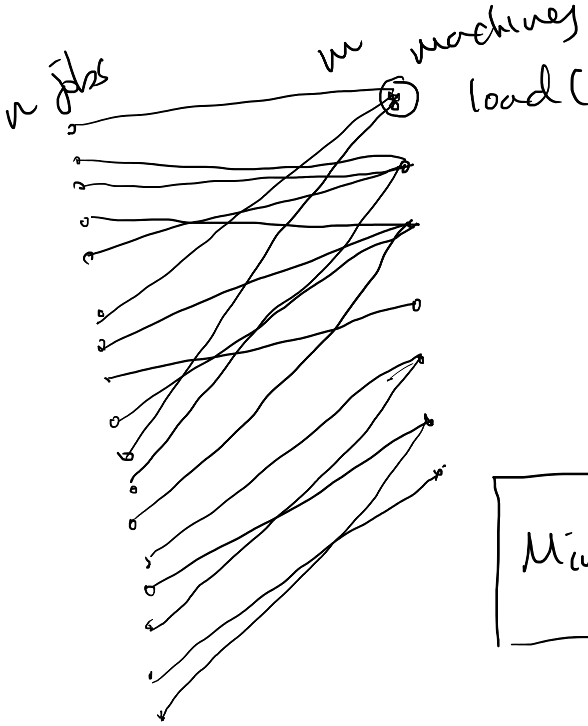
Graph colouring (extremely hard problem)

NP hard to get $n^{1-\varepsilon}$ approx. for colouring



Given n jobs, each with load $p_j \geq 0$, and m machines,

goal: assign jobs to machines to minimize the max load on any machine



$$\text{load}(i) = \sum_{j \to i} p_j$$

"sum load of all jobs assigned to M/C i"

$$\boxed{\min \max_i \text{load}(i)}$$

NP - complete, so seek Approximation Algorithms].

Potential Algorithms :-

① LP formulation ?

② Greedy Algorithm ?

Sort p_i's in descending order

Assign each job to M/C with least current load.

③ Algo ②, w/o sorting up front.

{ "Online Algorithm" }

{ Decisions are done as and when input is revealed to us }

Good News :-

Even this simple online Algorithm is a 2-approximation algo for the problem.

[Problem is called "Makespan Problem" in scheduling theory.]

Study of Approx Algos

① Need Algo

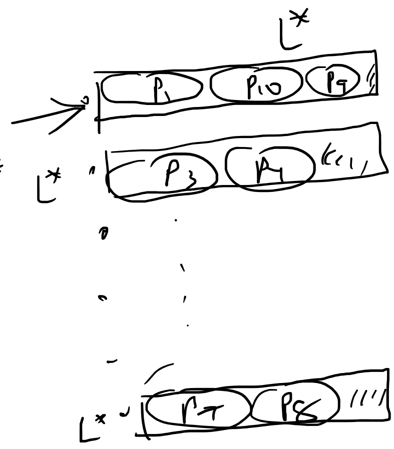
② Need understanding of OPT for analysis.

In our case, 2 simple facts give us this understanding.

Sps L^* = optimal Makespan,

① $\boxed{L^* \geq \max_j p_j}$

② $\boxed{L^* \geq \frac{\sum_j p_j}{m}}$



$$\sum_j p_j \leq m \cdot L^* \Leftarrow \begin{cases} p_1 + p_{10} + p_9 \leq L^* \\ p_3 + p_4 \leq L^* \\ \vdots \\ p_7 + p_8 \leq L^* \end{cases}$$

ANALYSIS of greedy :-

Let L_j denote the load on the M/C to which we assigned job j, <u>at the time we assigned job j</u>.

time we assigned job j.

Load of this M/C after we assigned job $= L_j + p_j$

At the time job j was considered, all M/Cs have a load of at least L_j!

$$\sum_{\text{all } j} p_j \geq \sum_{j' \text{ before } j} p_{j'} \geq m L_j$$

$$\Rightarrow m L_j \leq m L^* \quad (\text{from } ②)$$

$$\Rightarrow \boxed{L_j \leq L^*}$$

$\Rightarrow$ load of this M/C after we assign the job is

$$= L_j + p_j$$
$$\leq L^* + p_j$$
$$\leq 2L^* \quad (\text{from } ①).$$

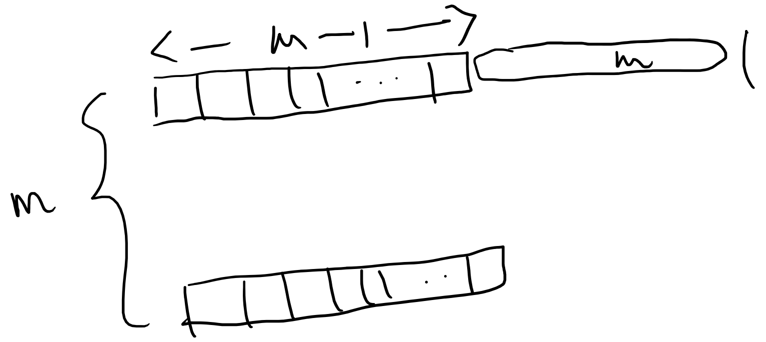
apply this to all jobs ~~to~~ infer that load on all machines is $\leq 2L^*$, giving us the desired 2-approximation $\square$

Q1) Can this algo have a better analysis?

<u>EXAMPLE</u>:

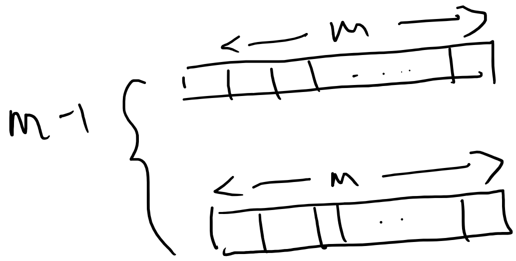
There are $m(m-1)$ jobs of size 1, and 1 job of size m.

Sps we process all small jobs before the large job, what will greedy algo do?



Greedy Makespan $= m-1+m$
$= 2m-1$

<u>Whereas</u> Opt



m

Optimal Makespan $= m = L^*$

factor: $\frac{2m-1}{m} = 2 - \frac{1}{m} \approx 2$ ☹

Point raised

Bad Example is somewhat cheating because Algo works in Online model but OPT reserved 1 M/C for large job.

(Crucially was offline nature of input).

fair, but the example was just to show that we couldn't have done a better analysis for this algo in the offline problem. □

You could try Algo 2 to see

if that does better.

↑ THINK / READ UP about what analysis this can give

Instead, we'll present a PTAS

↓ It will take long time to run, but will compute $(1+\varepsilon) \cdot OPT$ solution.

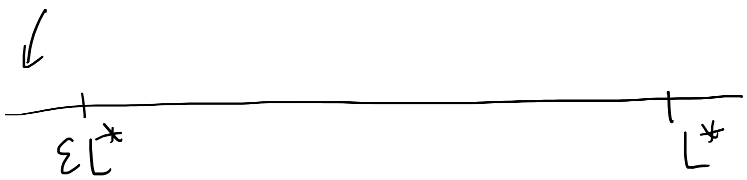
Ideas like for Knapsack

1. Discretization / Rounding
2. Enumeration (brute force "small instances")
3. Guessing framework.
4. Accommodate greedy Algorithm.

New ideas (2, 3)

Idea :-

Divide jobs into 2 categories



If $p_j \le \varepsilon L^*$, j is "small"

$p_j > \varepsilon L^*$, j is "large"

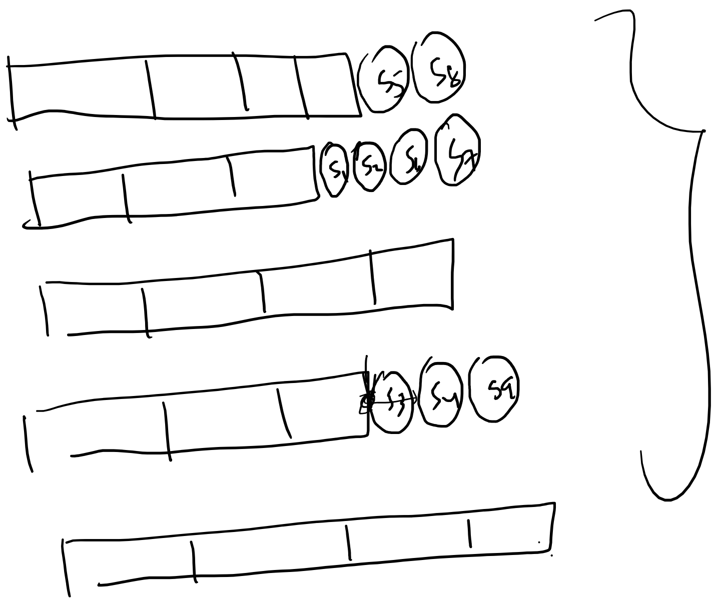
Note: We assume that we "know" the value of L^*. We'll assume for now & get rid of assumption at the end.

Guessing Framework

Idea:

If we run greedy Algo only on small jobs, then its analysis will be very good.

Use "enumeration + rounding" to find good assignment of large jobs.



Assignment of large jobs.

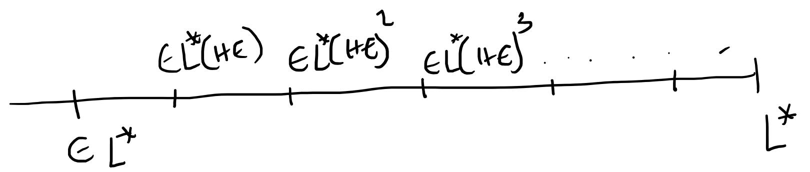
If large assignment is good, meaning all M/C have load $\leq L^*$, then Overall algo is good, all M/C have makespan $\leq (1+\varepsilon) C^*$

SAME ANALYSIS AS BEFORE.

It remains to find a good assignment of the large jobs alone.

23/02/2021.

How do we handle the large jobs?



Large jobs are jobs whose processing times range between ϵL^* and L^*.

Say a job belongs to class i if its $p_j \in [\epsilon L^*(1+\epsilon)^{i-1}, \epsilon L^*(1+\epsilon)^i)$

Moreover, "roundup" its processing time to
$$\hat{p}_j = \epsilon L^*(1+\epsilon)^i$$

Q1: How many non-empty classes are there?

The "last" class i^* is smallest value satisfying $\epsilon L^*(1+\epsilon)^{i^*} \geq L^*$

$$(1+\epsilon)^{i^*} \geq \frac{1}{\epsilon}$$

$$i^* \log(1+\epsilon) \geq \log\left(\frac{1}{\epsilon}\right)$$

Morally,

using $\log(1+\epsilon) \approx \epsilon$ for small enough ϵ

TAYLOR SERIES.

$$i^* \cdot \epsilon \geq \log 1/\epsilon$$

$$\boxed{i^* \approx 1/\epsilon \log 1/\epsilon}$$

$\Rightarrow$ There are only $K = \frac{1}{\epsilon}\log\frac{1}{\epsilon}$ many classes to consider!

(constantly many types).

OPT of "rounded instance $\hat{I}$" (only comprising large jobs w/ $\hat{p}_j$ values)
$\leq (1+\varepsilon) L^*$.

$$\therefore \boxed{\forall j : 1 \leq \frac{\hat{p}_j}{p_j} \leq (1+\epsilon)}$$

$\Rightarrow$ If we find a good sol^n for $\hat{I}$ of makespan $\leq (1+\epsilon)C^*$, then we are done.

Idea:

There are only constantly many job types and moreover, each M/c can accept $\leq$ constantly many jobs in the optimal sol^n.

Highly structured instance !!

We'll claim that there are only polynomially many types of schedules of the above structure, so we can enumerate over them and pick the best.

Let $K = \frac{1}{\epsilon} \log \frac{1}{\epsilon}$ (# Interesting Classes)

$L = \frac{1}{\epsilon}$ (Max # jobs any M/c gets in OPT sol^n).

b/c all jobs $p_j \geq \epsilon L^*$.

Let n_ℓ be the # jobs of class ℓ

Each Machine i receives $n_\ell(i)$ many jobs of class ℓ

s.t $\sum_{i=1}^{m} n_\ell(i) = n_\ell$

In any schedule

distinct "configurations" or distinct assignments any M/c can get ?

Each $0 \leq n_\ell(i) \leq L$ ($L+1$ choices) for all classes

$l = 0, 1, \ldots K$

$\Rightarrow$ # possible assignments any m/c can receive is

$$\leq (L+1)^{(K+1)} = \text{constant}$$

$C_1 \quad C_2, \; \ldots\ldots \quad C_M$

be these distinct configurations

$\Rightarrow$ Really we're asking

How many m/c's get config C_1,
config C_2,
⋮
config C_M?

enumerate over all such allocations and pick the best sol^n. which is feasible (ie) for all l,

$$\sum n_l(i) = n_l .$$

---

enumerations $\leq$

to go over

$$\text{o o} \mid \text{o} \mid \text{o o o} \mid \text{o o} \mid \text{o}$$

m balls for M/cs

M-1 sticks for the configs

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

$$\leq m^{M-1}$$

$$\leq m^{\text{constant}} \quad \left[M \text{ b/c is a constant} \right]$$

If we look closely: $m^{(1/\epsilon)^{1/\epsilon}}$, still polytime for constant ϵ !!

Overall Algo

1. Guess L^*
2. Enumerate all types of configs among m machines
3. Pick best feasible one
4. Greedily Allocate small jobs

gives $(1+\epsilon)L^*$ makespan.

L

(4) Greedily Allocate small jobs

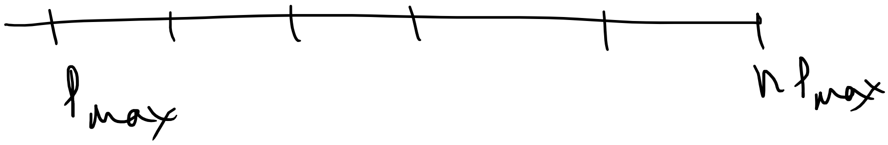
RUNTIME: $\boxed{poly(n)}$.

NOTE

More interesting from theoretical perspective, nobody is going to use this as is in practice.

For guessing L^*, note that

$$L^* \geq P_{max} \quad \& \quad L^* \leq n \cdot P_{max}$$



Say L^* belongs to class j, if

$$L^* \in \left[P_{max}(1+\epsilon)^{j-1}, \; P_{max}(1+\epsilon)^{j}\right)$$

Need to consider $\leq \frac{1}{\epsilon}\log n$ many classes for L^*.

Try running above algo for all "classes" of L^* (ie) $P_{max}(1+\epsilon)^j$ for all j, and choose the smallest which works.

Overall our makespan would be

$$(1+\epsilon)^2 L^*$$

One $(1+\epsilon)$ comes from approx. guess for L^*

another comes from Algo.

Set ϵ' to $\epsilon/3$ and run w/ ϵ'

to get overall

$$(1+\epsilon')^2 \cdot L^* \le (1+\epsilon) \cdot L^*$$

approximation

∎

What to do if P_j (load of jobs) is Machine dependent TOMORROW

actually, it's P_{ij}. (Non-Identical Machines)

aka Unrelated Machines in scheduling literature

Unrelated Machines

24 February 2021 07:57

- n jobs, m machines
- job j has a "load" p_{ij} on machine i

  Note that p_{ij} can be different from $p_{i'j}$.

Q: WHY study this extension?

A: - Machines can be human resources.
- " " " non-identical
  (ex. CPU, GPU, RAM, etc)

ex. if some M/C has too little RAM to accommodate a job, we can set $p_{ij} = \infty$ for i.

Can also capture speeds s_i on M/C i.

In that case we can think of

$$p_{ij} = \frac{p_j}{s_i}$$

Related Machines Scheduling

easier than unrelated Machines, PTAS is known for this problem as well

Brainstorming

① Candidate 1 : Greedy,

- Process jobs in arbitrary order,
- Insert j into M/c i with lowest resulting load.

cur-load $(i) + P_{ij}$

- Lets think for a bit

Here is an interesting "bad example" for this algo

Machines $m = 2^k$

$M_1 \quad M_2 \quad \ldots \ldots \quad M_{2^k}$

job j_1 has unit load on M_1 & M_2, ∞ on rest.

job j_2 " " " on M_3 & M_4 ∞ on rest.

⋮

job $j_{2^{k-1}}$ has unit load on M_{2^k-1} & M_{2^k} ∞ on rest.

Any "online Algo" (in particular greedy) has this "type" of lower bound.

We may assume greedy did the following:

job → Machine

1 ... 1 ... 2

	Job →	Machine
Jobs of Type 1	1	2
	2	4
	3	6
	⋮	
	2^{k-1}	2^k

Recurse on these M/C^s greedy used.

Jobs of Type 2

1 can go on M/c 2 & 4
∞ on rest.

2 " " " " 6 & 8
∞ on rest

2^{k-2} " " " " $2^k - 2$ & 2^k
∞ on rest.

Makespan of Greedy After Type 2 jobs :

Odd M/c have 0 load.
1/2 of even M/c have 1 load
1/2 of even M/c have 2 load

If we had just this instance, what would OPT look like ?

Type 1 jobs can go to odd M/cs,
All M/cs can have load 1.

To get worse example, recurse on even M/cs.

lets say greedy sent type 2 job to 4^{th} Machine.

Type 3 jobs only go on these ☹

$$\begin{cases} 1 \to 4 \text{ or } 8, \ \infty \text{ elsewhere} \\ 2 \to 12 \text{ or } 16, \ \infty \text{ elsewhere} \\ \vdots \\ 2^{k-3} \to 2^{k-1} \text{ or } 2^{k} \ \infty \text{ elsewhere.} \end{cases}$$

By continually focussing only on Machines of high load, we are forcing greedy to keep increasing its load by 1 in each type/phase.

By the end of $k = \log m$ rounds,
Greedy Makespan $= k$
Opt Makespan $= 1$

Lower bound

lower bound for any (deterministic) - ONLINE ALGORITHM which takes actions immediately upon seeing a new job.

Informal note - Needs to be more precise

Could try to think about randomized algorithm { if things look equal, pick randomly }

THIS is still a lower bound for RND Algorithms if the Nature of type of jobs can depend on Algos choices.

Not "as stated" a lower bound if input jobs is independent of Algos randomness.

Present an LP - based algorithm :-

by guessing to within $(1+\varepsilon)$, lets assume that we know L^* (optimal Makespan)

x_{ij} : denotes if job j is assigned to machine i.

what are the legitimate constraints we can enforce on $\{x_{ij}\}$ variables !

Min 0 (interestingly, no obj. fn)

$$\text{①}\left\{\begin{array}{ll} \sum_i x_{ij} = 1 & \forall j \text{ (Job assignment constraint)} \\ \text{②}\ \sum_j x_{ij} P_{ij} \leq L^* & \forall i \text{ (machine Load constraints)} \\ x_{ij} \geq 0 & \end{array}\right.$$

From ①, all x_{ij}'s will also never exceed 1.

Lets say we solve this LP & it is feasible.

How do we find an integral solution from it?

Q: what if we think of L^* as a variable and try to minimize L^*?

With hindsight, we don't follow this approach.

Really, LP is a "feasibility LP" and we are trying to exploit structure of BFS / Vertex / Extreme Points.

Idea ①

Randomized Rounding

- each job chooses its m/c according to $\{x_{ij}\}$ as a distribution.

[Can show $O(\log m)$ - approximations using Chernoff bounds.

we are equipped to study this algorithm! ✓

Idea ②: Exploit structure of BFS/Extreme pts.

- mn dimensional space,
- BFS / Vertices are identified by the intersection of mn hyperplanes.

$\Rightarrow$ mn constraints are satisfied at equality

$$\boxed{\Rightarrow \text{at least } mn - (m+n) \text{ of the } x_{ij}\text{s are actually } 0}$$

$\Rightarrow$ At Most $(m+n)$ variables are non-zero in the LP.

Lets ~~think~~ visualize the sol^n like a graph

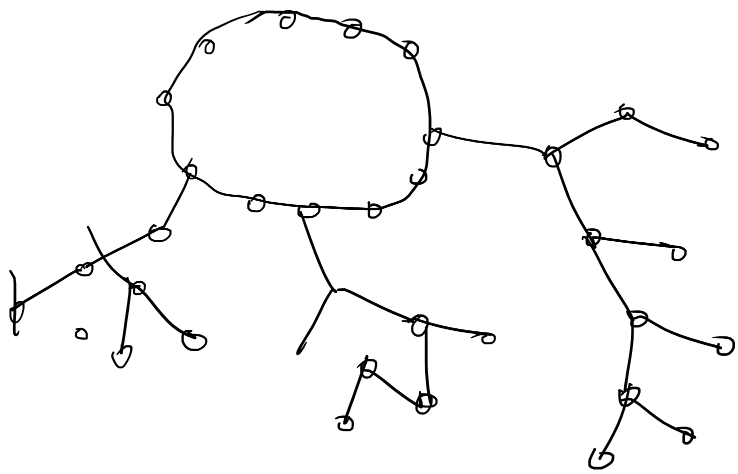
Vertex for every job & machine
Edge (i,j) if $x_{ij} > 0$

$$\left.\begin{array}{l} \#\text{ vertices} : \ m+n \\ \#\text{ edges} : \ \le m+n \end{array}\right\}$$

The most complicated soln to deal with is when # edges $= m+n$.

Fewer Non Zero variables ⇒ LP gives greater clarity.

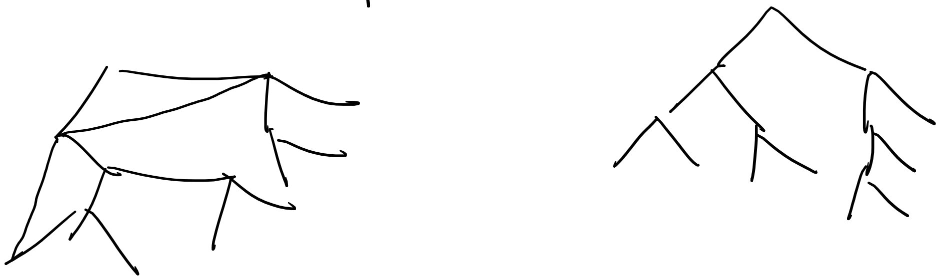
Example ① This LP-soln-Graph looks like a tree + one cycle.



Example ② Of graph with $m+n$ ...

vertices & $m+n$ edges;

> 1 components



$m_1 \quad n_1$ $\qquad$ $m - m_1 \quad n - n_1$

2 cycles $\qquad$ tree

We'll argue that BFS / extreme points can't have this structure!

THM:

In any BFS, it will look like every component is a tree with ≤ 1 cycle.

Recap: 26/02/2021

Min Makespan on Unrelated Machines

n jobs $\qquad$ m machines

Job $j \longrightarrow$ Machine i (P_{ij} load).

Assume we know OPT Makespan L^*

(we can "query" in powers of

Assume we know OPT Makespan L
(we can "guess" in powers of $(1+\varepsilon)$)

$\min 0$ (L^* is not a variable)

$$\left.\begin{aligned} \sum_i x_{ij} &= 1 \quad \forall j \\ \sum_j p_{ij} x_{ij} &\le L^* \quad \forall i \\ x_{ij} &\geq 0 \end{aligned}\right\} LP(M, J, L^*)$$

LP over Machine set M
Job set J
& Makespan L^*

We solve this LP ~~optimally~~ and find any basic feasible sol^n / Extreme point / Vertex sol^n.

Let x^* denote such a sol^n.

Q: How Many Non-Zero Variables can x^* have?

A: $\leq m+n$. (because x^* is BFS, it is a intersection of mn hyperplanes
$\Rightarrow \geq mn-(m+n)$ must be of the form $x_{ij} = 0$

$x_{ij} = 0$

$\Rightarrow \le (m+n)$ x_{ij}'s can be > 0.

ONE MORE OBSERVATION:

Let's construct a graph with vertices $= M \cup J$ and edges for non-zero variables

$m+n$ vertices, $\le m+n$ edges

Sps it has k - connected components

CLAIM 2

In Graph(x^*), each connected component is a "Pseudo Tree"

$\Downarrow$

Meaning a tree with at most one extra edge.

Proof:-

Suppose some connected component is not a pseudo-tree.

Suppose some connected component
is not a pseudo-tree.

- Sps it has $\bar{M} \subseteq M$ as Machine set &

$\bar{J} \subseteq J$ as job set.

Sps $|\bar{M}| = \bar{m}$ & $|\bar{J}| = \bar{n}$.

by connectivity #edges in component $\geq \bar{n} + \bar{m} - 1$

Now, b/c it is ~~not~~ not a pseudo-tree,

#edges in component $\geq \bar{m} + \bar{n} + 1$.

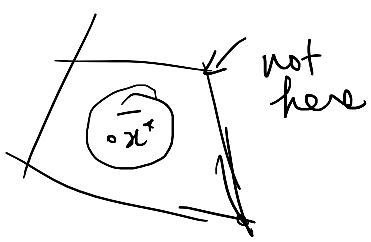
Let $\bar{x}^*$ denote the restriction of x^* to $\bar{J}, \bar{M}$.

We know that

$\bar{x}^*$ is feasible LP solution to $LP(\bar{M}, \bar{J}, L^*)$.

But it is not a BFS for this restricted LP.

Any BFS for restricted LP must have $\leq \bar{m} + \bar{n}$ Non-zero Variables !!



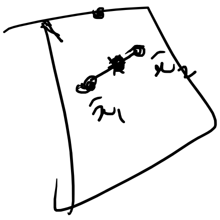
$\bar{x}^*$ is feasible for restricted LP $(\bar{M}, \bar{J}, L^*)$

but not BFS / Extreme Point
since it has more non-zero
Variables than
BFS's can have for
the restricted polytope.

$\Rightarrow$ can write $\bar{x}^* = \alpha\, \bar{x}_1 + (1-\alpha)\, \bar{x}_2$

where $\bar{x}_1$ & $\bar{x}_2$ are
feasible for
$LP(\bar{M}, \bar{J}, L^*)$

because $\bar{x}^*$ is
not a BFS,
it is in interior
of polytope



$\Rightarrow$ Can write x^* as a
convex combination of
x_1 and x_2

for $LP(M, J, L^*)$

$x_1 = \bar{x}_1$, rest of x^*
$x_2 = \bar{x}_2$, rest of x^*

$\Rightarrow$ x^* can't be a BFS
since we're able to write
x^* as convex combination
of 2 feasible

of 2 feasible solutions.

SUMMARY

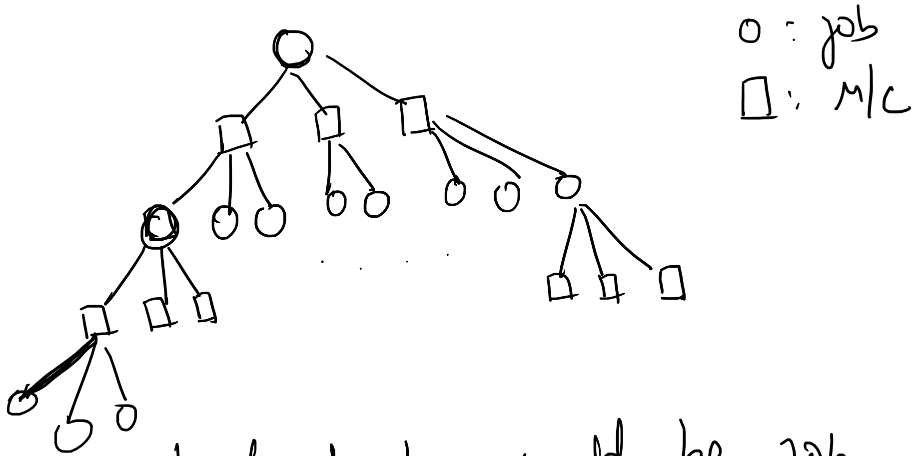
1. x^* is a BFS
2. $\Rightarrow x^*$ restricted to all components have to be BFS for restricted problem

$\Rightarrow \overline{x^*}$ for each component is a pseudo-tree

LAST STEP:

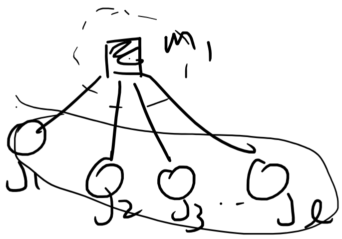
→ Rounding pseudo-trees.

→ Let's see how to round trees, pseudo-trees will be similar in spirit.

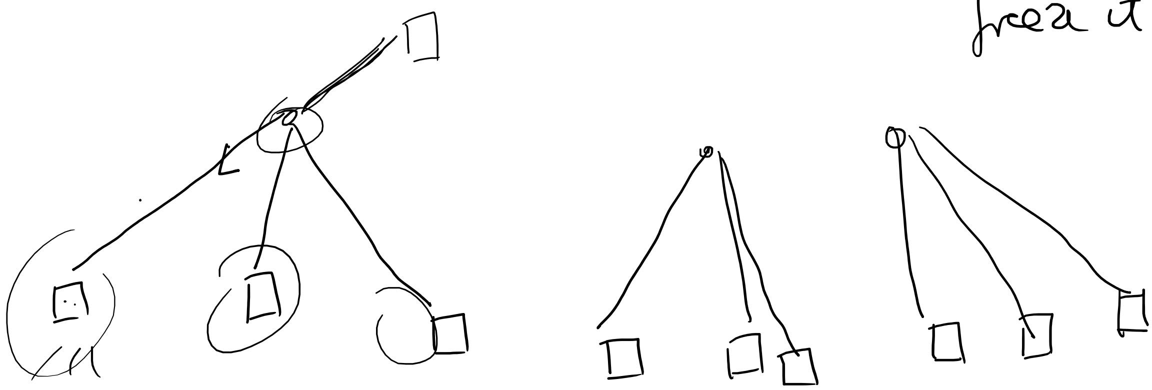


Leaf of tree could be job or Machine

leaf is a job $\Rightarrow x_{ij} = 1$ for that job, so we can freeze that assignment



all must be $= 1$, so we can freeze it.



Now all leaves are Machines.

Simply assign the parent job to an arbitrary child M/C.

contributes to an "excess" load on the child M/C

$\leq \max_{i,j} p_{ij}$

CAN WORK UPWARD for the rounding

Can round x^* using pseudotree structure to assign jobs $\Rightarrow$ M/Cs m/c's

such that each m/c gets

a load $\leq$ $L^* + \max_{ij} p_{ij}$

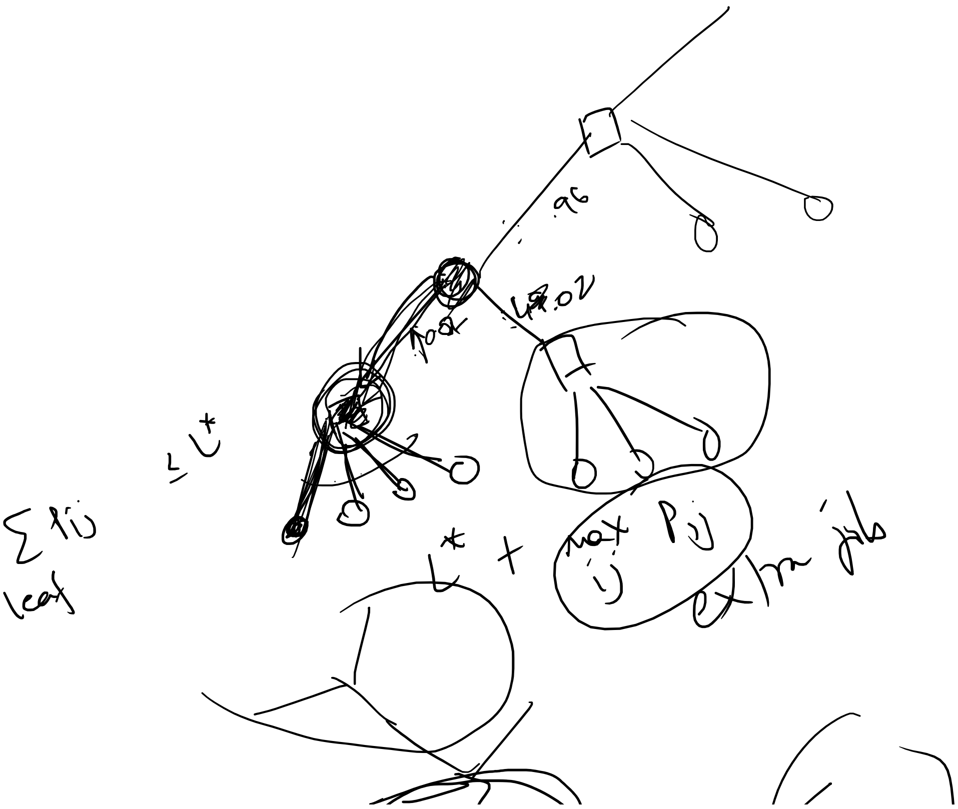
$\Rightarrow$ LAST STEP

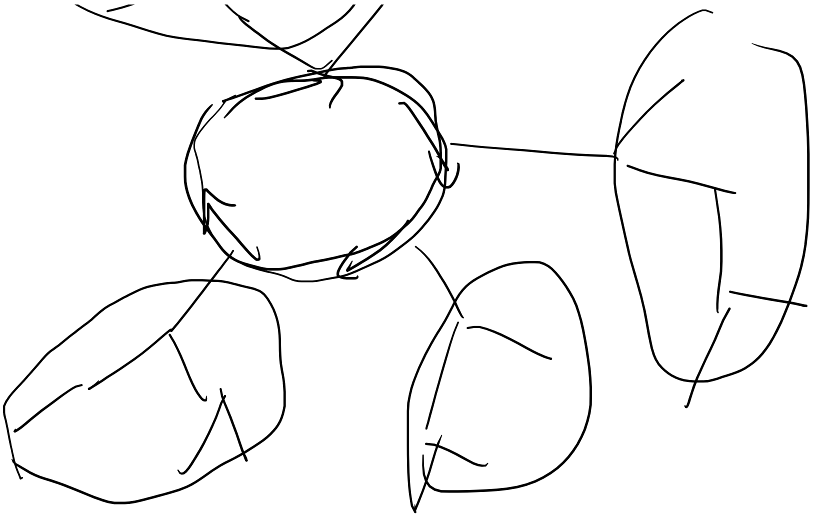
Since we know L^*, we can

for $x_{ij} = 0$ if $p_{ij} > L^*$ in the LP

(either explicit constraints or just delete these variables for large p_{ij} from original LP).

$\Rightarrow$ gives 2-approximation to Makespan on "unrelated m/c"





Min Makespan Scheduling

- n Jobs, m M/C
- p_{ij} = load of job j on M/C i.

Find assignment of jobs to machines to minimize maximum load on any machine

$$\boxed{\min \max_{i} \sum_{j \to i} p_{ij}}$$

Technique is "Iterative LP Rounding"

<u>High level idea</u> :-

Sps there is an LP with n variables, with

→ $0 \le x_i \le 1$ type of constraints

and → 'm' other linear constraints

<u>Totally</u>

$$\max c^T x$$

$$\rightarrow Ax \le b \quad \text{'m' constraints}$$

$$0 \le x_i \le 1 \quad \Leftarrow n \text{ variables}$$

If 'm' is very small, $\ll n$, then many variables will be satisfied Integrally.

in a BASIC FEASIBLE SOLUTION.

Proof: In a BFS, it is defined by the intersection of 'n' constraints satisfied @ equality

At most 'm' of these can come from $Ax \le b$

$\Rightarrow \ \ge n-m$ variables are satisfied integrally.

Idea of Iterative Rounding is to reduce the 'm' value as much as possible, by removing constraints which can be handled through other means

little vague, but we'll see soon

BACK TO SCHEDULING :-

What's the LP?

① We "guess" Optimal Makespan L^*

$$\min 0$$

$$\rightarrow \sum_j p_{ij} x_{ij} \leq L^* \quad \forall i$$

$$\rightarrow \sum_i x_{ij} = 1 \quad \forall j$$

$$0 \leq x_{ij} \leq 1 \quad \forall (i,j)$$

Moreover, delete x_{ij} variables with $p_{ij} > L^*$.

Lem ①

If guess of L^* is correct, then LP is feasible.

More general LP (allowing different loads on each machine).

$$\min 0$$

$$\sum_j p_{ij} x_{ij} \leq L_i \quad \forall i \in M$$

$$\sum_i x_{ij} = 1 \quad \forall j \in J$$

$$0 \le x_{ij} \le 1 \quad \forall\, (i,j) \in E$$

set of allowed variables

Let $\bar{L} = (L_1, L_2, \ldots L_m)$, initially $E = \{(i,j) \text{ st } p_{ij} \le L^*\}$

$$\boxed{\text{Then } LP(J, M, \bar{L}) \text{ is feasible for } \bar{L} = (L^*, L^*, \ldots L^*) \text{ for correct guess of } L^*.}$$

ALGORITHM

1. Guess L^* value
2. While $(J \ne \phi)$
3. Solve $LP(J, M, E, (L^*, L^*, \ldots, L^*))$ and compute a BFS. x^*
4. If $\exists\, j \in J$ and $i \in M$ st $x^*_{ij} = 1$,
   - 4a assign $j \to i$ in our schedule
   - 4b Reduce $L_i = L_i - p_{ij}$
   - 4c Remove j from J, update E appropriately
   - 4d Go back to step 2
5. If some $x^*_{ij} = 0$, remove this variable from E and go back to step 2
6. If some machine $i \in M$ has only one job j with $x^*_{ij} > 0$

   Assign job $j \to$ machine i.

   Remove j from J

   i from I

   and update E accordingly.

Goto Step 2 $\Leftarrow$ Can increase load on i beyond its capacity but "its a simple constraint"

7 if some M/C $i \in M$ has
only <u>2 jobs</u> j_1 & j_2
with $x^*_{ij_1} > 0$, $x^*_{ij_2} > 0$
and $x^*_{ij_1} + x^*_{ij_2} \geq 1$

then assign both jobs to M/C i,
remove them from J
remove i from M.
update E accordingly.
Goto step 2

L^*

job 1 has $p_{1i} = 0.9L^*$, $x^*_{1i} = 1$

$\downarrow$

freeze job 1 $\to$ M/C i

update $L_i = L^* - 0.9L^*$
$= 0.1L^*$

Next Step of Algo,
job 2 has $P_{2i} = L^*$, $x^*_{2i} = 0.1$
but this is only job going to M/C i.

Algo assigns job 2 to M/C i.

Is ALGO CLEAR?

Not yet, because it can potentially cycle in ∞-loop.

$\downarrow$

~~jee~~

Need to show

At any iteration, if x^* is a BFS, one of the conditions 4, 5, 6, 7 MUST HOLD !!

Proof:
Suppose x^* is a BFS where none of these conditions hold.

① Every job $j \in J$ has at least 2 non-zero x^*_{ij} variables

Proof: $\sum_i x^*_{ij} = 1$ in total, but NO single variable is 1.

② Every machine $i \in M$ has at least 2 incoming jobs with > 0 x^*_{ij} value.

Pf Otherwise it satisfies condition 6.

$\Rightarrow$ Total # of strictly positive variables in the LP?

If the # of jobs is $|J|$ and # machines is $|M|$, then

strictly +ve variables is $\geq |J| + |M|$

But x^* is a BFS to

$$\sum x_{ij} p_{ij} \le L_i \quad \forall i \in M$$
$$\sum x_{ij} = 1 \quad \forall j \in J$$
$$0 \le x_{ij} \le 1 \quad \forall (ij) \in E$$

In any BFS At least $|E| - |M| - |N|$ variables must be 0 or 1.

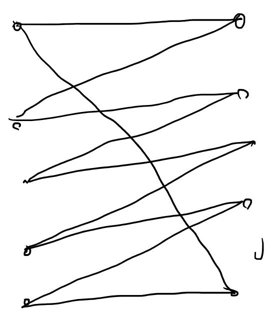
This says that $|E|$ has to $= |M| + |N|$.

$\Rightarrow$ If LP soln does not satisfy conditions (4), (5) or (6), it must be very structured

(ie) a) Exactly $|M| + |N|$ variables in LP,

b) Each job has 2 edges

c) Each M/C has 2 edges



Has to look like a disjoint collection of cycles.

$\Rightarrow$ at least 1 M/C satisfies

Condition 7 & SO
we make progress
(CONTRADICTION).

02/03/2021

J : set of jobs
M : set of Machines
E = { edges, valid variables }

Initially $J = [n]$
$M = [m]$
$E = \{ (i,j) : p_{ij} \leq L^* \}$

L_i = target residual load on M/C i
Initially, all $L_i = L^*$.

Min 0

$$\sum_{j : (i,j) \in E} p_{ij} x_{ij} \leq L_i \quad \forall i$$

$$\sum_{i : (i,j) \in E} x_{ij} = 1 \quad \forall j$$

$$0 \leq x_{ij} \leq 1 \quad \forall (i,j) \in E$$

Algo

1) Guess L^*
2) While $J \neq \phi$
Solve LP(J, M, E, (L_i)) to

② While $J \neq \phi$

③ Solve $LP(J, M, E, (L_1, L_2, \dots L_m))$ to get BFS x^*

④ if $\exists$ (i,j) st $x^*_{ij} = 1$

↳ Update $L_i = L_i - P_{ij}$

↳ Assign job j to M/c i

↳ ~~Remaitt~~ ~~jts~~ ~~frodgs~~ J ~~from~~ and E

↳ Goto step ②

⑤ if $\exists$ (i,j) st $x^*_{ij} = 0$

↳ Remove (i,j) from E

↳ Goto step ②

⑥ If $\exists$ M/C i with exactly <u>or NONE</u> one incoming job, (ie) $x^*_{ij} > 0$,

↳ Assign job j → M/c i

↳ Remove i & j from M & J and update E

↳ Goto step ②

7 If $\exists$ M/c i with exactly
2 +ve x^*_{ij} and
$x^*_{ij_1} + x^*_{ij_2} \geq 1$

↳ Assign both j_1 & j_2 to i

↳ Remove i, j_1, j_2 from
M and J resp'ly

↳ Goto Step 2.

Lemma 1

- Algo doesn't get stuck in an ∞ while loop

If (i, c) x^* is a BFS, then one of
4, 5, 6 or 7 holds true.

~~Proof~~:
We show If 4, 5, and 6 don't hold
for x^*, then 7 must hold.

Firstly, we claim that if 4 & 5
don't hold, then set of
allowed/active variables in the LP
is very small.

IP = Min T

LP = Min 0

$$\begin{cases} \sum p_{ij} x_{ij} \le L_i & \forall i \in M \\ \sum x_{ij} = 1 & \forall j \in N \end{cases}$$

$$0 \le x_{ij} \le 1 \quad \forall (i,j) \in E$$

In any BFS x^*, at most $|M| + |N|$ variables can be truly fractional, i.e.,

$$0 < x^*_{ij} < 1 .$$

Pf

We are looking for solutions in $|E|$-dim space

· These vertex pts of LP are intersections of $|E|$ tight constraints

$\Rightarrow \ge |E| - (m+n)$ must come from $0 \le x^*_{ij}$ & $x^*_{ij} \le 1$

But because ④ & ⑤ don't hold

$$|E| - (m+n) \le 0$$

$$\Rightarrow |E| \le (m+n) .$$

(**)

Next,

Because each job $j \in J$ (current set of active jobs)

① has $\sum_i x^*_{ij} = 1$ but no single variable is 1,
it has at least 2 active edges with $x^*_{ij} > 0$.

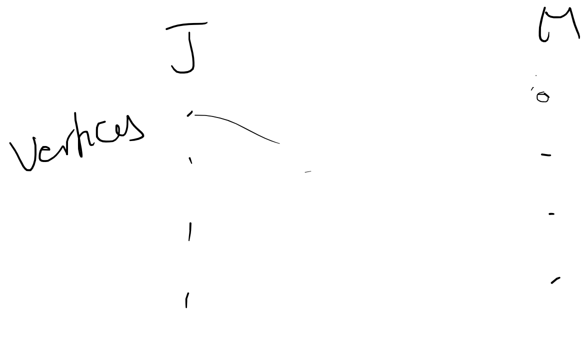
② Similarly, each M/C $i \in M$ (current set of active M/C)
has at least 2 active edges with $x^*_{ij} > 0$
[B/C step ⑥ also didn't occur]

$\Rightarrow$ Total # of variables $\geq 2|J|$
Total # of variables $\geq 2|M|$

(XX) $\Rightarrow$ By averaging, $|E| \geq |J| + |M|$.

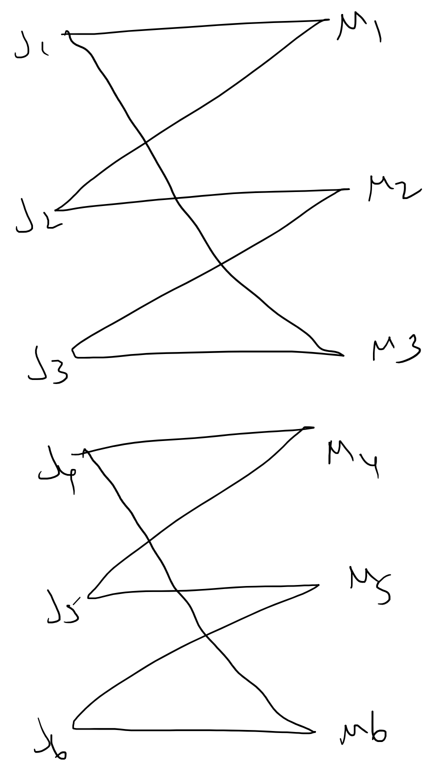
From (X) and (XX) we get $|E| = |J| + |M|$

Next imagine a graph



This graph has # edges = # vertices
and every vertex has
degree = 2.

$\Rightarrow$ This graph has to be a UNION of
cycles



From this, how do we conclude that
step (7) must happen?

In any cycle, $\sum_{i,j \in \text{cycle}} x^*_{ij}$ = # jobs in cycle

= # M/Cs in cycle.

$\Rightarrow$ $\exists$ some M/C i with $\sum x^*_{ij} \geq 1$.

$\Rightarrow$ Step (7) holds !! ☒

We find a good solution eventually.
(ie) it has Makespan $\leq 2L^*$

If i is an active machine,
L_i + Current load assigned to i
$\leq L^*$

By Induction
- Initially true since Current load = 0.
- load only increases Step (4) or (6) or (7)

Property continues to hold true in Step (4)

- Need not hold after Step (6) or (7), but M/C i is no longer active after these!

If we get to Step (6),
We may increase load of i by 1 jobs
which has $P_{ij} \leq L^*$
$\Rightarrow$ OK.

If we get to step (7),

we may increase load of i by 2 jobs

$\Rightarrow$ $3L^*$ guarantee is trivial

but we can do better by using $x^*_{ij_1} + x^*_{ij_2} \geq 1$

In our soln, both x^*_{ij} are rounded to 1

So total increase, when compared to L_i (which is excess load on $M(i)$)

$$= (1 - x^*_{ij_1}) p_{ij_1} + (1 - x^*_{ij_2}) p_{ij_2}$$

$$\leq (1 - x^*_{ij_1}) L^* + (1 - x^*_{ij_2}) L^*$$

$$\leq (2 - (x^*_{ij_1} + x^*_{ij_2})) L^*$$

$$\leq L^*$$

New load to i $- L_i \leq p_{ij_1} + p_{ij_2} - x^*_{ij_1} p_{ij_1} - x^*_{ij_2} p_{ij_2}$

$$\leq L^*$$

$\Rightarrow$ New load to i $\leq L_i + L^*$

Existing load + New load $\leq$ Existing load + $L_i + L^*$

$\leq 2L^*$ ∎

MAIN TAKEAWAY

1) LP BFS are powerful (almost integral many times)

2) If there is simple constraint blocking a BFS from becoming more integral, we can remove it hope to handle it in other ways

ITERATIVE ROUNDING AND RELAXATION

Removing constraints

Same scheduling problem, <u>different Algorithm</u>

n jobs $\longrightarrow$ m Machines
Job j has load P_{ij} on m/c i.
Find an assignment to Min Max Load.

We saw that LP BFS can be used to
- get 2 approximation (2 different methods)

- Today: "Simpler" "more efficient" "worse guarantee"
algorithm $O(\log m)$ - approximation.

- Nice feature: can assign jobs
One by One in arbitrary order
without knowledge of future jobs.

<u>Idea</u>:-
Running "greedy"-type of algorithms
to $\boxed{\text{minimize } \max(\text{load}(1), \text{load}(2), \ldots \text{load}(m))}$

Not a very smooth or easy-to-understand function.

We will actually try to optimize a
"different smoother objective function"
which is close enough to the
Max-Objective Function.

$$f(l_1, l_2, \ldots l_m) = l_1^p + l_2^p + \cdots + l_m^p$$
for large enough P.

"Intuition:" If p is larger and larger, then the large values of $\bar{l} = (l_1, .. l_d)$ start dominating the expression.

(ie) if $p >>$ large,

$$(\text{max load})^p \leq f(l_1, l_2, .. l_m) \leq m \cdot (\text{maxload})^p$$

Next lecture

Greedy Algo for scheduling using "surrogate" p-NORM objective

---

05/03/2021 Yet another algorithm for Makespan Minimization on Unrelated Machines

Idea: Come up with a suitable "smoother approximation" of the $\underset{i}{\text{Max}}(\text{load}(i))$ Objective.

In particular,

Minimizing $\text{Max}(x_1, x_2, .. x_m)$ closely related to

Minimizing $\sum_{i=1}^{m} x_i^q$ for large enough q.

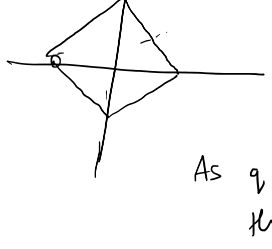
Intuition

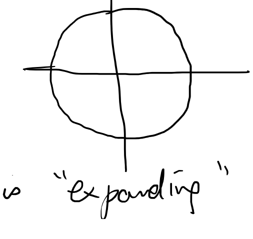
let's say $M = 2$. and plot

$$|X|^q + |Y|^q = 1$$

when $q = 1$

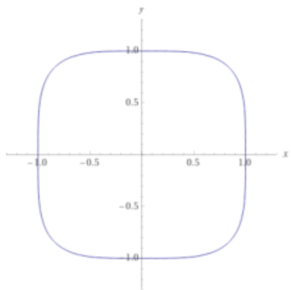
when $q = 2$





As q increases, the curve is "expanding" the curve is "expanding"

when $q = 4$



As q increases $x^q + y^q$ is dominated by $\max(x,y)^q$

We'll look to find a schedule which minimizes $\sum_{i=1}^{m} \text{load}(i)^q = \Phi$

Potential Function

Sort of a Surrogate Objective

"Soft-Max Objective"

Q) What's a nice greedy algo for Soft-Max objective?

Suppose $t-1$ jobs have been scheduled ($\text{load}_{t-1}(i)$ is the current load of M/C i at this time).

$$\Rightarrow \boxed{\Phi(t-1) = \sum_{i=1}^{m} \text{load}_{t-1}(i)^q}$$

When trying to schedule t^{th} job, try to schedule it to M/C which minimizes $\Delta\Phi$, (ie) $\Phi(t) - \Phi(t-1)$.

? ONLINE ALGORITHM! Can work with

ONLINE ...

Online Sequence of jobs.

THM :- for any sequence of job insertions, Max load of Algo $\leq \Theta(\log m)$ OPTIMAL MAX LOAD.

for $q = \Theta(\log m)$.

Says that even if we don't know all jobs ahead of time, we can do favourably against an all-knowing optimal sol^n.

Proof
We'll fix value of q later.
Let job insertions be numbered
$\rightarrow 1, 2, \ldots, n$
and $\phi(t) = \sum_{i=1}^{m} load_t(i)$ be the potential after t insertions

- $\phi(0) = 0$.

$load_t(i)$ = load of M/C i after first t jobs have been inserted.

Recall
$\forall j = 1, 2, \ldots n, \quad \forall i$
p_{ij} denotes load of job j if scheduled on M/C i.

Lets fix insertions upto time $t-1$ for $t \geq 1$ and consider the t^{th} job.

Suppose greedy Algo sent it to M/C $i(t)$.

Suppose Optimal solution of all jobs $1 \cdots n$ sends job t to M/C $i^*(t)$.

... assigning the t^{th} job

After assigning the t^{th} job

$$\phi(t) - \phi(t-1) = \text{load}_t(i(t))^q - \text{load}_{t-1}(i(t))^q$$

$$= \left(\text{load}_{t-1}(i(t)) + p_{i(t),t}\right)^q - \text{load}_{t-1}(i(t))^q$$

Because we run greedy Algorithm $\longrightarrow$

$$\leq \left(\text{load}_{t-1}(i^*(t)) + p_{i^*(t),t}\right)^q - \text{load}_{t-1}(i^*(t))^q \quad (*)$$

Now, a quick cheating proof.

Let's look @ (*)

$$= L_{t-1}(i^*(t))^q \left[\underbrace{\left(1 + \frac{p_{i^*(t),t}}{L_{t-1}(i^*(t))}\right)^q}_{} - 1\right]$$

SLIGHT CHEAT

$\approx$ if $p \ll L_{t-1}$

$$L_{t-1}(i^*(t))^q \left[\cancel{1} + q \cdot \frac{p_{i^*(t),t}}{L_{t-1}(i^*(t))} - \cancel{1}\right]$$

$$= q \cdot \text{Load}_{t-1}(i^*(t))^{q-1} \cdot p_{i^*(t),t}$$

$$\leq q \cdot \text{Load}_n(i^*(t))^{q-1} \cdot p_{i^*(t),t}$$

Also loads are increasing over time.

Back to cheating Analysis :-
Let's sum up over all arrivals.

$$\phi(t) - \phi(t-1) \leq q \cdot \text{load}_n(i^*(t))^{q-1} \cdot p_{i^*(t),t}$$

$$\sum_{t=1}^{n} \Rightarrow$$

$$\phi(n) - \phi(0) \leq q \cdot \sum_{i=1}^{m} \text{load}_n(i)^{q-1} \sum_{t: i^*(t)=i} p_{i,j}$$

$$= q \sum_{i=1}^{m} \text{load}_n(i)^{q-1} \cdot \text{load}^*(i)$$

$$= q \sum_{i=1}^{m} load_n(i)^{q-1} \cdot load^*(i)$$

$$\Rightarrow \boxed{\sum_{i=1}^{m} load_n(i)^q \le q \cdot \left(\sum_{i=1}^{m} load_n(i)^{q-1} \cdot load^*(i) \right)}$$

USE HÖLDER'S INEQUALITY.

$$\sum_i |a_i b_i| \le \left(\sum |a_i|^x \right)^{\frac{1}{x}} \left(\sum |b_i|^y \right)^{\frac{1}{y}}$$

as long as $\frac{1}{x} + \frac{1}{y} = 1$

dual norms.

We want

$\sum load^q$, $\sum (load^*)^q$, etc

so, what x & y should we use on RHS?

$$x = \frac{q}{q-1}, \quad y = \frac{q}{1}.$$

RHS

$$\le q \cdot \left(\sum_{i=1}^{m} \left(load_n(i)^{q-1} \right)^{\frac{q}{q-1}} \right)^{\frac{q-1}{q}} \cdot \left(\sum_{i=1}^{m} load^*(i)^q \right)^{\frac{1}{q}}.$$

$$= q \left(\sum_{i=1}^{m} load_n(i)^q \right)^{\frac{q-1}{q}} \left(\sum_{i=1}^{m} load^*(i)^q \right)^{1/q}$$

Overall, get

$$\left(\sum_{i=1}^{m} load_n(i)^q \right)^1 \le q \cdot \left(\sum_{i=1}^{m} load_n(i)^q \right)^{\frac{q-1}{q}} \left(\sum_{i=1}^{m} load^*(i)^q \right)^{1/q}$$

$$\Rightarrow \left(\sum_{i=1}^{m} load_n(i)^q \right)^{1/q} \le q \cdot \left(\sum_{i=1}^{m} load^*(i)^q \right)^{1/q}$$

$$\Rightarrow \sum_{i=1}^{m} load_n(i)^q \le q^q \cdot \sum_{i=1}^{m} load^*(i)^q. \quad \blacksquare$$

(*)

Suppose optimal Makespan $= L^*$.

Suppose optimal Makespan $= L^*$.
We'd like to show that all our machines have a low-load.

(**) $\Rightarrow$

$$\forall i, \quad \text{load}_n(i)^q \leq q^q \cdot m (L^*)^q$$

$$\Rightarrow \quad \text{load}_n(i) \leq q \cdot m^{1/q} \cdot L^*$$

set $q = \log_2 m$

$$m^{1/q} = m^{\frac{1}{\log_2 m}} = 2.$$

$$\Rightarrow \boxed{\text{load}_n(i) \leq (2 \cdot \log_2 m) L^* \quad \forall i}$$

---

lets try to not cheat

(*) $\Delta \phi = \phi(t) - \phi(t-1)$

$$L_{t-1}(i^*(t))^q \left[\left(1 + \underbrace{\frac{p_{i^*(t),t}}{L_{t-1}(i^*(t))}}_{T_1}\right)^q - 1 \right] - 1$$

either

$$\frac{p_{i^*(t),t}}{L_{t-1}(i^*(t))} \text{ is small } \leq \frac{1}{q}$$

or large $> \frac{1}{q}$

In small case, take full Taylor Series to get

$$T_1 \leq \cancel{1+} \; \underline{\underline{2 \cdot q}} \cdot \frac{p_{1^*(t),t}}{L_{t-1}(1^*(t))} \; \cancel{-1}$$

In large case

$$T_1 \leq \left(1 + \frac{p}{L}\right)^q - 1 \leq \left(1 + \frac{p}{L}\right)^q$$

$$\leq \left(q \cdot \frac{p}{L} + \frac{p}{L}\right)^q$$

$$\leq (q+1)^q \cdot \frac{p^q}{L^q}$$

Therefore, always

$$\Delta\phi \leq \underbrace{L^q \cdot 2q \cdot \frac{p_{1^*(t),t}}{L}}_{\text{good term}} + \underbrace{\cancel{L^q} \cdot (q+1)^q \cdot \frac{p^q_{1^*(t),t}}{\cancel{L^q}}}_{\text{bad term}}$$

good term proceeds as always,
bad term is also not bad

b/c we have only
Optimal sol^n terms
in it.
(NO ALG terms).

$$Alg \le q \cdot Alg^{\frac{q-1}{q}} \cdot OPT^{1/q} + OPT$$