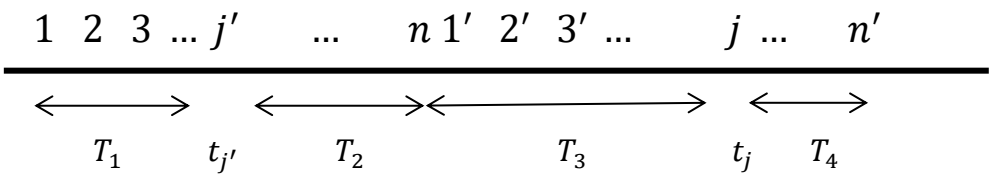
IND4301 Advanced Algorithms - Assignment 1

Student: Yuxuan Kang
Student Number:4504410

Question 1:

1. All the jobs can be regarded as a tuple (r, t), while r is the release time and t is the processing time. In order to minimize max(f) $(f = c - r)$, the task should start as soon as possible.

   The algorithm consists of two steps, first step is to sort all the jobs based on release time. The complexity of the step is the complexity of sorting algorithms, which is $O(n\log(n))$ for many sorting algorithms. The second step is to schedule the jobs in this order,
   So the complexity of the algorithm is $O(n\log(n))$.

2. The jobs are described as (duration, release time), after first step, the jobs are sorted as:
   (2,2),(3,2),(1,2),(1,3),(1,3)
   So the flow times are:
   2, 3, 6, 6, 7
   The maximum flow time is 7 according to this algorithm. There are only two kinds of job in this problem, one with a release time 2 and another with a release time 3. The total processing times for these two kinds of job are 6 and 2 respectively, in total 8. It's possible to choose to start with job with release time 2 or 3. Besides, there is no chance that the machine needs to wait during processing. So there are only two possibilities for the end time: 2+8=10 and 3+8=11. So the possibilities for f_j are: 10-2=8, 10-3=7, 11-2=9, 11-3=8. Among them, 7 is the minimum flow time.
3. 

   Let T_1 be the total processing time from first job1 with release time t to No. $j-1$, thus $T_1 = \sum_{i=1}^{j-1} t_i$.

   Similarly, let $T_2 = \sum_{i=j+1}^{n} t_i$, $T_3 = \sum_{i'=1}^{j'-1} t_{i'}$, $T_4 = \sum_{i'=j'+1}^{n'} t_{i'}$

   $t_{j'}$ be the processing time of job j', T_2 be the total processing time between $j+1$ and the very end of first kind of job, T_3 be the time gap from first job2 with release time r' to job $j'-1$, t_j be the processing

time of job j, T_4 be the time gap between $j+1$ to the end of job2.

If $T_1 < r' - r$, then machine needs to wait until j' can start to process:

$$f_{job1} = T_1 + waiting\ time + T_2 + t_{j'} + T_3 + t_j$$

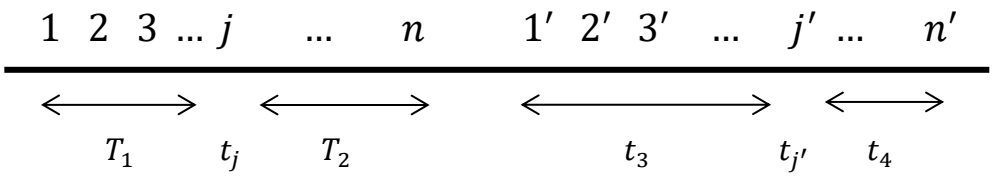
$$f_{job2} = T_2 + t_{j'} + T_3 + t_j + T_4$$

Else, the machine can work without waiting:

$$f_{job1} = T_1 + T_2 + t_{j'} + T_3 + t_j$$

$$f_{job2} = r + T_1 + T_2 + t_{j'} + T_3 + t_j - r'$$

If switch the order of j' and , the new time axis will be:



Now, if $T_1 < r' - r$:

$$h_{job1} = T_1 + T_2 + t_j + T_3 + t_{j'}$$

$$h_{job2} = waiting\ time + T_3 + t_{j'} + T_4$$

$$(waiting\ time \le t_j + T_2)$$

So $h_{job1} \le f_{job1}$ and $h_{job2} \le f_{job2}$

Else,

$$h_{job1} = T_1 + t_j + T_2$$

$$h_{job2} = r + T_1 + T_2 + t_{j'} + T_3 + t_j - r'$$

So $h_{job1} \le f_{job1}$ and $h_{job2} = f_{job2}$

In summary, interchanging the positions of j and j' cannot increase the maximum flow time.

4. According to former questions, it is known that in order to minimize $max_j(f_j)$, the rule 'first release first process' should be followed. So for 2 or more machines, jobs should be sorted based on release time. First released job should be processed first. Based on this rule, for jobs with the same release time, the problem becomes the same as minimizing makespan.

Therefore, finding a schedule that minimizes $max_j(f_j)$ on 2 or more machines can be divided into multiple sub-problems of minimizing makespan for 2 or more machines. The concept can be described as

follows:

Time axis

Group A	Group B	Group C	Group D	Group E
With release Time t_1	With release time t_2	with release time t_3	with release time t_4	with release time t_5
Minimizing makespan	Minimizing makespan	Minimizing makespan	Minimizing makespan	minimizing makespan

Jobs in Group A should be assigned to one of the machines first, then group B … until Group E. Compare to minimizing makespan problem, minimizing $max_j(f_j)$ only needs to consider the order of assigned job first, then do the same procedure as minimizing makespan problem. So makespan problem can be regarded as a special case of minimizing $max_j(f_j)$ where all the jobs have the same release time. So minimizing $max_j(f_j)$ can be reduced to minimizing makespan problem. Since minimizing makespan problem for 2 or more machines are NP-hard problem, minimizing $max_j(f_j)$ for 2 or more machines should also be NP-hard problem.

`

Question 2:

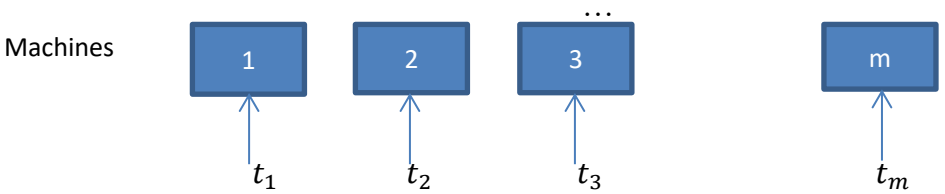
1. Let machine j have 3 or more jobs, thus $n_j \geq 3$. Then the makespan for j should be $T_j \geq n_j \times t_n > T^*$. Then the makespan of j is larger than the makespan for all the jobs which is impossible, because if so, T^* should equal to T_j.
2. Since every machine should have at most 2 jobs. So $n = n_1 + n_1 + \cdots + n_m \leq 2 \times$ m, thus $2m \geq n$.'
3. a. If $n \leq m$, all the machines should only have 1 jobs at most. In this case the makespan should be the longest processing time of all the jobs. Because Sorted-Balance already sorts the jobs in decreasing order or processing times, the makespan should be t_1. So Sorted-Balance produces an optimal schedule.

   b. Sorted-Balance sorts the jobs based on processing time. According to previous conclusion, in this case, all the machines only have 2 jobs at most.

   (1) if makespan happens on the machine with 1 job. Then T= t_1, Sorted-Balance gets the optimal schedule.

   (2) if makespan happens on the machine with 2 jobs.

   Since $t_1 \geq t_2 \geq t_3 \geq t_4 \geq \cdots \geq t_n$, when assigning job m+1 to machines, there appears machine with second job.



Then Sorted-Balance will assign job m+1 to machine m. Because

makespan happens on machine with two jobs, so for Sorted-Balance, makespan T should be :

$$T = \max(t_m + t_{m+1}, t_{m+2} + t_{m-1}, \dots, t_{2m-n+1} + t_n)$$
$$= \max(\dots \max(\max(t_m + t_{m+1}, t_{m+2} + t_{m-1}), t_{m+2} + t_{m-1}) \dots)$$

If job m+1 is assigned to machine m-1 instead of m:

$$t_m + t_{m+1} \le t_{m-1} + t_{m+1}$$
$$t_{m-1} + t_{m+2} \ge t_m + t_{m+2}$$
$$t_{m-1} + t_{m+1} \ge t_m + t_{m+1} \; and \; t_{m-1} + t_{m+1} \ge t_{m-1} + t_{m+2}$$
$$\text{so } \max(t_{m-1} + t_{m+1}, t_m + t_{m+2}) \ge \max(t_m + t_{m+1}, t_{m-1} + t_{m+2})$$

It can be observed that:

$$\min(m - 1 + m + 1, m + m + 2) = 2m$$
$$< \min(m + m + 1, m - 1 + m + 2) = 2m + 1$$

It is easy to draw a conclusion that:

if $\min(a + b, c + d) \ge \min(c + d, g + h)$ then $\max(t_a + t_b, t_c + t_d) \le \max(t_e + t_f, t_g + t_h)$.

Sorted-Balance finds the minimum min(corner mark sum) which is 2m+1. So Sorted-Balanced will find an optimal schedule.