Meta-heuristic Algorithm for Bounded-Splitting Jobs Scheduling Problem on a Single Machine

Hong-Son TRANG[1,2], Van-Huy NGUYEN[1,3], Nguyen HUYNH-TUONG[1], Van-Lang TRAN[4] and Ameur SOUKHAL[5]
[1] Faculty of Computer Science & Engineering, Ho Chi Minh city University of Technology, Vietnam
[2] Hoa Sen University, Vietnam
[3] Ho Chi Minh city University of Transport, Vietnam
[4] Institute of Applied Mechanics and Informatics, Vietnam Academy of Science and Technology
[5] Laboratoire d'Informatique, Université de François-Rabelais, France
Email: sonth@cse.hcmut.edu.vn

***Abstract*—This paper deals with scheduling problem with availability constraints on a single machine. The jobs are splitable 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 maximum completion time of jobs. The proposed scheduling problem was proved to be strongly NP-hard by a reduction from 3-SAT problem. We propose in this paper some effective heuristic algorithms are BMF - Based on Max Flow resolution and MAAS - MAtching and ASsignment approach. After that we apply Genetic Algorithm on BMF and MAAS and the last we use the .Net Parallel Task Library to evaluate the population. The computational results show the performance of the proposed heuristics in comparing with the exact method proposed in the literature.**

I. Introduction

People nowadays are in charge to perform several tasks in an instant. It is necessary to use a scheduling method for an individual in dealing with numerous complicated tasks. According to experts in productivity, we always need to perform at any moment about 50 to 150 small tasks [1].

In this paper, we consider a problem raised in real-life context: "*in order to perform individual tasks in the best possible way, it aims to schedule these tasks executed in some irregular available time-windows. The time-windows are determined by plenty of fixed appointments (meeting, lunch break, school hours, ...). Note that these jobs could be divided into several splits (also called sub-jobs) but each split duration must be larger than a given value. That value is normally greater than 20 or 30 minutes, while the processing time of practical tasks usually ranges from one to five hours*" [2].

This scheduling is being very popular nowadays as:

- LSP - Lot-sizing & Scheduling Problem [3]: addresses the problem of integrating lotsizing and scheduling of several products on a single, capacitated machine.
- VMAP - Virtual Machine Allocation Problem [4]: considers allocating the VM instances for one unit of time.
- JSP - Jobs Scheduling Problem [5]: consists of determining the best processing sequence for a set of jobs in order to minimize total costs.
- PSP - Personal Scheduling Problem [2]: personal plan is always changeable, have 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.

This paper focuses on the problem of scheduling splitable jobs on a single machine with available time windows. Availability on machine does not only mean the preventive maintenance [6], which have paid attention from several researchers (see surveys [7], [8], [9]) but also present the individually free time in personal scheduling [2]. According to [10], splitting job property means a job can be split into a number of independently processed sub-jobs. Since the size of such a sub-job may not be small, a supplementary min-split constraint is needed in this study.

In order to improve the quality of solution, the authors propose some effective heuristic algorithms are BMF - Based on Max Flow resolution and MAAS - MAtching and ASsignment approach. The experimental results illustrate the performance of the proposed heuristics in comparing with the exact method proposed in the literature.

This paper is organized as follows. The Jobs Scheduling problem is defined, demonstrated by a numerical example and notations will be used in Section II. Section III presents the preliminary results proposed in the literature, which include optimal structure properties, mathematical model formulated the considered problem. The best known LPT and the proposed heuristic algorithms, also meta-heuristic algorithms is presented in Section IV and Section V. Section VI gives computational results and Section VII concludes the paper.

II. Problem Definition

Let $C_{\max}$ be the maximum completion time for all jobs (also called the *"makespan"*) and the objective function aims to minimize the *makespan*. According to classical scheduling notations [11], this Jobs Scheduling problem is defined as:

$$1|splitable,\ split_{min}, p_i \geq split_{min},\ available - windows|C_{\max}.$$

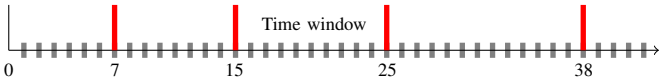
In order to well define the considering scheduling problem, let's present a numerical example which demonstrates the essential constraint with following input data and $split_{min} = 3$.

TABLE I
JOBS

Jobs	Processing-time
J_1	6
J_2	8
J_3	4
J_4	9
J_5	11

TABLE II
WINDOWS

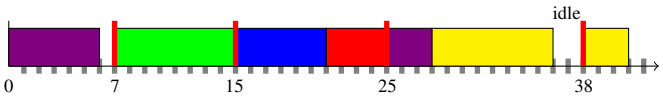
Windows	Available-time	Capacity
W_1	[0,7]	7
W_2	[7,15]	8
W_3	[15,25]	10
W_4	[25,38]	13
W_5	[38, $+\infty$)	$+\infty$



After calculating, the feasible schedule is maybe:

TABLE III
AN FEASIBLE SCHEDULE WITH THE $C_{max} = 41$

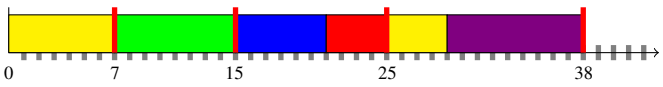
Job	Window	Processing-time	Time-window
J_1	W_3	6	[15-21]
J_2	W_2	8	[7-15]
J_3	W_3	4	[21-25]
J_4	W_1	6	[0-6]
J_4	W_4	3	[25-28]
J_5	W_4	8	[28-36]
J_5	W_5	3	[38-41]



And the optimal schedule is maybe:

TABLE IV
AN OPTIMAL SCHEDULE WITH THE $C_{max} = 38$

Job	Window	Processing-time	Time-window
J_1	W_3	6	[15-21]
J_2	W_2	8	[7-15]
J_3	W_3	4	[21-25]
J_4	W_4	9	[29-38]
J_5	W_1	7	[0-7]
J_5	W_4	4	[25-29]



The following notations will be used in this problem:

- n: Number of jobs
- m: Number of windows
- J_i: The i^{th} job
- p_i: Processing time for job i
- rj_i: Remaining time for job i
- b_t: The t^{th} time break
- W_t: The t^{th} window
- w_t: Size of the t^{th} window
- rw_t: Remaining time of the t^{th} window

III. PRELIMINARY RESULTS

A. Complexity of the problem

This scheduling problem was proved to be NP-Hard in [12] by a reduction from 3-SAT problem (which is known to be strongly NP-Hard [13]).

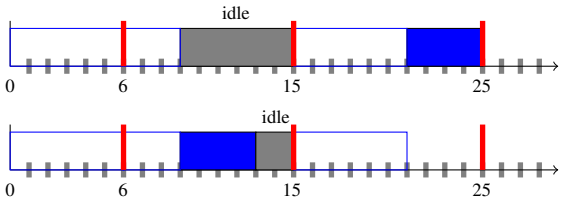
B. Optimal Properties

Some structural properties of a particular optimal solution were proposed in [14].

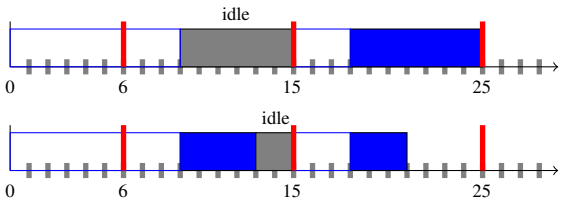
Proposition *1:* There exits an optimal schedule such that the jobs are scheduled without any *idle-time* with size $\geq 2 \times split_{min}$.

For example, we consider two cases:

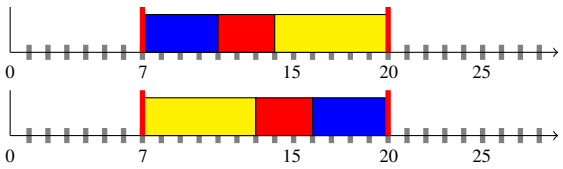
- Last sub-job $\leq$ idle-time



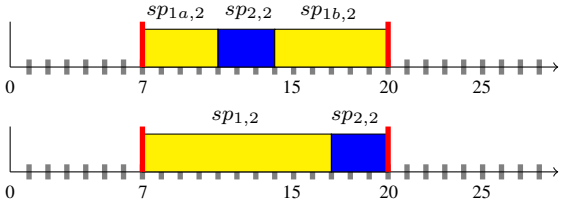
- Last sub-job $\geq$ idle-time



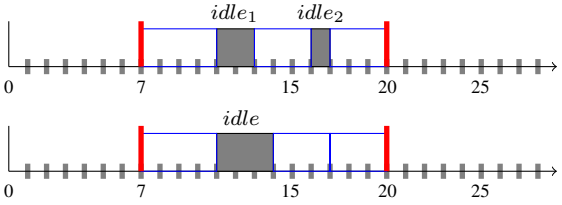
Proposition *2:* There exists an optimal schedule such that the sub-jobs in an available window are scheduled in arbitrary order.



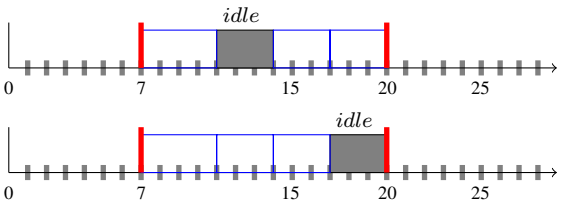
Proposition *3:* There exists an optimal schedule such that each job has only zero or one sub-job in an available time window.



Proposition *4:* There exists an optimal schedule such that there is at most one idle time in an available time window.



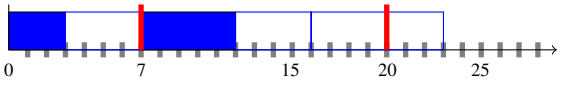
Proposition *5:* There exists an optimal schedule such that if there exists an idle time in an available window, it should be at the end of the time window.



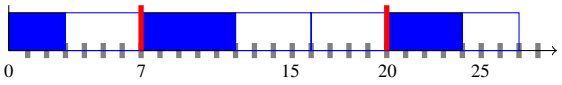
Proposition *6:* There exists an optimal schedule such that a job J_i is split into at most S_i, with $S_i = \min\{\left\lfloor \frac{p_i}{split_{min}} \right\rfloor ; m\}$.

For example:

- $p_i = 8, split_{min} = 3, m = 3 \Rightarrow S_i = 2$



- $p_i = 12, split_{min} = 3, m = 3 \Rightarrow S_i = 3$



Based on these structural properties, a solution of the scheduling problem is determined by the following decision sets:

1) job assignment in each available time-window (which defines a sub-job executed in this time-window),
2) the size of each sub-job.

C. MILP

In what follows, a MILP model is constructed based on the above structurally optimal properties [14].

According to the above decision sets, decision variables are defined as follows:

- $x_{i,t} = \begin{cases} 1 & \text{if there exits a sub-job } J_i \text{ executed in} \\ & \text{the window } W_t \\ 0 & \text{otherwise} \end{cases}$
- $y_{i,t}$: processing time of a sub-job J_i in the window W_t corresponding to $x_{i,t}$

All constraints of the problem are described in linear form of decision variables as follows:

- Total size of splits of a job should be equal to processing time of this job:

$$\sum_{t=1}^{m} y_{i,t} = p_i, \forall i = 1, \ldots, n \quad (1)$$

- Total size of splits executed in a time window could not be exceeded the size of this time-window:

$$\sum_{i=1}^{n} y_{i,t} \leq w_t, \forall t = 1, \ldots, m \quad (2)$$

- Processing time of a split should be greater than $split_{min}$ if exists:

$$y_{i,t} \geq x_{i,t} \times split_{min}, \forall i = 1, \ldots, n, \forall t = 1, \ldots, m \quad (3)$$

- Processing time of a sub-job could not be greater than p_i if exists or equals zero if not exists:

$$y_{i,t} \leq x_{i,t} \times p_i, \forall i = 1, \ldots, n, \forall t = 1, \ldots, m \quad (4)$$

Objective function of the problem is presented by:

$$\min \max_{t=1,\ldots,m} \left(\left\lceil \frac{\sum_{i=1}^{n} x_{i,t}}{n} \right\rceil \times b_{t-1} + \sum_{i=1}^{n} y_{i,t} \right) \quad (5)$$

The objective function (5) represents the makespan minimization since the first term determines the starting time of the last executed time-window whilst the second term represents the total processing time of all sub-jobs scheduled on the last executed time-window. Combination of constraints (3) and (4) imposes the relation between $x_{i,t}$ and $y_{i,t}$: if $x_{i,t} = 0$ then $y_{i,t}$ should be zero, and reciprocally remains valid ($y_{i,t} = 0 \Rightarrow x_{i,t} = 0$). Consequently, if $x_{i,t} \neq 0$ (i.e. $x_{i,t} = 1$), $y_{i,t}$ should be greater than or equal to $split_{min}$, and no greater than p_i. Totally the proposed MILP model has $(2 \times m \times n)$ decision variables, and $(n + m + 3 \times n \times m)$ constraints.

It can be optimally solved by solvers for small size problem. Therefore, a heuristic has been developed for large and more complicated problems. The proposed heuristics will be presented in next section.

IV. Heuristic Algorithms

A. BMF - Based on Max Flow resolution

Remarks that the scheduling decision depends on two subsets: job assignment in each available time-window, and the size of each sub-job. Without min-split constraints, the problem is polynomial solvable by reduction to Max Flow problem. When considering min-split constraint, method reduced to Max Flow resolution will be achieved good solution if the order of jobs considering is adequate when choosing each flow. The jobs are ordered according to shortest processing time rule (SPT). Then, the Jobs Scheduling problem can be transformed to the Max Flow problem with two additional constraints:

- for each an arc (i, j) with flow transferred greater than $split_{min}$

- for each J_i an arc (s, J_i) with remaining capacity equal to 0 or no less than $split_{min}$.

The network N includes the following vertices:

- a source s and a sink d
- job vertices J_i
- window vertices W_t

The arcs in N are:

- for each J_i an arc (s, J_i) with capacity p_i (processing time for job i).
- for each $i = 1, \ldots, n$ and $t = 1, \ldots, m$ an arc (J_i, W_t) with capacity p_i.
- for each W_t with $t = 1, \ldots, m$ an arc (W_t, d) with capacity w_t (size of the t^{th} window).

The example of section II is presented to the network N at figure 1.

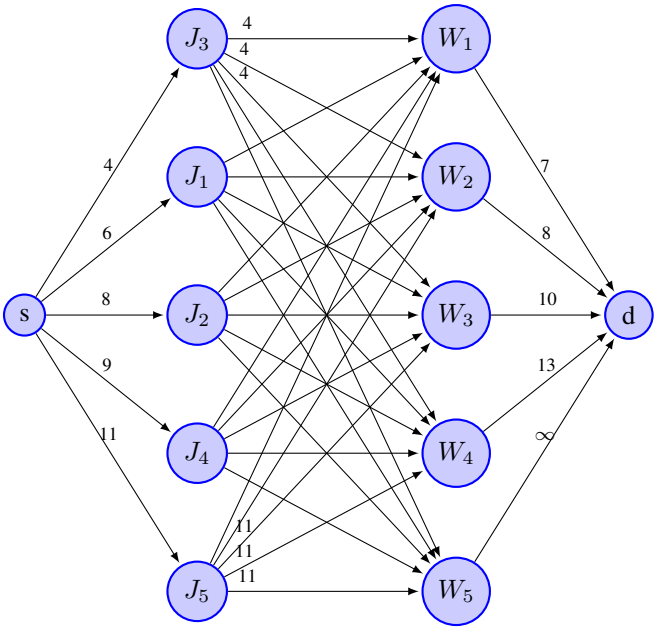


Fig. 1. Modeling by the network flow N

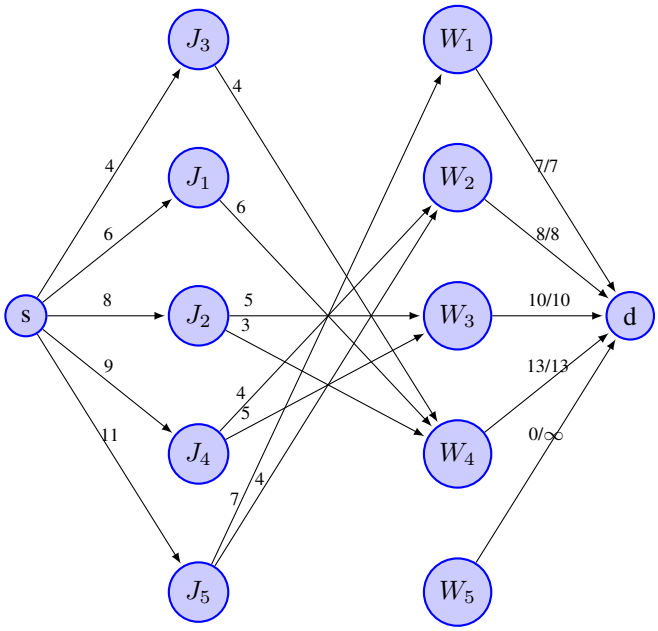


Fig. 2. Max Flow on the network N, with $C_{max} = 38$

The makespan is calculated by the sum of capacity from W_1 to W_{k-1} and the flow transferred through W_k which is the last executed time-window.

The Max Flow problem can be solved by Ford-Fulkerson algorithm with applying BFS strategy to find the available path. This algorithm's complexity is $O((n+m+n\times m)\times f)$ [15].

B. MAAS - MAtching and ASsignment approach

The idea of this heuristic will be divided into two main steps are matching and assignment.

First, all jobs are put into a list $Jobs$ and all windows are put into a list $Wins$. Second, in the matching step, all jobs are traversed to find some jobs have the sum of processing time is equal to the size of window, if found then some jobs are assigned to this window. The end, in assignment step, the other jobs will be assigned respectively based on the two rules:

- If the processing time of job is smaller than the remaining time of window (rw), then the job is allocated to be executed in this window.
- Otherwise, if the processing time is greater than $rw + split_{min}$ (the remaining window rw is also greater than $split_{min}$), then this job is divided into two parts; first part is allocated in this window that the size of part is equal rw, the second part is put back into the list $Jobs$.

The example of section II is considered in two steps are matching step:

Algorithm 1: Matching, with $O(n^2 \times m)$

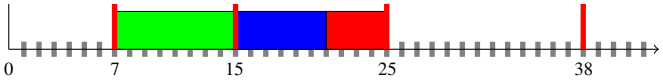
input: $Jobs$: list jobs
$Wins$: list windows

```
begin
    for i ← 1 to Jobs.Count do
        foreach window W_t ∈ Wins do
            if rw_t == rj_i then
                Assign J_i into window W_t;
                continue for-loop i;
            end
        end
        for j ← i + 1 to Jobs.Count do
            sumIJ ← rj_i + rj_j;
            sumI2J ← rj_i + rj_{i+1} + rj_{i+2} + ... + rj_j;
            foreach window W_t ∈ Wins do
                if rw_t == sumIJ then
                    Assign J_i into window W_t;
                    Assign J_j into window W_t;
                    continue for-loop i;
                else if rw_t == sumI2J then
                    Assign J_i, J_{i+1}, J_{i+2}, ..., J_j into
                     window W_t;
                    continue for-loop i;
                end
            end
        end
    end
end
```

Algorithm 2: Assignment, with $O(n \times m)$

input: $Jobs$: list jobs
$Wins$: list windows

```
begin
    foreach job J_i ∈ Jobs do
        foreach window W_t ∈ Wins do
            if rj_i ≥ split_min and rw_t ≥ split_min then
                if rj_i ≤ rw_t then
                    Assign job J_i with the size of rj_i into
                     window W_t;
                else
                    if rj_i − rw_t ≥ split_min then
                        Assign job J_i with the size of rw_t
                         into window W_t;
                    else if rj_i − split_min ≥ split_min then
                        Assign job J_i with the size of
                         rj_i − split_min into window W_t;
                    end
                end
            end
        end
    end
end
```



And assignment step:

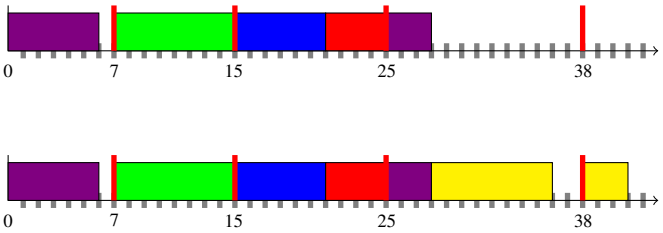


Fig. 3. MAAS with the $C_{max} = 41$

V. Meta-heuristic Algorithms

Genetic Algorithm (GA) is a meta-heuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic Algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

Genetic Algorithm Flow:

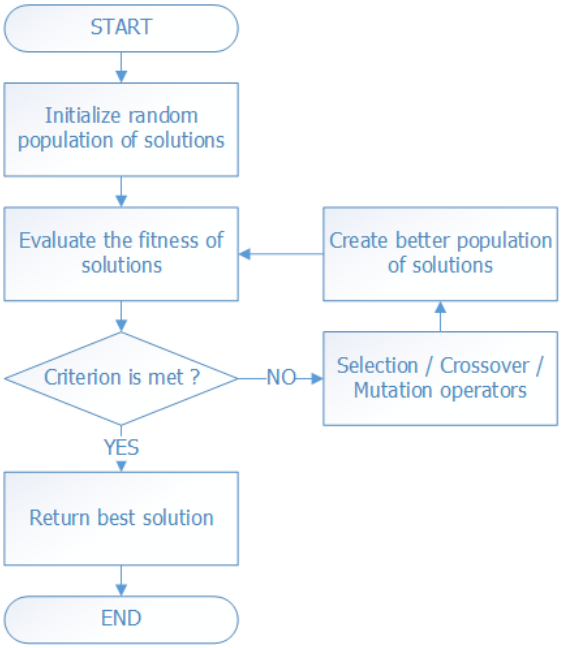


Fig. 4. GA Flow

Genetic Algorithm Operators:

- Selection (elite) operator:
  - allows a percentage of the best solutions within the population to pass through to the next generation without being modified. For example, a percentage of 10% with a population of 100, will be that the top 10 individuals form part of the next generation. This means that any future generation is at least as good as the current generation.
  - helps protect good individuals, however, it can be shown that as the percentage of elites is increased,

the number of duplicated solutions within the next generation increases, thereby reducing the performance of the GA.

- Crossover operator: is simply a value between 0 and 1 that represents the probability of parents crossing over to produce children. For example, a value of 1 means that all selected parents will crossover and create children, a value of 0.85 means that 85% will be crossed over and the remainder of parents will be pass through untouched to the new generation.

CHROMOSOME 1	T1	T2	T3	T4	T5	T6
CHROMOSOME 2	T1	T3	T2	T5	T4	T6
CHILD	T1	T2	T3	T5	T4	T6

Fig. 5. Single crossover

- Mutation operator: is to add diversity to the population, and to avoid local minimal. This probability should be set low, if it is set too high, the search will turn into a primitive random search.

CHROMOSOME 1	T1	T2	T3	T4	T5	T6
CHROMOSOME 2	T4	T2	T3	T1	T5	T6

Fig. 6. Swap mutation

A. GA.BMF - Genetic Algorithm on BMF

In the Max Flow problem with two additional constraints at IV-A, we realized that the order of jobs affects flow transferred on the network.

The example of section II, the order of jobs is:

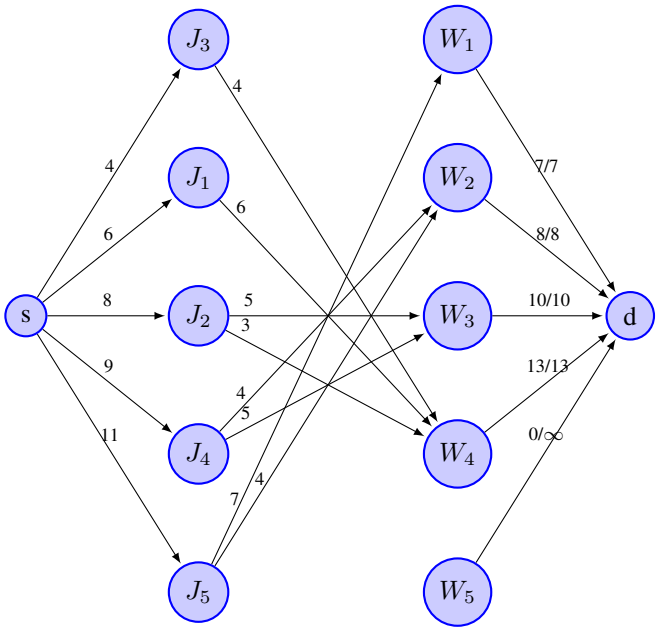


Fig. 7. $J_3 = 4, J_1 = 6, J_2 = 8, J_4 = 9, J_5 = 11 \Rightarrow C_{max} = 38$

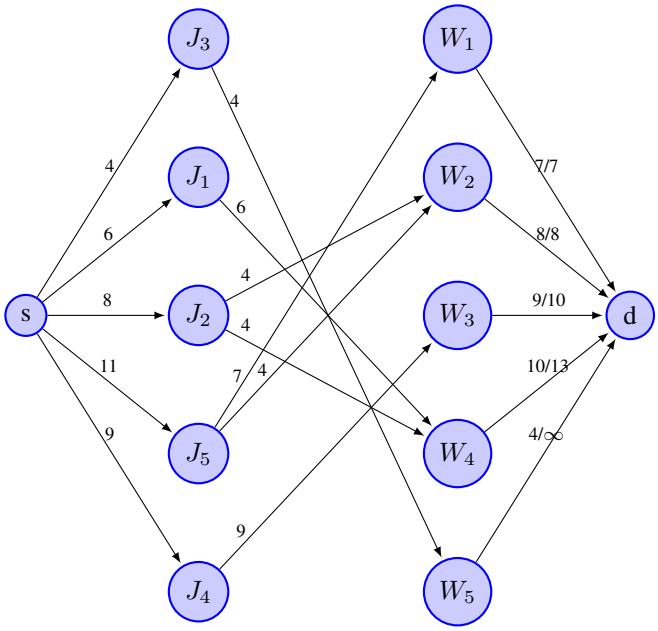


Fig. 8. $J_3 = 4, J_1 = 6, J_2 = 8, J_5 = 11, J_4 = 9 \Rightarrow C_{max} = 42$

So combining the order of jobs is achieved the best results and genetic algorithm approach is applied to solve combinatorial optimization problems.

Coding of a solution:

- Each feasible solution is a chromosome.
- Each chromosome is a set of job's position. For example:

Jobs	Position
J_1	1
J_2	2
J_3	3
J_4	4
J_5	5

Jobs	J_2	J_3	J_5	J_4	J_1
Chromosome 1	2	3	5	4	1

Jobs	J_3	J_2	J_4	J_1	J_5
Chromosome 2	3	2	4	1	5

Genetic Algorithm Operators:

- Selection elite operator: allows a percentage of the best solutions (based on fitness values) within the population to pass through to the next generation without being modified.
- Single crossover operator: takes two chromosomes, randomly selects an index, takes preceding section from chromosome-1 and succeeding section from chromosome-2 and generates a new one.

		⇓			
Chromosome 1	2	3	5	4	1
Chromosome 2	3	2	4	1	5
Child	2	3	4	1	5

- Swap mutation operator: swaps the values of some randomly chosen genes of a chromosome.

		⇓		⇓	
Chromosome	2	3	5	4	1
New	2	4	5	3	1

Fitness refers to the assessment of a solution provided by the genetic algorithm. And the range of the fitness value is [0,1], calculated by:

$$fitness = LB/C_{max} \tag{6}$$

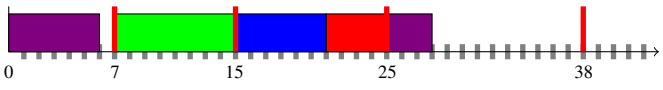
with the lower bound LB determined by total processing time of all jobs, and the C_{max} found by BMF heuristic.

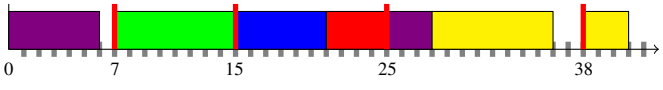
B. GA.MAAS - Genetic Algorithm on MAAS

In the MAAS heuristic at IV-B, we also realized that the order of jobs affects the result at the assignment step.

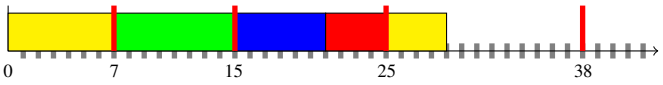
The example of section II:

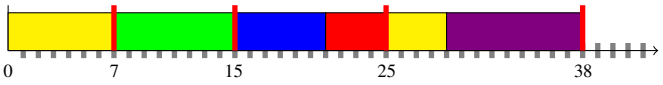
- $Jobs : \{J_1 = 6, J_2 = 8, J_3 = 4, J_4 = 9, J_5 = 11\} \Rightarrow C_{max} = 41$





- $Jobs : \{J_5 = 11, J_3 = 4, J_1 = 6, J_4 = 9, J_2 = 8\} \Rightarrow C_{max} = 38$





With the similar coding as the section V-A and the C_{max} found by MAAS heuristic, so applying genetic algorithm is achieved the best results.

VI. Experimental Results

A. Evaluation criteria and Dataset

In order to evaluate the quality of solutions, the objective value of the obtained solutions are compared with the lower bound LB determined by total processing time of all jobs. Remark that, the quality of solutions could be evaluated through two facts:

- number of optimal solutions found (if exist)
- gap (%) from known optimal solutions

However, these measures are not trivial to determine. In this paper, we propose to use:

- number of instances (#) which objective value of obtained solution is equal to LB
- relative percentage gap (%) from the LB, is computed according to following formula:

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

  where $Z^{\#}$ is objective value of obtained solution.
- run-time (t) in second.

In dataset, there are 10 instances for each combination of $n = \{10, 20, 30, 50, 100, 200\}$, $m = \{5, 10, 20, 30, 50, 100\}$ and $split_{min} = \{2, 3, 4\}$ or $\{5, 6, 7\}$ with each instance is generated as follows:

- Processing time is generated randomly by integer uniform distribution in $[split_{min}, 20]$;
- Window size is generated randomly by integer uniform distributions in $[2 \times split_{min}, 30]$.

B. The GA parameters

Applying Genetic Algorithm on BMF and MAAS with these parameters:

- The size of population PS={60, 80, 100, 120}
- Selection (elite) operator SO={0.05, 0.1, 0.2, 0.3, 0.4}
- Crossover operator CO={0.7, 0.8, 0.9}
- Mutation operator MO={0.02, 0.05, 0.1}

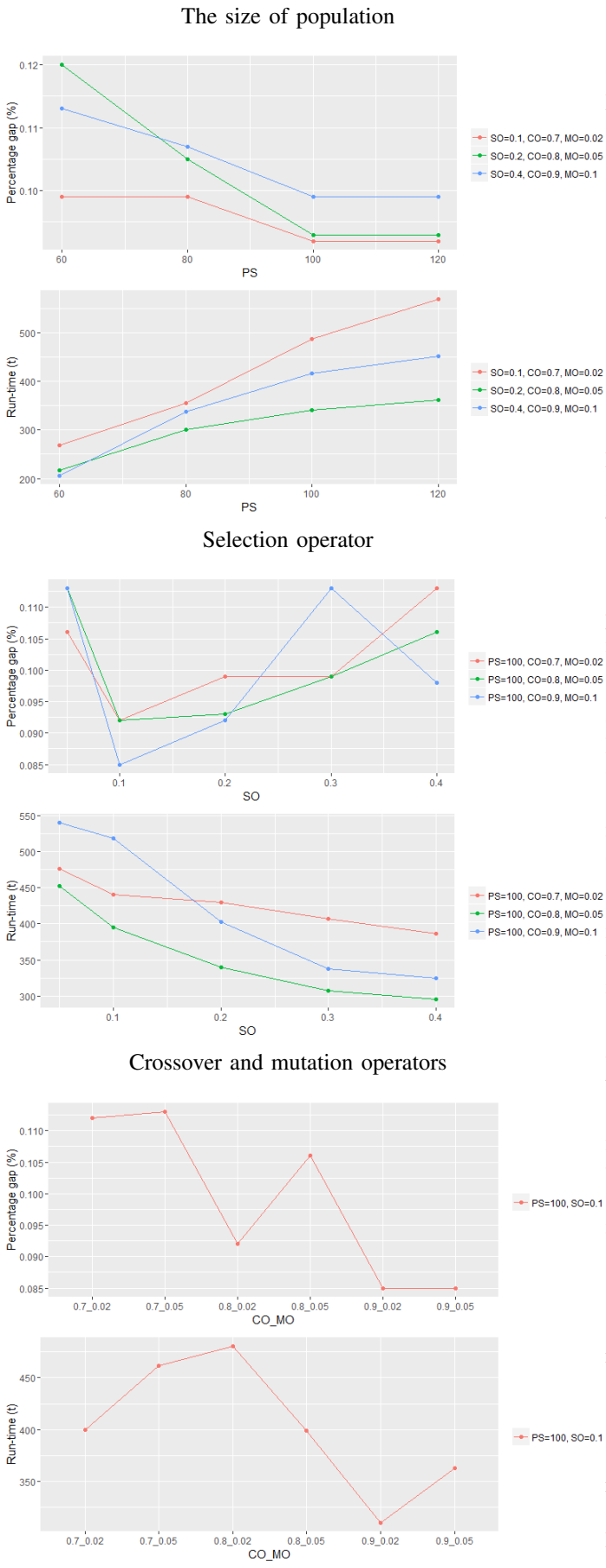


Fig. 9. The GA parameters

We decide the value of the parameter based on the criteria are percentage gap (%) first and the run-time then. Figure 9 show that these parameters PS=100, SO=0.1, CO=0.9, MO=0.02 are selected to apply GA on the heuristic algorithms.

C. Benchmarking

We have implemented the MILP model on GUROBI 5.1.0 solver, LPT heuristic [12] and the some proposed meta-heuristics. We have measured the performance of those methods on computer with configuration: Intel(R) Core(TM) i7-4650U 1.70GHz, 8GB memory with Windows 8.1 professional OS.

Table V shows the summarized results including relative percentage gap (%), number of instances (#) which the solution is equal to LB and run-time (t) in second.

Our experiment results indicate that GUROBI provides the quality of solutions (%, #) is the better than other heuristics. In particular, with the instance where $n \leq 50$ and $m \leq 30$, GUROBI finds the optimal solution with small run-time (under 10 seconds). However, with the other instances, GUROBI needs a lot of memory and calculating time (OutOfMemory exception with $n \geq 100$ and $m \geq 100$).

The run-time of LPT is impressive in all cases (the average is approximately 0.00 s), but the quality of solutions found is the worst among some approaches implemented (i.e., % factor is the highest, # factor is the lowest).

The proposed BMF provides a compromise approach between quality and performance. The quality of solutions is better than the one determined by LPT heuristic (% gap factor: 1.98% compared with 2.09%, # factor: average 0.94 optimal found compared with average 0.69 optimal found). The run-time is worse than (the average is about 112.05 s). With the scale of sample size is larger, the run-time of BMF will be increased steadily.

The proposed MAAS gives the quality of solutions (%, #) and the run-time is the best in heuristic algorithms group, with % factor: 1.36 is the lowest, # factor: 4.56 is the highest and run-time is approximately 0.00 s.

In meta-heuristic algorithms group, these genetic algorithms are GA.BMF and GA.MAAS provides the quality of solutions (%, #) is the better than other algorithms in heuristic algorithms group. However, genetic algorithms have to spend a lot of time in evolution, so the run-time of these algorithms are higher.

TABLE V
SUMMARY OF EXPERIMENTAL RESULTS

ID	n	m	msp	GUROBI			LPT			BMF			MAAS			GA.BMF			GA.MAAS		
				#	%	t	#	%	t	#	%	t	#	%	t	#	%	t	#	%	t
1	10	5	2	10	0.00	0.02	5	0.46	0.00	6	0.38	0.00	6	0.37	0.00	10	**0.00**	0.04	10	**0.00**	0.01
2	10	5	3	10	0.00	0.03	5	1.26	0.00	4	1.01	0.00	7	0.82	0.00	10	**0.00**	0.04	10	**0.00**	0.00
3	10	5	4	10	0.00	0.04	1	3.04	0.00	3	1.57	0.00	4	2.02	0.00	10	**0.00**	0.04	10	**0.00**	0.00
4	10	10	2	8	0.17	0.36	3	0.90	0.00	4	2.14	0.00	2	1.12	0.00	8	**0.17**	0.15	8	**0.17**	0.04
5	10	10	3	10	0.00	0.30	1	2.84	0.00	1	2.44	0.00	1	2.73	0.00	10	**0.00**	0.09	10	**0.00**	0.00
6	10	10	4	9	0.23	0.35	0	4.14	0.00	0	3.64	0.00	1	4.54	0.00	9	**0.23**	0.12	9	**0.23**	0.02
7	20	10	2	10	0.00	0.05	4	0.27	0.00	6	0.21	0.02	8	0.08	0.00	10	**0.00**	0.53	10	**0.00**	0.01
8	20	10	3	10	0.00	0.07	2	0.65	0.00	2	0.98	0.02	9	0.07	0.00	10	**0.00**	0.54	10	**0.00**	0.01
9	20	10	4	10	0.00	0.12	1	1.86	0.00	1	1.37	0.02	3	1.03	0.00	10	**0.00**	0.95	10	**0.00**	0.01
10	20	20	2	8	0.08	2.24	0	0.94	0.00	4	0.83	0.05	0	1.25	0.00	8	**0.08**	2.47	8	**0.08**	0.06
11	20	20	3	9	0.08	2.31	0	2.13	0.00	0	2.14	0.05	0	3.50	0.00	9	**0.08**	2.32	9	**0.08**	0.04
12	20	20	4	9	0.12	2.03	0	3.81	0.00	0	3.56	0.06	0	5.99	0.00	7	0.19	6.13	5	0.40	0.22
13	30	20	2	10	0.00	0.99	1	0.52	0.00	1	0.64	0.16	5	0.36	0.00	10	**0.00**	4.43	10	**0.00**	0.01
14	30	20	3	10	0.00	2.79	0	1.68	0.00	0	1.79	0.16	2	2.43	0.00	9	0.03	11.21	10	**0.00**	0.04
15	30	20	4	10	0.00	1.74	0	2.99	0.00	0	2.94	0.17	0	2.10	0.00	2	0.53	28.10	9	0.06	0.14
16	30	30	2	10	0.00	9.74	1	0.54	0.00	0	1.01	0.24	0	1.23	0.00	10	**0.00**	6.66	10	**0.00**	0.01
17	30	30	3	8	0.11	8.58	0	1.68	0.00	0	1.57	0.27	0	2.96	0.00	6	0.17	24.54	7	0.14	0.21
18	30	30	4	7	0.20	8.27	0	2.94	0.00	0	2.39	0.26	0	5.92	0.00	2	0.60	42.99	1	0.79	0.56
19	50	30	2	10	0.00	2.56	0	0.41	0.00	2	0.27	1.01	6	0.14	0.00	10	**0.00**	32.96	10	**0.00**	0.02
20	50	30	3	10	0.00	1.48	0	1.07	0.00	0	0.88	1.08	4	0.22	0.00	8	0.03	142.01	10	**0.00**	0.02
21	50	30	4	10	0.00	4.92	0	2.00	0.00	0	2.37	1.20	2	1.33	0.00	0	0.60	241.96	8	0.30	0.15
22	50	50	2	9	0.02	85.89	1	0.55	0.00	0	0.51	1.81	0	1.05	0.00	9	**0.02**	69.76	9	**0.02**	0.10
23	50	50	3	8	0.04	68.99	0	1.74	0.00	0	1.35	1.99	0	2.20	0.00	3	0.20	322.55	0	0.32	1.12
24	50	50	4	8	0.08	70.24	0	2.92	0.00	0	3.15	2.16	0	4.90	0.00	0	1.11	372.70	0	1.35	1.17
25	100	30	5	10	0.00	4.92	0	1.19	0.01	0	1.38	9.04	9	0.03	0.00	0	0.75	1277.01	10	**0.00**	0.12
26	100	30	6	10	0.00	10.49	0	1.87	0.01	0	1.87	14.23	10	**0.00**	0.00	0	1.08	1941.56	10	**0.00**	0.10
27	100	30	7	10	0.00	42.80	0	2.83	0.01	0	2.42	16.66	10	**0.00**	0.00	0	1.89	1623.47	10	**0.00**	0.10
28	100	50	5	10	0.00	42.26	0	2.80	0.01	0	2.37	31.16	9	0.02	0.00	0	1.87	3711.49	10	**0.00**	0.06
29	100	50	6	10	0.00	255.57	0	3.35	0.01	0	3.65	32.02	6	0.14	0.00	0	3.08	3287.97	10	**0.00**	0.06
30	100	50	7	10	0.00	642.27	0	5.65	0.01	0	5.33	41.77	3	0.27	0.00	0	4.11	3571.90	10	**0.00**	0.06
31	200	50	5	(ex)	(ex)	(ex)	0	0.92	0.01	0	1.26	146.73	10	**0.00**	0.01	(na)	(na)	(na)	10	**0.00**	0.68
32	200	50	6	(ex)	(ex)	(ex)	0	1.54	0.01	0	1.55	273.06	10	**0.00**	0.01	(na)	(na)	(na)	10	**0.00**	0.66
33	200	50	7	(ex)	(ex)	(ex)	0	2.08	0.01	0	1.90	402.58	10	**0.00**	0.01	(na)	(na)	(na)	10	**0.00**	0.72
34	200	100	5	(ex)	(ex)	(ex)	0	2.60	0.01	0	2.55	783.82	9	0.03	0.01	(na)	(na)	(na)	10	**0.00**	0.29
35	200	100	6	(ex)	(ex)	(ex)	0	3.69	0.01	0	3.38	1026.24	10	**0.00**	0.01	(na)	(na)	(na)	10	**0.00**	0.26
36	200	100	7	(ex)	(ex)	(ex)	0	5.20	0.01	0	4.56	1245.85	8	0.02	0.01	(na)	(na)	(na)	10	**0.00**	0.25
Average				9.43	0.04	42.41	0.69	2.09	0.00	0.94	1.98	112.05	4.56	1.36	0.00	6.00	0.56	557.56	8.69	0.11	0.20

Notes:

- each case is the average results of 10 different instances
- number of instances (#): higher is better
- percentage gap (%) and runtime (t): lower is better
- (ex) is OutOfMemory exception
- (na) is not acceptable runtime (less than one hour)

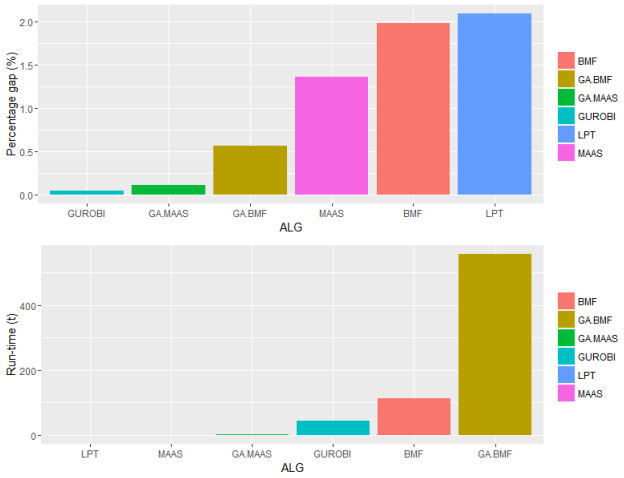


Fig. 10. Summary of experimental results

In summary, due to the ability of determining good solution and the quality of obtained solutions, the proposed meta-heuristic GA.MAAS gives best solution in the short time for practical data set (from 50 to 150 tasks [1] as presented in Section I) where the size of instance (n, m and msp) is large and complicated.

VII. CONCLUSION

In this paper, we considered the Jobs Scheduling problem with availability time windows, splitable jobs and min-split constraint so as to minimize the makespan. Some meta-heuristic algorithms based on BMF and MAAS are proposed. The quality of solution determined by these algorithms and the calculating time are compared with the exact method implemented by GUROBI solver, and compare with the best known heuristic LPT in the literature. The experimental results show GA.MAAS algorithm achieves a better compromise between the quality and the performance in the case when the size of input increases ($n \geq 50$).

Future works may address on solving Teamwork Scheduling problem is a expanding parallel machine of this problem.

ACKNOWLEDGMENT

This research was supported in part by Vietnam National Foundation for Science and Technology Development

(NAFOSTED) under grant no. 102.01-2012.01; in part by European Union through the Erasmus STA program.

REFERENCES

[1] D. Allen. *Getting things done, the art of Stress-free productivity*. Penguin Group, 2003.

[2] D.Q. Tran, N. Huynh Tuong, G.L. Hoang Ngoc, T.L. Mai, T.L. Tran, Q.T. Mai, and T.T. Quan. A personal scheduling system using genetic algorithm and simple natural language processing for usability. In *Multi-disciplinary International Workshop on Artificial Intelligence (MIWAI'2010)*, Mahasarakham, Thailand, 2010.

[3] B. Fleischmann and H. Meyr. The general lotsizing and scheduling problem. *OR Spektrum*, 19(1):11–21, 1997.

[4] S. Zaman and D. Grosu. Combinatorial auction-based allocation of virtual machine instances in clouds. *Journal of Parallel and Distributed Computing*, 73(4):495–508, 2013.

[5] M.T. Almeida and M. Centeno. A composite heuristic for the single machine early/tardy job scheduling problem. *Computers & Operations Research*, 25(7-8):625–635, 1998.

[6] C. Low, M. Ji, C-J Hsu, and C-T Su. Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, 34(2):334–342, 2010.

[7] E. Sanlaville and G. Schmidt. Machine scheduling with availability constraints. *Acta Informatica*, 9:795–811, 1998.

[8] G. Schmidt. Scheduling with limited machine availability. *European Journal of Operational Research*, 121:1–15, 2000.

[9] Y. Ma, C. Chu, and C. Zuo. A survey of scheduling with deterministic machine availability constraints. *Computer & Industrial Engineering*, 58:199–211, 2010.

[10] W. Xing and J. Zhang. Parallel machine scheduling with spliting jobs. *Discrete Applied Mathematics*, 103:259–269, 2000.

[11] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[12] V.H. Nguyen, N. Huynh Tuong, H.P. Nguyen, and T.H. Nguyen. Single-machine scheduling with splitable jobs and availability constraints. *REV Journal on Electronics and Communications*, 3(1-2):21–27, 2013.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1979.

[14] V.H. Nguyen, N. Huynh Tuong, V.H. 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)*, pages 397–400, Ho Chi Minh, Vietnam, 2013.

[15] L.R. Ford and D.R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399, 1956.