Journal of Computer Science and Cybernetics, V.34, N.2 (2018), 97–111
DOI 10.15625/1813-9663/34/2/12667

MINIMIZING MAKESPAN OF PERSONAL SCHEDULING PROBLEM IN AVAILABLE TIME-WINDOWS WITH SPLIT-MIN AND SETUP-TIME CONSTRAINTS*

TRANG HONG SON[1,2], TRAN VAN LANG[3], NGUYEN HUYNH-TUONG[1]

[1]*Ho Chi Minh City University of Technology, Vietnam*
[2]*Hoa Sen University, Vietnam*
[3]*Institute of Applied Mechanics and Informatics, VAST*
[2]*son.tranghong@hoasen.edu.vn*

Abstract. This paper deals with personal scheduling problem in available time-windows with split-min and setup-time constraints. The jobs are splittable into sub-jobs and a common lower bound on the size of each sub-job is imposed. The objective function aims to find a feasible schedule that minimizes the total completion time of all jobs. The proposed scheduling problem was proved to be strongly NP-hard by a reduction to previous problem in the preliminary results. We propose in this paper an exact method based on MILP model to find optimal solution, some heuristics to find feasible solution and a meta-heuristic based on tabu search algorithm to find good solution. The computational results show the performance of proposed exact method, some heuristics and tabu search algorithm.

Keywords. Splitting-job, available time-window, setup-time, assignment approach, SPT/LPT rule, tabu search algorithm.

1. INTRODUCTION

In everyday life, there are many things that people need to carry out (called job), for instance, writing up the lesson plan, teaching, checking and correcting students' homework, writing tests, doing research and so on are jobs of a teacher. Each job has a processing-time and a preparation time (or setup-time in literature). The preparation time presents the period to recuperate the job status from the last execution while this job can be split in several sub-jobs. For example, when the teacher wants to evaluate the student works from examination, they cannot do marking for hundreds of students at once. In order to fairly mark a group of students for each evaluation time interval (sub-job), once the teacher wants to continue the marking, he should read the test details carefully, check the scoring in the answer keys and then match for each work of students. We all have available time-windows to do work and unavailable time-windows not to do work. For simplicity of modeling problem, unavailable time-windows are reduced to milestones (called break-time).

*This paper is selected from the reports presented at the 11th National Conference on Fundamental and Applied Information Technology Research (FAIR'11), Thang Long University, 09 - 10/08/2018.

The personal scheduling problem is arranging jobs depending on the available time-windows and basing on different criteria so as to maximize their productivity. The constraints in this problem are that the jobs can be splittable into some sub-jobs which can not be less than a threshold called $split_{\min}$ (note that if we divide the job into as many parts as we can, setup-time for broken sections), and jobs are not assigned into unavailable time-windows (i.e., jobs are not scheduled through break-times). The main goal of this problem is to find the solution so that all jobs are completed as soon as possible.

According to classical scheduling notations [5], this scheduling problem with $C_{\max}$ is the time when all jobs are completed, which is denoted as follows:

$$1|splitting - job, split_{\min}, setup - time, time - window|C_{\max}$$

The other notations used in the problem are:

- n: number of jobs
- m: number of windows
- J_i: the i^{th} job
- p_i: processing-time for job J_i
- s_i: setup-time for job J_i
- W_t: the t^{th} window
- w_t: size of window W_t
- b_t: the t^{th} break-time.

Consider a simple example to understand this problem, the input data is presented in Tables 1 and 2 with $n = 3$ jobs (J_1, J_2, J_3) with the corresponding processing-time of each job is 12, 7, 13 and the setup-time is 2, 1, 2; and $m = 4$ windows (W_1, W_2, W_3, W_4) with corresponding available time [0, 8], [8, 18], [18, 33], [33, $+\infty$); and 3 break-times at times $t = 8$, $t = 18$, $t = 33$ as Figure 1.

Table 1. Jobs

Jobs	**Processing time**	**Setup time**
J_1	12	2
J_2	7	1
J_3	13	2

Table 2. Windows

Windows	**Available time**	**Size**
W_1	[0,8]	8
W_2	[8,18]	10
W_3	[18,33]	15
W_4	[33, $+\infty$)	$+\infty$

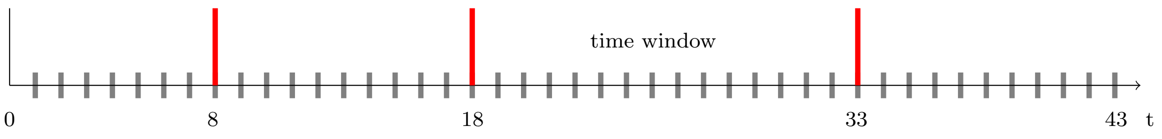

Figure 1. Demonstration of available time-windows

With $split_{\min} = 5$, a feasible solution has $C_{\max} = 43$ with an idle-time at $[16, 18]$ as Figure 2.

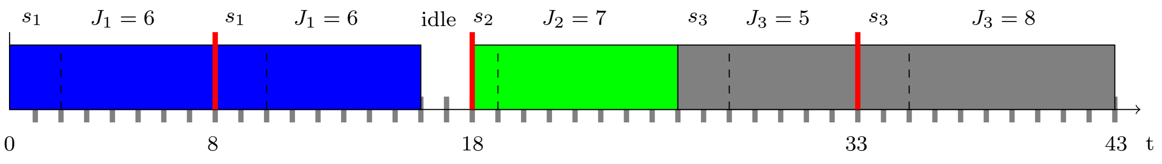

Figure 2. A feasible solution has $C_{\max} = 43$

And an optimal solution has $C_{\max} = 40$ with an idle-time at $[32, 33]$ as Figure 3.

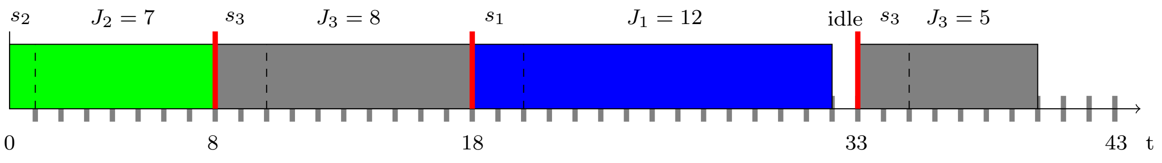

Figure 3. An optimal solution has $C_{\max} = 40$

The previous personal scheduling problem in [7] is a special case of this considered problem with setup-time = 0, in other words, this problem is a generalization of the previous problem. Since the previous problem was proved to be strongly NP-hard by a reduction from 3-PARTITION problem in [7], this problem is also strongly NP-hard. The properties of the optimal solution are also presented in [8], that is there exists an optimal solution such that in an available time window:

a) Each job does not have to be executed or has only one sub-job; and each job J_i is split up into S_i sub-jobs, with $S_i = \min\{\left\lfloor \frac{p_i}{split_{\min}} \right\rfloor; m\}$.

b) The order of sub-jobs is optional.

c) If exiting, the size of idle-time is smaller than $2 \times split_{\min}$ and should be at the end of the time-window.

Based on the above optimal properties, one must answer two questions to solve the problem:

1. Is job or sub-job assigned to an available time-window?
2. If so, what is the size of this job or sub-job?

The map of related scheduling problem is presented as in Figure 4.

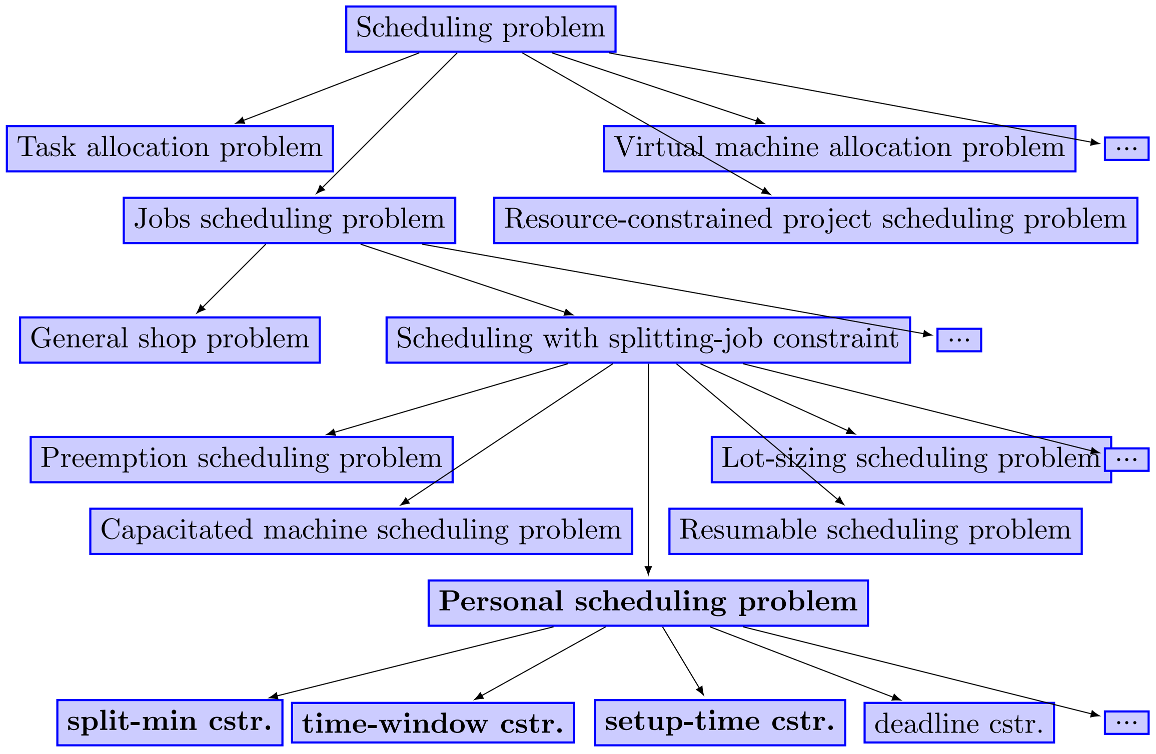
Scheduling problem
Task allocation problem
Virtual machine allocation problem
···
Jobs scheduling problem
Resource-constrained project scheduling problem
General shop problem
Scheduling with splitting-job constraint
···
Preemption scheduling problem
Lot-sizing scheduling problem
···
Capacitated machine scheduling problem
Resumable scheduling problem
Personal scheduling problem
split-min cstr.
time-window cstr.
setup-time cstr.
deadline cstr.
···

Figure 4. The map of related scheduling problem

The kind of scheduling problem is being very popular in many areas like:

- Jobs scheduling problem (JSP): consists of determining the best processing sequence of jobs to minimize total costs [1].
  - General shop problem (GSP): job shops, flow shops, open shops, and mixed shops are special cases of the general shop [2].
  - Scheduling with splitting-job constraint (SwSJC):
    - Preemption scheduling problem (PrSP): a job is interrupted solely by another job having higher priority if preemption is allowed [2].
    - Lot-sizing scheduling problem (LSP): focused in the context where several customer orders share production capacity and finite planning situation from production to inventory status depends on fluctuating demands [3].
    - Capacitated machine scheduling problem (CSP): scheduling with the availability on working machine has been limited [9].

  - ∗ Resumable scheduling problem (RSP): a special case of the capacitated scheduling problem, in which jobs do not need to be restarted in the case when it has been interrupted by the starting time of a non-availability period [4].
  - ∗ Personal scheduling problem (PSP): personal plan is always changeable, has to be flexible, may break a large task into many small ones, but each split part of the task cannot last less than a specified minimum duration [10].

- Resource-constrained project scheduling problem (RCPSP): consists of activities that must be scheduled subject to precedence and resource constraints such that the makespan is minimized [6].
- Task allocation problem (TAP): aims to minimize total execution cost and inter-node communication cost in traditional parallel computing systems [11].
- Virtual machine allocation problem (VMAP): considers allocating the VM instances for one unit of time [12].

This paper is organized as follows. The next section presents the proposed approaches for solving this scheduling problem. The results of the experiment are demonstrated in Section 3. And the final section is discussion and conclusion of the study.

2. PROPOSED APPROACHES

2.1. Exact method - MILP model

In this problem, some decision variables are:

- $x_{i,t} \in \{0,1\}$ is 1 if the sub-job of J_i is assigned to the window W_t, otherwise is 0.
- $y_{i,t}$ is integer, is processing-time of the sub-job of J_i in the the window W_t corresponding to $x_{i,t}$.

And intermediate variables are:

- $\alpha_t = \left\lceil \frac{\sum_{i=1}^{n} x_{i,t}}{n} \right\rceil$, $\alpha_t \in \{0,1\}$ is 0 if there does not exist sub-job of J_i is assigned to the window W_t, otherwise is 1.
- $\beta_t = \alpha_t \times b_{t-1}$ is integer, is the start of the window W_t.
- $\gamma_t = \beta_t + \sum_{i=1}^{n}(s_i \times x_{i,t} + y_{i,t})$ is integer, is the end of sub-job in the window W_t.
- $C_{\max} = \max_{t=1,\dots,m}(\gamma_t)$ is integer, is the maximum of the end of sub-job in all windows.

MILP model formulation is presented as:

Objective function

$$\min(C_{\max})$$

Subject to

$$\sum_{t=1}^{m} y_{i,t} = p_i;\ \forall i = 1, \dots, n, \tag{1}$$

$$\sum_{i=1}^{n} (s_i \times x_{i,t} + y_{i,t}) \leq w_t;\ \forall t = 1, \dots, m, \tag{2}$$

$$split_{\min} \times x_{i,t} \leq y_{i,t} \leq p_i \times x_{i,t};\ \forall i = 1, \dots, n, \forall t = 1, \dots, m, \tag{3}$$

$$\gamma_t \leq C_{\max};\ \forall t = 1, \dots, m. \tag{4}$$

These constraints are described below:

- Constraint (1): the total processing-time of all sub-jobs of a job is equal to the completion time of that job.
- Constraint (2): the total processing-time and setup-time of sub-jobs in a window should not exceed the size of that window.
- Constraint (3): if a sub-job is assigned to a window, the processing-time of that sub-job must be greater than $split_{\min}$; in addition, this constraint also ensures that if $x_{i,t} = 0$ then $y_{i,t} = 0$.
- Constraint (4): linearization of max function.

2.2. Assignment (ASS)

Some notations are used in this algorithm:

- rj_i is remaining time for job J_i, after any sub-job of J_i is assigned into the window.
- rw_t is remaining size of window W_t, after any sub-job or job is assigned into this window.

Main idea: traverse each window from LEFT to RIGHT, at each window, the job is considered one of three cases, see more details in Algorithm 1:

- If $(rj_i + s_i) \leq rw_t$, then assign job J_i with the size of rj_i into window W_t.
- If $(rj_i + s_i) \geq (rw_t + split_{\min})$, then assign job J_i with the size of $(rw_t - s_i)$ into window W_t, after that the remaining job is pushed back into list jobs.
- If $rj_i \geq (2 \times split_{\min})$, then assign job J_i with the size of $(rj_i - split_{\min})$ into window W_t, after that the remaining job is pushed back into list jobs.

With the above input data in Table 1, 2, the solution from ASS algorithm presents as Figure 5.

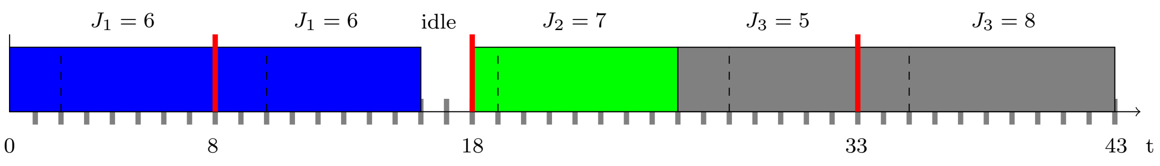

Figure 5. ASS algorithm has $C_{\max} = 43$

Algorithm 1: ASS, with $O(m \times n)$

```
    input: $Jobs$: list jobs;
           $Wins$: list windows
1   begin
2       foreach $window\ W_t \in Wins$ do
3           foreach $job\ J_i \in Jobs$ do
4               if $rj_i \geq split_{\min}$ and $rw_t \geq split_{\min}$ then
5                   if $(rj_i + s_i) \leq rw_t$ then
6                       Assign job $J_i$ with the size of $rj_i$ into window $W_t$;
7                   else
8                       if $(rj_i + s_i) - rw_t \geq split_{\min}$ then
9                           Assign job $J_i$ with the size of $(rw_t - s_i)$ into window $W_t$;
10                      else if $rj_i - split_{\min} \geq split_{\min}$ then
11                          Assign job $J_i$ with the size of $(rj_i - split_{\min})$ into window $W_t$;
12                  end
13              end
14          end
15      end
16  end
```

2.3. Shortest setup processing time (SPT) and longest setup processing time (LPT)

Main idea: improved from the Assignment algorithm, at each window, the jobs are sorted according to the total setup-time and processing-time by ascending (SPT) or descending (LPT), see more details in Algorithm 2. With the above input data in Tables 1 and 2, the solution from SPT algorithm presents as Figure 6.

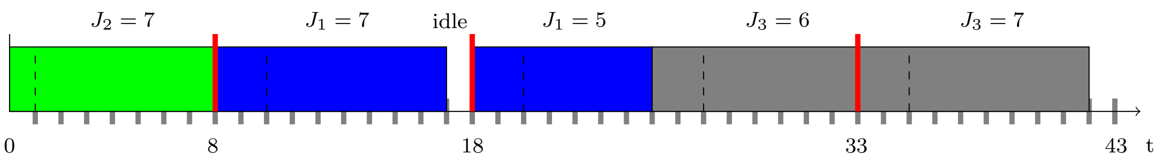

Figure 6. SPT algorithm has $C_{\max} = 42$

Algorithm 2: SPT/LPT, with $O(m \times n \times \log n)$

```
input: $Jobs$: list jobs;
       $Wins$: list windows
begin
    foreach window $W_t \in Wins$ do
        List jobs sorted in SPT/LPT order;
        foreach job $J_i \in Jobs$ do
            if $rj_i \geq split_{\min}$ and $rw_t \geq split_{\min}$ then
                if $(rj_i + s_i) \leq rw_t$ then
                    Assign job $J_i$ with the size of $rj_i$ into window $W_t$;
                else
                    if $(rj_i + s_i) - rw_t \geq split_{\min}$ then
                        Assign job $J_i$ with the size of $(rw_t - s_i)$ into window $W_t$;
                    else if $rj_i - split_{\min} \geq split_{\min}$ then
                        Assign job $J_i$ with the size of $(rj_i - split_{\min})$ into window $W_t$;
                end
            end
        end
    end
end
```

2.4. Tabu search on Assignment (TSASS)

Note that, in the Assignment algorithm shown in subsection 2.2, the order of jobs will affect the result $C_{\max}$, for example with input data in Tables 1 and 2 presents as Figure 8. Because combining ordering works better, the Tabu search algorithm is applied to this problem such as the combinatorial optimization problem. Tabu search uses a local or neighborhood search procedure to iteratively move from one potential solution x to an improved solution x' in the neighborhood of x, until some stopping criterion has been satisfied (generally, an attempt limit or a score threshold). Local search procedures often become stuck in poor-scoring areas or areas where scores plateau. In order to avoid these pitfalls and explore regions of the search space that would be left unexplored by other local search procedures, Tabu search carefully explores the neighborhood of each solution as the search progresses.

- Encoding solution related to each solution represents a list of job orders, for example at Figure 7, $S1 = J_2|J_3|J_5|J_4|J_1$ and $S2 = J_3|J_2|J_4|J_1|J_5$.

Solution 1	2	3	5	4	1

Solution 2	3	2	4	1	5

Figure 7. Encoding presentation

- Creating neighbor solutions from initial solution $S = J_1|J_i|J_2|J_j|J_3$ by using the following operators:

- SWAP: a neighbor of S is created by interchanging the jobs in position i and j, leading to sequence $S' = J_1|J_j|J_2|J_i|J_3$.
- EBSR (extraction and backward shifted re-insertion): a neighbor of S is created by extracting J_j and re-inserting it backward just before J_i, leading to sequence $S' = J_1|J_j|J_i|J_2|J_3$.
- EFSR (extraction and forward shifted re-insertion): a neighbor of S is created by extracting J_i and re-inserting it forward immediately after J_j, leading to sequence $S' = J_1|J_2|J_j|J_i|J_3$.

• Evaluate the fitness of neighbor solutions by using the Assignment algorithm at subsection 2.2 to find the $C_{\max}$.

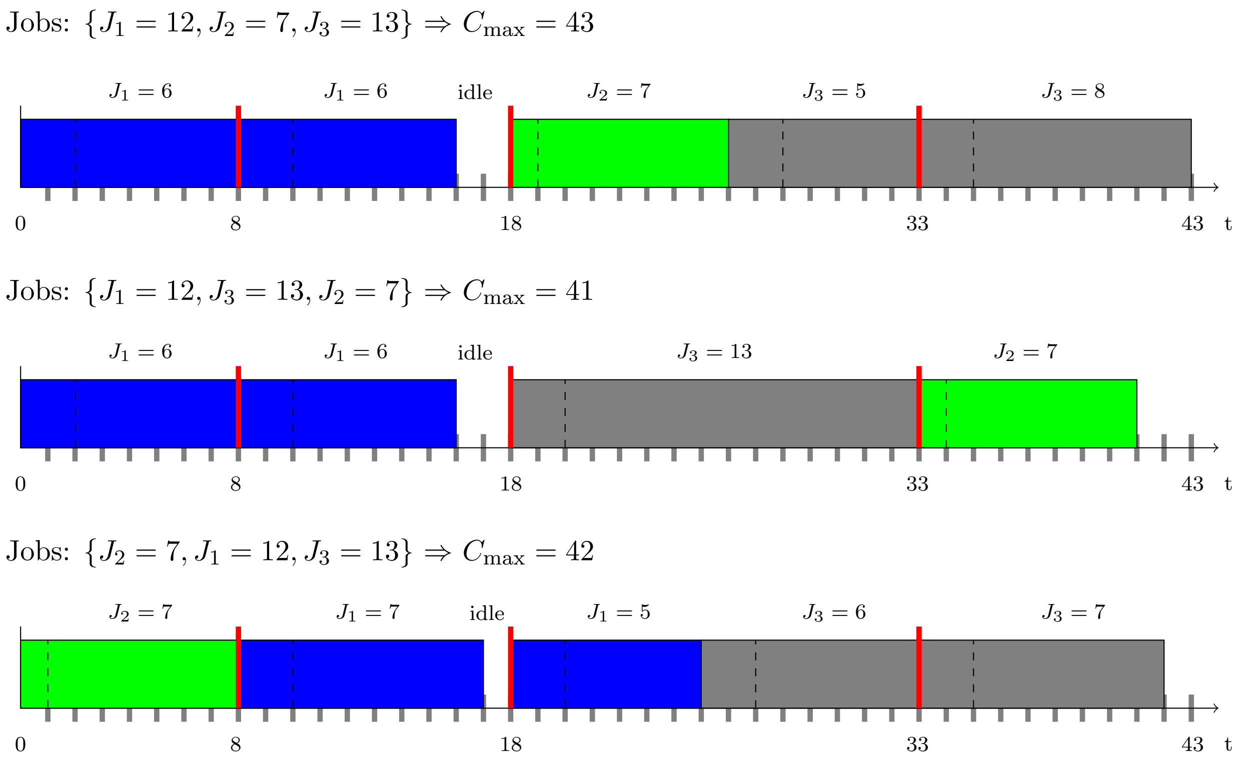

Figure 8. The value $C_{\max}$ is different when ordering different jobs

3. EXPERIMENTAL RESULTS

3.1. Dataset

There are two datasets, DS1 and DS2, which are used to evaluate the methods. In particular, DS1 contains small sample sizes of 10 to 20 jobs, which are used to compare the exact method with the heuristic methods; and DS2 contains large sample sizes of 100 to 200 jobs (scale up to 10 times), which are used to compare between heuristic methods. In each dataset, 18 tuples $(n, m, split_{\min})$ are considered, and for each such tuple, there are 10 sample instances generated according to the following rules:

- $split_{\min} = \{5, 6, 7\}$,
- p_i is randomly generated by integer uniform distribution in $[split_{\min}, 30]$,
- w_t is randomly generated by integer uniform distribution in $[1.5 \times split_{\min}, 1.5 \times 30]$,
- $s_i = p_i \div 10$.

In the education context, the unit time is often 45 minutes, so it can be flexibly scaled from 30 minutes to 1 hour. If the time unit is equivalent to 0.5 hours, jobs will be processed continuously from 2.5 hours to 3.5 hours, and any job is split too low below this threshold then the effect of the job will not be high. The size of the window will be 1.5 times the processing-time of the job. And the setup-time of a job is about 1/10 the processing-time of that job.

3.2. Evaluate of the exact method

All experimental results for the evaluation were conducted on a computer configured with Intel(R) Core(TM) i7-4650U 1.70GHz, 8GB memory with Windows 8.1 professional OS. The MILP model was implemented on the CPLEX 12.7.1 solver, which was tested on the dataset DS1 and evaluated on two criteria:

- Percentage gap (%) between the optimal solution Z^* and the lower bound $LB = \sum_{i=1}^{n}(s_i + p_i)$, as follows
$$\%LB = \frac{(Z^* - LB)}{LB} \times 100.$$
- Runtime (t) finds the optimal solution.

Table 3 shows that the average gap (%) between the optimal Z^* and the lower bound LB is very small (about 0.98%), which proves that the proposed lower bound is pretty good. And the Figure 9 also presents that when the number of jobs (n) and the number of window frames (m) are large, the runtime (t) increases rapidly. Within the time limit for finding an acceptable solution (less than 10 minutes), $n = 20, m = 15$ are the maximum values that CPLEX can find the optimal solution.

Table 3. CPLEX for dataset DS1

ID	n	m	$split_{\text{min}}$	%LB	t
1	10	5	5	0.85	0.4
2	10	5	6	0.92	0.36
3	10	5	7	0.99	0.38
4	10	7	5	1.35	1.2
5	10	7	6	1.69	1.07
6	10	7	7	1.77	1.34
7	15	7	5	0.58	3.97
8	15	7	6	0.48	4.05
9	15	7	7	0.67	3.64
10	15	10	5	0.97	20.72
11	15	10	6	1.41	22.74
12	15	10	7	1.46	20.75
13	20	10	5	0.5	35.31
14	20	10	6	0.32	32.26
15	20	10	7	0.45	36.49
16	20	15	5	0.83	250.68
17	20	15	6	0.98	248.7
18	20	15	7	1.49	198.25
Average				0.98	-

Notes: each case is the average results of 10 different instances

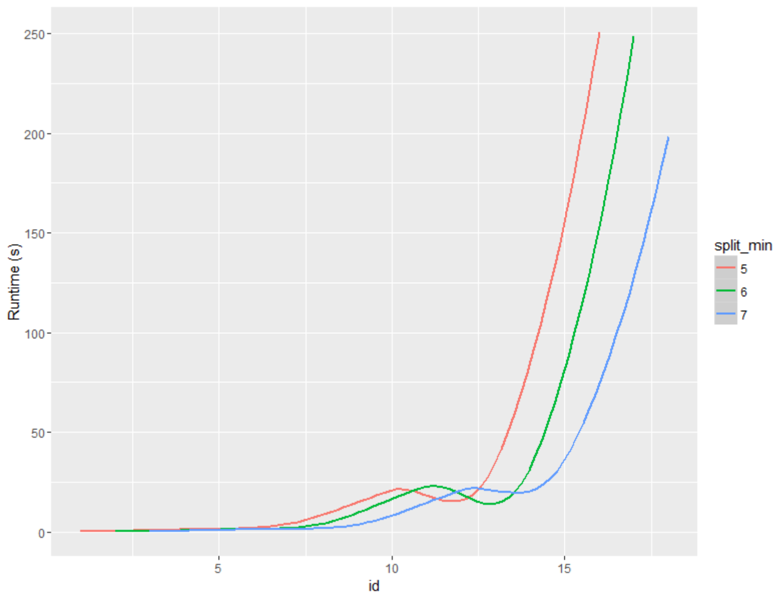

Figure 9. Runtime on CPLEX for dataset DS1

3.3. Evaluate of the heuristic methods

These heuristic methods such as ASS, SPT/LPT and TSASS were conducted experimentally on a small dataset DS1 with three criteria:

- Percentage gap (%) between the found solution $Z^{\#}$ and the lower bound LB, as follows

$$\%LB = \frac{(Z^{\#} - LB)}{LB} \times 100.$$

- Percentage gap (%) between the found solution $Z^{\#}$ and the optimal solution Z^* (found by solver CPLEX in the evaluate subsection 3.2), as follows

$$\%OPT = \frac{(Z^{\#} - Z^*)}{Z^*} \times 100.$$

- Runtime (t) in seconds.

Experiment results in Table 4 and Figure 10 showed that the ASS algorithm yielded the highest (7.07% compared to the lower bound and 6.03% versus optimal) because ASS is a very simple heuristic algorithm. The SPT/LPT algorithms result in a (%) gap less than the ASS algorithm because these algorithms apply the SPT/LPT rule before proceeding to assign to windows, however the improvement is not much (only about 1%). The TSASS algorithm yields the smallest (%) gap because the strategy of the Tabu search algorithm is trying to find the best solution in the set of neighbor solutions to the current solution, but TSASS has to undergo many iterations, leading to higher runtime (t) costs than other heuristics.

Table 4. Summary of experimental results for dataset DS1

ID	n	m	$split_{\min}$	ASS			LPT			SPT			TSASS		
				%LB	%OPT	t	%LB	%OPT	t	%LB	%OPT	t	%LB	%OPT	t
1	10	5	5	4.55	3.67	0	4.98	4.1	0	4.26	3.38	0	1.89	1.04	0.01
2	10	5	6	5.47	4.52	0	5.55	4.6	0	5.15	4.2	0	1.52	0.59	0.01
3	10	5	7	6.55	5.51	0	6.53	5.49	0	6.14	5.1	0	1.39	0.4	0.01
4	10	7	5	6.42	5.01	0	6.55	5.14	0	7.36	5.93	0	3.13	1.76	0.02
5	10	7	6	8.58	6.78	0	7.24	5.47	0	7.04	5.27	0	3.28	1.57	0.01
6	10	7	7	9.05	7.16	0	8.58	6.67	0	7.78	5.91	0	2.99	1.2	0.01
7	15	7	5	4.79	4.19	0	4.82	4.22	0	4.18	3.58	0	1.53	0.94	0.04
8	15	7	6	5.29	4.79	0	5.29	4.78	0	5.48	4.98	0	1.82	1.34	0.04
9	15	7	7	6.09	5.39	0	5	4.31	0	5.68	4.98	0	1.65	0.97	0.04
10	15	10	5	6.73	5.71	0	7.07	6.04	0	6.87	5.84	0	3	2.01	0.05
11	15	10	6	7.76	6.25	0	8.8	7.29	0	8.76	7.25	0	3.57	2.13	0.05
12	15	10	7	9.44	7.87	0	8.88	7.31	0	9.07	7.5	0	3.92	2.43	0.05
13	20	10	5	5.52	5	0	5.47	4.95	0	5.3	4.78	0	1.91	1.41	0.11
14	20	10	6	5.42	5.08	0	5.78	5.44	0	5.39	5.05	0	1.94	1.61	0.11
15	20	10	7	7	6.52	0	6.37	5.9	0	7.05	6.58	0	2.34	1.88	0.11
16	20	15	5	8.5	7.61	0	7.94	7.05	0	8.12	7.23	0	4.56	3.69	0.15
17	20	15	6	9.62	8.56	0	8.67	7.61	0	8.94	7.87	0	4.19	3.18	0.14
18	20	15	7	10.49	8.87	0	9.42	7.82	0	9.97	8.36	0	5.1	3.56	0.15
Average				7.07	6.03	-	6.83	5.79	-	6.81	5.77	-	2.76	1.76	-

Notes: each case is the average results of 10 different instances

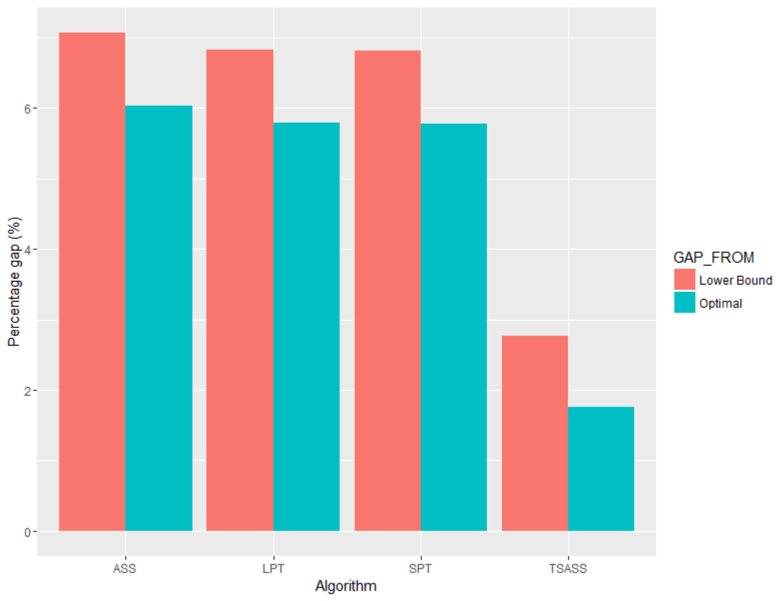

Figure 10. The chart compares the methods on the dataset DS1

Experimenting on the large dataset DS2, due to the lack of optimal solution results, only two criteria (%LB) were the results found $Z^{\#}$ with lower bound LB and runtime (t). Experiment results on the large dataset DS2 are similar to the small data set DS1 (see details at Table 5 and Figure 11), in which the ASS algorithm yields the highest gap, followed by the SPT/LPT algorithms, and the lowest is the TSASS algorithm. However, the runtime (t) cost of the TSASS algorithm is high, especially for the case of $n = 200, m = 150$, the computation time is very high (about 8 minutes) but still acceptable for large data.

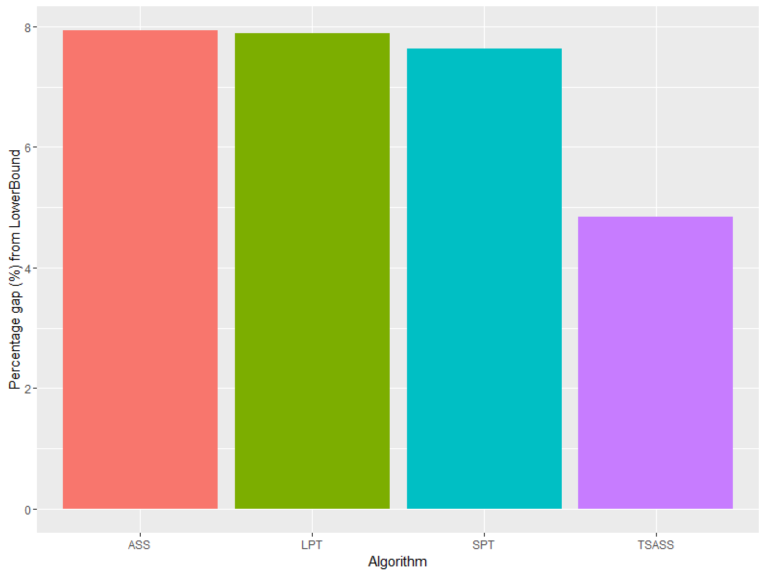

Figure 11. The chart compares the methods on the dataset DS2

Table 5. Summary of experimental results for dataset DS2

ID	n	m	$split_{\min}$	ASS		LPT		SPT		TSASS	
				%LB	t	%LB	t	%LB	t	%LB	t
1	100	50	5	5.78	0	6.12	0	5.53	0	3.36	19.14
2	100	50	6	6.61	0	6.47	0	6.41	0	3.27	18.83
3	100	50	7	7.06	0	7.49	0	6.99	0	3.47	18.95
4	100	70	5	8.3	0	8.29	0	7.51	0	5.3	26.16
5	100	70	6	9.76	0	8.86	0	9.04	0	5.3	26.5
6	100	70	7	10.01	0	10.23	0	10.25	0	5.75	26.61
7	150	70	5	5.39	0	5.66	0	5.15	0	3.21	83.91
8	150	70	6	6.22	0	6.45	0	5.81	0	3.51	81.5
9	150	70	7	6.75	0	7.37	0	6.67	0	3.7	81.38
10	150	100	5	8.08	0	8.32	0	7.69	0	5.34	117.84
11	150	100	6	8.93	0	8.66	0	8.62	0	5.62	148.66
12	150	100	7	10.21	0	9.66	0	9.68	0	5.87	182.11
13	200	100	5	6.12	0	6.25	0	5.53	0	4.03	346.84
14	200	100	6	6.28	0	6.62	0	6.28	0	4.02	324.6
15	200	100	7	7.45	0	7.68	0	7.48	0	4.48	321.95
16	200	150	5	8.89	0	8.53	0	8.51	0	6.48	486.97
17	200	150	6	10.16	0	9.09	0	9.84	0	6.88	487.95
18	200	150	7	10.92	0	10.12	0	10.6	0	7.67	477.01
Average				7.94	-	7.88	-	7.64	-	4.85	-

Notes: each case is the average results of 10 different instances

4. CONCLUSION

In this paper, the personal scheduling problem in available time-windows with $split_{\min}$ and setup-time constraints has been proposed. The MILP model was introduced and implemented using the CPLEX solver to find the optimal solution for the problem. In addition, some heuristic algorithms such as ASS, SPT/LPT as well as the TSASS based on Tabu search algorithm have been proposed to solve this problem. Experiments to evaluate the methods have also been carried out and the results showed that the TSASS algorithm achieves a good compromise between solution quality and acceptable execution time in both small and large dataset.

Adding more constraints to this personal scheduling problem is an issue that needs to be considered in the future, such as the deadline constraint for each job or the parallel machine constraints.

REFERENCES

[1] M. Almeida and M. Centeno, "A composite heuristic for the single machine early/tardy job scheduling problem," *Computers & Operations Research*, vol. 25, no. 7-8, pp. 625–635, 1998.

[2] P. Brucker, *Scheduling Algorithms - Fifth Edition.* Springer, 2007.

[3] B. Fleischmann and H. Meyr, "The general lotsizing and scheduling problem," *OR Spektrum*, vol. 19, no. 1, pp. 11–21, 1997.

[4] S. Gawiejnowicz and A. Kononov, "Complexity and approximability of scheduling resumable proportionally deteriorating jobs," *European Journal of Operational Research*, vol. 200, pp. 305–308, 2010.

[5] R. Graham, E. Lawler, J. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.

[6] S. Hartmann and D. Briskorn, "A survey of variants and extensions of the resource-constrained project scheduling problem," *European Journal of Operational Research*, vol. 207, pp. 1–14, 2010.

[7] V. Nguyen, N. H. Tuong, H. Nguyen, and T. Nguyen, "Single-machine scheduling with splitable jobs and availability constraints," *REV Journal on Electronics and Communications*, vol. 3, no. 1-2, pp. 21–27, 2013.

[8] V. Nguyen, N. H. Tuong, V. Tran, and N. Thoai, "An milp-based makespan minimization model for single-machine scheduling problem with splitable jobs and availability constraints," in *International Conference on Computing, Management and Telecommunications (ComManTel)*, Ho Chi Minh, Vietnam, 2013, pp. 397–400.

[9] S. Raut, S. Swami, and J. Gupta, "Scheduling a capacitated single machine with time deteriorating job values," *International Journal of Production Economics*, vol. 114, pp. 769–780, 2008.

[10] D. Tran, N. H. Tuong, G. H. Ngoc, T. Mai, T. Tran, Q. Mai, and T. Quan, "A personal scheduling system using genetic algorithm and simple natural language processing for usability," in *Multidisciplinary International Workshop on Artificial Intelligence (MIWAI'2010)*, Mahasarakham, Thailand, 2010.

[11] J. Yang, "Complexity analysis of new task allocation problem using network flow method on multicore clusters," *Mathematical Problems in Engineering, Hindawi Publishing Corporation*, vol. 2014, pp. 1–7, 2014.

[12] S. Zaman and D. Grosu, "Combinatorial auction-based allocation of virtual machine instances in clouds," *Journal of Parallel and Distributed Computing*, vol. 73, no. 4, pp. 495–508, 2013.

Received on September 09, 2018
Revised on September 18, 2018