Operations Research, Spring 2022 (110-2) Suggested Solution for Case Assignment 2

Tim Kuo[*], Yuan Ting Lin[†], Yu-Chieh Kuo[‡]

May 21, 2022

1 A heuristic algorithm

We present a heuristic algorithm, which is both simple to implement and yields an adequate performance. The idea is to schedule all the stage-1 jobs first, before the stage-2 jobs. While the idea makes the algorithm easy to implement, it is also reasonable because the algorithm ensures those jobs with only one stage are scheduled sooner. This avoids unintended tardiness, making it easier to achieve better performance.

First, the stage-1 jobs are sorted with the longest-job-first rule. We choose this rule rather than earliest-due-first because we intend to decrease the tardiness, by finishing those jobs with only one stage first. Applying the earliest-due-first rule may result in scheduling those jobs due earlier, but with two stages first, leaving no time for completing long one-stage jobs. Next, according to the order, the jobs are assigned to the legal machine that is most unoccupied. A similar procedure is applied to the stage-2 jobs with a little difference. The stage-2 jobs are sorted by the earliest-due-first rule. If a job cannot meet its due time, the algorithm skips it and schedule other jobs first to minimize the tardiness. Finally, the algorithm schedule those skipped jobs and return the result. The time complexity of this algorithm is $O(mn \log(m + n))$, where n is the number of jobs, and m is the number of machines.

The Pseudocode of our algorithm is described below.

2 Designing numerical experiments

The performance of heuristic algorithms may vary under different situations, so we perform numerical experiments to analyze them. In general, we may set a benchmark such as integer program or linear relaxation, and compare them with the performance of heuristic algorithms under different criteria. For example, in our problem, we may set the number of machines to ”small”, ”medium”, or ”large”. The processing time of the two stages may be ”short” or ”long”, and may be ”evenly” or ”unevenly” distributed. There may be scenarios where ”machines can conduct all routines” or ”machines can only conduct specific routines”, and the due time may also vary a lot. Analyze the performance of heuristic algorithms, and show how they outperform the benchmark under different scenarios.

[*]Department of Information Management, National Taiwan University. E-mail: r10725025@ntu.edu.tw.

[†]Department of Information Management, National Taiwan University. E-mail: b07705036@ntu.edu.tw.

[‡]Department of Information Management, National Taiwan University. E-mail: b07611039@ntu.edu.tw.

```
function HEURISTIC(instance)
    $m$ represents number of machines.
    $n$ represents number of jobs.
    $S_1[1...n]$ represents stage-1 jobs.
    $S_2[1...n]$ represents stage-2 jobs.
    $M[1...n][1...2] \leftarrow 0$
    $C[1...n][1...2] \leftarrow 0$
    $M_t[1..m] \leftarrow 0$
    $D \leftarrow []$

    $S_1$.sort(key=lambda x: x.processTime)
    for job in $S_1$ do
        $M_j \leftarrow$ job.machines
        $p \leftarrow$ job.processTime
        $k \leftarrow$ job.id

        $M_j$.sort(key=lambda x: $M_t$[x])
        $M_t[M_j[1]] \leftarrow M_t[M_j[1]] + p$
        $C[k][1] \leftarrow M_t[M_j[1]]$
        $M[k][1] \leftarrow M_j[1]$
    end for

    $S_2$.sort(key=lambda x: x.dueTime)
    for job in $S_2$ do
        $M_j \leftarrow$ job.machines
        $p \leftarrow$ job.processTime
        $d \leftarrow$ job.dueTime
        $k \leftarrow$ job.id

        $M_j$.sort(key=lambda x: $M_t$[x])
        if $\max(M_t[M_j[1]], C[k][1]) + p \leq d$ then
            $M_t[M_j[1]] \leftarrow M_t[M_j[1]] + p$
            $C[k][2] \leftarrow M_t[M_j[1]]$
            $M[k][2] \leftarrow M_j[1]$
        else
            $D$.append(job)
        end if
    end for

    for job in $D$ do
        $M_j \leftarrow$ job.machines
        $p \leftarrow$ job.processTime
        $k \leftarrow$ job.id

        $M_j$.sort(key=lambda x: $M_t$[x])
        $M_t[M_j[1]] \leftarrow M_t[M_j[1]] + p$
        $C[k][2] \leftarrow M_t[M_j[1]]$
        $M[k][2] \leftarrow M_j[1]$
    end for

    return $M, C$
end function
```