Regular Article

Single-machine Scheduling with Splitable Jobs and Availability Constraints

Van Huy Nguyen[1], **Nguyen Huynh Tuong**[1], **Hua Phung Nguyen**[1], **Thanh Hien Nguyen**[2]

[1] Faculty of Computer Science & Engineering, Ho Chi Minh city University of Technology, Vietnam
[2] Faculty of Information Technology, Ton Duc Thang University, Vietnam

Correspondence: Nguyen Huynh Tuong, htnguyen@cse.hcmut.edu.vn

Manuscript communication: received 18 September 2012, accepted 24 June 2013

***Abstract*– This paper deals with a single machine scheduling problem with availability constraints. The jobs are splitable and lower bound on the size of each sub-job is imposed. The objective is to find a feasible schedule that minimizes the makespan. The proposed scheduling problem is proved to be *NP*-hard in the strong sense. Some effective heuristic algorithms are then proposed. Additionally, computational results show that the proposed heuristic performs well.**

***Keywords*– single machine scheduling, availability constraint, splitable jobs, makespan, *NP*-hard, heuristic.**

1 Introduction

Scheduling is a decision process indicating how to allocate resources over the time axis to optimally perform a set of tasks while satisfying given constraints [1–5]. This paper addresses on the problem of scheduling splitable jobs on a single discontinuously-available machine. According to [6], splitting job property splitting figures out a job can be split into a number of independently processed sub-jobs. Since the size of such sub-job is not so small, we recruit a supplementary min-split constraint in this paper.

In current technology era, people have to perform many tasks at the same time. This scheduling could be utilized for each individual in dealing with a numerous complicated tasks. According to experts in productivity, we always have about 50 to 150 small tasks to process at any moment [7]. This paper studies in the context of real-life: "*in order to perform individual tasks in the best, it aims to schedule these tasks executed in some irregular available time-windows. The time-windows are caused 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 split must be larger than a given value. They are actually greater than 20 or 30 minutes, but the processing time of practical tasks are from 1 to 5 hours*" [8].

Formally, this scheduling problem is defined as follows.

- A set $\mathcal{N}$ consists of n jobs needs to be processed on a single machine.
- Machine can process only one job at a time and is not always available every time. This constraint is represented by m available time-windows W_k with size denoted by w_k, $1 \leq k \leq m$. For sake of simplicity, these windows could be also defined by time-breaks b_k with $1 \leq k < m$, i.e. that means $W_1 = [0, b_1)$, $W_2 = [b_1, b_2)$, ..., $W_m = [b_{m-1}, +\infty)$).
- All the jobs are available at time $t = 0$; the processing times are known in advance, deterministic and integer, p_i denotes the processing times of J_i, with $1 \leq i \leq n$.
- Each job could be divided into several sub-jobs. Each sub-job could be so small, but its size should be greater than or equal to a given value, denoted by $split_{min}$; each sub-job has to be processed in an available time-window under non-stopping mode, without any preemption during execution process. Let's denote $sp_{i,j}$ the j-th sub-job of job J_i.

Let $C_{\max}$ be the maximum completion time of all jobs (so called "*makespan*"). The objective function aims to minimize *makespan*. According to classical scheduling notation proposed by [9], the scheduling problem is defined as:

$$1|splitable,\ split_{min}, p_i \geq split_{min},\ available - windows, w_k \geq 2split_{min}|\mathrm{C}_{\max}.$$

In this paper, we will show that the makespan-minimization problem of single machine scheduling in available time windows with job splitting properties is strongly-*NP*-hard. Then, some efficient heuristics based on LPT (**L**ongest **P**rocessing **T**ime) rule are proposed with analysis of experimental results.

The rest of this paper is organized as follows. Some related works are presented in Section 2. Section 3, we propose structural properties of an optimal schedule. *NP*-completeness of problem will be presented in Section 4. Then, a heuristic to solve the considered problem is proposed in Section 5. Section 6 presents the computational experiments and analysis. Section 7 concludes this study and and provides some discussions on future work.

2 Related Works

Availability machine scheduling problems actually present in some planning as production scheduling and preventive maintenance [10], which are the most common and significant problems faced by the manufacturing industry and have paid attention from several researchers (for survey of scheduling problems with availability constraints, refer to [11–13]). Here we cite some interesting results concerning machine scheduling relational to minimizing makespan. Yang *et al*. [14] studied a single flexible maintenance activity. The problem was proved to be *NP*-hard. The authors provided an efficient heuristic algorithm with complexity $O(n \log n)$. Liao and Chen [15] proposed to solve scheduling problem with periodical maintenance and non-resumable constraints. They gave a heuristic algorithm for finding the near-optimal solution for large-sized problems in order to minimize the maximum tardiness. Yong *et al*. [16] approached the problem of scheduling a set of jobs on a single machine on which a rate-modifying activity could be performed. They assumed that the rate-modifying activity can take place only at certain pre-determined time points. One of the objectives was to minimize makespan. The analysis showed that the problems were *NP*-hard even for some special cases, then the authors provided a pseudo-polynomial time optimal algorithm for the problems. Ji *et al*. [17] dealt with single-machine scheduling with periodical maintenance to minimize makespan. They proved that the worst-case ratio of the classical LPT algorithm is 2, and there is no polynomial time approximation algorithm with a worst-case ratio less than 2 unless $P = NP$, which implies that the LPT algorithm is possibly the best as well. Chen [18] investigated in a single machine scheduling problem with periodic maintenance, where the machine was assumed to be stopped periodically for maintenance for a constant time during the scheduling period. Two mixed binary integer programming models were provided for deriving the optimal solution. Additionally, an efficient heuristic was proposed for finding the near-optimal solution for large-sized problems. Xu *et al*. [19] showed that there is no polynomial time approximation algorithm with a worst-case performance bound less than 2 unless $P = NP$ for the problem solved in [18]. The result implied that Chen's heuristic algorithm is the best possible polynomial time approximation algorithm for the considered scheduling problem.

Without availability machine constraints and min-split constraint, the considered scheduling problem becomes trivial and there is no interesting practical application corresponding. That's why there are a few results on single machine scheduling with only job splitting properties. But related to the parallel machine scheduling problems, there are some results. Kim *et al.* [20] showed that it can be applied in PCB manufacturing systems. Blocher and Chhajed [21] showed that the problem with the objective of minimizing total completion time is *NP*-hard and proposed several heuristics and lower bounds in order to minimize average completion time. Note that this objective is equivalent to minimizing total completion time or minimizing total Work in Progress (WIP [22]). In [22], Yang and Posner developed heuristics and gave worst-case bounds for these heuristics. With a more complicated objective to optimize, unrelated parallel machine problems with a job-splitting property for the objective of minimizing maximum weighted tardiness were considered [23]. For identical parallel machine scheduling problem to minimize a simpler objective- makespan, Xing and Zhang [6] proposed a heuristic algorithm and analyze the worst case performance $7/4 - 1/m$ of the algorithm with $m \geq 2$, number of machines.

In [8], Tran *et al.* considered a personal scheduling problem with splitable jobs for minimizing total weighted tardiness. An application with simple natural langague processing was implemented. Solutions were found by using genetic algorithm when availability machine constraints and min-split constraints.

3 Optimal Structure

Before determining structure of an optimal solution, let's show a numerical example which demonstrates the constraint essential of the scheduling problem with following input data.

- There are 4 jobs, J_1, J_2, J_3 and J_4.
- Let $split_{min} = 3$.
- Processing time of each job is respectively defined: $p_1 = 9$, $p_2 = 6$, $p_3 = 4$ and $p_4 = 8$.
- Available time-windows are [0,7], [7,15], [15,25] and [25, $+\infty$).

After calculating, an optimal decision is found and presented as follows (the corresponding solution described by Gantt chart in Figure 1).

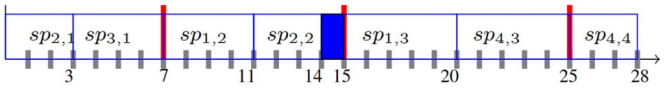

Figure 1. An optimal schedule of numerical example

- Job J_1 has two sub-jobs which correspond to $sp_{1,2} = 4$ (starts at $t = 7$) and $sp_{1,3} = 5$ (starts at $t = 15$);
- Job J_2 has two sub-jobs which correspond to $sp_{2,1} = 3$ (starts at $t = 0$), $sp_{2,2} = 3$ (starts at $t = 11$);
- Job J_3 is not split, i.e., it has only one sub-job which corresponds to $sp_{3,1} = p_3 = 4$ (starts at $t = 3$);
- Job J_4 has two sub-jobs which correspond to $sp_{4,3} = 5$ (starts at $t = 20$) and $sp_{4,4} = 3$ (starts at $t = 25$).

It is clear that optimal solution is not unique: sub-job $sp_{4,4}$ could interchange with a sub-jobs $sp_{2,1}$. In other words, we could determine another optimal solution where:

- Job J_1 has two sub-jobs which correspond to $sp_{1,2} = 4$ (starts at $t = 7$) and $sp_{1,3} = 5$ (starts at $t = 15$);

- Job J_2 has two sub-jobs which correspond to $sp_{2,2} = 3$ (starts at $t = 11$), $sp_{2,4} = 3$ (starts at $t = 25$);
- Job J_3 is not split, i.e. it has only one sub-job which corresponds to $sp_{3,1} = p_3 = 4$ (starts at $t = 3$);
- Job J_4 has two sub-jobs which correspond to and $sp_{4,1} = 3$ (starts at $t = 0$) and $sp_{4,3} = 5$ (starts at $t = 20$).

In the following, we will determine some structural properties of a particular optimal solution.

Proposition 1. *There exits an optimal schedule such that the jobs are scheduled without any idle time with size is greater than or equal to* $2 \times split_{min}$.

Proof: Suppose that there exists an optimal solution S having an idle-time (called X) with size greater than $\delta = 2 \times split_{min}$. Consider the last executed sub-job $J_{(i,j)}$ of S:

- if the processing time of this sub-job less than δ (i.e. $p_{(i,j)} < \delta$), we could create another solution by moving this sub-job into the idle time with size greater than δ. Then, the makespan value will be decreased an amount equal to the processing time of the considering sub-job.
- otherwise ($p_{(i,j)} \geq \delta$), we divide the sub-job into two splits: the first one with the size of $split_{min}$ and the second one with the size $p_{(i,j)} - split_{min}$. Then, the first sub-job could be move into the idle time X and therefore the makespan is decreased an amount of $split_{min}$.

So, in both cases, we could construct another solution that is better than S (which contradicts to the fact that S is an optimal solution). ■

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

Proof: Clearly, it is possible to interchange the positions of two consecutive sub-jobs in an available window. When the exchange occurs, the global objective value doesn't change. So, it could permute positions between two arbitrary sub-jobs (which are executed in the same time window), the makespan value does not change. ■

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

Proof: If there exists an optimal solution S in which a job J_i has more than two sub-jobs executed in a time window, then some permutation operations are performed in order to specify another solution S' in which two these sub-jobs executed sequentially. Since two these sub-jobs belong to the same jobs, we could form only one bigger sub-job instead of two these sub-jobs in S'. Furthermore, $C_{\max}(S') = C_{\max}(S)$. So, we could determine another optimal schedule that each job has at most one sub-job executed 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.*

Proof: Suppose that there is an optimal solution such that there are two idle times in an available time window. It's similar with the proof for Proposition 3 when we consider these idle times as two sub-jobs of a dummy job. So, some permutations are operated and it could determine another optimal solution so as to possess 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.*

Proof: Continuing with the proof in Prosition 4, the sub-job of dummy job (representing idle-time) could be permuted with other sub-jobs in order to be at the last position of the time window. This action define another optimal solution satisfying constraint in Proposition 5. ■

Based on these structural properties, the scheduling problem should determine the following decision sets:

1) number of splits (or sub-jobs) of each job,
2) assignment of sub-jobs in available time-windows,
3) the size of each sub-job.

In the next section, we will show that the problem with the above decision set is *NP*-complete.

4 *NP*-Completeness

Let's consider the following theorem.

Theorem 1. *Scheduling problem* $1|\ splitable,\ split_{min},\ available windows|C_{\max}$ *is "strongly NP-hard".*

In order to prove the theorem, let's consider the corresponding decision problem as follows.

Decision problem SPLITSCHE

Data input: Consider single machine scheduling has n splitable jobs to be executed with processing time p_1, p_2,..., p_n; m available window to execute these jobs could be defined $m-1$ time break b_1, b_2,..., b_{m-1}; processing time of each sub-job is greater than $split_{min}$ time unit.

Question: Does there exist a schedule such that $C_{\max} \leq y$, with y non-negative integer?

Proposition 6. *Decision problem SPLITSCHE is NP-complete.*

Proof: First, we should prove *SPLITSCHE* belongs to *NP* class. Given an"*yes-instance*" of *SPLITSCHE*, there exists permutation of decision set (defined in Section 3) that we construct a feasible solution to *SPLITSCHE* and verify in polynomial time such that the objective function gives rise to a solution with $C_{\max} \leq y$.

We next prove that *SPLITSCHE* belongs to *NP*-complete class by a reduction to $3-PARTITION$ which is known to be is "strongly *NP*-complete" [24].

Recall that $3-PARTITION$ is defined as follows:
Decision problem $3-PARTITION$

Data Input: Given an non-negative integer B and a multi-set $\mathcal{A} = \{a_1, \ldots, a_{3r}\}$ of $3r$ positive integers with $B/4 < a_k < B/2$ ($k = 1, \ldots, 3r$) and $\sum_{k=1}^{3r} a_k = rB$.
Question: Is there a partition of $\mathcal{A}$ into r mutually disjoint sub-set $\mathcal{A}_1, \ldots, \mathcal{A}_r$ such that the elements in $\mathcal{A}_k$ sum up to B for each $k = 1, \ldots, r$?

Given an instance of $3 - PARTITION$, We construct the following instance of the *SPLITSCHE* problem with $3r + 1$ jobs and $r + 1$ breaks ($r + 2$ time-windows) as follow:

- $\mathcal{N} = \{1, 2, \ldots, 3r, 3r + 1\}$
- Job $J_i, i \in \{1, \ldots, 3r\}$: $p_i = a_i$
- Job J_{3r+1}: $p_{3r+1} = B/2$
- $split_{min} = B/4$
- Breaks $b_t, t \in \{1, \ldots, r\}$: $b_t = tB$
- Breaks $b_{r+1} = (r + 1/4)B$
- $y = (r + 1/2)B$.

The remainder must be proved such that $3 - PARTITION$ is solvable if only if *SPLITSCHE* is solvable.

($\Rightarrow$) Suppose the answer of $3 - PARTITION$ is 'yes'. Then we determine solution of *SPLITSCHE* as follows: construct r block jobs from jobs $J_i, i \in \{1, \ldots, 3r\}$ such that block k ($k = 1, \ldots, r$) contains jobs that have the corresponding index with index of elements in sub-set A_r. Job J_{3r+1} is split into two equal-size parts which are executed on W_{r+1} and W_{r+2} respectively. Therefore, $C_{\max}$ is $(r + 1/2)B = y$. The answer for the question of decision problem *SPLITSCHE* is also 'yes'.

($\Leftarrow$) Suppose that there exists a solution σ which satisfies constraints in problem *SPLITSCHE* with $C_{\max} \leq y$. Since total processing time of jobs is equal to y, there does not exist any idle time between two sub-jobs in solution σ and without idle time in the r first time-window.

Note that $split_{min} = B/4$ and processing time of jobs J_i ($i \in \{1, \ldots, 3r\}$) is between $B/4$ and $B/2$ (but not equal to one of two margins). So these jobs couldn't be split into two parts and couldn't be executed in W_{r+1} and W_{r+2}; job J_{3r+1} is not split or split into two parts with the same size $split_{min}$.

As there is no idle time in σ and $w_{r+1} = B/4 = split_{min}$, J_{3r+1} should be split into two parts: first part $sp_{3r+1,1}$ executed in time-window W_{r+1} and another part $sp_{3r+1,2}$ executed in W_{r+2}.

Consequently, each remaining time windows W_t ($t = 1, \ldots, r$) has the size of B and contains a sub-set of jobs J_i ($i \in \{1, \ldots, 3r\}$). The sub-set have exactly three jobs, since processing time of jobs J_i ($i \in \{1, \ldots, 3r\}$) is between $B/4$ and $B/2$ (but not equal to $B/4$ or $B/2$). These jobs sub-sets help to determine r disjoint partitions of the corresponding $3 - PARTITION$. Hence, the answer of $3 - PARTITION$ problem is also 'yes'. ■

5 Heuristic

The proposed heuristic is composed by three phases. The first phase of heuristic is to construct a "traversing list" of jobs according to the following conditions:

- splitable jobs are located at the beginning of the list, and
- non-splitable jobs are located at the end of the list.

Note that LPT order (**L**ongest **P**rocessing **T**ime) could answer the above conditions. The second and third phases of heuristic are to make decision for the first part and second part of the traversing list respectively.

5.1 Special case: $p_i < 2split_{min}$

In a special case where the traversing list contains only non-splitable jobs (i.e. the third part). It means that the second part is null.The problem becomes $1|available - windows|C_{\max}$ which is well-known in the literature. We apply then LPT rule since according to [17], this heuristic gives a 2-approximation and the bound is tight.

5.2 General case

In the following, let's consider the traversing list composed by two parts (the first part contains initially all splitable jobs; the second one contains non-splitable sub-jobs). We consider now how to decide for the first part of the list which contains entirely the splitable jobs.

Let rp_i be remaining processing time of the considering job J_i, i.e. at the beginning, $rp_i = p_i$. Let rw_k be size of remaining available time of window W_k.

The available time windows are considered from left to right respectively (i.e. we start with window W_1). The following procedure performing firstly on the first part of the list (which contains all splitable jobs/sub-jobs) is proposed to solve differently for each of following cases concerning about relations between values of rp_i, $split_{min}$ and rw_k.

1) $rp_i \leq rw_k - split_{min}$:
   put J_i to be processed in window W_k
2) $rp_i > rw_k - split_{min}$:
   a) $rp_i \leq rw_k$:
      cut J_i into two sub-jobs with size respectively $(rp_i - split_{min})$ and $(split_{min})$,
      then put the sub-job with the size $rp_i - split_{min}$ to execute in window W_k,
      sub-job with the size $split_{min}$ of J_i is put at the beginning of the traversing list.
   b) $rp_i > rw_k$:
      i) $rp_i \geq rw_k + split_{min}$:
         cut J_i into two sub-jobs with size respectively (rw_k) and $(rp_i - rw_k)$,
         then put sub-job with size rw_k to execute in window W_k,
         sub-job with size $rp_i - rw_k$ of J_i is put at the beginning of the traversing list.
      ii) $rp_i < rw_k + split_{min}$:
         A) $rw_k \geq 2split_{min}$:
            cut J_i into two sub-jobs with size respectively $(rw_k - split_{min})$ and $(rp_i - rw_k + split_{min})$,
            then put sub-job with size $rw_k - split_{min}$ to execute in window W_k,
            sub-job with size $rp_i - rw_k + split_{min}$

of J_i is put at the beginning of the traversing list.

B) $rw_k < 2split_{min}$:
push this sub-job at the end of the traversing list (in second part).

When this phase is finished, the list contains only second part which contains non-splitable jobs and non-splitable sub-jobs (created from this phase). Then, we apply LPT rule for schedule all jobs and sub-jobs in this part. During last phase, in each iteration, considered job (or sub-job) will start at the first available moment with possible size in time axis. Remark that after finishing the second phase of the proposed procedure, there is no idle time between two jobs or sub-jobs. The most difficult case in this phase is case (2.b.ii.B). If the algorithm runs without encountering the case (2.b.ii.B), the obtained solution will be an optimal solution.

Running time: Each behavior of the first part will finish making decision of a job or a time window. So the complexity of the above procedure is bound by $O(n+m)$. The jobs and sub-jobs in second part will be scheduled in $O(n \log n)$ times (ordered according to LPT rule). These non-splitable ones are decided when executed in $O(m \times n)$. So, the overall complexity of the proposed heuristic is upper-bounded by $O(mn + n \log n)$.

6 Numerical Study

Remark that the total processing time of jobs is a lower bound since if a solution without idle time is feasible, this should be optimal. Let LB denote lower bound which is defined by total processing time of jobs.

In the following, we will compare the performance of the above heuristic proposed in Section 5 with some improvement strategies derived from well-known LPT rule:

- Heuristic 1: use LPT to schedule jobs without splitting.
  First, we put all jobs into a list and apply LPT rule. Second, the available time windows are considered from left to right respectively. For each available time window, we traverse the list which starts from first element. If the processing time of job is smaller than the time window, then the job is allocated to be executed in this window. This heuristic has complexity of $O(n \times m)$.
- Heuristic 2: use LPT to schedule jobs and jobs is splits if possible and needed.
  The principal is improved from Heuristic 1 by considering additionally some sub-jobs. These sub-jobs is created in the case where the job processing time is splitable and greater than the remaining window size (this size is also greater than $split_{min}$): we cut the considering job into sub-job 1 and sub-job 2; the sub-job 1 is allocated to be executed the window and sub-job 2 is put to the list. The complexity is bounded by $O(n \times m)$.

We have measured the performance of all the heuristics on machine with the following configuration: Intel Core i5 3.00GHz, 4GB memory under the Windows 7 professional operating system. In this experiment, there are 3 instances for each combination of $n = \{10, 20, 30\}$, $m = \{5, 10, 20\}$ and $split_{min} = \{2, 3, 4\}$. Each instance is generated as follows:

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

In order to evaluate the quality of solutions from a heuristic, the solutions are compared with the lower bound LB. Table I shows the summarized results including percentage deviation and number of solutions that obtained makespans achieve LB correspondingly. In this table, Heuristic 1, Heuristic 2, Heuristic 3 (proposed in Section 5) show the percentage gap between the solution of the heuristics and the lower bound, respectively. They are computed by following formula:

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

where $Z^{\#}$ obtained by the heuristics and lower bound LB the total processing time of jobs. Note that each case in Table I is the average results of 3 different instances.

Our numerical experiments indicate that the results of the three heuristics are very close to the lower bound. Especially, Heuristic 3 performs very well under all parameter combinations setting: the percentage gap is never exceeded 3%; the average optimality gap for all setting remains fairly small (about 0.45%). This heuristic, in addition, has an asymptotic optimality since obtained solutions are confirmed to be optimal for 60 of 117 testing problems. This outperforms in comparing with optimal results that could be found by Heuristic 1 and Heuristic 2.

Furthermore, average optimality gaps for all setting of triplet (n,m,$split_{min}$), presented on each line of Table I, tend to decrease from left to right, i.e. the value of the solution is obtained by Heuristic 3 is smaller than the ones is obtained by Heuristic 1 or Heuristic 2. Heuristic 3 is then the most efficient performance, followed by Heuristic 2, and then Heuristic 1. In summary, the results which are derived from Heuristic 3, are impressive in all cases. We could conclude that the heuristic proposed in Section 5 is efficient to find the good solution which is very close to the optimal solution. Thus, it is possibly to apply in practical setting as well.

7 Conclusions

In this paper, we have considered the problem of personal scheduling in availability time windows with splitable jobs and min-split constraint so as to minimize the *makespan*. After presenting several basic properties for an optimal solution to the problem, we proved that the problem is strongly NP-hard for the general case. An efficient heuristic is proposed then. Experimental results show that obtained solutions have never exceeded 3% and average deviation of about 0.45%.

Table I
Average Percentage of Gaps

n	m	$split_{min}$	Heuristic 1		Heuristic 2		Heuristic 3	
			%	# inst.	%	# inst.	%	# inst.
10	5	2	4.71	0	0.29	2	0.00	3
		3	4.20	0	1.67	1	0.00	3
		4	6.58	0	1.17	1	0.35	2
10	10	2	21.52	0	0.58	1	1.87	1
		3	32.03	0	8.05	0	1.49	0
		4	7.96	0	3.48	0	0.94	2
10	20	2	15.32	0	0.29	2	0.00	3
		3	10.91	0	2.77	0	2.10	1
		4	10.20	0	4.06	0	1.16	1
20	5	2	0.61	1	0.00	3	0.00	3
		3	0.86	1	0.71	1	0.00	3
		4	1.94	0	1.28	1	1.13	2
20	10	2	3.00	0	0.44	1	0.15	2
		3	3.20	0	1.35	1	0.40	1
		4	4.11	0	0.38	2	0.00	3
20	20	2	4.33	0	0.71	2	0.47	2
		3	3.61	0	1.32	0	1.33	0
		4	7.81	0	3.36	0	1.55	0
30	10	2	1.15	0	0.10	2	0.09	2
		3	1.10	0	0.49	0	0.21	0
		4	0.83	0	0.65	1	0.10	2
30	20	2	6.88	0	0.66	0	0.27	2
		3	8.45	0	1.60	0	0.28	1
		4	4.45	0	2.81	0	1.18	0
50	10	2	0.29	1	0.12	2	0	3
		3	0.11	1	0.35	1	0	3
		4	0.73	0	0.34	1	0.17	2
50	20	2	0.77	0	0.31	0	0.06	2
		3	1.6	0	0.29	1	0.06	2
		4	1.96	0	0.93	0	0.34	2
50	30	2	3.64	0	0.55	0	0.06	2
		3	3.39	0	0.89	0	0.4	0
		4	1.96	0	0.93	0	0.34	2
100	20	2	0.12	0	0.21	0	0.03	0
		3	0.48	0	0.58	0	0.06	1
		4	0.5	0	0.91	0	0.47	0
100	30	2	0.09	1	0.15	0	0.06	2
		3	0.42	0	0.45	0	0.17	0
		4	0.67	0	0.9	0	0.34	0
Average			4.67	-	1.18	-	0.45	-
Total			-	5	-	26	-	60

Further research can be undertaken to construct mathematical model based on properties of an optimal solution established in section 3, to improve the proposed heuristics and to test with larger-size instances.

Acknowledgement

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.01-2012.01.

References

[1] J. Y.-T. Leung, *Handbook of scheduling : algorithms, models, and performance analysis*. Boca Raton, Florida: Computer and information science series, Chapman and Hall/CRC, 2004.

[2] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, *Handbook on scheduling : from theory to applications*. Springer, 2007.

[3] P. Brucker, *Scheduling algorithms*, 4th ed. Berlin, Germany: Springer-Verlag, 2004.

[4] M. Pinedo, *Scheduling : theory, algorithms, and systems*, 2nd ed. Upper Saddle River, New Jork, USA: Precentice Hall, 2002.

[5] V. T'Kindt and J.-C. Billaut, *Multicriteria scheduling : theory, models and algorithms*, 2nd ed. Springer, 2006.

[6] W. Xing and J. Zhang, "Parallel machine scheduling with splitting jobs," *Discrete Applied Mathematics*, vol. 103, pp. 259–269, 2000.

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

[8] D. Q. Tran, N. H. Tuong, G. L. H. 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 *Proc. of Multi-disciplinary International Workshop on Artificial Intelligence (MIWAI'2010)*, Mahasarakham, Thailand, 2010.

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

[10] 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*, vol. 34, pp. 334–342, 2010.

[11] E. Sanlaville and G. Schmidt, "Machine scheduling with availability constraints," *Acta Informatica*, vol. 9, pp. 795–811, 1999.

[12] G. Schmidt, "Scheduling with limited machine availability," *European Journal of Operational Research*, vol. 121, pp. 1–15, 2000.

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

[14] D. Yang, C. Hung, C. Hsu, and M. Chern, "Minimizing the makespan in a single machine scheduling problem with a flexible maintenance," *Journal of the Chinese Institute of Industrial Engineers*, vol. 19, pp. 63–66, 2002.

[15] C. J. Liao and W. J. Chen, "Single-machine scheduling with periodic maintenance and nonresemable jobs," *Computers and Operations Research*, vol. 30, pp. 1335–1347, 2003.

[16] H. Yong, M. Ji, and T. Cheng, "Single machine scheduling with a restricted rate-modifying activity," *Naval Research Logistics*, vol. 52, pp. 361–369, 2005.

[17] M. Ji, Y. He, and T. Cheng, "Single-machine scheduling with periodic maintenance to minimize makespan," *Computers and Operations Research*, vol. 34, pp. 1764–1770, 2007.

[18] J. S. Chen, "Scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan," *European Journal of Operations Research*, vol. 190, pp. 90–102, 2008.

[19] K. S. D. Xu and H. Li, "A note on scheduling of nonresumable jobs and flexible maintenance activities on a single machine to minimize makespan," *European Journal of Operations Research*, vol. 197, pp. 825–827, 2009.

[20] Y. D. Kim, S. O. .Shim, S. B. Kim, Y. C. Choi, and H. M. Yoon, "Parallel machine scheduling considering a job splitting property," *International Journal of Production Research*, vol. 42, pp. 4531–4546, 2004.

[21] J. D. Blocher and D. Chhajed, "The customer order lead-time problem on parallel machines," *Naval Research Logistics*, vol. 43, pp. 629–654, 1996.

[22] J. Yang and M. Posner, "Scheduling parallel machines for the customer order problem," *Journal of Scheduling*, vol. 8, pp. 49–74, 2005.

[23] P. Serafini, "Scheduling jobs on several machines with job splitting property," *Operations Research*, vol. 44, pp. 617–628, 1996.
[24] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman & Company, 1979.

Van Huy NGUYEN is now, a lecturer in the Faculty of Information Technology, Ho Chi Minh City University of Transport. He received his B.Eng. degree in Information Technology from HCM University of Technical Education in 2008 and received Master degree in 2012 from Bordeaux 1, France. His research interests include scheduling and embedded systems.

Nguyen HUYNH TUONG is a senior lecturer in the Faculty of Computer Science and Engineering, HCMC University of Technology, Vietnam. He has received his Ph.D. (2009) and M.Eng (2006) in Computer Science from Tours University (France), and B.Eng (2001) in Computer Engineering from HCMC University of Technology, Vietnam. His research interests are in the areas of scheduling, high performance computing and network security.

Hua Phung NGUYEN is a senior lecturer in the Faculty of Computer Science and Engineering, HCMC University of Technology, Vietnam. He has received his Ph.D. (2005) in Computer Science from UNSW, and M.Eng (1999) and B.Eng in Computer Engineering from HCMC University of Technology, Vietnam. His research interests include scheduling, data mining, program analysis and verification.

Thanh Hien NGUYEN has served as the Dean, Faculty of Information Technology, Ton Duc Thang University since 2012. His research interests include Job Shop Scheduling, Information Extraction, Machine Learning, Natural Language Processing, Web/Text Mining, Semantic Web, and Networks Analysis. He received Ph.D. in Computer Science, M.S. in Computer Science, and B.Eng. in Computer Engineering from Ho Chi Minh City University of Technology, in 2011, 2005, and 2002 respectively. Contact him at Faculty of Information Technology, Ton Duc Thang University, Nguyen Huu Tho St., District 7, Ho Chi Minh City, Vietnam.