ITC 1/50

Information Technology and Control
Vol. 50 / No. 1 / 2021
pp. 5-12
DOI 10.5755/j01.itc.50.1.25531

Max-Min Processors Scheduling

Received 2020/03/19 | Accepted after revision 2021/02/18

http://dx.doi.org/10.5755/j01.itc.50.1.25531

HOW TO CITE: Alquhayz, H., Jemmali, M. (2021). Max-Min Processors Scheduling. *Information Technology and Control*, 50(1), 5-12. https://doi.org/10.5755/j01.itc.50.1.25531

Max-Min Processors Scheduling

Hani Alquhayz

Department of Computer Science and Information, College of Science at Zulfi, Majmaah University, Al-Majmaah 11952, Saudi Arabia

Mahdi Jemmali

Department of Computer Science and Information, College of Science at Zulfi, Majmaah University, Al-Majmaah 11952, Saudi Arabia

Mars Laboratory, University of Sousse, Tunisia

Department of Computer Science, Higher Institute of Computer Science and Mathematics of Monastir, University of Monastir, Monastir, 5000, Tunisia

Corresponding author: m.jemmali@mu.edu.sa

This study focuses on the maximization of the minimum completion time using identical parallel processors. The objective of this maximization is to ensure fair distribution. A set of processes is to be scheduled to several identical parallel processors, and this problem is proved as NP-hard. The research for this paper is based primarily on the performance of the proposed heuristics with other methods cited in the literature review. Our heuristics are developed mainly using the randomization method and the iterative utilization of the knapsack problem to solve the above-mentioned problem. The heuristics are assessed by several instances represented in the experimental results. The results shew that the knapsack-based heuristic provides a performance that is almost like the heuristic in the literature review but with better running time.

KEYWORDS: Parallel processors; algorithms; heuristic; knapsack problem.

1. Introduction

Nowadays, it is crucial to reduce manufacturing costs in companies across the world. The machines used in manufactories comprise rare resources because, in general, machines are costly. Thus, the proper distribution of processes is essential from an industrial and financial point of view.

Suitable scheduling that reduces costs in the industrial sectors can help managers make decisions. There is a need to have a system that provides data and information. Several heuristics algorithms were compared for this study, while the first heuristics were based on the randomized algorithms, and the second was based on meta-heuristics. The presented work is focused on improved methods for scheduling problems, which provides an extension to help decision-analytics using methods that help organizations and firms. The scheduling problem has several applications in manufacturing, networks, communication, budgeting, and many more fields. The solutions to these problems affect everyday activities and managers with relevant effects on the public and private domains and largely on all society. Various applications of the studied problem and their scheduling in several areas have been studied. The scheduling problems were used, and a presentation of a wide variety of the distribution of resource models by assigning a mathematical formulation and model to compute optimal solutions to these models was formulated.

In our study, we focus on the maximization of the minimum load machines for the identical parallel machines problem. In several works, researchers use the term "machine covering problem" to describe the same problem. Deuermeyer et al. [5] introduced the machine covering problem. Several definitions for the studied problem and results are described [13].

Different industrial applications implement the solution of the machine covering problem to reduce costs. Moreover, the application of the solution to gas turbine aircraft engine maintenance is developed, and some proposed solutions based on mathematical modeling is presented [9]. Additionally, this research presents the results of the implementation coded in Cplex.

The literature review of the studied problem is not extensive. However, some works related to the covering machine are examined in several cases.

Semi-online scheduling for identical parallel machines and the study of the machine covering problems is presented [22]. Notably, in the latter work an optimal solution with several semi-online versions was developed.

Jiang et al. [17] shew that the offline scheduling version can provide a solution in $O(mn)$. In addition, the latter research shews that the ratio measuring the competitiveness of a randomized online method has several fixed values, for m-uniform-machine and m-identical machine problems.

Other researches articulated the machine covering problem but only on two uniform machines [4], [12], [20] and [21]. Among the more recent works, a specific problem regarding the preemptive scheduling in the case of the semi-online problem was developed [12]. In this work, an exact solution using an algorithm for the semi-online case for every machine fixed speed ratio of s was developed. In addition, the latter study has shown that an idle time must exist when the procedure of the algorithms was assigned for any bounded s.

An improvement of the $(2+\varepsilon)$-competitive algorithm with constant reassignment factor was developed [19]. The main result for the latter work is that for any $\varepsilon>0$, one can maintain a $(1+\varepsilon)$-competitive solution for several constant rescheduled factors $r(\varepsilon)$.

Wu et al. [24] focused their research on the machine covering problems on two hierarchical machines, with added constraints, such as tasks that are correspondingly grouped into two hierarchical groups.

A random method was presented for the online problem with a running time of $O(\sqrt{mlnm})$ [3]. A deterministic algorithm enhanced this recent work with a competitive ratio of $11/6 \leq 1.834$ [6].

Machine covering was also applied with partial information on identical parallel machines [25].

Gálvez et al. [7] presented a theorem that bounded the migration factor for the online algorithms for the machine covering problem with migration. In this context, the latter research shew that there exists, for any $\varepsilon>0$, $(4/3+\varepsilon)$-competitive algorithm with a migration factor $O(1/\varepsilon^5)$, and an approximation ratio of the local search algorithm in the interval [1.691,1.75] .

Gerke et al. [8] considered a specific problem based on a stochastic variant of the Santa Claus problem.

Recently Walter et al. [23] proposed a new exact algorithm based on the branch-and-bound algorithm to solve the problem of maximizing the minimum. Several enhanced lower bounds and heuristics were also developed in this work. In addition, in the latter work, a comparison study of the results given [11] was developed. The studied problem has several applications in our real-life and is practical. Our study is based on the mathematical modeling of two new lower bounds. The model utilizes the randomized method and the iterative solution of several subset problems.

Recently, several studies have focused on the fair distribution [1], [2], [14], [15] and [16].

In this study, we organize the work as follows. In Section 2, we describe the studied problem with examples and provide variable definitions and notations that will be used in the study. The proposed heuristics are presented in Section 3. Section 4 is devoted to the experimental results that shew a comparison between the proposed heuristics and those cited in the literature review. Finally, a conclusion is provided in the last Section.

2. Problem Definition

In this Section, the definition is presented of the studied problem. This problem can be described as follows. Denoted by J, the set of n independent processes will be scheduled on p identical parallel processors represented by the set $\{pr_p,...,pr_p\}$. Each process j is defined by its processing time, which is denoted by p_j. All processes have positive processing time $\{p_1,\dots,p_n\}$. C_i^p denotes the load of the i^{th} processor. The load of a processor is calculated by the summation of all the processing times corresponding to the processes scheduled on the processor. Let C_j be the time when the process j finishes its execution. The minimum completion time is denoted by C_{min}. The goal is to determine a suitable schedule that maximizing C_{min}. The problem is denoted by $P||C_{min}$ using the notation described in [10]. This problem has an inextinguishable theoretical interest because the impact of the application in real-life. Indeed, the studied problem van be applied on different domain of application (financial, industrial, computer science, aircraft, health care, railway, etc.) In this work, we present some heuristics to provide a solution to the problem.

Example 1. Let $n=6$ and $p=2$. We display the processing time for each process in Table 1.

Figure 1 presents a schedule to assign the processes on the processors.

Table 1

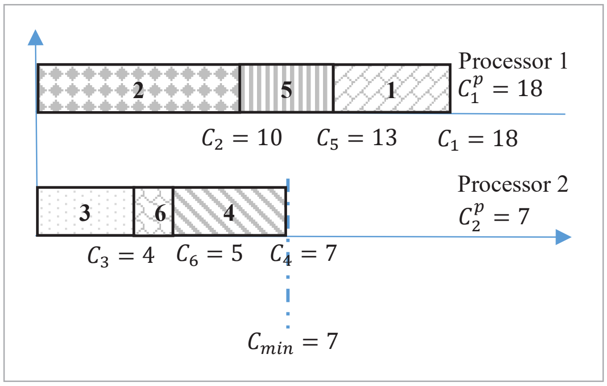
Processing time of 6-processes

j	1	2	3	4	5	6
p_j	5	10	4	2	3	1

From Figure 1, we observe that the minimum completion time C_{min} on the processors is 7. The objective is to search a schedule that maximizes the obtained C_{min}. Here for this example the gap between the first processor and the second one is *18-7=11*.

Figure 1

6-2 processes-processors processing time distribution

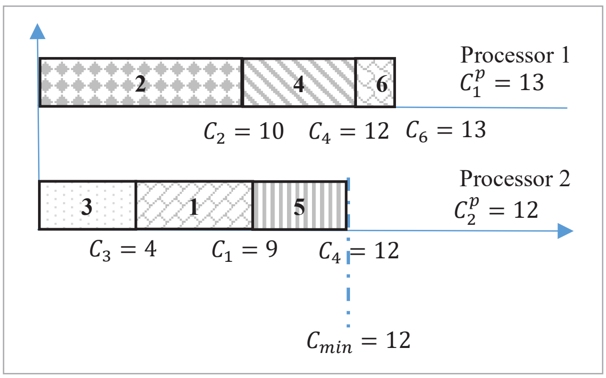


Applying another algorithm can minimize the gap. Let the following schedule (Figure 2) obtained for the same instance in Table 1.

In Figure 2, the C_{min}=*12*. Thus, the gap between the two processors is 13-12=1. Comparing with schedule illustrated in Figure 1, the new schedule is better and give a minimum gap by maximizing the minimum completion time.

Figure 2

Maxi-min schedule



3. Heuristics

In this work, several heuristics will be presented. The first one is based on the iterative probabilistic method, and the second is based on the repeating resolution of the subset problems generated from two processor problems.

This work compares the *LPT* rule [11] and the proposed heuristics.

3.1. Iterative Randomized Heuristic (*IR*)

The *LPT* rule is based on the ordering of all processes in the non-increasing order of their processing time. After that, we schedule the first longest process on the most available processor, and so on. The idea of the proposed heuristic is to extend the selection of the longest process. This implies that we do not select the longest process, but randomly select a process between the two longest processes. A generalization of this idea is to iterate the procedure multiple times. Indeed, the selected process is chosen among k processes having the longest processing time. The choice of the process is given a probability α.

In practice, the probability is calculated as follows. A random number r will be chosen between 1 and k. The selected process will be the process that has the r^{th} longest processing time among the unscheduled processes. When the number of unscheduled processes n_u is less than r, we choose r randomly between 1 and n_u.

The iterative randomized heuristic is displayed in Algorithm 1.

Algorithm 1: Iterative randomized heuristic algorithm

***Step* 0**	Set $it=1, k=2$.
***Step* 1**	$J_k=J$.
***Step* 2**	$R=random[1-min(k,/J_k/)]$.
***Step* 3**	Assign the r^{th} longest process L_r to the most available processor.
***Step* 4**	$J_k=J_k \backslash L_r$, if $J_k \neq \varnothing$ goto ***Step* 2**
***Step* 5**	Calculate $C_{\min}^{k,it}$.
***Step* 6**	$it=it+1$
***Step* 7**	If $it<lim$ go to ***Step* 1**
***Step* 8**	$C_{\min}^{k} = \max_{1 \le it \le 1000} C_{\min}^{k,it}$
***Step* 9**	$k=k+1$, if $k<7$ then $it=1$ and go to ***Step* 1**
***Step* 10**	$C_{\min} = \max_{1 \le k \le 5} C_{\min}^{k}$
***Step* 11**	Return C_{min}. **Stop**.

3.2. Iterative Knapsack Problem Heuristic (*IK*)

This heuristic is articulated primarily for the given idea. An upper bound was calculated for the studied problem denoted by *UB*. After that, we solve a knapsack problem searching the assignment of the maximum processes on the first processor, not reaching the *UB* value. The resolution of the knapsack problem will give as a set of processes that will be scheduled for the first processor. Now, we apply the same procedure to the remaining processes and search for the processes that will be assigned to the second processor, and so on, until we assign all the given processes.

For this heuristic, a greedy iterative method is adopted to solve a knapsack (*KS*) family problem $KS(l)$ with $l=\{0,...,p-1\}$.

$$KS(l): \left\{ \begin{array}{l} Z_l = \max \sum_{j \in J_l} p_j y_j \\ s.t: \sum_{j \in J_l} w_j y_j \le UB(J_l, p-l) \\ y_j \in \{0,1\}, \forall j \in J_l \end{array} \right\}$$

where:

- $J_1=J$ and $J_{l+1}=J_l \backslash O_l$, where O_l is an optimal set given by $KS(l)$.
- $w_j = \dfrac{|J_l|}{p-l} p_j - 1$.
- $UB(O,k)$ is an upper bound for the $P||C_{min}$ the problem of a reduced instance defined on $k \le p$ processors and a subset of processes O $O \subseteq J$. In practice, we choose the trivial upper bound.

Consequently, the algorithm for the iterative knapsack heuristic begins by solving $KS(O)$ to determine a subset of processes J_0 where total processing time is maximal but not more significant than the value of an upper bound. These processes will be scheduled on the first processor. Then, the algorithm computes an upper bound on the remaining processes and processors. This is mean that the new problem will be defined by $p-1$ processors and process-set $J \backslash J_1$. Now, the knapsack problem is solved to determine an optimal subset of processes that will be assigned to a second processor, and so on.

The processes that belong to $Op=J_{p-1} \backslash O_{p-1}$ are scheduled on the p^{th} processor. Since *KS* is an NP-hard, we utilize, in our algorithm, the pseudo-code developed in [18], which is based on the resolution of the problem in pseudo-polynomial time using dynamic programming. The iterative knapsack heuristic is displayed in Algorithm 2.

Algorithm 2: Iterative knapsack heuristic

***Step* 0**	Initialize $l=0$, $J_0=J$.
***Step* 1**	**For** $l=0$ **to** $p-1$ **do**
***Step* 2**	$u=UB(J_l,p-l)$
***Step* 3**	$Z_l=KS(l)$, O_l is the list returned by $KS(l)$.
***Step* 4**	Schedule O_l on processor pr_{l+1}
***Step* 5**	$J_{l+1}=J_l \setminus O_l$
	End For
***Step* 6**	Calculate C_{min} value.
***Step* 7**	Return C_{min}. ***Stop***.

Algorithm 2, given above, shows that the iteration while solving several knapsack problems until we schedule all processes on the processors. In practice, we choose the trivial upper bound U_0 cited in [11].

4. Experimental Results

After the written algorithms to give a lower bound of the studied problems, we show the results returned by the developed heuristics with a statistic analysis. These algorithms were coded using Microsoft Visual C++, then executed on an Intel(R) Core (TM) i7-3337U CPU @ 1.8GHz and 8GB RAM. A set of instances were being generated to test the proposed lower bounds. We list the generating manner of instances from several classes described in [11] and [23]. Indeed, the processing time p_j was generated based on two distributions.

The first distribution is the uniform one (U), and the second is the normal one (N). The classes are:

- Class 1: $p_j \in U[1,100]$.
- Class 2: $pj \in U[20,300]$.
- Class 3: $pj \in U[5,100]$.
- Class 4: $pj \in N[50,100]$.
- Class 5: $pj \in N[20,100]$.

The choice of n,p and Class will fix the number of generated instances. Therefore, the pair (n,p) has several possibilities, as given in Table 2.

Table 2
Generation of (n,p)

n	p
10	2,3,5
20,50	2,3,5,10,15
100,250,500,1000,2500,5000,10000	3,5,10,15

For each triplet ($n,p,Class$), we generate ten instances of the processing time.

In Table 2, the total number of instances is 2050 instances. To measure the performance of heuristics, we must define some metrics as follows:

- LB: the best heuristic value obtained after running of all lower bounds.
- L: the studied heuristic.
- Max: the number of instances when $L=LB$.
- $Gap = \frac{L-LB}{L}100$
- $Agap$: the average Gap .

$Time$: the spent time to run heuristic in seconds, and we denote by "-" if the time is less than 0.001 s.

In Table 3, we present the average of the indicators Max, $Perc$, $Agap$ and $Time$, for each heuristic over the 2050 instances. As shown in the table, the best heuristic is MSS having 85.3% and an average time 0.340 s compared with IK which has 78.6% and an average time of 0.004s. The advantage of the proposed heuristic IK is that it is faster than MSS, and there exists only a 6.7% difference between them in terms of $Perc$.

Table 3
Overall heuristics results in comparison

	LPT	*MSS*	*IK*	*IR*
Max	838	2046	1886	1294
Perc	34.9%	85.3%	78.6%	53.9%
Agap	0.01	0.00	0.01	0.00
Time	0.000	0.340	0.004	10.928

Table 4
Gap and *Time* variation according to *n*

n	*LPT*		*MSS*		*IK*		*IR*		*Total*	
	Gap	*Time*	*Gap*	*Time*	*Gap*	*Time*	*Gap*	*Time*	*Gap*	*Time*
10	0.02	0.000	0.00	0.002	0.02	0.000	0.00	0.002	0.01	0.001
20	0.02	0.000	0.00	0.004	0.03	0.001	0.01	0.007	0.01	0.003
50	0.02	0.000	0.00	0.005	0.00	0.001	0.01	0.014	0.01	0.005
100	0.01	0.000	0.00	0.006	0.00	0.001	0.01	0.036	0.00	0.011
250	0.00	0.000	0.00	0.012	0.00	0.001	0.00	0.131	0.00	0.036
500	0.00	0.000	0.00	0.033	0.00	0.002	0.00	0.364	0.00	0.100
1000	0.00	0.000	0.00	0.093	0.00	0.003	0.00	1.115	0.00	0.303
2500	0.00	0.000	0.00	0.344	0.00	0.006	0.00	5.673	0.00	1.506
5000	0.00	0.000	0.00	0.637	0.00	0.011	0.00	20.034	0.00	5.171
10000	0.00	0.000	0.00	2.297	0.00	0.021	0.00	84.613	0.00	21.733

Table 4 shows that the behavior of *Gap* and Time, according to *n*. From this table, we can observe that when *n* is greater than 100, all heuristics are assigned a zero *Gap* value. This reflects the ease of the problem when the number of processors is less than the number of processes.

Table 4 shows that for *IK*, there is only *n= {10,20}* where *Gap* is not equal to zero. However, for all remaining values of *n* *Gap* is 0. The maximum *Gap* value is 0.03 and obtained for heuristic *IK* when *n=20*.

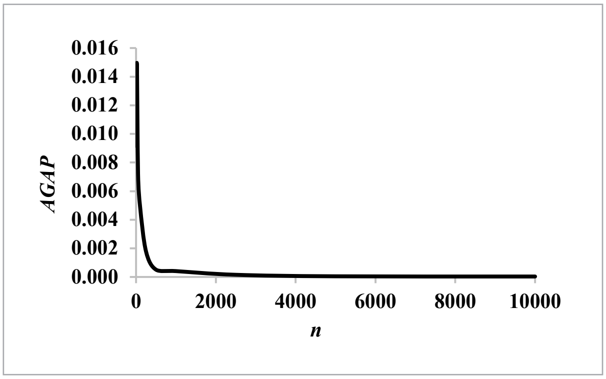
The second maximum value of *Gap* is 0.02 and obtained for heuristic *LPT* when *n= {10,20,50}* and *IK* when *n=10*. It is clear from Table 4 that heuristic *IR* is the most time consuming compared with heuristics *MSS* and *IK* reaching 84.613 s when *n=10000*.

Figure 2 gives the variation of the average gap (*Agap*) according to *n*.

Figure 3 shows that *Agap* decreases when *n* increase. In addition, the *Agap* is around 0.01 when *n* in {10,20,50}. The *Agap* becomes to decrease from *n=100*. As shown in Figure 3, when *n=10000*, *Agap* is less than 0.00003.

Further, Table 5 gives the variation of *Gap* and *Time* according to the number of processors. For all heuristics, excluding *MSS*, the time increases as *p* increases. The average *Gap* of all heuristics, given in the column *Total*, shows that instances when $p \geq 10$ are more difficult to solve because the *Agap* is not equal to zero. We

Figure 3
Agap variation according to *n*



observe that, in Table 5, when *p* is increasing, the running time also increases. This is due to the increasing complexity of the problem.

Table 6 shows the behavior of *Gap* and Time according to Class. For each class, we have 410 instances. Table 6 shows that the classes have almost the same difficulty for all heuristics. A slight difference in *IK* heuristic and *IR*. Indeed, for *IK* the higher *Agap* is obtained for Classes 1 and 4. However, for *IR* the higher *Agap* is obtained for classes 3 and 5.

The results show that more time is consumed for the *IR* heuristic when *n=10000* and *p=15* with 84.949 s. The maximum value of *Agap* is obtained for *IK* heuristic with 0.05.

Table 5
Gap and *Time* variation according to *p*

p	*LPT*		*MSS*		*IK*		*IR*		*Total*	
	Gap	*Time*	*Gap*	*Time*	*Gap*	*Time*	*Gap*	*Time*	*Gap*	*Time*
2	0.01	0.000	0.00	0.001	0.00	0.000	0.00	0.006	0.00	0.002
3	0.01	0.000	0.00	0.385	0.00	0.001	0.00	10.847	0.00	2.809
5	0.00	0.000	0.00	0.227	0.00	0.003	0.00	11.248	0.00	2.870
10	0.01	0.000	0.00	0.269	0.01	0.006	0.00	12.578	0.01	3.213
15	0.01	0.000	0.00	0.576	0.01	0.009	0.01	12.645	0.01	3.308

Table 6
Gap and *Time* variation according to Class

Class	*LPT*		*MSS*		*IK*		*IR*		*Total*	
	Gap	*Time*	*Gap*	*Time*	*Gap*	*Time*	*Gap*	*Time*	*Gap*	*Time*
1	0.00	0.000	0.00	0.209	0.01	0.003	0.00	11.024	0.00	2.809
2	0.01	0.000	0.00	0.347	0.00	0.004	0.00	10.554	0.00	2.726
3	0.01	0.000	0.00	0.496	0.00	0.005	0.01	11.011	0.00	2.878
4	0.01	0.000	0.00	0.220	0.01	0.004	0.00	11.033	0.00	2.814
5	0.01	0.000	0.00	0.429	0.00	0.006	0.01	11.017	0.00	2.863

5. Conclusion

In this study, we presented the problem of the maximization of the C_{min} (Max-Min) on the identical parallel processors. The problem is exhibited strong NP-hard characteristics. We developed novel heuristics to solve the problem approximately with an acceptable time execution. The first method iteratively solves by randomly selecting the processes having a fixed largest completion time. The second method is based on the utilization of the knapsack problems by dividing the initial problem into several sub-problems and an upper bound as a limit to schedule processes on the fixed processor. The experimental results show that the knapsack-based heuristic gives the same result as the heuristic given in literature review *MSS* with better processing time.

Acknowledgment

The authors extend their appreciation to the Deanship of Scientific Research at Majmaah University for funding this work under project number (RGP-2019-13).

References

1. Alharbi, M., Jemmali, M. Algorithms for Investment Project Distribution on Regions. *Computational Intelligence and Neuroscience, 2020*, 2020. https://doi.org/10.1155/2020/3607547
2. Alquhayz, H., Jemmali, M., Otoom, M. M. Dispatching-Rule Variants Algorithms for Used Spaces of Storage Supports. *Discrete Dynamics in Nature and Society, 2020*, 2020. https://doi.org/10.1155/2020/1072485
3. Azar, Y., Epstein, L. On-line Machine Covering. Journal of Scheduling, 1998, 1, 67-77. https://doi.org/10.1002/(SICI)1099-1425(199808)1:2<67::AID-JOS6>3.0.CO;2-Y
4. Chen, X., Epstein, L., Tan, Z. Semi-online Machine Covering for Two Uniform Machines. *Theoretical Computer Science*, 2009, 410, 5047-5062. https://doi.org/10.1016/j.tcs.2009.08.001

5. Deuermeyer, B. L., Friesen, D. K., Langston, M. A. Scheduling to Maximize the Minimum Processor Finish Time in a Multiprocessor System. *SIAM Journal on Algebraic Discrete Methods*, 1982, 3, 190-196. https://doi.org/10.1137/0603019
6. Ebenlendr, T., Noga, J., Sgall, J., Woeginger, G. A Note on Semi-Online Machine Covering. *International Workshop on Approximation and Online Algorithms*, 2005, 110-118. https://doi.org/10.1007/11671411_9
7. Gálvez, W., Soto, J. A., Verschae, J. Improved Online Algorithms for the Machine Covering Problem with Bounded Migration. *12th Workshop on Models and Algorithms for Planning and Scheduling Problem*s, 2015, 21.
8. Gerke, S., Panagiotou, K., Schwartz, J., Steger, A. Maximizing the Minimum Load for Random Processing Times. *ACM Transactions on Algorithms (TALG)*, 2015, 11, 17. https://doi.org/10.1145/2651421
9. Gharbi, A. Scheduling Maintenance Actions for Gas Turbines Aircraft Engines. *Constraints*, 2014, 10, 4.
10. Graham, R. L., Lawler, E. L., Lenstra, J. K., Kan, A. R. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics,* 1979, vol. 5, 287-326. https://doi.org/10.1016/S0167-5060(08)70356-X
11. Haouari, M., Jemmali, M. Maximizing the Minimum Completion time on Parallel Machines. *4OR*, 2008, 6, 375-392. https://doi.org/10.1007/s10288-007-0053-5
12. He, Y., Jiang, Y. Optimal Semi-Online Preemptive Algorithms for Machine Covering on Two Uniform Machines. *Theoretical Computer Science*, 2005, 339, 293-314. https://doi.org/10.1016/j.tcs.2005.02.008
13. Imreh, C. Maximizing the Minimum Machine Load. *Encyclopedia of Algorithms*, 2008, 1-3. https://doi.org/10.1007/978-3-642-27848-8_503-1
14. Jemmali, M. Approximate Solutions for the Projects Revenues Assignment Problem. *Communications in Mathematics and Applications*, 2019, 10, 653-658. https://doi.org/10.26713/cma.v10i3.1238
15. Jemmali, M., Alquhayz, H. Equity Data Distribution Algorithms on Identical Routers. *International Conference on Innovative Computing and Communications*, 2020, 297-305. https://doi.org/10.1007/978-981-15-0324-5_26
16. Jemmali, M., Melhim, L. K. B., Alharbi, S. O. B., Bajahzar, A. S. Lower Bounds for Gas Turbines Aircraft Engines. *Communications in Mathematics and Applications*, 2019, 10, 637-642. https://doi.org/10.26713/cma.v10i3.1218
17. Jiang, Y., Tan, Z., He, Y. Preemptive Machine Covering on Parallel Machines. *Journal of Combinatorial Optimization*, 2005, 10, 345-363. https://doi.org/10.1007/s10878-005-4923-5
18. Pisinger, D. Dynamic Programming on the Word RAM. *Algorithmica*, 2003, 35, 128-145. https://doi.org/10.1007/s00453-002-0989-y
19. Skutella, M., Verschae, J. A Rbust PTAS for Machine Covering and Packing. *European Symposium on Algorithms*, 2010, 36-47. https://doi.org/10.1007/978-3-642-15775-2_4
20. Tan, Z., Cao, S. Semi-Online Machine Covering on Two Uniform Machines WITH Known Total Size. *Computing*, 2006, 78, 369-378. https://doi.org/10.1007/s00607-006-0187-x
21. Tan, Z., He, Y., Epstein, L. Optimal On-line Algorithms for the Uniform Machine Scheduling Problem with Ordinal Data. *Information and Computation*, 2005, 196, 57-70. https://doi.org/10.1016/j.ic.2004.10.002
22. Tan, Z., Wu, Y. Optimal Semi-Online Algorithms for Machine Covering. Theoretical Computer Science, 2007, 372, 69-80. https://doi.org/10.1016/j.tcs.2006.11.015
23. Walter, R., Wirth, M., Lawrinenko, A. Improved Approaches to the EXACT SOLUTION OF THE Machine Covering Problem. *Journal of Scheduling*, 2017, 20, 147-164. https://doi.org/10.1007/s10951-016-0477-x
24. Wu, Y., Cheng, T., Ji, M. Optimal Algorithms for Semi-Online Machine Covering on Two Hierarchical Machines. *Theoretical Computer Science*, 2014, 531, 37-46. https://doi.org/10.1016/j.tcs.2014.02.015
25. Wu, Y., Yang, Q., Huang, Y. Machine Covering with Combined Partial Information. *Journal of Statistical Planning and Inference*, 2010, 140, 2351-2354. https://doi.org/10.1016/j.jspi.2010.01.030