Suppose we have a new problem to solve.

We try

① To devise an efficient algorithm for the problem, using the techniques we've covered, or even more advanced techniques.

② To show that it is unlikely an efficient optimal algorithm exists.

②' Show that the instances we encounter in practice do/don't have some sort of special structure.

③ ?
$\Longrightarrow$ Try to design an algorithm that gets us a "good" (if not optimal) solution in polynomial time.

One such approach is Approximation-Algorithms.

Approximation Algorithms (AA)

- Run in polynomial time
- Provide a solution that is provably close to optimal.

Key difficulty - show that the solution we find is not far from an optimal solution (Note: in many cases where we apply AA, OPT is hard to compute).

Example: The load balancing problem

Given: A set of m machines $M_1, \dots, M_m$ and a set of n jobs such that each job j has processing time t_j.

Find: An assignment of jobs to machines that minimizes the maximum makespan.

$T = \max_i T_i$ where

$$T_i = \sum_{j \in A(i)} t_j$$

and $A(i)$ is the set of jobs assigned to machine i.

Note: The load balancing problem is NP-Hard

A greedy Approx. Algo for Load Balancing

- Assign job j to the machine with the smallest load so far

Greedy-Balance:

Set $T_i = 0$ and $A(i) = \emptyset$ for all M_i

for $j = 1, \ldots, n$:

Let M_i be a machine with a minimum $\min_k T_k$

$A(i) = A(i) \cup \{j\}$

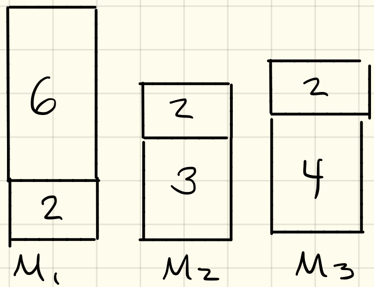
$T_i = T_i + t_j$

End

E.g. Consider the instance $M = \{M_1, M_2, M_3\}$, $J = \{1, 2, 3, 4, 5, 6\}$

$t_1 = 2$, $t_2 = 3$, $t_3 = 4$, $t_4 = 6$, $t_5 = 2$, $t_6 = 2$

Greed-Balance would give



The makespan here is 8 (not optimal; we could achieve 7 ... how?).

Let T be the Greedy-Balance makespan, wish to show that it is not much larger than T^*, the optimal makespan.

- Don't have a good way of computing T^* generally
- Will consider T vs. a "lower bound" on the optimal solution. A lower bound is always at least as small as T^*.

E.g. $T^* \geq \frac{1}{m} \sum_j t_j$

because there must be at least one machine that does at least $1/m$ fraction of the work (i.e. the average work).

But, what if the t_j are very uneven? We could find an optimal solution that still doesn't match the lower bound. We want a LB as tight as possible. Consider another:

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

Because some machine must run the slowest job.

Theorem: Greedy-Balance produces an assignment with makespan

$T \leq 2T^*$

Proof: When we assigned job j to M_i, we know M_i had the smallest load of any machine. Just before assigning job j, M_i had total load $T_i - t_j$. Since this was the smallest load, every other machine also had a load at least as large. Thus:

$$\sum_k T_k \geq m(T_i - t_j) \quad \text{or} \quad T_i - t_j \leq \underbrace{\frac{1}{m}\sum_k T_k}_{\substack{\text{our first} \\ \text{lower bound}}} \leq T^*$$

Now, we account for the remaining load on M_i, which is just t_j. Our second lower bound gives us that $T^* \geq \max_k t_k \geq t_j$.

So, after assignment of t_j, M_i has load

$$T_i = (T_i - t_j) + t_j \leq T^* + T^* = 2T^*$$

So, our makespan is no longer than $2T^*$

Put another way, before the addition of j, our makespan was at most lower bound 1, and we added at most lower bound 2, so our total makespan is $\leq 2T^*$

We can indeed come close to this factor of 2 in practice (i.e. not actually do better than $2T^*$). Consider the following instance.

m machines and $n = m(m-1)+1$ jobs. The first $n-1 = m(m-1)$ jobs have $t_j = 1$, the last job has $t_j = m$.

Our greedy algorithm schedules the first $n-1$ jobs evenly across machines, and then assigns the last job to one of these machines resulting in a makespan of $T = 2m-1$, while the optimal solution has a makespan of m

$$\lim_{m \to \infty} \frac{2m-1}{m} = 2$$

- We can do better!

Sorted-Balance

$T_i = 0$, $A(i) = \emptyset$ for all M_i

Sort jobs in <u>decreasing</u> order of processing time

for $j = 1, \ldots, n$:

let M_i be the machine with $\min_k T_k$

$A(i) = A(i) \cup \{j\}$

$T_i = T_i + t_j$

End

Consider yet one more lower bound. If there are $> m$ jobs then $T^* \geq 2t_{m+1}$. The first $m+1$ jobs in the sorted order take at least t_{m+1} time, but they are run on only m machines. Some machine is assigned 2 such jobs and has processing time $\geq 2t_{m+1}$.

Theorem: Sorted-Balance produces an assignment with makespan $T \le (3/2)T^*$

Proof: Consider a machine M_i with maximum load. If M_i has only 1 job, the schedule is optimal (why?)

Assume M_i has at least 2 jobs, let t_j be the time required for the last job assigned. $j \geq m+1$ (since the first m jobs go to distinct machines). So $t_j \le t_{m+1} \le 1/2\, T^*$

Proceeding as in the previous proof, we know that $T_i - t_j \le T^*$ and $t_j \le (1/2)T^*$ so

$$(T_i - t_j) + t_j \le T^* + \tfrac{1}{2}T^* = (3/2)T^*$$