New Heuristic For Single Machine Semi-online Total Completion Time Minimization

Hajar Nouinou, Taha Arbaoui, Alice Yalaoui

Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 53-2 (2020) 10676–10681

New Heuristic For Single Machine Semi-online Total Completion Time Minimization

H. Nouinou * **T. Arbaoui** * **A. Yalaoui** *

* *Logistics and Industrial Systems Optimization team, Charles Delaunay Institute, FRE CNRS 2019, University of Technology of Troyes, 12 Rue Marie Curie, 10004 Troyes, Cedex, France (e-mails: {hajar.nouinou; taha.arbaoui; alice.yalaoui}@utt.fr).*

Abstract: This paper addresses a semi-online setting of the minimization of the total completion time scheduling problem on a single machine, where jobs arrive over-time, i.e, each job has a corresponding release date at which it becomes available for processing. In this study, the case where the release dates of the jobs are known at the beginning of the decision process is considered while processing times remain unknown. A semi-online algorithm that makes use of the available information in order to produce better schedules compared to its online peers is presented. A numerical analysis is established, showing the impact of having this information about release dates.

Keywords: Total completion time, Semi-online scheduling, Numerical analysis.

1. INTRODUCTION

The problem of minimizing the total completion time on a single machine with release dates is a fundamental problem that received considerable attention during the last decade. In offline settings, characteristics of jobs are all available at the beginning of the decision process. An offline algorithm must define a proper sequence of these jobs in order to optimize the objective function. However, even if all information are available in advance, the decisions can be very hard to make since the problem is classified as NP-hard (Lenstra et al., 1977). On the other hand, when preemption is allowed, the problem can be solved in polynomial time using the SRPT (Shortest Remaining Processing Time) algorithm (Lawler et al., 1993).

In online settings, a sequence of n jobs arriving over time must be scheduled on the machine. Each job J_j has a release date r_j at which it becomes available and a processing time p_j. Job's characteristics become available for the decision maker when the job actually arrives. The goal is to minimize the total completion time $\sum C_j$, where C_j is the completion time of job J_j. The problem can be denoted $1|r_j, online|\sum C_j$ in the Graham's notations (Graham et al., 1979). The quality of an online algorithm is often measured by the value of the competitive ratio. An online algorithm is ρ-competitive if for any instance, the value of the objective function of the schedule obtained by this algorithm is not worse than ρ times the value obtained by the optimal offline algorithm. List scheduling, with the objective to minimize the makespan, is the first problem studied by competitive analysis (Graham, 1966). For the online problem of minimizing the total completion time on a single machine with release dates, Mao et al. (1995) proved that SPT (Shortest Processing Time) has a performance guarantee of n, where n is the total number of jobs. SPT consists in choosing a job with the smallest processing time among all available and unscheduled jobs. However, the problem can be solved optimally using SPT if release dates of jobs are equal.

The online problem of minimizing the total completion time on a single machine with jobs arriving over time was later studied by Hoogeveen and Vestjens (1996). They presented an online algorithm called D-SPT with a competitive ratio of 2. They also proved that their algorithm is the best possible by deriving a lower bound of 2 on the competitive ratio of any online algorithm. In the classical D-SPT algorithm, when an available job J_j with the smallest processing time denoted by p_j is selected, it must verify $p_j \leq t$ in order to be scheduled, with t the current decision time. This decision making prevents the algorithm from performing too bad compared to the offline schedule.

Between offline and online models, we find the semi-online models where some information about incoming jobs is available in advance. A Recent survey of semi-online models was presented by Tan and Zhang (2013) for the problem of minimizing the total completion time on a single machine. For the objective of minimizing makespan on parallel machines, many semi-online models were considered. Information about some of the models can be found in recent surveys (Albers, 2013; Tan and Zhang, 2013; Epstein, 2018).

Some semi-online models can be found in literature where an information about processing times is known in advance. Liu et al. (2012) studied the problem with deteriorating jobs. They provided a semi-online algorithm

10.1016/j.ifacol.2020.12.2838

Table 1. Existing online and semi-online algorithms

Reference	Added information	Algorithm
Hoogeveen and Vestjens (1996)	Non	D-SPT
Tao et al. (2009)	$\frac{p_{max}}{p_{min}} \leq \gamma$	αD-SPT
Liu et al. (2012)	$p_j = a_j t$	D-SGR
Hall et al. (2009)	T_k for $k \in \{1, ..., m\}$	CSWPT
Section 2	r_j for $j \in \{1, ..., n\}$	VD-SPT

called D-SGR (Delayed Smallest Growth Rate) with a competitive ratio of $1+a_{max}$, where $a_{max} = max_{J_j \in l}\{a_j\}$, and $a_j \geq 0$ and t are the deteriorating rate and start time of processing of jobs, respectively. Tao et al. (2009) studied the problem denoted as $1|r_j, online, \frac{p_{max}}{p_{min}} \leq \gamma| \sum C_j$, where p_{max} and p_{min} denote the longest and the shortest processing times, respectively, and $\gamma \geq 1$ is a problem's data known at the beginning of the decision process. They presented a semi-online algorithm called αD-SPT, which is a modified version of D-SPT. The condition to verify in order to schedule an available job is $p_j \leq \alpha t$ where $\alpha = \frac{1+\sqrt{1+\gamma(\gamma-1)}}{\gamma-1}$. They proved that αD-SPT is $1 + \frac{1}{\alpha}$ competitive. Hall et al. (2009) presented a semi-online algorithm for the problem of minimizing the average weighted completion time on a single machine. They considered an online planning period scheduling problem which means that they know the possible arrival times, denoted T_k, with $k \in \{1, ..., m\}$ and m refers to the total number of potential arrival times. However, they do not necessarily know if a job is going to be released at this arrival time or how many jobs are going to be released (Table 1).

The D-SPT algorithm makes decisions in an online manner, which means that no information regarding the incoming jobs are available at the beginning of the decision making, which justifies the need to define a fixed waiting time that depends only on the processing time of an available job and the value of the current decision time. However, in this article we study the semi-online variant of the problem where release dates of incoming jobs are known in advance. The first advantage of adding this information is that the semi-online algorithm does not always have to wait. For example, if some jobs are available and we know that no more jobs are going to be released in the future, a semi-online algorithm will no longer insert idle time and jobs will be scheduled by SPT order which is optimal in this case. The answer to the question: are more jobs going to be released in the future? becomes crucial in this case. Another example displaying the importance of having this information about release dates is when some jobs are available and the next release date is greater than the sum of the processing times of these available jobs and the current decision time. In this case, jobs can also be scheduled in an offline manner without inserting any idle time since we know that the next release date will not interfere with the scheduling of available jobs.

We present in this paper a new algorithm, called VD-SPT (Variable Delayed Shortest Processing Time) which is a result of the reflections described above. It makes use of the information in hand, which is the release dates of incoming jobs.

The paper is organised as follows: In Section 2 we present the proposed approach for designing the semi-online algorithm denoted VD-SPT. In Section 3 an experimental study is presented by implementing both the proposed semi-online algorithm and the online algorithm D-SPT. Conclusions are presented in Section 4.

2. THE PROPOSED APPROACH

In designing a semi-online or an online algorithm, the same logic is used as for designing an offline algorithm. The main difference lays on the amount of information used in the execution of the algorithm. An online algorithm is allowed to use only information available at the current decision time, while an offline algorithm has all the information needed to construct a good schedule. However, in the design of VD-SPT we make use of an existing information which is the release dates of jobs which implies that we also know the number n of jobs that are going to be released.

2.1 Definition of t_{max}

For the semi-online problem of minimizing the total completion time, a semi-online algorithm must answer the following questions: If some job is available, is it better to wait or to schedule immediately the available job? and if it chooses to wait, then for how long? Since we have an information about release dates, we must use this information in the construction of our algorithm.

Hoogeveen and Vestjens (1996) proved that D-SPT is the best possible algorithm for the problem $1|r_j, online| \sum C_j$ by deriving a lower bound on the competitive ratio of any online algorithm. The lower bound was achieved by considering a set of instances that represent the worst case scenario for any online algorithm and for which no online algorithm can guarantee an outcome strictly less than twice the optimum. In order to find these instances, the adversary method was employed. In this method, the worst case is obtained by playing role of adversary that tries to make the online algorithm perform as bad as possible compared to its own performance when serving the same instance in an offline manner. The instances can be described as follows: a first job arrives at time 0 with a processing requirement p. The first scenario is when the online algorithm decides to schedule the first job at time S while the adversary releases no more job in the future. The second scenario is when $n-1$ small jobs with processing times equal to 0 are released immediately after the online algorithm decides to schedule the first job.
By intuition and based on Hoogeveen and Vestjens (1996) demonstration of lower bound, we can imagine two possible cases that represent the worst case scenarios for a semi online algorithm that has the advantage of knowing release dates of jobs. First, among available and unscheduled jobs, a job with the smallest processing time is chosen, denoted by p. Moreover, two possible worst cases might occur. The first case will be when the adversary releases a great

number of jobs with small processing times after the online algorithm decides to schedule a long job. The second case is when the algorithm decides to wait for next released jobs and they turn out to have the same processing time as the first. These two case are studied in order to determine the value of t_{max}, which is the maximum time that a semi-online algorithm can afford to wait in order to have some sort of a compromise between these two worst cases.
We must note that in these two cases, we only focus on the jobs arriving in the scheduling time interval $]t, t+p[$, with t is the current decision time, since jobs released later than $t+p$ will not affect the decision making regarding the current job. We denote by t' the arrival time of n' jobs, such as $t' \in]t, t+p[$ and $1 \leq n' \leq n-1$ where n denotes the total number of jobs. As a result of studying the following two worst cases, the value of t_{max} is determined. It depends on the number of jobs n' to be released and the time t' at which they will be released. In the following, C_j^* refers to the completion time of job J_j in the offline schedule.

Case 1: The semi-online algorithm decides to schedule immediately the available job with processing time p at time t and the adversary releases n' jobs with processing time equal to 0 at time t' (Figure 1). Hence, the total completion time of the semi-online algorithm becomes:

$$\sum C_j = t + p + n'(t+p) \tag{1}$$

$$\Leftrightarrow \sum C_j = (n'+1)(t+p) \tag{2}$$

Since we study the worst case, which is a bad decision made by a semi-online algorithm compared to the offline algorithm, we consider that $t' < \frac{n'}{n'+1}p + t$, which is the case where the offline algorithm decides to wait for small jobs since its cost will be better than the online one (Figure 1), while on the opposite case, the adversary will make the same decisions as the semi-online algorithm. Hence, Equations (3) and (4) are only valid if t' verifies this inequality.

$$\sum C_j^* = n't' + t' + p \tag{3}$$

$$\Leftrightarrow \sum C_j^* = (n'+1)t' + p \tag{4}$$

Hence the ratio of the two objective values is

$$\frac{\sum C_j}{\sum C_j^*} = \frac{(n'+1)(t+p)}{(n'+1)t' + p} \tag{5}$$

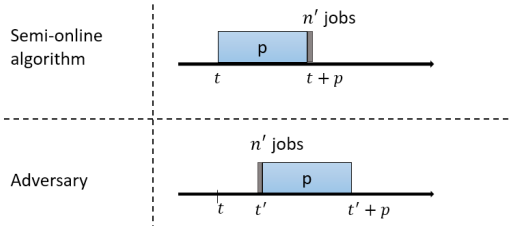

Fig. 1. Schedules for case 1

Case 2: For this second case, the semi-online algorithm decides to wait for the n' jobs arriving at time t' and the adversary releases these jobs with processing time p equal to the processing time of the first job (Figure 2). then the total completion time of the semi-online algorithm will be as follows

$$\sum C_j = t' + p + \frac{n'(n'+1)}{2}p + n'(t'+p) \tag{6}$$

$$\Leftrightarrow \sum C_j = (n'+1)(t'+p) + \frac{n'(n'+1)}{2}p \tag{7}$$

The offline algorithm will obviously schedule the first job immediately since any waiting time would be wasted (Figure 2).

$$\sum C_j^* = t + p + \frac{n'(n'+1)}{2}p + n'(t+p) \tag{8}$$

$$\Leftrightarrow \sum C_j^* = (n'+1)(t+p) + \frac{n'(n'+1)}{2}p \tag{9}$$

Hence the ratio of the two objective values is

$$\frac{\sum C_j}{\sum C_j^*} = \frac{2t' + (n'+2)p}{2t + (n'+2)p} \tag{10}$$

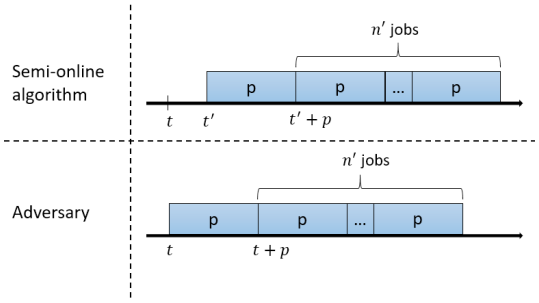

Fig. 2. Schedules for case 2

Equation (5) is a decreasing function with respect to t' while equation (10) is an increasing function with respect to t'. In order to find t_{max} which represents the compromise between these two cases we have to study the equality of these two functions. In the following, let t_{max} be the value of t' that results from studying the equality.

$$\frac{(n'+1)(p+t)}{(n'+1)t' + p} = \frac{2t' + (n'+2)p}{2t + (n'+2)p} \tag{11}$$

It can be seen from equation (13) below that the value of t_{max} depends on three parameters, n' as the number of jobs to be released, t as the current decision time and p as the smallest processing time of an available job. In what follows, a semi-online algorithm is presented making use of the expression of t_{max}.

$$2(n'+1)t_{max}^2 + ((n'+1)(n'+2)+2)pt_{max} - (n'^2+5n'+4)tp - (2n'+2)t^2 - n'(n'+2)p^2 = 0 \tag{12}$$

$$t_{max} = \frac{-(2+(n'+1)(n'+2))p + \sqrt{(2+(n'+1)(n'+2))^2p^2 + 8(n'+1)[(n'^2+5n'+4)tp + (2n'+2)t^2 + n'(n'+2)p^2]}}{4(n'+1)} \tag{13}$$

2.2 VD-SPT algorithm

In the case studied in subsection 2.1, we assumed that n' jobs could arrive at t'. Now, in order to use the previous result, we are going to consider not only the next n' jobs which are going to arrive at the next release date, but the n'' jobs which are going to be released in the interval $]t, t+p[$. Hence, in the following, equation (13) will be used using n'' instead of n' and for $t = 0$ in the expression of t_{max}, while the maximum time that VD-SPT can afford to wait is $t+t_{max}(n'', p)$. Moreover, $t+t_{max}$ will be compared to the last release date in the interval $]t, t + p[$, denoted $r_{n''}$, in an iterative manner, such as if the condition is not verified, the last job is deleted and is replaced by the antecedent one if it exists.

In step 1 of algorithm VD-SPT, an available job with the smallest processing time p_m is selected. Then in step 2, set A is filled with the n'' jobs released in the interval $]t, t + p_m[$, we sort and denote release dates of these jobs by $A = \{R_1, ..., R_{n''}\}$. In case no job is released within the considered interval, we schedule immediately the selected job (with minimum processing time p_m) and update the value of the current decision time ($t = t + p_m$). In step 3 we compute the value of $t_{max}(n'', p_m)$. If the time we can afford to wait for the last job in set A is greater than or equal to the release date of this last then we wait, otherwise we update set A by deleting the last job. We repeat the above procedure until set A is empty or the decision to wait is made and the value of t is updated.

Step 0: Set t at the first release date, N the set of jobs available at time t, and N' the set of jobs with release dates greater than t.

Step 1: Set $A = \emptyset$. At time t, among jobs in N, choose a job J_0 with the smallest processing time, denote it p_m.

Step 2: If some jobs are released in the interval $]t, t + p_m[$, then add the release dates of these jobs to the set A and denote them as $A = \{R_1,..., R_{n''}\}$ sorted in an increasing order and go to step 3. Else go to step 6.

Step 3: Compute t_{max} for n'' and p_m, and go to step 4.

Step 4: If $R_{n''} \leq t + t_{max}$ then add jobs in A to set N and delete them from set N', then go to step 5. Else delete $R_{n''}$ from A and update the set with $n'' = n'' - 1$. If $A = \emptyset$ then go to step 6, else go back to step 3.

Step 5: If the first job in A can be completed before $R_{n''}$ then schedule it immediately and delete it from set N, else go to the next job until all except the last job in A are verified. Finally, update the decision time with $t = R_{n''}$ and go to step 1.

Step 6: Schedule job J_0 at time t and put $t = t + p_m$. If $N = \emptyset$ and $N' \neq \emptyset$ then set t at the next release date and go to step 1. If $N \neq \emptyset$ then go to step 1. If $N = \emptyset$ and $N' = \emptyset$ then Stop.

Consider the instance in table 2 as an example to apply VD-SPT algorithm. Release dates are known but the processing time of a job only becomes known when the current decision time t is greater than or equal to the release date of that job. In step 0, we put $t = r_1 = 0$, $N = \{J_1, J_2\}$, and $N' = \{J_3, J_4\}$. Next, in step 1, the smallest job J_2 is chosen and is denoted J_0 with processing time $p_2 = p_m = 6$. In step 2, we find that in the interval $]t, t + p_m[=]0, 6[$ there are $n'' = 2$ jobs released, we denote their release dates $A = \{R_1, R_2\} = \{r_3, r_4\} = \{1, 2\}$. In step 3, we find the value of $t_{max} = 2.8$ for $n'' = 2$ and $p_m = 6$. In step 4, we compare $R_{n''} = 2$ to $t + t_{max} = 2.8$, and since the second is greater than the first then we update sets with $N = \{J_1, J_2, J_3, J_4\}$ and $N' = \emptyset$ and go to step 5. Since the first job in A with release date $R_1 = 1$ can be completed before $R_{n''}$ then we schedule it and update set $N = \{J_1, J_2, J_4\}$. Moreover, we update the decision time with $t = R_{n''}$ and go to step 1. The same procedure is repeated and as a result job J_3 is scheduled first at $t = 1$ followed by J_4, J_2 and J_1 starting from $t = 2$.

Table 2. Instance example

j	1	2	3	4
r_j	0	0	1	2
p_j	8	6	0	4

3. COMPUTATIONAL EXPERIENCE

In the following, a numerical analysis is presented by implementing four different algorithms D-SPT, VD-SPT, SPT and SRPT. The online algorithm D-SPT is compared to our semi-online algorithm VD-SPT. SRPT is used as a lower bound on the objective value of an optimal offline algorithm for the problem and thus an upper bound on the competitive ratio for each algorithm can be computed. Moreover, SPT allows to display cases where no waiting strategy is better and thus identify which of the two algorithms inserts less idle time. Firstly, an average case study is presented where instances generated by probability distributions are tested. Secondly, a worst case analysis is presented by testing an instance where jobs with long processing requirements arrive followed by jobs with small processing requirements.

Many experimental studies can be found in literature. A recent work proposed by Albers (2013) focused on online scheduling algorithms for minimizing the makespan on parallel machines. They analysed algorithms on various job sequences. Some of them were generated by probability distributions, while others were real world data. Hall and Posner (2001) provided specific proposals for the generation scheme of a variety of machine scheduling problems including the problems with release dates.

3.1 Average case analysis

VD-SPT, D-SPT, SPT, and SRPT were tested on problems with 10, 15 and 20 jobs. For processing times, we generate for each job, an integer processing time p_j from the uniform distribution $U[0, 10]$. Furthermore, since range of release dates is likely to influence the effectiveness of the algorithms, release dates are integers generated as follows (Hall and Posner, 2001):

$$r_1 = 0 \text{ and } r_j = r_{j-1} + X_j,\ j = 2, ..., n$$

Where X_j is a random variable generated from a uniform distribution $U[0, 10]$. For each value of n, 10 instances were generated, producing a total of 30 problems.

The algorithms were coded in C++ on Microsoft Visual Studio and run on a $1,7$ GHz i5-8350U intel CPU.

Figures 3, 4 and 5 display the upper bounds on the competitive ratios resulted from running 10 instances on the algorithms SRPT, SPT, D-SPT, and the semi-online algorithm VD-SPT. Figure 3 shows results for the 10-jobs problems while Figures 4 and 5 show results for the 15 and 20-jobs problems, respectively. For the 10-jobs problem, it can be seen that the semi-online algorithm VD-SPT yields a better performance than the online one D-SPT, while in some cases such as instance 9 of Figure 3, the three algorithms have approximately the same performance. However, for the 15-jobs problem (Figure 4), we notice that for some instances D-SPT guarantees a slightly better performance than VD-SPT. Figure 5 shows that the performance of the four algorithms maintains relatively the same level, while for some instances such as instances 4 and 7, we can see that VD-SPT yields a better performance than D-SPT.

Overall, from Table 3, it can be seen that on average VD-SPT yields a better performance than D-SPT for 10, 15, and 20 jobs problems. Furthermore, For a small number of jobs we can notice a large difference in the performance of the three algorithms while the difference decreases when tested on bigger sized problems. It must be noted that approximately for all instances, SPT yields the best performance, which means that the strategy of not waiting proves better than other strategy when an average case study is conducted. This is due to the fact that online or semi-online algorithms are inserting waiting times in order to avoid worst case instances such as described in previous sections. Therefore, however tempting it is to apply SPT for its better performance on average, one worst case instance might make us regret the decision.

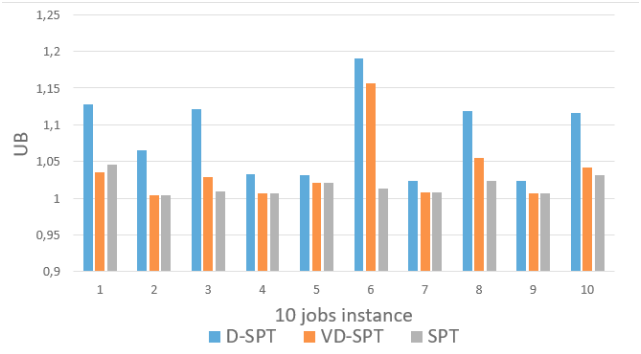

Fig. 3. Upper Bound on the competitive ratio for 10-jobs problem

3.2 Worst case analysis

In order to show the advantage of the proposed algorithm VD-SPT, a special instance is tested for which VD-SPT

Fig. 4. Upper Bound on the competitive ratio for 15-jobs problem

Fig. 5. Upper Bound on the competitive ratio for 20-jobs problem

Table 3. Mean UB over 10 instances

	10 jobs	15 jobs	20 jobs
VD-SPT	$1,03$	$1,02$	$1,02$
D-SPT	$1,08$	$1,05$	$1,03$
SPT	$1,01$	$1,01$	$1,01$

Table 4. Instance characteristics

Number of jobs	Release dates	Processing times
1	22	22
11	23	ϵ
1	45	45
7	46	ϵ
1	91	91
9	92	ϵ

algorithm presents a clear advantage. The instance is characterised by jobs with long processing requirements which are released followed by jobs with small processing requirements (Table 4). Considering that the D-SPT condition is verified for long jobs, i.e, $p_j \leq t$, the online algorithm decides to schedule immediately a long job once it arrives. This means that small jobs will have to wait until the long job is completed since preemption is not allowed (Figure 7). On the other hand, as shown in Figure 6, VD-SPT will choose to wait for small jobs to arrive since the condition of t_{max} is verified, which will give it the advantage on its online peer.

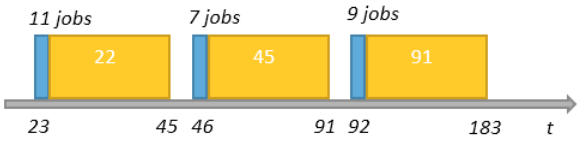

Fig. 6. Schedule by VD-SPT for worst case instance

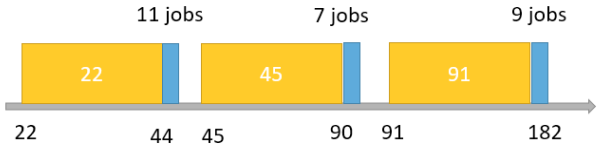

Fig. 7. Schedule by D-SPT for worst case instance

Figure 8 below shows the variation of the objective function values when the instance is tested on both the online and the semi-online algorithms and by considering that ϵ, the processing times of small jobs, tend to 0. We can see that D-SPT yields a very bad performance compared to VD-SPT. Moreover, the schedule showed in Figure 6 is the schedule constructed by VD-SPT for the considered instance, which also represents the optimal schedule in an offline setting. Therefore, for this type of instances the proposed semi-online algorithm VD-SPT presents a better performance than D-SPT by constructing a schedule close the optimal one.

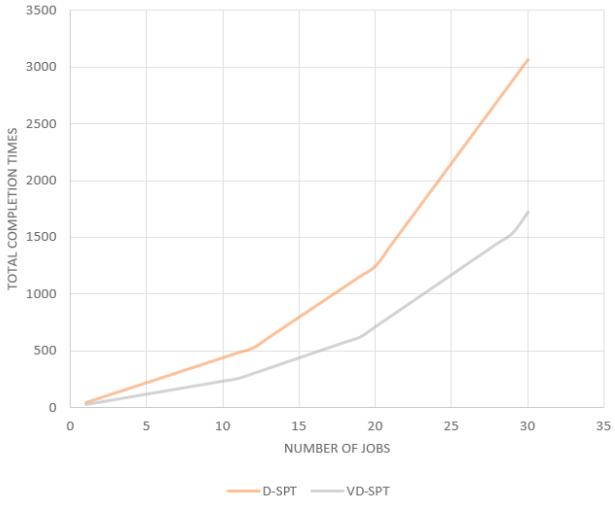

Fig. 8. Total completion time by VD-SPT and D-SPT for worst case instance

4. CONCLUSION

In this article, a semi-online algorithm called VD-SPT is presented where release dates are known at the beginning of the decision process. Furthermore, a numerical analysis is presented for average cases where instances are generated by a probability distribution, we saw that the performance of VD-SPT algorithm remains close to other algorithms while it yields slightly better performance compared to D-SPT in most cases. In addition, a worst case analysis is presented where VD-SPT and D-SPT algorithms were evaluated on a special instance and for which VD-SPT guarantees a clear advantage on its online peer. An interesting approach for future research would be to characterize the instances for which VD-SPT guarantees the best performance and to test the online and the semi-online algorithm on real-world data in order to evaluate their performances in practice. In addition, the semi-online scheduling problem on parallel machines where release dates are known in advance can also be considered.

REFERENCES

Albers, S. (2013). Recent advances for a classical scheduling problem. In *International Colloquium on Automata, Languages, and Programming*, 4–14. Springer.

Epstein, L. (2018). A survey on makespan minimization in semi-online environments. *Journal of Scheduling*, 21(3), 269–284.

Graham, R.L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9), 1563–1581.

Graham, R.L., Lawler, E.L., Lenstra, J.K., and Kan, A.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, volume 5, 287–326. Elsevier.

Hall, N.G. and Posner, M.E. (2001). Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49(6), 854–865.

Hall, N.G., Posner, M.E., and Potts, C.N. (2009). Online scheduling with known arrival times. *Mathematics of Operations Research*, 34(1), 92–102.

Hoogeveen, J.A. and Vestjens, A.P. (1996). Optimal on-line algorithms for single-machine scheduling. In *International Conference on Integer Programming and Combinatorial Optimization*, 404–414. Springer.

Lawler, E.L., Lenstra, J.K., Kan, A.H.R., and Shmoys, D.B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4, 445–522.

Lenstra, J.K., Rinnooy Kan, A.H., and Brucker, P. (1977). Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, 343–362. Elsevier.

Liu, M., Zheng, F., Wang, S., and Huo, J. (2012). Optimal algorithms for online single machine scheduling with deteriorating jobs. *Theoretical Computer Science*, 445, 75–81.

Mao, W., Kincaid, R.K., and Rifkin, A. (1995). Online algorithms for a single machine scheduling problem. In *The impact of Emerging Technologies on Computer Science and Operations Research*, 157–173. Springer.

Tan, Z. and Zhang, A. (2013). Online and semi-online scheduling. *Handbook of combinatorial optimization*, 2191–2252.

Tao, J., Chao, Z., and Xi, Y. (2009). A novel way to analyze competitive performance of online algorithms. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1.