Exact Algorithm for Batch Scheduling on Unrelated Machine

Hemmak Allaoua
Department of Computer Science, Mohamed Boudiaf University,
Laboratory of Informatics and its Applications of M'sila, Algeria
allaoua.hemmak@univ-msila.dz

Abstract: *In this paper, we propose a new linear algorithm to tackle a specific class of unrelated machine scheduling problem, considered as an important real-life situation, which we called Batch Scheduling on Unrelated Machine (BSUM), where we have to schedule a batch of identical and non-preemptive jobs on unrelated parallel machines. The objective is to minimize the makespan (C_{max}) of the whole schedule. For this, a mathematical formulation is made and a lower bound is computed based on the potential properties of the problem in order to reduce the search space size and thus accelerate the algorithm. Another property is also deducted to design our algorithm that solves this problem. The latter is considered as a particular case of* $Rm||C_{max}$ *family problems known as strongly NP-hard, therefore, a polynomial reduction should realize a significant efficiency to treat them. As we will show, Batch BSUM is omnipresent in several kind of applications as manufacturing, transportation, logistic and routing. It is of major importance in several company activities. The problem complexity and the optimality of the algorithm are reported, proven and discussed.*

Keywords: *Scheduling, unrelated machine, exact method, parallel machine, batch scheduling*.

Received September 9, 2022; accepted May 4, 2023
https://doi.org/10.34028/iajit/20/4/8

1. Introduction

In most cases, unrelated machine scheduling without preemption is considered among the hardest scheduling problems in the strong sense regarding their complicatedness and the astronomic number of candidate solutions especially when the number of jobs and machines are large enough. In the other hand, these kinds of problems are omnipresent in many company activities such as manufacturing, transportation, logistic and routing. Indeed, finding optimized machine schedules is an important and challenging task, as a large number of jobs need to be processed every day. They reveal a significant impact on the income of the company, especially when we need to minimize the last completion time known in scheduling field as makespan (C_{max}). The problem treated in this work consists on a specific case of the class of problems denoted as $Rm| |C_{max}$ which is proved as strongly NP-hard [5]. This case consists to schedule a batch of identical and non-preemptive jobs on unrelated machines which we have called Batch Scheduling on Unrelated Machine (BSUM) and we denoted as $Rm|p_{ij}=p_j|C_{max}$. Its resolution consists to find the optimal vector of job numbers to be assigned to the machines that minimize the makespan. A new linear algorithm is proposed, proven then implemented to tackle this problem. We aim to provide a tool that determine a polynomial reduction for the $Rm||C_{max}$ family problems to $Rm|p_{ij}=p_j|C_{max}$ that is to contribute to tackle NP-complet scheduling problems under "divide and conquer" paradigm. Two potential properties of the problem will be stated then proven that allow to elaborate then prove our algorithm. The first property allows to find a lower bound of the optimal solution that is to reduce the search space size and thus the processing time of the algorithm. The second one leads to compute the optimal solution of the problem.

In the rest of this paper, a related work is presented in section two, then we give a full description of the treated problem in section three. Section four is dedicated to expose the design, phases and complexity of our algorithm. Some comments and discussion are reported in section five.

2. Related Work

A significant amount of research on scheduling problems in general, and on those of unrelated machines has been studied extensively. Scheduling identical jobs on unrelated machines have been the subject of thorough research in the past, and two surveys by Allahverdi *et al*. [6] and Allahverdi [7] give an overview of the related literature. Relevant research in this type of problems includes approximation algorithms [4, 6, 8, 29] exact algorithms [16, 17, 20], mathematical programming techniques [14, 22, 25] optimization techniques [4, 8, 9, 23], and metaheuristic approaches [1, 18]. Mokotoff [21] and Pinedo [26] provides an extended survey for multiprocessor jobs problems in general.

The problems of scheduling on unrelated machines to minimize the makespan were also been well studied in the literature [5, 12, 19, 27]. Since this class of scheduling problems is known and proved as strongly NP-hard, all these works gave approximate algorithms [3] to solve them. in some work, just the case of two types of jobs is considered, Vakhania *et al*. [28] Hernandez presented a polynomial time algorithm. Ebenlendr *et al*. [10] elaborate a O(n^2)-algorithm to tackle a special case of the class $Rm||C_{max}$ [2, 21]. Fanjul-Peyro and Ruiz [13]. Some research focus on the equal processing times of jobs [11, 15]. Munir *et al*. [24] propose novel approaches for Scheduling task graphs in heterogeneous distributed computing environment that tackle a similar problem.

3. Problem Definition and Overview

3.1. Problem Statement

A batch of n identical and non-preemptive jobs to be scheduled on m unrelated parallel machines. The processing time of one job of the batch on the machine j is p_j (assuming that p_j is integer and $p_j>0$); p_j is the time spent by the machine j to proceed one job of the batch without preemption. That show that the speed of the machine j is inversely proportional to the processing time p_j. We aim to find the schedule of these n jobs with minimum last completion time (makespan) C_{max}. This scheduling problem can be denoted as $Rm|p_{ij}=p_j|C_{max}$.

Below (Table 1) an instance of BSUM problem:

Table 1. An instance of BSUM.

n = 5 jobs ; m = 3 machines			
Machine j	1	2	3
Processing time p_j	5	10	8

Since, in unrelated machine scheduling, each machine has its own speed $v_j=p_{ij}/p_i$, but in our case all jobs are identical, so the processing time p_j determine the speed of the machine j.

- **Problem Formulation**

The resolution of this problem consists to dispatch the n jobs on the m machines such that C_{max} is minimal. Therefore, the biggest task to do is to determine the number x_j of jobs to be assigned to the machine j for j=1, m; thus, the solution of this problem can be represented as an integer vector $x=(x_j)_{j=1,m}$ that describe this assignment. Hence the problem can be formulated as below:

$$\begin{cases} \min_{x\in N^m} C_{max} = \max(C_j) = \max(x_j p_j); \\ x = (x_j)\ ;\ j = 1, m \\ s.t.\ x_j p_j \le Cmax\ ;\ x_j \in N\ ;\ j = 1, m \\ \sum_{j=1}^{m} x_j = n \end{cases}$$

Exhaustive research of the optimal solution amounts to seek all ways to partition the integer n as an arrangement of m integers whose sum is n.

For the instance above, there are 21 different manners to dispatch the 5 jobs between 3 machines shown with their respective makespan (Table 2):

Table 2. Exhaustive list of solutions.

(5,0,0) 25	(1,4,0) 40	(4,1,0) 20
(0,5,0) 50	(1,0,4) 32	(4,0,1) 20
(0,0,5) 40	(0,1,4) 32	(0,4,1) 40
(2,3,0) 30	(3,2,0) 20	(1,1,3) 24
(2,0,3) 24	(3,0,2) 16	(1,3,1) 30
(0,2,3) 24	(0,3,2) 30	**(3,1,1) 15**
(2,1,2) 16	(2,2,1) 20	(1,2,2) 20

The optimal solution is (3, 1, 1) with the makespan C_{max}=15 represented in the diagram below (Figure 1):

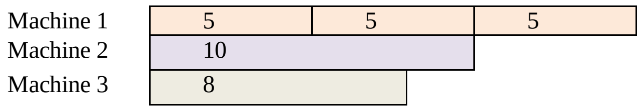

Figure 1. Schedule diagram.

Construct a solution to this problem consists to dispatch n jobs one by one between m machines, that leads to separate a sequence of n "1"s with (m-1) ","s to form m subsequences then add the "1"s of each subsequence.

Example: for n=5 and m=3 the sequence 111, 11 define the solution (3,0,2); that means we assign 3 jobs to the first machine, any job to the second machine and 2 jobs to the third machine and so on.

Thus, the number of ways to dispatch n jobs between m machines equals to the number of ways to separate n "1"s by (m-1) ","s ; therefore, the number of candidate solutions is: $C_{n+m-1}^{m-1} = \frac{(n+m-1)!}{n!\times(m-1)!}$, that explodes when n is big enough.

- **Example**

With n=5 jobs and m=3 machines, we have $C_7^2 = 21$ different ways to dispatch 5 jobs on 3 machines as given above. For 100 jobs and 20 machines, the search space size becomes astronomic: 491037×10^{16} candidate solutions. Result: since $C_{n+m-1}^{m-1} \approx O(2^n)$, exhaustive algorithm still inefficient. It is why we have to look for a polynomial algorithm based on potential properties of the problem BSUM to solve it.

3.2. Problem Properties

- **Property 1**

$LB = \left\lceil \frac{n}{\sum_{j=1}^{m} \frac{1}{p_j}} \right\rceil$ is a lower bound of C_{max} for BSUM.

- **Proof**

From the constraints in the BSUM formulation, we have: $x_j \le \frac{Cmax}{p_j}$. Hence: $\sum_{j=1}^{m} x_j \le \sum_{j=1}^{m} \frac{C_{max}}{p_j}$

Thus: $n \leq C_{max} \times \sum_{j=1}^{m} \frac{1}{p_j}$

Therefore: $C_{max} \geq \frac{n}{\sum_{j=1}^{m} \frac{1}{p_j}}$

Since C_{max} is integer: $C_{max} \geq LB$

That means:

$$LB = \begin{cases} \frac{n}{\sum_{j=1}^{m}\frac{1}{p_j}} ;\, if\, n\ modulo \sum_{j=1}^{m}\frac{1}{p_j} = 0 \\ \left(i.e.\ if\ \frac{n}{\sum_{j=1}^{m}\frac{1}{p_j}} \in N \right); \\ \frac{n}{\sum_{j=1}^{m}\frac{1}{p_j}} + 1 \quad otherwise \ ; \end{cases}$$

For the instance above: $LB = \left\lceil \frac{5}{\frac{1}{5}+\frac{1}{10}+\frac{1}{8}} \right\rceil = 12.$

- **Property 2**

The minimal value of C_{max} for BSUM is the smallest multiple λ of one of the integers p_j that satisfy: $\sum_{j=1}^{m} \left\lfloor \frac{\lambda}{p_j} \right\rfloor \geq n.$

- **Proof**

First, we have to prove that λ exists. So, for a given instance (n;m;p_j,j=1,m), there exists at least the makespan: $n \times \min_{j=1,m}(p_j)$ which is a multiple of one of the p_j integers and satisfy $\sum_{j=1}^{m} \left\lfloor \frac{\lambda}{p_j} \right\rfloor \geq n$, that is when we assign all jobs to the fastest machine where the solution is in the form (0,….,0,n,0,..,0), therefore, λ exists. Assume that x_j is the number of jobs to be proceeded by the machine j, the completion time C_j of the machine j is then $C_j = x_j \times p_j$. Since the makespan of the schedule is defined as $Cmax = max_{j=1,m}(C_j)$; Thus: $\exists\, i \in [1, m] : \lambda = x \times p_i ; x \in N^*$.
(λ is one among the $C_j, j = 1, m$).

So, λ is a multiple of one of the integers pj.

Since λ is a makespan of the schedule then:
$\forall j \in [1, m]: \lambda \geq x_j \times p_j$, Hence: $\forall j \in [1, m]: \frac{\lambda}{p_j} \geq x_j$

By adding: $\sum_{j=1}^{m} \left\lfloor \frac{\lambda}{p_j} \right\rfloor \geq \sum_{j=1}^{m} x_j$; $\sum_{j=1}^{m} \left\lfloor \frac{\lambda}{p_j} \right\rfloor \geq n$

In the other hand $\lambda = \min_{j=1,m}(C_{max})$, so λ is the smallest multiple of one the integers p_j that satisfy: $\sum_{j=1}^{m} \left\lfloor \frac{\lambda}{p_j} \right\rfloor \geq n.$

From these two properties, we deduct the following corollary:

- **Corollary**

The minimal value of C_{max} for BSUM is the smallest multiple λ of one the integers p_j that satisfy: $\sum_{j=1}^{m} \left\lfloor \frac{\lambda}{p_j} \right\rfloor \geq n$ and $\lambda \geq \frac{n}{\sum_{j=1}^{m}\frac{1}{p_j}}$.

These two properties allow to find the makespan and the optimal solution in polynomial time. The property (1) implies that the search start from LB, the property (2) implies that we have to look for the smallest multiple of one the processing time p_j that is the last completion time in the schedule.

4. Algorithm Description

In this section we will describe and discuss all phases of our proposed linear algorithm for solving BSUM. This approach consists of three phases:

The first phase is computing lower bound of the makespan. Based on the property (1) above we elaborate the Algorithm (1) below:

Algorithm 1: Int LB(int n ; int m ; int[] p)

```
 # Computing the lower bound LB.
# Input: number of jobs n, number of machines m and the
        respective processing time table p[].
# Output : LB.
   {
     sum = 1 / p[0]
     for ( j = 1 to m-1)
              sum = sum+ 1 / p[j]
     If ( n mod s = 0 )
              Return (n / sum)
     else
              return LB=((int)(n / sum) + 1)
   }
```

It is clear that this algorithm is linear, in O(m).

The second step consists to compute the makespan based on the property (2) of BSUM. The main idea is as follow: starting from the LB computed in the Algorithm (1) above, we seek progressively for the multiple of the integers p_j that satisfy the property (2), whence the Algorithm (2) below:

Algorithm 2: Int MinCmax(Int n ; Int m ; Int[] p)

```
# Computing the makespan min Cmax.
# Input: number of jobs n, number of machines m and the
  respective processing time table p[].
# Output: min Cmax.
   {
      Cmax = LB( n , m , p[]) ; # call LB function
      SumQ = 0 # Sum of Quotients
      while (True)
     {
        for (j = 0 to m-1)
          if (Cmax Mod p[j]) = 0) break # exit for
        if (j <= m)
       {
         SumQ = Cmax / p[0]
         for ( j = 1 to m-1)
               SumQ = SumQ  + Cmax / p[j]
         if (SumQuotients >= n)
                Return MinCmax = Cmax  # exit while
        }
     Cmax = Cmax +1
 }
   }
```

Once the LB is computed, we can assign $x_j^0 = \left\lfloor \frac{LB}{p_j} \right\rfloor$ jobs to the machine j to construct the initial solution x^0=(x_j^0) where $\sum_{j=1}^{m} x_j^0 \le n$.

If $\left(\sum_{j=1}^{m} x_j^0 = n \right)$ then x^0 is the optimal solution and LB =min C_{max}.

If $\left(\sum_{j=1}^{m} x_j^0 < n \right)$ then x^0 is not feasible solution and LB < min C_{max}, we will have to assign the remaining jobs not yet assigned whose the number is: $=$ $n - \sum_{j=1}^{m} \left\lfloor \frac{LB}{p_j} \right\rfloor$. As upper bound, we can assign them to the fastest machine (i.e., the machine with min(p_j)), so there exists a feasible solution x with:

C_{max}=*LB*+*r*×*min*(p_i). (x may be not optimal).

Therefore, this algorithm terminates and converges because λ exists as proven in property (2).

The while loop makes at most r iterations as much as the two inner successive loops are in O(m). Therefore, In the worst case, the Algorithm (2) is in O(rm) (that is when C_{max}= C_{max}=*LB*+*r*×$\min_{j=1,m}(p_i)$).

By replacing LB by its value in the expression of r, we find r≈ m. Since, in practice m<<n, say m ≈ c.n (c<1) this algorithm is at least in O(n).

The third step consists to find the optimal solution using the second part of the property (2), the optimal solution is a vector $x=(x_j)_{j=1,m}$ that describe the assignment of the n jobs to the m machines; that is done by dividing C_{max} respectively by the processing times p_j, that means assigning the n jobs to the m machines one by one. The number of jobs assigned to the machine j is $x_j = \frac{C_{max}}{p_j}$ (the quotient of C_{max} by p_j), as shown in the Algorithm (3) below:

Algorithm 3: Int[] Solution (Int n; Int m; Int[] p; Int Cmax ;)

```
#Finding an optimal solution.
#Input: number of jobs n, number of machines m, the
  respective processing time table p[].
#Output: schedule of jobs (x m-vector of jobs number
  assigned to the m machines) and Cmax.
{
   Int Sol (m)
   Int AllJobs = 0
   for (machine = 0 to m-1)
   {
       Sol [machine]=0
       for (Jobs=1 to Cmax / p[machine])
       {
                   Sol[machine]++
                   AllJobs = AllJobs+1
                   if (AllJobs = n) return Solution = Sol
       }
     }
  }
```

In order to show the algorithm efficiency, we have implemented it with the interface shown in the figure bellow (Figure 2). The data instances are generated randomly, that allow us to introduce instances with big size (large number of jobs and/or machines). For each case of these two dimensions, ten instances are generated. The algorithm has been implemented in the C programming language and compiled with gcc version 4.8.2. The computational experiments have been performed on one core of a system with Intel Core i5-4210U processor at 1.7 GHz and 10 GB of RAM under a Linux OS.

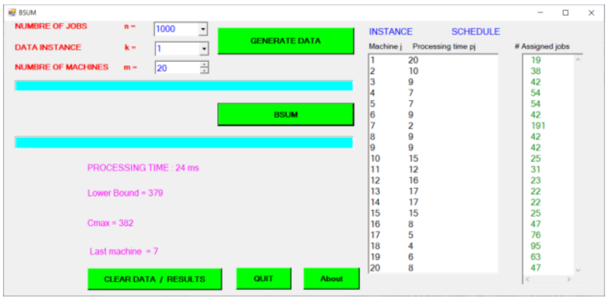

Figure 2. Implementation interface.

In order to show the efficiency and the robustness of our algorithm a set of random input data is generated using our own random generator that is to run the algorithms with same input. The following experimental settings is used:

# of jobs n ∈{50,100,500,1000,10000,100000};

# of machines m ∈{n/20 , n/10 , n/5};

instance k ∈ [1,10] .

Processing time p_j: random integer in the range [1, 20]. Following an example of results fo 10 instances for the set n=1000 and m=100 (Table 3).

Table 3. results for n=1000 and m=100.

instance	**Time (ms)**	C_{max}
1	76	70
2	65	72
3	73	52
4	74	72
5	75	60
6	76	52
7	86	57
8	71	60
9	71	54
10	72	72
Average	73.9	

The algorithm was run for all the data set, then we have constructed the curve representing the CPU time average in terms of n for each case of m values (Figure 3).

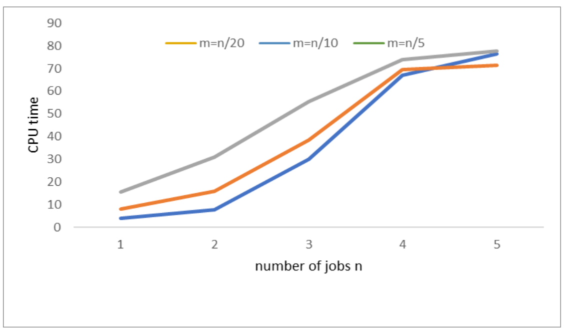

Figure 3. Average CPU time in terms of n.

This curve shows clearly the linearity of the algorithm complexity whatever the choice of m.

- **Comparison with other Exact Approaches**

As we are about to discuss exact approaches, where the optimality must be formally proven, we have compared our algorithm to the exact ones elaborated for the same problem found in the literature. the results are summarized in the Table 3 below. All these cases are reported in several papers and formally proven. Some of them was served to measure and justify the efficiency of heuristics [5, 7].

Table 3. Comparison with other algorithms.

Approach	Complexity
Linear Assignment	$O(mn^2)$
Dynamic Programming	$O(mn^{2m+1})$
Integer Linear Programming	O(n log m)
Linear programming relaxation	$O(n + m\log m)$

Note that, the least expensive metaheuristic as simulated annealing will make not less than $O(n^2)$ time to give just a good approximate solution (the number of iteration must at least be linear in n and the computation of a solution neighbour costs O(n) .

5. Conclusions

In this paper, a new algorithm was proposed then implemented for solving a specific class of unrelated machine scheduling problem where we have to schedule a batch of same jobs on unrelated machines which we have called BSUM. The algorithm is designed based on the potential properties of the problem. We showed that this algorithm is quadratic complexity in worse case. For this, a mathematical formulation is made and a lower bound is computed based on the potential properties of the problem in order to reduce the search space size and thus accelerate the algorithm. Another property is also deducted to design our algorithm that solves this problem. The latter is considered as a particular case of Rm| |Cmax family problems known as strongly NP-hard, therefore, a polynomial reduction should realize a significant efficiency to treat these problems. As we will show, BSUM is omnipresent in several kind of applications as manufacturing, transportation, logistic and routing. it is of major importance in many company activities. The problem complexity and the optimality of the algorithm are reported, proven and discussed.

References

[1] Adan J., Adan I., Akcay A., Van den Dobbelsteen R., and Stokkermans J., "A Hybrid Genetic Algorithm for Parallel Machine Scheduling At Semiconductor Back-End Production," *in Proceedings of the International Conference on Automated Planning and Scheduling,*, vol. 28, pp. 298-302, 2018. DOI: https://doi.org/10.1609/icaps.v28i1.13913

[2] Afzalirad M. and Rezaeian J., "A Realistic Variant of Bi-Objective Unrelated Parallel Machine Scheduling Problem: NSGA-II and MOACO Approaches," *Applied Soft Computing*, vol. 50, pp. 109-123, 2017. https://doi.org/10.1016/j.asoc.2016.10.039

[3] Afzalirad M. and Shafipour M., "Design of An Efficient Genetic Algorithm for Resource-Constrained Unrelated Parallel Machine Scheduling Problem with Machine Eligibility Restrictions," *Journal of Intelligent Manufacturing*, vol. 29, no. 2, pp. 423-437, 2018. DOI:10.1007/s10845-015-1117-6

[4] Afzalirad M. and Rezaeian J., "Resource-Constrained Unrelated Parallel Machine Scheduling Problem With Sequence Dependent Setup Times, Precedence Constraints and Machine Eligibility Restrictions," *Computers and Industrial Engineering*, vol. 98, pp. 40-52, 2016. https://doi.org/10.1016/j.cie.2016.05.020

[5] Allahverdi A., "The Third Comprehensive Survey on Scheduling Problems with Setup Times/Costs," *European Journal of Operational Research*, vol. 246, no. 2, pp. 345-378, 2015. https://doi.org/10.1016/j.ejor.2015.04.004

[6] Allahverdi A., Ng C., Cheng T., Kovalyov M., "A Survey of Scheduling Problems with Setup Times or Costs," *European Journal of Operational Research*, vol. 187, no. 3, pp. 985-1032, 2008. https://doi.org/10.1016/j.ejor.2006.06.060

[7] Allahverdi A., "The Third Comprehensive Survey on Scheduling Problems with Setup Times/Costs," *European Journal of Operational Research*, vol. 246, no. 2, pp. 345-378, 2015. https://doi.org/10.1016/j.ejor.2015.04.004

[8] Chen Z., "Parallel Machine Scheduling with Time Dependent Processing Times," *Discrete Applied Mathematics*, vol. 70, no. 1, pp. 81-93, 1996. https://doi.org/10.1016/0166-218X(96)00102-3

[9] Cheng T., Ding Q., and Lin B., "A Concise Survey of Scheduling with Time-Dependent Processing Times," *Discrete Applied Mathematics*, vol. 152, no. 1, pp. 1-13, 2004. https://doi.org/10.1016/0166-218X(96)00102-3

[10] Ebenlendr T., Kral M., and Sgall J., "Graph Balancing: A Special Case of Scheduling Unrelated Parallel Machines," *in Proceedings of the nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, pp. 483-490, 2008. DOI:10.1145/1347082.1347135

[11] Fanjul-Peyro L. and Ruiz R., "Iterated Greedy Local Search Methods for Unrelated Parallel Machine Scheduling," *European Journal of Operational Research*, vol. 207, no. 1, pp. 55-69, 2010. https://doi.org/10.1016/j.ejor.2010.03.030

[12] Fanjul-Peyro L. and Ruiz R., "Scheduling

Unrelated Parallel Machines with Optional Machines and Jobs Selection," *Computers and Operations Research*, vol. 39, no. 7, pp. 1745-1753, 2012. https://doi.org/10.1016/j.cor.2011.10.012

[13] Fanjul-Peyro L. and Ruiz R., "Size-Reduction Heuristics for the Unrelated Parallel Machines Scheduling Problem," *Computers and Operations Research*, vol. 38, no. 1, pp. 301-309, 2011. https://doi.org/10.1016/j.cor.2010.05.005

[14] Gawiejnowicz S., *Time-Dependent Scheduling*, Springer, 2008.

[15] Goldwasser M. and Pedigo M., "Online, Non-Preemptive Scheduling of Equal-Length Jobs on Two Identical Machines," *in Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT)*, Riga, pp. 113-123, 2006.

[16] Ji M. and Cheng T., "Parallel-Machine Scheduling of Simple Linear Deteriorating Jobs," *Theoretical Computer Science*, vol. 410, no. 38, pp. 3761-3768, 2009. https://doi.org/10.1016/j.tcs.2009.04.018

[17] Kononov A. and Gawiejnowicz S., "Np-Hard Cases in Scheduling Deteriorating Jobs on Dedicated Machines," *The Journal of the Operational Research Society*, vol. 52, no. 6, pp. 708-717, 2001. https://www.jstor.org/stable/254283

[18] Kravchenko S. and Werner F., "Parallel Machine Problems with Equal Processing Times: A Survey," *Journal of Scheduling*, vol. 14, pp. 435-444, 2011. DOI:10.1007/s10951-011-0231-3

[19] Lenstra J., Shmoys D., and Tardos E., "Approximation Algorithms for Scheduling Unrelated Parallel Machines," *Mathematical Programming*, vol. 46, pp. 259-271, 1990.

[20] Li S. and Yuan J., "Parallel-Machine Scheduling with Deteriorating Jobs and Rejection," *Theoretical Computer Science*, vol. 411, no. 40, pp. 3642-3650, 2010. https://doi.org/10.1016/j.tcs.2010.06.008

[21] Mokotoff E., "Parallel Machine Scheduling Problems: A Survey," *Asia-Pacific Journal of Operational Research*, vol. 18, no. 2, pp. 193-202, 2001.

[22] Moser M., Musliu N., Schaerf A., and Winter F., "Exact and Metaheuristic Approaches for Unrelated Parallel Machine Scheduling," *Journal of Scheduling*, vol. 25, no. 5, pp. 507-534, 2022. https://doi.org/10.1007/s10951-021-00714-6

[23] Mosheiov G., "Multi-Machine Scheduling with Linear Deterioration," *Information Systems and Operational Research*, vol. 36, no. 4, pp. 205-214, 1998. https://doi.org/10.1080/03155986.1998.11732359

[24] Munir E., Ijaz S., Anjum S., Khan A., Anwar W., and Nisar W., "Novel Approaches for Scheduling Task Graphs in Heterogeneous Distributed Computing Environment," *The International Arab Journal of Information Technology*, vol. 12, no. 3, pp. 270-277, 2015. https://www.iajit.org/PDF/vol.12%2Cno.3/6131.pdf

[25] Ouazene Y. and Yalaoui F., "Identical Parallel Machine Scheduling with Time-Dependent Processing Time," *Theoretical Computer Science*, vol. 721, pp. 70-77, 2018. https://doi.org/10.1016/j.tcs.2017.12.001

[26] Pinedo M., *Scheduling, Theory, Algorithms, and Systems*, Springer Science+Business Media, 2012. DOI 10.1007/978-3-319-26580-3

[27] Shchepin E. and Vakhania N., "An Optimal Rounding Gives A Better Approximation for Scheduling Unrelated Machines," *Operations Research Letters*, vol. 33, no. 2, pp. 127-133, 2005. https://doi.org/10.1016/j.orl.2004.05.004

[28] Vakhania N., Werner F., and Alberto J., "Scheduling Unrelated Machines with Two Types of Jobs," *International Journal of Production Research*, vol. 52, no. 13, pp. 3793-3801, 2014. DOI:10.1080/00207543.2014.888789

[29] Yin N., Kang L., Sun T., Yue C., and Wang R., "Unrelated Parallel Machines Scheduling with Deteriorating Jobs and Resource Dependent Processing Times" *Applied Mathematical Modelling*, vol. 38, no. 19, pp. 4747-4755, 2014. https://doi.org/10.1016/j.apm.2014.03.022

Hemmak Allaoua associate professor at department of computer science, faculty of mathematics and informatics since, Mohamed Boudiaf University of M'sila, Algeria since 2007. Having taught several syllabuses such as combinatorial optimization, metaheuristics, network, language theory. Director of laboratory of informatics and its application of M'sila (LIAM) and head of the team of optimization and artificial intelligence. Having several publications, talks, activities in these research fields. Chair of ISIA'20 (International Symposium of Informatics and its Application). Actually, TPC member of several conferences and reviewer for several journals in the field.