Prod. Eng. Res. Devel.
DOI 10.1007/s11740-016-0680-9

COMPUTER AIDED ENGINEERING

Priority allocation rules for single machine total weighted linear and square tardiness problems

Aihua Yin[1] · Abraham P. Punnen[2] · Dongping Hu[1]

Received: 30 September 2015 / Accepted: 1 June 2016

Abstract The single machine scheduling problems minimizing total weighted tardiness and square tardiness objectives have been studied in literature for many years. Applications of the model include sequencing problems in manufacture and logistics. This paper proposes two new priority allocation rules, PAR 1 and PAR 2, for solving these two problems. Unlike most known dispatch rules and constructive algorithms, our new rules take advantage of not only the jobs' static characters values such as the process time, the due date and the weight, but also their dynamic characters values, i.e., the slack and the values of the objective function for different choices of some jobs. At any time when a job is being selected to process, some of the unprocessed jobs are delayed while the others are not. It means that the characters of these two sorts of jobs are different from each other. So, combining these characters with the objective function's value can obtain effective dispatch rule. Experimental analysis based on the instances from the OR-Library discloses that our priority allocation rules, PAR 1 and PAR 2, are efficient and have significant advantages over traditional approaches.

✉ Aihua Yin
aihuayin@jxufe.edu.cn

Abraham P. Punnen
apunnen@sfu.ca

Dongping Hu
hdp337@126.com

[1] School of Software and Communication Engineering, Jiangxi University of Finance and Economics, Nanchang 330032, China

[2] Department of Mathematics, Simon Fraser University, Surrey, BC V3T 0A3, Canada

Keywords Single machine scheduling · Priority allocation rule · Heuristic · Constructive algorithm

1 Introduction

This paper considers two single machine scheduling problems with the objective of minimizing the total weighted tardiness and the total squared tardiness [1, 2]. Formally, the problems can be defined stated as follows. Let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of n jobs to be processed on one machine. For each job J_i, $i = 1, 2, \ldots, n$, a processing time p_i, a due date d_i and a weight w_i are prescribed. At any moment, no more than one job is allowed to be processed and the operation of each job can not be preempted. The objective is to generate the start times of each job so that the total weighted tardiness (TWT) is minimized or the total weighted square tardiness (TWT^2). Let

$$TWT = \sum_{i=1}^{n} w_i T_i \tag{1}$$

$$TWT^2 = \sum_{i=1}^{n} w_i T_i^2 \tag{2}$$

where T_i is the tardiness of job i, which is $\max\{0, C_i - d_i\}$, and C_i is its competition time.

The single machine scheduling problems TWT and TWT^2 appear in a variety of practical situations. For example, in real world production system such as semiconductor wafer fabrication facility and logistics engineering, the single machine total weighted tardiness and square tardiness problems are important [3]. Scheduling models with a single machine are also applicable for problems with multiple processing stages. In fact, the performance of many production systems is often determined by the quality of the

Published online: 11 June 2016

schedules for a single machine. Furthermore, the results and insights obtained from these single machine problems can be applied to more complex scheduling problems, such as flow shop and job shop problems.

The single machine total weighted tardiness and squared tardiness problems are known to be NP-hard [4]. Several exact and heuristic algorithms are available in literature for solving the single machine total weighted tardiness problems.

Exact algorithms are guaranteed to produce an optimal solution for all instances of the problem, but the running time could be prohibitively large. Hence, exact algorithms are not used in practice when the problem size is very large. The branch and bound algorithm proposed by Potts and Van Wassenhove [5] can solve problem instances with up to 50 jobs in reasonable computation time. The advancement of computers and general purpose integer programming solvers enhanced our capability of solving problems of reasonably large size, heuristic algorithms are still a valuable tool, especially when quick solutions are desired.

Construction heuristics are among the first class of heuristic algorithms proposed for the single machine total weighted tardiness problem. Construction heuristics generate a schedule efficiently in n iterations. Due to the wide gap between the solution generated by them and the optimal solution, constructive heuristics are often adopted to generate initial schedules for more complex improvement heuristics. Well-known construction heuristics for *TWT* include Weighted Shortest Processing Time (WSPT) [1], Apparent Tardiness Cost (ATC) [6], Weighted Modified Due Date (WMDD) [7]. Recently, Valente and Schaller published efficient dispatching heuristics for quadratic tardiness scheduling problems [8, 9].

Local search algorithms presented by Avci et al. [10] and Ergun and Orlin [11] can produce a much better solutions than construction heuristics by employing additional computational time. Local search algorithms start from an initial solution, which is usually generated by a construction algorithm, and then iteratively generate a sequence of improved solution. Some meta-heuristics such as tabu search discussed by Bilge et al. [12], and simulated annealing presented by Nearchou [13] are improvements over local search algorithms.

This paper is organized as follows. In Sect. 2 we describe two effective constructive heuristic algorithms, PAR 1 and PAR 2. Section 3 deals with experimental analysis of our algorithms followed by concluding remarks in Sect. 4.

2 The priority allocation rules

To process all the jobs, a priority allocation rule or a construction algorithm usually processes the jobs one by one. Initially, all the jobs are not processed, and then jobs are selected one-by-one to process using appropriate selection rules. By the kth state, k jobs are already processed while n–k jobs are still waiting in the job queue. The process terminates when all jobs processed. More specifically, the selection of jobs to be processed is identified specific characteristics, called patterns, of the jobs. Each of the patterns of interest to us can be uniquely identified by three elements, i.e., the processed job sequence, the set of unprocessed jobs and the current time t meaning that the earliest start time for the next job to be processed. Figure 1 shows the second pattern of an instance of seven jobs. After the job 2 and the job 6 have been processed, the value of the current time t becomes $p_2 + p_6$.

So, a pattern can be noted by a three tuple, an ordered subset S of J with the jobs already processed, the complement $S^* = J \backslash S$ of S in J, and the current time t. Thus, a pattern is described as follows.

$$\theta = (S, S^*, t) \tag{3}$$

where, the value of the current time t is the sum of process time of the jobs in S.

Generally, the process from a certain current pattern θ_{cur} to a new one θ_{new}, needs three steps. First, the job with highest priority level in the set S^* is selected by some heuristic rules or by some function value, such as, the objective function of the problem. Then, after this selected job is processed it is moved to the set S. Last, the features of all those jobs in set S^* (if they exist), such as, whether a job is tardy, and the starting processing time for the next selected job, are updated. By this way, while the last job is processed, a schedule is obtained. So, a basic priority allocation rule can be described as follow.

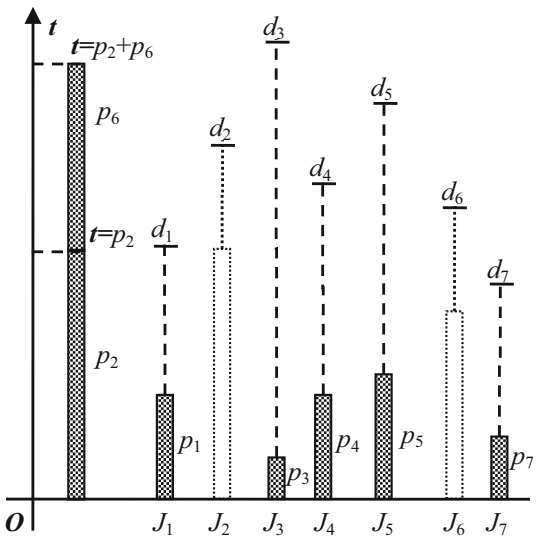

Fig. 1 The second pattern of an instance with seven jobs

Priority allocation rule (PAR)

*Step*1. Select a job with highest priority level from S^*.
*Step*2. Move this job from S^* to S, update the current time t and the jobs' features in S^*.
*Step*3. If $S^* = \phi$, stop. Otherwise, return to *Step*1.

For any instance, while the jobs are processed one by one, it is not difficult to divide all the patterns from the initial moment, $t = 0$, to a schedule into three classes, no matter how many jobs are there and what the jobs characteristics are, such as, p_j, d_j and w_j ($j = 1, 2, \ldots, n$). Furthermore, as the next job is being selected to process, it is easy to identify the current pattern belonging to which class by calculating the slacks $d_i - p_i - t$ for all the jobs J_i in S^*. The specifics are elaborated below:

Class I: Each pattern has no tardy unprocessed job, i.e., in any of these patterns, the job in S^* has nonnegative slack.
Class II: Each pattern has both tardy and non-tardy unprocessed jobs.
Class III: Each pattern has all unprocessed job are tardy, i.e., in any of these patterns, all of the jobs in S^* have negative slack.

Based on the partition of the patterns into classes *I*, *II* and *III*, it can be verified that a pattern in one class can be followed by a new pattern in another class. Preciously, a pattern in the *Class I* or *Class II* can be followed by a new pattern in any another class, but a pattern in the *Class III* can only be followed by a new pattern in the *Class III*. Figure 2 shows this relationship between the patterns in which a directed arc means a pattern in one class can be followed by another pattern in another class, such as from the *Class I* or *Class II* to the *Class III*. The second fact is that the last pattern belongs to either the *Class I* or the *Class III*.

So, a pattern has its fourth element, denoted by C, showing which class the pattern belongs to. Obviously, the value of C belongs to $\{I, II, III\}$. Adding this fourth element to our notation of a pattern, we denote it by Θ. Then,

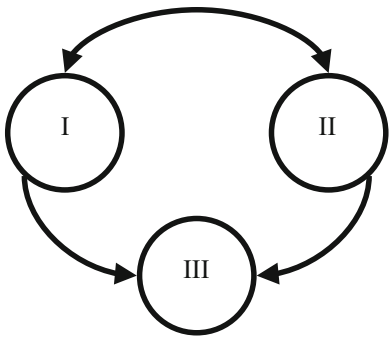

Fig. 2 The relationship between the patterns in different classes

$$\Theta = (S, S^*, t, C) \tag{4}$$

Thus, in the example above, there are $\Theta_0 = (\phi, J, 0, I)$, $\Theta_1 = (\{J_2\}, \{J_1, J_3, J_4, J_5, J_6, J_7\}, p_2, II)$ and $\Theta_2 = (\{J_2, J_6\}, \{J_1, J_3, J_4, J_5, J_7\}, p_2 + p_6, I)$, where $\Theta_0(C) = \Theta_2(C) = I$ and $\Theta_1(C) = II$.

In the literature on priority rules or construction algorithms [2, 3, 6], the authors adapted different effective rules to direct one pattern to next one. All these rules take the job's characteristics into account, such as d, p and w. In this paper, we not only take into account the job's characteristics but also the objective function's value, as well.

In detail, if a pattern is in the *Class I*, the job with the smallest due date is selected to be processed, and if a pattern is in the *Class II*, two jobs are selected by some rule and then their incremental objective value is compared and the smaller is selected to be processed first. And if a pattern is in the *Class III*, the Weighted Shortest Processing Time rule is certainly adapted, c.f. the theorem of Smith [1].

To describe our algorithm precisely, we first let $JUD = \{J_k | \ d_k \geq t + p_k, J_k \in J\}$, be the non-tardy job set, and $JD = \{J_k | \ d_k < t + p_k, J_k \in J\}$, be the tardy job set. Let t be the current time of schedule, so $t = 0$ at the initial time. Then the first version of our algorithm, priority allocation rule 1 (PAR 1) is described as follows.

PAR 1

*Step*1. If $JUD \neq \phi$, $JD = \phi$, select $J_i \in JUD$ with the smallest due date. Go to *Step*5.
*Step*2. If $JUD = \phi$, $JD \neq \phi$, select $J_i \in J$ with the smallest p_i/w_i. Go to *Step*5.
*Step*3. If $JUD \neq \phi$, $JD \neq \phi$, determine $J_i \in JUD$ with the smallest due date and $J_j \in JD$ with the smallest p_j/w_j. Then Calculate $w_j \cdot (t + p_i + p_j - d_j)$ and $w_j \cdot (t + p_j - d_j) + w_i \cdot \max\{t + p_j + p_i - d_i, 0\}$.
*Step*4. If $w_i \cdot (t + p_i + p_j - d_j) \geq w_j \cdot (t + p_j - d_j) + w_i \cdot \max\{t + p_j + p_i - d_i, 0\}$, select $J_j \in JD$. Otherwise, select $J_i \in JUD$.
*Step*5. Let the selected job be J_l, set $t = t + p_l$. Set $J = J \backslash \{J_l\}$.
*Step*6. If $J = \phi$, stop. Otherwise, go to *Step*1.

The complexity of PAR 1 is $O(n^2)$, because the complexities of each of the steps of *Step*1, *Step*2 and *Step*3 is $O(n)$, which selects just one job each time. Furthermore, both the job's characteristics and the objective function's value are taken into consideration, and the computing test shows that PAR 1 is effective.

PAR 1 compares the two increments of the objective function caused by two different pre-sequences respectively. However, if some dynamic features, such as, the slacks of the jobs are considered, this strategy can easily be modified. So, we get the second version of our rule.

If $\Theta(C) = I$, then, the job with the smallest $(d_i - t)/w_i$ is selected to be processed.
If $\Theta(C) = II$, then, the two jobs with the smallest $(d_i - t)/w_i$ and p_j/w_j respectively are selected, and then their incremental is compared to the objective function's value, finally the job with the smaller incremental is selected to being processed.
If $\Theta(C) = III$, then, the Weighted Shortest Processing Time (WSPT) rule is certainly adapted. And the priority allocation rule 2 is described as follows in detail.

PAR 2

*Step*1. If $JUD \neq \phi$, $JD = \phi$, select $J_i \in JUD$ with the smallest slack, $(d_i - t)/w_i$. Go to *Step*5.
*Step*2. If $JUD = \phi$, $JD \neq \phi$, select $J_i \in J$ with the smallest p_i/w_i. Go to *Step*5.
*Step*3. If $JUD \neq \phi$, $JD \neq \phi$, $J_i \in JUD$ with the smallest slack, $(d_i - t)/w_i$ and $J_j \in JD$ with the smallest p_j/w_j. Calculate $w_j \cdot (t + p_i + p_j - d_j)$ and $w_j \cdot (t + p_j - d_j) + w_i \cdot \max\{t + p_j + p_i - d_i, 0\}$.
*Step*4–6. are the same with PAR 1.

It is obvious that the complexity of PAR 2 is also $O(n^2)$, because it does not add the search of the jobs in J. In the next section, the efficiency is shown by computing experiments.

3 Computing experiments

The algorithms PAR 1 and PAR 2 are tested on benchmark instances presented in OR-Library [14] (http://logistik.hs-uhh.de/SMTWTP). The tested benchmark instances are in the following three sets (a), (b) and (c). The proposed algorithms are implemented by using C#.NET 2005. Numerical experiments are conducted on a PC with two Intel Core TM2 T6670 @ 2.2 GHz processors and 2 GM RAM.

(a) 125 benchmark instances with exactly 40 jobs for each of them.
(b) 125 benchmark instances with exactly 50 jobs for each of them.
(c) 125 benchmark instances with exactly 100 jobs for each of them.

These instances are randomly generated as follows. For each job J_i $(i = 1, 2, \ldots, n)$, an integer processing time p_i is generated from the uniform distribution [1, 100] and an integer processing weight w_i is generated from the uniform distribution [1, 10]. Instance classes of varying hardness are generated by using different uniform distributions for generating the due dates. For a given relative range of due dates RDD (RDD = 0.2, 0.4, 0.6, 0.8, 1.0) and a given average tardiness factor TF (TF = 0.2, 0.4, 0.6, 0.8, 1.0), an integer due date d_i for each job J_i is randomly generated from the uniform distribution

$$[\mathrm{P}(1 - \mathrm{TF} - \mathrm{RDD}/2),\ \mathrm{P}(1 - \mathrm{TF} + \mathrm{RDD}/2)] \quad (5)$$

where $\mathrm{P} = \Sigma p(i)$ $(i = 1, \ldots, n)$. Five instances are generated for each of the 25 pairs of values of RDD and TF, yielding 125 instances for each value of n.

For the two problems in (1) and (2), the priority allocation rules, PAR 1 and PAR 2, are compared with other two effective constructive heuristic algorithms named as WMDD [7] and ATC [6], which are among the best constructive algorithms up to now. Results are reported in Tables 1 and 2, respectively. The gap (%) is expressed as $100 \cdot (V_{CHA} - V_{OPT})/V_{OPT}$ or $100 \cdot (V_{CHA} - V_{\mathrm{PAR}\ 2})/V_{\mathrm{PAR}\ 2}$, where V_{CHA} denotes the value of the feasible solution that is generated by either one of the heuristics and V_{OPT} denotes either the value of the optimal solution (if applicable) or the value of the best-known solution in the OR library, and the mean of $V_{\mathrm{PAR}\ 2}$ is obvious. NP denotes the number of problems (out of 25) where the associated heuristic finds an optimal solution. If the denominator, V_{OPT}, is 0, the gap (%) cannot be calculated. Hence, this paper does not use problem instances if the value of the optimal solution is zero.

For each set of benchmark instances, PAR 1 and PAR 2 have got the best performance among the four algorithms. However, PAR 1 and PAR 2 get the most times both in average gap (%) and in NP, even when TF = 0.2 it overwhelms the other three algorithms. Especially, PAR 1 and PAR 2 have obtained the optimal solution (the *TWT* value is 123,893) of the 44th instance in (b) for the first time as a constructive algorithm. Furthermore, based on the Tables 1 and 2, it is easy to find that PAR 1 and PAR 2 dominate each other in the problem (1) and (2), respectively.

The results in [8, 9] are based on the data generated by their own, and the method used to generate the problems was the same as the one that was used to create the linear weighted tardiness problem instances available in [14]. Since we can not get those data, we do not compare our rules with theirs.

4 Conclusions

In this paper, we presented two effective priority allocation rules for the single machine total weighted tardiness and squared tardiness problem. Our algorithms have been tested on 375 benchmark instances which are available in OR-Library. Comparisons with other classic constructive heuristic algorithm show that these two constructive heuristic algorithms are effective. For further research work, some other priority rules can be used when all the

Table 1 Comparison with the other two effective dispatching rules for the problem $\sum w_i T_i$

TF	Jobs	ATC		WMDD		PAR 1		PAR 2	
		Average gap	NP	Average gap	NP	Average gap	NP	Average gap	NP
0.2	40	0.404	0	0.558	0	0.153	**4**	**0.14**	**4**
	50	0.259	0	0.641	0	0.141	0	**0.114**	**1**
	100	0.344	0	**0.085**	0	0.156	0	0.139	
0.4	40	12.105	0	1.389	0	1.144	1	**0.901**	**4**
	50	175.106	0	14.728	0	10.797	2	**8.171**	**4**
	100	107.154	0	6.364	0	6.008	0	**3.877**	
0.6	40	1.033	0	0.497	0	0.184	0	**0.127**	**2**
	50	1.275	0	0.572	0	0.235	0	**0.183**	
	100	1.410	0	**0.061**	0	0.226	0	0.193	
0.8	40	7.236	0	1.631	0	0.389	3	**0.294**	**4**
	50	4.262	0	0.754	0	0.245	0	**0.137**	
	100	46.037	0	0.565	0	0.458	0	**0.330**	
1.0	40	2.072	0	0.977	0	0.058	0	**0.029**	**1**
	50	2.128	0	0.847	0	0.230	0	**0.119**	
	100	23.353	0	0.312	0	0.146	0	**0.120**	

Average gap $= \sum$ (compt $-$ Opt)/25, where, compt means the computing results of the corresponding algorithms

The numbers highlighted in bold mark the best value among the four values in the same line

Table 2 Comparison with the other two effective dispatching rules for the problem $\sum w_i T_i^2$

TF	Jobs	(ATC–PAR 2)/PAR 2 Average gap	(WMDD–PAR 2)/PAR 2 Average gap	(PAR 1–PAR 2)/PAR 2 Average gap
0.2	40	10.10	19.47	0.19
	50	7.80	19.74	**−0.30**
	100	11.28	**−0.26**	0.02
0.4	40	0.24	572.10	20.68
	50	384.36	47.23	**−0.06**
	100	3283	28.93	**−0.14**
0.6	40	1611	77.68	**−0.72**
	50	12,098	31.86	**−1.06**
	100	123,037	**−0.05**	**−0.53**
0.8	40	1,077,841	240.41	**−0.79**
	50	634.93	59.60	**−0.47**
	100	12,116	33.91	**−0.63**
1.0	40	43,114	732.45	**−0.53**
	50	325,568	238.16	**−0.08**
	100	77,066	460.01	**−0.57**

Average gap $= \sum$ (compt $-$ PAR 2)/25, where, compt means the computing results of the corresponding algorithms

The negative numbers highlighted in bold mark the computing result of the corresponding algorithm is optimal for PAR 2

jobs are not delayed. For example, a job selected from the two jobs, one with the smallest due date and another with the smallest weight due date. Furthermore, the concept of partition of patterns can be used in some local search algorithms, such as, Tabu search or evaluation computation, to get more effective neighborhood structure or generation characteristics.

Acknowledgments This research is supported by the National Natural Science Foundation of China No. 61262011 and the Natural Science Foundation of Jiangxi Province No. 20142BAB207024.

References

1. Smith WE (1956) Various optimizers for single-stage production. Naval Res Logist Q 3:59–66
2. Townsend W (1978) The single machine problem with quadratic penalty function of completion times: a branch-and-bound solution. Manag Sci 24(5):530–534
3. Mason SJ, Fowler JW, Carlyle WM (2002) A modified shifting bottleneck heuristic for minimizing total weighted tardiness in complex job shops. J Sched 5:247–262
4. Lenstra JK, Rinnoy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann Discret Math 1:343–362
5. Potts CN, Van Wassenhove LN (1985) A branch and bound algorithm for total weighted tardiness problem. Oper Res 33:363–377
6. Vepsalainen AP, Morton TE (1987) Priority rules for job shops with weighted tardiness costs. Manag Sci 33:1035–1047
7. Kanet JJ, Li X (2004) A weighted modified due date rule for sequencing to minimize weighted tardiness. J Sched 7:261–276
8. Valente Jorge MS, Schaller Jeffrey E (2012) Dispatching heuristics for the single machine weighted quadratic tardiness scheduling problem. Comput Oper Res 39:2223–2231
9. Schaller Jeffrey E, Valente Jorge MS (2012) Minimizing the weighted sum of squared tardiness on a single machine. Comput Oper Res 39:919–928
10. Avci S, Akturk MS, Storer RH (2003) A problem space algorithm for single machine weighted tardiness problems. IIE Trans 35:479–486
11. Ergun Ö, Orlin JB (2006) Fast neighborhood search for the single machine total weighted tardiness problem. Oper Res Lett 34:41–45
12. Bilge Ü, Kurtulan M, Kıraç F (2007) A tabu search algorithm for the single machine total weighted tardiness problem. Eur J Oper Res 176:1423–1435
13. Nearchou AC (2004) Solving the single machine total weighted tardiness scheduling problem using a hybrid simulated annealing algorithm. In: Proceedings of the 2nd IEEE international conference on industrial informatics INDIN'04, Berlin, Germany, pp 513–516
14. Geiger M (2011) New instances for the single machine total weighted tardiness problem. http://edoc.sub.unihmburg.de/hsu/volltexte/2011/2808/pdf/RR-10-03-01.pdf