and $c = \frac{1}{72000}$ (after calculation)

Lecture 21 : 6th April

APPROXIMATION ALGOS

P = problems that can be solved in time that is polynomial in size of input

NP = time polynomial in size of input, using non-determinism

→ $P \subseteq NP$ but opposite not true

→ Evidence to $P \neq NP$ is coming from the existence of NP-complete problems.

NP-Complete Problems are

1. NP
2. every other problem in NP should be reducible in polynomial time to this problem.

Approximation Algo

→ For a problem P, let A be an approx algo.

Suppose that P is a minimization problem. — minimises the value of some objective function.

Then, the performance of algo A for P is measured as its approximation ratio defined :

(value of input variables becomes an instance)

For an instance I of P let OPT(I) denote value of best possible soln.

Let A(I) denote value of soln produced by algo A.

The approx ratio of A is

$$\max_{I} \left(\frac{|A(I)|}{|OPT(I)|} \right)$$

This ratio is always at least 1

∵ minimization problem

$\Rightarrow OPT(I) \leq A(I)$

If P is a maximisation problem

$$\text{Ratio} = \max_I \left(\frac{|OPT(I)|}{|A(I)|} \right)$$

Ratio always at least 1

as $OPT(I) \geq A(I)$.

→ IN MAXSAT,

we want to check maximum no. of clauses we can satisfy.

we got 3/4 th of best possible soln.

Here, the ratio = 4/3 (inverse)

as $\frac{opt(I)}{A(I)}$

$$\Rightarrow \frac{OPT(I)}{\frac{3}{4}|OPT(I)|} = \frac{4}{3}$$

so, for maximisation,

$$|A(I)| \geq \frac{3}{4} |OPT(I)|$$

minimisation

$$|A(I)| \leq \frac{3}{4} |OPT(I)|$$

Date
Page

LOAD BALANCING

most applications are single threaded - so they have to be mapped to core of your computer.

So, cores schedule jobs.

These cores are machines that are identical to each other : M_1, M_2, ---M_m

We have 'n' jobs J_1, J_2 --- J_n

Our goal is to minimise the time spent by any machine while executing the jobs it is assigned

Here, all jobs have same priority, and we have to figure out which job will be executed by which machine.

eg. 4 cores - M_1 M_2 M_3 M_4

6 jobs - 4 (1) 2 (2) 5 (3) 3 (4) 7 (5) 2 (6)

M_1	M_2	M_3	M_4
J_1, J_4	J_3, J_6	J_5	J_2
$= 4+3$	$= 5+2$	7	2
$= 7$	$= 7$	7	2

~~minimise~~ minimise the max

∴ no machine is more heavily loaded than other machines, we do this kind of minimisation.

Let us define the following:

$A(i)$ = jobs assigned to M_i

t_j = time requirement for job j (for $j = 1$ to n)

The makespan of a machine M_i is $T_i = \sum_{j \in A(i)} t_j$

Makespan of an assignment $T = \max_i T_i$

Goal: find an assignment A that minimizes the makespan T.

greedy

```
GreedyAssign (Jobs J, m) {
    A(i) = empty for all i ∈ (1, m)
    T(i) = 0 for all i ∈ (1, m)
    for (j = 1 to n) {
        find i s.t. Mi has
        minimum Ti
        A(i) = A(i) ∪ {j}
        Ti = Ti + tj
    }
}
```

eg.

J_1	J_2	J_3	J_4
4	2	3	5

M_i	M_1	M_2
A_i	4	2
	5	3
T_i =	9	5

$T = 9$

eg. 2 3 2 2 4 | 2 1

Greedy:

M_i	M_1	M_2	M_3	M_4
A_1	2	3	2	2
	4	1	1	2
T_i	6	4	3	4

$T = 6$

Best possible:

M_1	M_2	M_3	M_4
4	3	2	2
1	1	2	2
5	4	4	4

OPT(T) = 5

So, greedy doesn't give best possible solution

Observations

Let T^* denote the best possible makespan.

- $\sum_{j=1}^{n} t_j$ ← total work

1. $$T^* \geq \frac{\sum_{j=1}^{m} t_j}{m}$$ if we divide jobs uniformly across all 'm' machines.

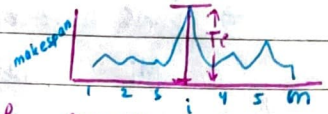
2. $$T^* \geq \max_j t_j$$

The longest job will be on some machine which will work for at least that much time

So, we have provided the lower bound on the value of the best possible solution.

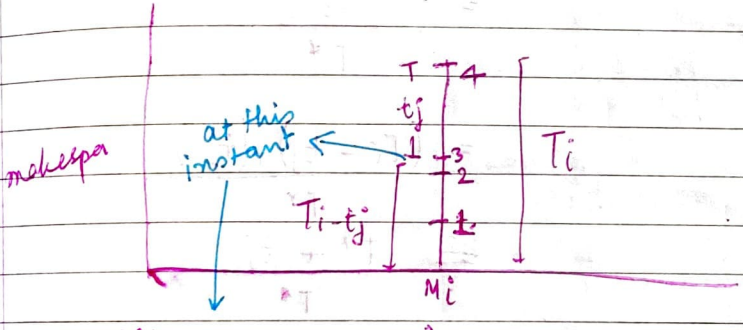
i.e. lower bound on OPT(I) which is in denominator of approx ratio, hence this gives upper bound on the ratio.

Let M_i be machine that has the largest makespan of T_i



Let t_j be the last job assigned to M_i.

Why did we give t_j to M_i? because M_i had the least load before assigning t_j to M_i.



every other machine had load at least $T_i - t_j$

and at this instant —

the total runtime of jobs ~~have~~ —

→ total runtime is at least $m \times (T_i - t_j)$

$$= \sum_{j=1}^{n} t_j \geq m(T_i - t_j)$$

$\sum_{j=1}^{n} t_j$: sum of job durat^n

$m(T_i - t_j)$: sum of duration of jobs upto t_j

$$= T_i - t_j \leq \frac{\sum_{j=1}^{n} t_j}{m} \leq T^*$$

makespan of M_i before assigning t_j

$\Rightarrow \underbrace{T_i - t_j}_{\text{makespan of } M_i \text{ before assigning } t_j} \leq \underbrace{T^*}_{\text{global makespan}}$ — (2)

$\Rightarrow$ just before we assign t_j to our machine M_i, its makespan is smaller than the smallest possible makespan

Now, ratio $= \frac{|A(I)|}{|OPT(I)|}$

$= \frac{T_i}{T^*}$

$= \frac{(T_i - t_j) + (t_j)}{T^*}$

$\leq \frac{T^* + t_j}{T^*}$

$(\; t_j \leq \max_k t_k \leq T^* \;)$

$\leq \frac{2T^*}{T^*}$

≤ 2

For any input, the makespan produced by our algo is no bigger than 2 times the best possible makespan

SORTED GREEDY ASSIGNMENT

Sort jobs in descending order.

when $n \leq m$, we can easily produce the best possible makespan through this algo as it assigns one job to each machine and

$$T_i = \max_k t_k$$

So, for $n > m$, decreasing order of t_j –

$$t_1 \geq t_2 \geq t_3 \geq \dots \geq t_m \geq t_{m+1} \dots \geq t_n$$

and

M_1 M_2 $\dots$ M_i $\dots$ M_m machines

So, we place first m jobs on m machines

J_1	J_2	J_3	$\dots$	J_i	$\dots$	J_m
M_1	M_2	M_3	$\dots$	M_i	$\dots$	M_m

J_{m+1}

and $t_{m+1} \leq t_i$

So, $T^* \geq 2\, t_{m+1}$

as $t_i \geq t_{m+1}$

$\therefore n > m$ some machines will get at least 2 jobs and one of the jobs would be at least as long as t_{m+1}.

this is true for any assignment

If t_j is last job assigned to M_i

then, $t_j \leq t_{m+1}$ (as $j \geq m+1$ and duration sorted)

but we already know,

$$t_{m+1} \leq \frac{T^*}{2}$$

So,

$$T_i = T_i - t_j + t_j$$

$$\leq T^* + t_j$$

$$\leq T^* + \frac{T^*}{2}$$

$$\leq \frac{3}{2} T^*$$

Example to show sorted greedy does not produces best possible makespan

Class 22 : 9th April

BIN PACKING

Given a set of items with weights $w_1, w_2, \ldots w_n$ b/w 0 and 1, find the smallest number of unit sized bins that can pack all the items. (NP- Complete)

Algo 1 : FIRST FIT

- consider all currently available bins
- if any bin can keep the current item, add it
- If no current bin can keep the item, then a new bin is added.

eg. 0.4 0.4 0.3 0.5 0.5 0.2 0.7 0.4

Best soln = [0.7, 0.3] [0.5, 0.5] [0.4, 0.4, 0.2] [0.4] = 4

First fit = [0.4, 0.4, 0.2] [0.3, 0.5] [0.5, 0.4] [0.7] = 4

Date
Page

Prove = For any input algo will use no more than $2 * OPT(I)$

Proof:

Let our algo use k bins i.e. $|A(I)| = k$

$B_1 B_2 \dots B_k$

$B_i + B_{i+1} \geq 1$

For every pair of bins,

$[B_1 B_2]$ $[B_3 B_4]$ $\dots$ $[B_{k-1} B_k]$

≥ 1 ≥ 1 ≥ 1

$\Rightarrow \frac{1}{2} \sum_i w_i \geq \frac{k}{2}$

$\Rightarrow$ Sum of weights of all items $\geq \frac{k}{2}$

Now,

sum of weights of all items is a lower bound on the no. of bins needed.

$\Rightarrow |OPT(I)| \geq \sum_i w_i$

$\Rightarrow |OPT(I)| \geq k/2$

and $\frac{|A(I)|}{|OPT(I)|}$ (minimisation problem) $\leq \frac{k}{k/2}$

$$\frac{|A(I)|}{|OPT(I)|} \leq 2$$

what kind of inputs drive this algorithm to do so bad

SORTED FIRST FIT - Apply first fit in BIN ORDER
First fit in decreasing order of weights

Proof = let $|A(I)| = k$

Consider bin numbered $t = 2k/3$

Case 1 = Bin t contains an item of size greater than 1/2. (heavy item)

Case 2 = Bin t does not contain an item of size greater than 1/2. (light items only)

Case 1 = if bin t contains heavy item

⇒ each of first t bins contain heavy items (∵ decreasing order).

and no two heavy items will share a bin so will require at least 1 bin of their own in optimal soln.

⇒ even best possible soln would use at least 't' bins.

so, $|A(I)| = k$

$|OPT(I)| \geq t$

$$\frac{|A(I)|}{|OPT(I)|} \leq \frac{k}{t} \leq \frac{k}{2k/3} \leq \frac{3}{2}$$

Case 2 : lots of small items

given $|A(I)| = k$,

here bin t does not have size at least 1/2

① ⇒ we put all heavy items till t-1

② ⇒ none of these items fit in t-1 bins (because we created new bins)

③ → Each such bin (t to k) will have at least 2 items.

Because if (<1/2) we cannot create new bin for an item (<1/2)

So it has to go

④ all items in all bins from t to k have sizes no bigger than 1/2.

This allows us to place a lower bound on the sum of sizes of all items in the input.

Consider bins t through k,

So, bin t+1 --- k = 2 items each and t has at least 1 item

$\Rightarrow 2(k-(t+1)+1) + 1$

$\Rightarrow 2(k-t) + 1$ ← bin t

each of these $2(k-t)+1$ items do not fit in bins 1 through t-1.

$\Rightarrow$ total weight of all items exceeds $2(k-t)+1$.

How?

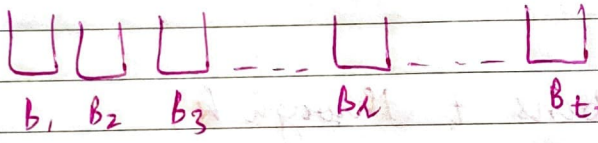
Consider light items - I_1 I_2 I_3 ... $I_{2(k-t)+1}$

If we keep/map each of these to bins 1 to t-1 we know,

$$W(B_1) + I_1 \geq 1$$

$$W(B_2) + I_2 \geq 1$$

$$W(B_{k}) + I_{2(k-t)+1} \geq 1$$



$$\Rightarrow \sum w_i \geq 2(k-t) + 1 \quad \text{--- (1)}$$

Heavy items

→ since adding light items to first t-1 bins is not possible as it will result in overflowing

$$\Rightarrow \sum w_i \geq t-1 \quad \text{--- (2)}$$

$\Rightarrow$ Total weight

So, combine (1) & (2)

$$\sum w_i \geq \min(t-1,\ 2(k-t)+1)$$

~~$3t = 2k + 2$~~

~~$t - 1 = 2(k - k) + 1$~~

~~$t = 2t - 2k + 2$~~

~~$2(k+1) = t$~~

$$t - 1 = 2k - 2t + 1$$

$$3t = 2k + 2$$

$$t = \frac{2k}{3} + \frac{2}{3}$$

Since $t = 2k/3$, we conclude

$$\sum w_i \geq \frac{2k}{3}$$

So, $|OPT| \geq \sum w_i \geq 2k/3$

$$|A(I)| = k$$

So, $\dfrac{|A(I)|}{|OPT(I)|} = \dfrac{k}{2k/3} = \dfrac{3}{2}$