1. Introduction to Approximation Algorithm

Bingyue Bei

January 24, 2025

Exercise 1.1

i For the maximization problem, we have definition:

$$\textsc{alg}(\mathrm{I}) \geq \rho \cdot \textsc{opt}(\mathrm{I}) \quad \forall \text{ I as input.}$$

If $\rho > 1$, we have

$$\textsc{alg}(\mathrm{I}) \geq \rho \cdot \textsc{opt}(\mathrm{I}) > \textsc{opt}(\mathrm{I}),$$

which contradicts the definition of an optimal solution.

ii For the minimization problem, we have definition:

$$\textsc{alg}(\mathrm{I}) \leq \rho \cdot \textsc{opt}(\mathrm{I}) \quad \forall \text{ I as input.}$$

If $\rho < 1$, we have

$$\textsc{alg}(\mathrm{I}) \leq \rho \cdot \textsc{opt}(\mathrm{I}) < \textsc{opt}(\mathrm{I}),$$

which contradicts the definition of an optimal solution.

Exercise 1.2

We say that the approximation ratio ρ is tight when:

$$\rho = \inf_{I} \frac{\textsc{alg}(\mathrm{I})}{\textsc{opt}(\mathrm{I})}$$

where the infimum is taken over all possible inputs I.

Exercise 1.3

i Consider the job array [80, 80, 40]. Note here that OPT(I) = 80 + 40 = 120. The work distribution is:

- Machine 1: [80, 40]
- Machine 2: [80]

In this case, ALG(I) returns the optimal result, and is 1.05-approximation.

ii We started by building an intuition for the problem we are facing. Since the tasks all have very small sizes, in the optimal case, they will be distributed evenly between the two machines, returning a makespan which is approximately 100. So the result for a 1.05-approximation algorithm should be slightly above and very close to 105.
Here is a formal proof: Since $t_1, t_2, \ldots, t_n \leq 10$, let M_i be the workload on machine o_1 and M_2 be the workload on machine o_2. We have $|M_1 - M_2| \leq 20$ for an optimal solution.
We proof this claim by constructing a proof by contradiction. Without loss of generality, assume $M_1 \geq M_2 + 20$. Remove any task, represented by t', from the workload of M_1, and added it into the workload of M_2. The new workload for o_1, $M_1' = M_1 - t' \geq M_1 - 10$ since $t' \leq 10$. The new workload for o_2, $M_2' = M_2 + t' \leq M_2 + 10$. Since $M_1 \geq M_2 + 20$, we have $M_1 - 10 \geq M_2 + 10$, which is equivalent to $M_1' \geq M_2'$. The makespan have been reduced from M_1 to M_1' We thus created a new solution better than the previous one which claimed to be the optimal.
Since $M_1 + M_2 = 200$, and $|M_1 - M_2| \leq 20$. For an optimal solution, we have $M_1, M_2 \leq \frac{200+20}{2} = 110$. So the optimal solution has an upper-bound of 110, $\textsc{opt}(\text{I}) \leq 110$. Our algorithm is an $1.05 -$ approximation, meaning $\textsc{alg}(\text{I}) \leq 1.05 \cdot \textsc{opt}(\text{I}) = 1.05 \cdot 110 = 115.5$.
Professor's claim has to be false.

Exercise 1.4

By definition of approximation algorithm, we have:

- $\textsc{opt}(\text{I}) \leq \textsc{alg1}(\text{I}) \leq 2 \cdot \textsc{opt}(\text{I})$
- $\textsc{opt}(\text{I}) \leq \textsc{alg2}(\text{I}) \leq 4 \cdot \textsc{opt}(\text{I})$

(a) This statement is false.
The problem only claim that ALG2 is a 4-approximation algorithm. It does not mean the approximation ratio 4 is tight. It is possible that ALG2 is the optimal solution itself, which also satisfy the quaternion for 4-approximation algorithm. If that is the case, the fact that ALG1 is a 2-approximation algorithm guarantees that there would be no solution satisfying $\textsc{alg2}(\text{I}) = \textsc{opt}(\text{I}) \geq 2 \cdot \textsc{alg1}(\text{I})$.

(b) This statement is true.
Proof by contradiction. Suppose we have $\textsc{alg1}(\text{I}) > 2 \cdot \textsc{alg2}(\text{I})$. By definition for 4-approximation algorithm, we have

$$\textsc{alg1}(\text{I}) > 2 \cdot \textsc{alg2}(\text{I}) \geq 2 \cdot \textsc{opt}(\text{I})$$

Also using the definition for 2-approximation algorithm, we have

$$\textsc{alg1}(\text{I}) \leq 2 \cdot \textsc{opt}(\text{I})$$

Combining the two inequalities, we have

$$2 \cdot \textsc{opt}(\text{I}) \geq \textsc{alg1}(\text{I}) > 2 \cdot \textsc{alg2}(\text{I}) \geq 2 \cdot \textsc{opt}(\text{I})$$

which is a contradiction since the left most end and the right most end of the strict inequality represents the same value.

Exercise 1.5

i Given the constraints of the problem we have:

$$\sum_{j=1}^{n} t_j \geq 500, \quad t_1, t_2, ..., t_n \leq 25$$

Let M_i be the machine whose workload determines the makespan. Consider the scheduling of its last job j^* with time t_{j^*}. Before that job get scheduled, the total processing time of all the job which has been scheduled is at least 475.
Let $load(M_i^*)$ represents the load of machine M_i before its last job gets scheduled.

$$load(M_i^*) = \frac{1}{5}(\sum_{j=1}^{j<j^*} t_j) \geq \frac{1}{5}(\sum_{j=1}^{n} t_j - t_{j^*}) \geq \frac{(500-25)}{5} = \frac{475}{5}$$

Since that machine is where the last job is assigned to, workload of that machine has an upper-bound of $95 + 25 = 120$.

ii Considering the task array of [1, 1, ..., 1, 25], which is an array composed by $500 - 25 = 475$ of 1's and one 25 lingering at the end. Apply our greedy scheduling algorithm to this array. Before the last job with finish time 25 is assigned, the jobs with finishing time 1 is distributed evenly between the five machines, which makes each of the five machine have a total workload of $\frac{500-25}{5} = \frac{475}{5} = 95$, with its work stack looks like [1, 1, ..., 1](composed of all 1's).
The last job would be assigned to a machine with the least total workload. In this case, since all machine have the same workload, which machine we assigned it to does not change the resulting makespan, which is $95 + 25 = 120$.
However, if we assigned the job with finish time 25 first to one of the machine, and then assign jobs with finish time 1 based on greedy scheduling, we would have one machine with task stack that looks like [25, 1, 1, ..., 1] and the rest four machines with task stack that looks like [1, 1, ..., 1], all of them have a total finish time of $\frac{500}{5} = 100$, which is also the optimal solution.
The ratio between optimal solution and the results given by greedy scheduling algorithm is $\frac{120}{100} = 1.2$.
This case demonstrate perfectly what we have discussed in lecture that the pitfall of Greedy Scheduling is when we have a gigantic job assigned at the end of the array, when each machine have similar total workload. That gigantic job would drastically increase the resulting makespan.

Exercise 1.6

Constructing a worst case scenario using similar idea from the previous question. Consider the input array [1, 1, ..., 1, m], which consists of $m * (m-1)$ jobs of size 1 followed by a single job of size m. The total workload of the array would be $m*(m-1)*1+1*m = m*[(m-1)+1] = m*m = m^2$, being distributed into m machines.
The optimal solution would be a makespan of m, which one machine being assigned a job of m, and all other machines being assigned m jobs of size 1.
Now consider how the jobs would be distributed with *Greedy-Scheduling*. Before the last big job of size m is assigned, all the smaller jobs of size 1 would be distributed evenly across all m machines. All machines would have a workload array which looks like [1, 1, ..., 1], consist of $m-1$ jobs of size 1. After the last job is assigned, one machine would have a workload increased by m, thus dictating the makespan of the system, which would be $(m-1)+m = 2*m-1$.
Therefore, the ration bewteen ALG(I) and OPT(I) is $\frac{2m-1}{m} = 2 - \frac{1}{m}$.
We have thus constructed an example which makes the approximation ratio tight.
Intuition for constructing our example: The proof of the $2-\frac{1}{m}$ ratio contains two inequalities. The first one being $\frac{1}{m}\sum_{j=1}^{n} t_j \leq$ OPT(I). The second being $t_j{}^* \leq$ OPT(I). To make the approximation ratio tights, both inequalities should be equalities instead.

Exercise 1.7

Consider the input array [3, 3, 2, 2, 2], distributed into 2 machines. Since the array has already been sorted into descending order, the *Greedy-Scheduling* and the *Ordered-Scheduling* will essentially run the same routine.
Both algorithm would return the same results:

- Machine 01: [3, 2, 2]
- Machine 02: [3, 2]

With the resulting makespan being $3 + 2 + 2 = 7$.
However, the optimal solution is:

- Machine 01: [3, 3]
- Machine 02: [2, 2, 2]

With the optimal makespan being $2 + 2 + 2 = 3 + 3 = 6$.
In this case, neither algorithm returns the optimal solution. In Exercise 1.8 and 1.9, we will see how this case is constructed.

Intuitive Example: *Ordered-Scheduling* is (4/3)-approximation

Consider the input array [$2m$, $2m$, $2m-1$. $2m-1$, ... , $m+1$, $m+1$, m], distributed into m machines. Noted that the input array has already been sorted into descending order, so *Ordered-Scheduling* essentially runs the same routine as the *Greedy-Scheduling* algorithm. There are a total of $2 * [(2m) - (m+1) + 1] + 1 = 2m + 1$ jobs in the input array.
Before the last job is scheduled, the $2m$ jobs should be distributed optimally, creating a makespan of $3m + 1$, which is also the workload of any machine.

- Machine 1: [$2m$, $m+1$]
- Machine 2: [$2m-1$, $m+2$]
  $\vdots$
- Machine $m-1$: [$m+2$, $2m-1$]
- Machine m: [$m+1$, $2m$]

After the last job is scheduled, the makespan of the system would be $(3m+1) + m = 4m + 1$.
We then consider the optimal solution for this input. First, calculate the lower bound for the optimal solution.

$$\frac{1}{m}\sum_{j=1}^{n} t_j = \frac{1}{m}[(m) * (3m+1) + m] = \frac{1}{m}(3m^2 + 2m) = 3m + 2$$

We then provide a scheduling scheme which shows this lower bound is feasible.

- Machine 1: [$m+1$, $m+1$, m]
- Machine 2: [$2m$, $m+2$]
- Machine 3: [$2m-1$, $m+3$]
  $\vdots$

- Machine $m-1$: $[m+3,\ 2m-1]$
- Machine m: $[m+2,\ 2m]$

The approximation ratio based on this specific input is:

$$\frac{\text{ALG(I)}}{\text{OPT(I)}} = \frac{4m+1}{3m+2} \to \frac{4}{3}, m \to \infty$$

Note: The full proof is provided by Graham in 1969, showing the 4/3 approximation ratio both feasible and TIGHT. It is not included here since the proof is not part of the curriculum of this course. However the above example is quite useful to think of when you are trying to construct any example with scheduling problems. Fix $m = 2$ and you have the example we provide for Exercise 1.7.

Exercise 1.8

Intuition: Consider when the optimal solution is the same as $t_m + t_{m+1}$. One situation is that there is precisely one more task then there is machines. In this case, one machine gets assigned two tasks (with its workload being at least $t_m + t_{m+1}$) while all other gets assigned one tasks. Consider the input array $[t,\ t,\ t,\ t,\ t]$, with t being any positive real number indicating size of a job, distributed into 4 machines. The *Ordered-Scheduling* algorithm returns a makespan of $2t$ with the following job distributed:

- Machine 01: $[t,\ t]$
- Machine 02: $[t]$
- Machine 03: $[t]$
- Machine 04: $[t]$

Now calculate the three lower bound:

- The average individual job size

$$\text{LB}_1 = \frac{1}{m}\sum_{j=1}^{n} t_j = \frac{5t}{4}$$

$$\frac{3}{2}\text{LB}_1 = \frac{3}{2} * \frac{5t}{4} = \frac{15t}{8} < \frac{16t}{8} = 2t$$

- Maximum job size: $\text{LB}_2 = \max_{1 \le j \le n} t_j = t$. $\frac{3}{2}\text{LB}_2 = \frac{3}{2}t < 2t$
- Sum of the last two jobs get scheduled: $\text{LB}_3 = t_m + t_{m+1} = t + t = 2t = \text{OPT(I)}$.

Exercise 1.9

If there are less job than machines, than each machine gets assigned one job or less and the solution is optimal. Therefore, we could assume $n > m$ in the subsequent proof.
Consider the input array be $[t_1,\ t_2,\ ...\ t_n]$ after sorting. We therefore have the ordering $t_1 \ge t_2 \ge ... \ge t_n$. Let Machine M_{i^*} be assigned the maximum workload and therefore determine the makespan. Let t_{j^*} be the last job gets scheduled to that machine. To have t_{j^*} assigned to M_{i^*}, the machine needs to have the minimum workload right before the assignment. Since its

workload at that time is the minimum across all machine, it has to be less than or equal to the average workload.

$$load(M_{i^*}) = load(M'_{i^*}) + t_{j^*} \leq \frac{1}{m}[(\sum_{j=1}^{n} t_j) - t_{j^*}] + t_{j^*} = \frac{1}{m}(\sum_{j=1}^{n} t_j) + \frac{m-1}{m} t_{j^*}$$

The first term is one of our three lower bound for the optimal solution.

$$\frac{1}{m}(\sum_{j=1}^{n} t_j) \leq \text{OPT(I)}$$

For the second term, we use the fact $j^* > m$. Since jobs are sorted after assigned, we have

$$\frac{m-1}{m} t_{j^*} \leq \frac{m-1}{m}[\frac{1}{2}(t_m + t_{m+1})] \leq (1 - \frac{1}{m})(\frac{1}{2}\text{OPT(I)})$$

Combining the two inequalities, we have our desired results of $(\frac{3}{2} - \frac{1}{2m})\text{OPT(I)}$.

Exercise 1.10

(i) For $m = 2$ and $n \leq 4$, assume we are given the input array $[t_1, t_2, t_3, t_4]$, sorted into descending order. For cases where we have strictly less than 4 jobs, let $t_i = 0$. In this way we fix $n = 4$. Apply our algorithm. After the two jobs are assigned, we have

- Machine 01: $[t_1]$
- Machine 02: $[t_2]$

Since the second machine has the smaller workload, job 3 would be assigned to that machine.

- Machine 01: $[t_1]$
- Machine 02: $[t_2, t_3]$

If Machine 01 has the smaller workload, job 4 gets assigned to Machine 01, the resulting makespan a lower bound for our optimal results.

$$\text{ALG(I)} = \min(t_1 + t_4, t_2 + t_3) \leq t_2 + t_3 = t_m + t_{m+1} \leq \text{OPT(I)}$$

If Machine 02 has the smaller workload, job 4 gets assigned to Machine 02, the resulting makespan equals t_1, which is also one of the lower bound for the optimal results. Our results is optimal.

$$t_1 = \max_{1 \leq j \leq n} t_j \leq \text{OPT(I)}$$

(ii) Our example for Exercise 1.7 gives the $\frac{7}{6}$ approximation ratio.

(iii)

Summary

Here are some important takeaways from this exercise:

- What is the worst case input scenario for *Greedy-Scheduling*? (Note: Exercise 1.6 shows that this input would make the approximation ratio is tight).
- Based on Exercise 1.7, consider the following statement: For **ANY** input, if *Ordered-Scheduling* cannot return the optimal solution, nether could *Greedy-Scheduling*. Is this statement true? (Hint: try to construct an input where *Ordered-Scheduling* returns a worse results than *Greedy-Scheduling*.)