Operations Research, Spring 2024 (112-2)

Midterm Project

B11705004 Tzu-Hsin Chou B11705027 Cheng-Yu Chen

B11705018 Shu-Ting Hsu 111356509 Yun-Yun Jhuang B11705033 Jui-Chen Chiang

April 29, 2024

Problem 1

Suppose there are n_j jobs and n_m machines. Let $J = \{1, ..., n_j\}$ be the set of jobs, and $M = \{1, ..., n_m\}$ be the set of machines.

p_{ij} is the processing time of job i at stage j. $\forall i \in J,\ j = 1, 2$.

D_i is the due time of job i. $\forall i \in J$.

t_i is the tardiness of job i. $\forall i \in J$.

c_{ij} is the complete time of the stage j of job i. $\forall i \in J,\ j = 1, 2$.

x_{ijk} is 1 if job i can be processed on machine k at stage j; otherwise, is 0.

$\forall i \in J,\ j = 1, 2,\ k \in M$.

b_{ijkrs} is 1 if stage j of job i is processed on machine k before stage s of job r; otherwise, is 0.

$\forall i, r \in J,\ j, s = 1, 2,\ k \in M$.

m_{ijk} is 1 if stage j of job i is processed on machine k; otherwise, is 0. $\forall i \in J,\ j = 1, 2,\ k \in M$.

w_{ij} is 1 if stage j of the job i not exist; otherwise, is 0. $\forall i \in J,\ j = 1, 2$.

$$
\begin{aligned}
\min \quad & \sum_{i \in J} t_i \\
\text{s.t.} \quad & N = \sum_{i \in J} \sum_{j=1}^{2} p_{ij} \\
& t_i \geq c_{i2} - D_i, \, \forall i \in J \\
& c_{i1} + p_{i2} \leq c_{i2}, \, \forall i \in J \\
& c_{i1} + p_{i2} + 1 \geq c_{i2}, \, \forall i \in J \\
& m_{ijk} \leq x_{ijk}, \, \forall i \in J, \, j = 1, 2, \, k \in M \\
& \sum_{k \in M} m_{ijk} = 1 - w_{ij}, \, \forall i \in J, \, j = 1, 2 \\
& 2b_{ijkrs} \leq (m_{ijk} + m_{rsk}), \, \forall i, r \in J, \, j, s = 1, 2, \, k \in M \\
& b_{ijkrs} + b_{rskij} \leq 1, \, \forall i, r \in J, \, j, s = 1, 2, \, k \in M, \, s \neq j \text{ or } i \neq r \\
& b_{ijkrs} + b_{rskij} \geq m_{ijk} + m_{rsk} - 1, \, \forall i, r \in J, \, j, s = 1, 2, \, k \in M, \, s \neq j \text{ or } i \neq r \\
& c_{ij} + p_{rs} - c_{rs} \leq N(1 - b_{ijkrs}), \, \forall i, r \in J, \, j, s = 1, 2, \, k \in M, \, s \neq j \text{ or } i \neq r \\
& c_{ij} \geq p_{ij}, \, \forall i \in J, \, j = 1, 2 \\
& c_{ij} \geq 0, \, \forall i \in J, \, j = 1, 2 \\
& t_i \geq 0, \forall i \in J \\
& x_{ijk} \in \{0, 1\}, \, m_{ijk} \in \{0, 1\}, \, \forall i \in J, \, j = 1, 2, \, k \in M \\
& b_{ijkrs} \in \{0, 1\}, \, \forall i, r \in J, \, j, s = 1, 2, \, k \in M \\
& w_{ij} \in \{0, 1\}, \, \forall i \in J, \, j = 1, 2.
\end{aligned}
$$

Let $m_{\max}$ be the max of makespan.

z be the total tardiness of above result.

$$
\begin{aligned}
\min \quad & m_{\max} \\
\text{s.t.} \quad & \sum_{i\in J} t_i \le z \\
& N = \sum_{i\in J}\sum_{j=1}^{2} p_{ij} \\
& t_i \ge c_{i2} - D_i,\ \forall i \in J \\
& c_{i1} + p_{i2} \le c_{i2},\ \forall i \in J \\
& c_{i1} + p_{i2} + 1 \ge c_{i2},\ \forall i \in J \\
& m_{ijk} \le x_{ijk},\ \forall i \in J,\ j = 1,2,\ k \in M \\
& \sum_{k\in M} m_{ijk} = 1 - w_{ij},\ \forall i \in J,\ j = 1,2 \\
& 2b_{ijkrs} \le (m_{ijk} + m_{rsk}),\ \forall i,r \in J,\ j,s = 1,2,\ k \in M \\
& b_{ijkrs} + b_{rskij} \le 1,\ \forall i,r \in J,\ j,s = 1,2,\ k \in M,\ s \ne j \text{ or } i \ne r \\
& b_{ijkrs} + b_{rskij} \ge m_{ijk} + m_{rsk} - 1,\ \forall i,r \in J,\ j,s = 1,2,\ k \in M,\ s \ne j \text{ or } i \ne r \\
& c_{ij} + p_{rs} - c_{rs} \le N(1 - b_{ijkrs}),\ \forall i,r \in J,\ j,s = 1,2,\ k \in M,\ s \ne j \text{ or } i \ne r \\
& m_{\max} \ge c_{ij},\ \forall i \in J,\ j = 1,2 \\
& c_{ij} \ge p_{ij},\ \forall i \in J,\ j = 1,2 \\
& c_{ij} \ge 0,\ \forall i \in J,\ j = 1,2 \\
& t_i \ge 0, \forall i \in J \\
& x_{ijk} \in \{0,1\},\ m_{ijk} \in \{0,1\},\ \forall i \in J,\ j = 1,2,\ k \in M \\
& b_{ijkrs} \in \{0,1\},\ \forall i,r \in J,\ j,s = 1,2,\ k \in M \\
& w_{ij} \in \{0,1\},\ \forall i \in J,\ j = 1,2.
\end{aligned}
$$

Problem 2

In instance05, the minimized tardiness is 2.30, and the minimized makespans is 9.0. Figure 1 shows the optimal job scheduling.

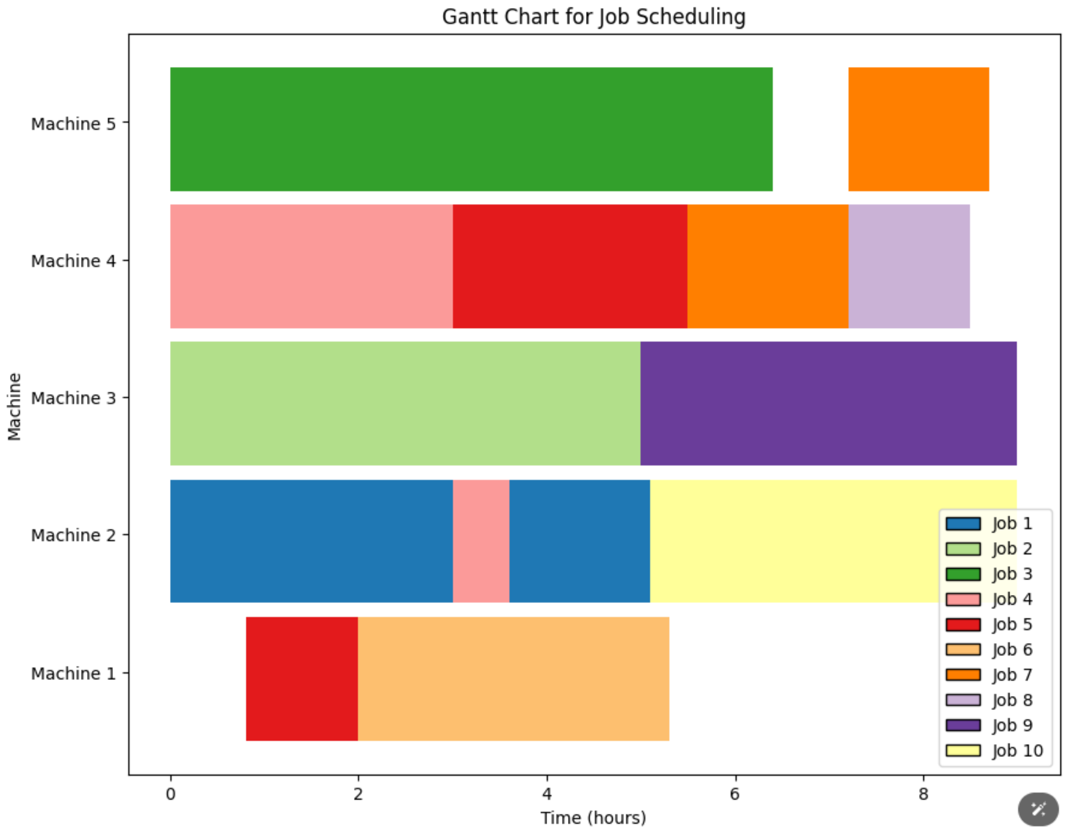

Figure 1: A schedule for the instance05

Problem 3

The program is in the folder along with this doc.

Problem 4

In our Heuristic algorithm, we will find the machine that has the least recent complete time, which means the machine that has already finished the job we assigned, we can then assign a new job for that machine, this make sure we always assign the job to the machine that can complete it the fastest. Among all the jobs, we want to find the most urgent one to run, so the `Find()` function will find the job that is able to run on that machine and has the most processing time as well as the least due time.

Algorithm 1 Find

```
procedure FIND(k)
    for i in jobs do
        if job i can run in machine k in once then
            Find the most urgent
        end if
    end for
    if any job can run on this machine then
        return urgent
    else
        return 300
    end if
end procedure
```

Since there is a for loop running through n_j, which represents the number of jobs, the time complexity of the `Find()` function is $O(n_j)$.

To simplify the problem, we assume that all the job run in once unless it could not (*e.g.,* stage 1 can run on machine 1, 2 and stage 2 can run in machine 3, 4, 5)(we will call it Condition1 after), although this is not the best way to order the job, this is indeed a good and feasible way. The jobs that can not run in once will be ordered in the same way(we will call it Condition2 after), which is finding the minimum recent complete time and match stage 1 and stage 2 to feasible.

Since we are not sure about whether we should run the Condition1 first or Condition2 first, so we try both and find the better one.

To arrange the job of Condition1, we start from a machine that seems to appear least, we assume that it is less importance and we hope it to do as more job as possible since there is not much it can do, so we arrange it first. We use a while loop until there is no left work to do. In every loop, we use `Find()` function to find the job and choose the machine that already finish its job for the next loop. If there is no more job the machine can do, we kick it out.

Algorithm 2 Condition1

while true **do**
    $goal \leftarrow Find(machine)$
    if can't run any job **then**
        delete machine
        continue
    end if
    $time[machine] \leftarrow time[machine] + left[goal]$
    $c[goal] \leftarrow time[machine]$
    for each machine k in machines **do**
        set *goal* can not run on any machine
    end for
    $left[goal] \leftarrow 0$
    continue
    Find machine with minimum time
    if $\sum left - \sum left[i]$ for each i in $not\_in\_one \leq 0$ **then**
        break
    end if
end while

The while loop will run through all the jobs, and the `Find()` function will be process every time. Hence, the time complexity of Condition1 is $O(n_j^2)$.

To arrange the job of Condition2, we find the 2 machines that can finish the 2 stages fastest, and check the complete time of both machines to make share the answer is feasible.

Algorithm 3 Condition2

```
for each task i in not_in_one do
    Stage 1:
    available_machines ← machines that can execute task i
    if available_machines is not empty then
        Find the machine machine1 with the minimum time among available_machines
        Increase the time on machine1 by the processing time of task i for Stage 1
    end if
    Stage 2:
    available_machines ← machines that can execute task i for Stage 2
    if available_machines is not empty then
        Find the machine machine2 with the minimum time among available_machines
        if the time on machine2 is greater than the time on machine1 plus 1 then
            Increase the time on machine2 by the processing time of task i for Stage 2
            Set the time on machine1 to be just before the start time of machine2
        else if the time on machine2 is greater than the time on machine1 then
            Increase the time on machine2 by the processing time of task i for Stage 2
        else
            Increase the time on machine2 to be just after the time on machine1 finishes
        end if
    end if
    Set c[i] to the time on machine2
end for
```

The time complexity of our heuristic algorithm is $O(n_j^2)$.

Problem 5

Our simple algorithm is designed as it will run on current machine until there is no more any possible job that can run, then it will switch to the next machine.

The first scenario is in the following description:

Let the number of jobs be the multiples of 5 (that is, 5, 10, 15, 20, 25) and the number of machines be fixed at 5. Each job has 30% of probability to have only one stage and 70% to have two stages. For each piece of each job, the processing time is uniformly drawn from $[1, 10]$. For each job, the first piece can be processed by all machines, and the second piece can only be processed by machines 2 to 5. For each job, the due time is randomly assigned to be 5 or 10 with equal probability.

scenario	condition		total tardiness			optimal gap (%)		
	jobs	machines	relaxed	heuristic	simple heuristic	realxed vs. heuristic	relaxed vs. simple heuristic	heuristic vs. simple heuristic
1	5	5	24.7	24.7	109.4	0	342.915	342.915
	10	5	24	61.4	265.6	155.8333	1006.667	332.5733
	15	5	32.8	185.8	754.9	466.4634	2201.524	306.2971
	20	5	58.8	379.2	1180.7	544.898	1907.993	211.366
	25	5	61.9	555.2	1719.8	796.9305	2678.352	209.7622

Figure 2: Result in scenario 1

The Figure 2 shows the result. We can find that the optimal gap between relaxed algorithm and heuristic algorithm is smaller than that between heuristic algorithm and simple heuristic algorithm. The average optimal gaps in each comparison are 392.83, 1627.49, 280.58, respectively (rounding to the second decimal place). The standard deviation in each comparison are 317.15, 941.82, 65.30, respectively (rounding to the second decimal place).

The second scenario is in the following description:

All jobs have two stages. All the other settings are identical to scenario 1.

scenario	condition		total tardiness			optimal gap (%)		
	jobs	machines	relaxed	heuristic	simple heuristic	realxed vs. heuristic	relaxed vs. simple heuristic	heuristic vs. simple heuristic
2	5	5	12.3	16.7	128.8	35.77236	947.1545	671.2575
	10	5	27.7	117.7	565.7	324.9097	1942.238	380.6287
	15	5	41.4	268.7	1222	549.0338	2851.691	354.7823
	20	5	81.5	638.1	2737.3	682.9448	3258.65	328.9766
	25	5	69.4	819.5	3555.6	1080.836	5023.343	333.8743

Figure 3: Result in scenario 2

The Figure 3 shows the result. We can find that the optimal gap between relaxed algorithm and heuristic algorithm is smaller than that between heuristic algorithm and simple heuristic algorithm. The average optimal gaps in each comparison are 534.70, 2804.62, 413.90 respectively (rounding to the second decimal place). The standard deviation in each comparison are 391.60, 1526.84, 145.30, respectively (rounding to the second decimal place).

We've also tried the scenario 3, 4, 5 the problem 5 offer, and we find out that the simple heuristic algorithm also works well since it often switch to next machine and do not stuck in the particular operating one. Although our heuristic algorithm is still much better than the simple heuristic algorithm, it is not that significant. Therefore, we don't display the result.

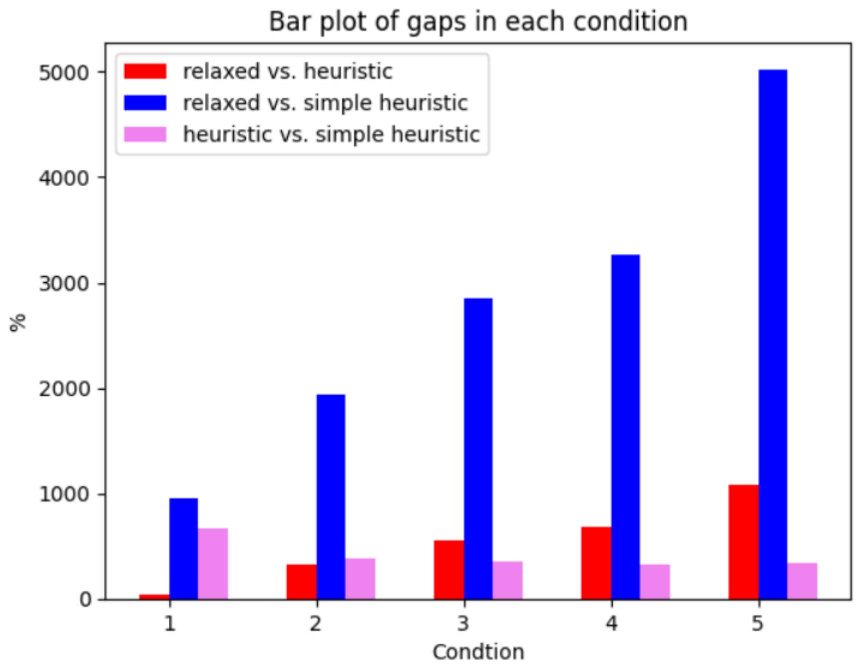

Figure 4: Bar plot of gaps in each condition

In Figure 4, we can find that all the optimal gaps between relaxed algorithm and heuristic algorithm are better than those between relaxed algorithm and simple heuristic algorithm. Compared with the simple heuristic algorithm, heuristic algorithm performs quite well, which indicates that it's much closer to the optimal solution. Though the gaps between heuristic algorithm and simple heuristic algorithm are comparatively smaller. The fact that the heuristic algorithm perform better than simple heuristic algorithm is still true.