REAL-TIME ASSIGNMENT OF WORKERS SUBJECT TO FATIGUE

SIMON BERNARD, LOUIS FOUQUET, AND ROMAIN TUI

1. Introduction

In industry, organization of the tasks between workers and machines is an important issue. Mathematic models aim at finding a way to organize these tasks in order to optimize an objective. Some models, more realistic, consider the evolution of the workers' fatigue along the working day. In a former study, *A. Ferjani, A. Ammar, H. Pierreval and S. Elkosantini* presented their own approach to the problem. They used a decision-making tool: TOPSIS (Technique for Order of Preference by Similarity to Ideal Solution). TOPSIS determines the "best" solution by considering positive and negative criteria. The aim of the work is to reduce the average processing time of a product also called MFT (Mean Flow Time). In this study, there was a single positive criteria SPT (Shortest Processing Time which chooses the machine where the first task is the shortest) and a single negative criteria LNQ (Longest Number in Queue, which chooses the machine where the total processing time of a machine is the longest).

Are these criteria relevant? Is this approach appropriate? Is it possible to find another approach which is more efficient?

Our work was about replicating the original study from *A. Ferjani, A. Ammar, H. Pierreval and S. Elkosantini*, to formulate the same problem and to have the experimental conditions as closest as possible to theirs. The second part of our work was to offer new heuristics which could improve the original results. We presented two heuristic: a new version of TOPSIS with two more criteria (*New TOPSIS*), and a heuristic based on *Dynamic Programming*. We also added in our results a random method to measure the efficiency of each heuristic.

Our results are questioning and challenging. In some cases, none of our heuristic seems to work (the random assignment is always better). In most of the cases, our heuristics *Dynamic Programming* and *New TOPSIS* seem to surpass the *Original TOPSIS* and the others heuristics.

2. Problem Formulation

2.1. **Job-shop model.** We consider a job-shop with a set O of n workers who can be assigned to a set M of m machines, with $n < m$. Each worker can be assigned to a machine according to a skill matrix $Q(n,m)$ with the binary $Q(i,j)$ indicating if the worker i is skilled to work on machine j.

Machines process r different product types with determined routines. The assignment of a worker to a machine is made following two triggering events:

- A worker i finishes his task and is available for another assignment.
- A worker is waiting for a task and a machine on which he can work needs a worker.

When the triggering event occurs at t_i, the worker i has to be assigned to a machine within the set $S_i^{t_i}$ of available machines for worker i at instant t_i (such that $S_i^{t_i} \subseteq M$ and $Q(i,j) = 1,\ for\ all\ i \in S_i^{t_i}$. Three cases can then appear depending on the number of machines available to worker i:

- $|S_i^{t_i}| = 0$: There is no machine available to worker i. He has to wait for the arrival of a new product or for another worker to release a machine on which he is skilled to work. Then he will be assigned to the first available machine.
- $|S_i^{t_i}| = 1$: The worker is immediately assigned to the only avalaible machine.
- $|S_i^{t_i}| > 1$: One machine should be chosen among the available machines for worker i. This is the only case for which a decision has to be taken.

The objective of the assignment decisions is to minimize the mean flowtime (MFT) Z.

2.2. **Product arrival and processing model.** For the r types of products, job orders are arriving randomly following an exponential distribution with parameters $\lambda_1(t), \lambda_2(t), ..., \lambda_r(t)$. The value of the parameters $\lambda_1(t), \lambda_2(t), ..., \lambda_r(t)$ depend on time to represent the varying demand of customers over time. Jobs in the machines queues are processed according to the First In First Out (FIFO) dispatching rule. The theoretical (without the added value due to fatigue) are uniformly distributed within an interval which varies from one type of product to another and from one machine to another. The theoretical processing times are determined when the product arrives in the system.

2.3. **Fatigue model.** The task duration gets deteriorated as the fatigue level of the workers increases over time. The corrected processing time will thus be a function of the theoretical processing times T_{kj} (theoretical processing time of task k on machine j).

Let $F_i(t)$ be the fatigue level of worker i at instant t which represents all the fatigue he have accumulated before t on the shift. When the worker is not assigned to a machine, his fatigue level does not vary.

Each machine increases the fatigue at a different speed regarding its penibility. Therefore each machine j is associated with a penibility coefficient d_j such that $0 \leqslant d_j \leqslant 1$.

The increase of fatigue generated during a new task on machine j, between its beginning t_i and the current time t is expressed by $\Delta_{ij}(t_i, t)$ as follows:

$$\Delta_{ij}(t_i, t) = \left(1 - F_i(t_i)\right)\left(1 - e^{-d_j(t-t_i)}\right). \tag{1}$$

Then the level of fatigue is updated as follows:

$$F_i(t) = F_i(t_i) + \Delta_{ij}(t_i, t). \tag{2}$$

Finally, we define $T'_{kj}(t_i)$ the corrected processing time of task k (waiting in the k-th position of queue) on machine j to which the worker i can be assigned on instant t_i, as follows:

$$T'_{kj}(t_i) = T_{kj}\Big(1 + \delta d_j\Big(\ln\left(1 + F_i(t_i)\right)\Big)\Big). \tag{3}$$

The parameter δ represents the influence of fatigue to adapt this correction to different cases.

3. Algorithmic solutions and contributions

3.1. **Dynamic Programming.**

3.1.1. *A first model without fatigue.* We consider the case of two machines, one worker, all products are already present in the system (static setting), and every product has to visit only one machine. The objective is to minimize the mean flow time. We claim that this case is polynomial.

We model now this problem with the notion of shuffle. A *shuffle* of two sequences $s = s_1, \dots, s_k$ and $\boldsymbol{t} = t_1, \dots, t_\ell$ is a sequence obtained by interlacing them in an arbitrary way. Formally, a shuffle of two such sequences is a sequence $w_1, \dots, w_{k+\ell}$ such that $s_i = w_{\alpha(i)}$ for all $i \in [k]$ and $t_j = w_{\beta(j)}$ for all $j \in [\ell]$ where $\alpha : [k] \to [k+\ell]$ and $\beta : [\ell] \to [k+\ell]$ are two increasing maps with disjoint images. The set of all shuffles of s and $\boldsymbol{t}$ is denoted by $s \sqcup \boldsymbol{t}$.

The feasible solutions of our problem are exactly the shuffles of the two queues of products. Denote by m and n the number of products waiting respectively in the first queue and in the second queue. For a feasible solution in which the processing time of the ith product processed by the worker is denoted by w_i, the mean flow time is given by:

$$\frac{1}{m+n}\sum_{i=1}^{m+n}(m+n-i)w_i \tag{4}$$

Given two sequences $s = s_1, \dots, s_m$ and $\boldsymbol{t} = t_1, \dots, t_n$ of positive real numbers, our problem consists thus in computing a sequence $w \in s \sqcup t$ minimizing:

$$\sum_{i=1}^{m+n}(m+n+1-i)w_i \tag{5}$$

The following lemma shows that the problem can be solved in polynomial time via a dynamic programming approach.

Lemma 1. *For $k \in \{0, \dots, m\}$ and $\ell \in \{0, \dots, n\}$, define*

$$\pi(k,\ell) = \min_{w}\sum_{i=1}^{m+n}(m+n+1-i)w_i$$

where the minimum is taken over all shuffles w of $s_1, \dots, s_k$ and $t_1, \dots, t_\ell$. The following equality holds then for all $k \in [m]$ and $\ell \in [n]$:

$$\pi(k,\ell) = \min\big(\pi(k-1,\ell) + (m+n+m'+n'+1-k-\ell)s_k, \pi(k,\ell-1) + (m+n+1-k-\ell)t_\ell\big).$$

Theorem 1. *The problem can be solved in $O\big((m+n')(n+m')\big)$.*

Proof. This is a direct consequence of Lemma 1. □

3.1.2. *Dynamic programming considering fatigue.* The new definition of the mean flow time is now:

$$\frac{1}{m+n}\sum_{i=1}^{m+n}(m+n+1-i)\omega(w,i) \tag{6}$$

where w_i is equal to the original processing time for the ith product and $\omega(w,i)$ is the updated processing time for this task. This value can be computed thanks to:

$$\omega(w,i) = w_i\phi\big(d(w,i), f(w,i-1)\big) \textit{ for all } i \in [m+n]$$

$$f(w,i) = \begin{cases} 0 & \text{if } i = 0 \\ \psi\big(d(w,i), f(w,i-1), \omega(w,i)\big) & \text{otherwise.} \end{cases}$$

In this expression we have:

- $\phi(d, f) = 1 + \delta d ln(1 + f)$
- $\psi(d, f, \omega) = f + (1 - f)(1 - e^{-d\omega})$
- $f(w, i)$ is the fatigue of worker after the ith task has been processed.
- $d(w, i)$ is the penibility coefficient d_j of the machine that processes the ith product in w.

Then we can define for $k \in \{0, ..., m\}$, $l \in \{0, ..., n\}$, and $f \in [0, 1]$:

$$\pi(k,l,f) = \inf \sum_{i=1}^{k+l} (m+n+1-i)\omega(w,i) \ : \tag{7}$$

where w is a shuffle of $s_1, ..., s_k$ and $t_1, ..., t_l$, and $f(w, k + l) \leq f$

$\pi(k, l, f)$ is the minimal mean flow time to process the first $k+l$ tasks and finishing with a fatigue level being at most f. So the mean flow time of our problem is the quantity $\pi(m, n, 1)$.

To compute $\pi(k, l, f)$, we have the following equality for all $k \in \{0, ..., m\}$, $l \in \{0, ..., n\}$, and $f \in [0, 1]$:

$$\begin{aligned} \pi(k,l,f) = \min\Big(&\inf\big\{\pi(k-1,l,f') + (m+n+1-k-l)s_k\phi(d_1,f') \\ &f' \geq 0 \text{ and } \psi(d_1,f',s_k) \leq f, \inf\big\{\pi(k,l-1,f') + (m+n+1-k-l)t_l\phi(d_2,f') \\ &f' \geq 0 \text{ and } \psi(d_2,f',t_l) \leq f\Big) \end{aligned} \tag{8}$$

It is clear that it is not possible to compute this value for all f. But we can approximate the situation. Let R be a positive integer, to be thought as the level of accuracy. For positive integers $k \leq m, l \leq n$, and $r \leq R$, define:

$$\rho^R(-1,l,r) = \rho^R(k,-1,r) = +\infty \text{ and } \rho^R(0,0,r) = 0 \tag{9}$$

and when at least one of k and l is nonzero define

$$\begin{aligned} &\rho^R(k,\ell,r) = \\ &\min\bigg(\min\Big\{\rho^R(k-1,\ell,r') + (m+n+1-k-\ell)s_k\phi\big(d_1,\tfrac{r'}{R}\big) \,:\, r' \in \{0,1,\dots,r\} \text{ and } \psi\big(d_1,\tfrac{r'}{R},s_k\big) \leqslant \tfrac{r}{R}\Big\}, \\ &\min\Big\{\rho^R(k,\ell-1,r') + (m+n+1-k-\ell)t_\ell\phi\big(d_2,\tfrac{r'}{R}\big) \,:\, r' \in \{0,1,\dots,r\} \text{ and } \psi\big(d_2,\tfrac{r'}{R},t_\ell\big) \leqslant \tfrac{r}{R}\Big\}\bigg). \end{aligned}$$

We have $\rho^R(m, n, R) \geqslant \pi(m, n, 1)$ for all positive integers R, and $\lim_{R\to+\infty} \rho^R(m, n, R) = \pi(m, n, 1)$ (and the convergence is monotone).

Looking at these equations, it would appear that it is not difficult to build an algorithm which is able to approximate the MFT and to build the optimal shuffle. But we have to remind that our problem is more complex. The problem that we have to solve is when $|S_i^{t_i}| > 1$. But this sub-problem is different from the original problem:

- The number of machines could be more than just two.
- Not all tasks are known from the beginning of the problem.
- If the worker is not the only one able to work on a machine, he is not going to do all the tasks of this machine.

Considering these problems and accepting some of them, we propose the following heuristic:

(1) Create a list of machine's pairs and possibly a single machine (in odd cases). Obviously, each machine of $S_i^{t_i}$ must appear only once.
(2) For each pair, compute $\pi(m, n, 1)$ and the associated path.
(3) The first machine of the path is considering as the "winning machine" and is added to the "winners" list. The possible single machine is also added to this list.
(4) If the "winners" list contains more than an single machine, return to step 1.
(5) The machine chosen is the one which is the last winner.

The idea is to organize a kind of tournament between the machines included in the set $S_i^{t_i}$. With this tournament we can bring ourselves back to the subproblem of one worker - two machines. Intuitively, this heuristic is better suited to the case where the worker is the only one working on the machines for which he is qualified.

3.2. **TOPSIS method.**

3.2.1. *Classical TOPSIS. A. Ferjani, A. Ammar, H. Pierreval and S. Elkosantini* proposed a heuristic based on the TOPSIS. We remind the grand features of such a method. TOPSIS can be described as a multi-criteria decision analysis method that relies on the transposition of observable phenomena into quantitative criteria that will constitute coordinates for a corresponding situation in the space of possible situations. Assuming there exists an ideal situation and a negative ideal situation, TOPSIS aims at choosing the alternative that has the shortest geometric distance from the positive ideal situation and the longest geometric distance from the negative ideal situation.

We can decompose the TOPSIS process into 5 different steps.

(1) We create a matrix $A = (a_{ij})$ in which each line represents a possible situation and each column represents a different criterion of interest.
(2) From A we create a normalized matrix $V = (v_{ij})$ by dividing each column j by its norm, calculated as $\sqrt{\sum_i a_{ij}^2}$.
(3) From V we create a weighted matrix $W = (w_{ij})$ by multiplying each column j by a weight ν_j, ranging from 0 to 1 and corresponding to the importance given to each criterion. To illustrate the fact that only these criteria are considered, we impose that $\sum_j \nu_j = 1$.
(4) We evaluate the best situation and the worst one:
    - $I_{best} = \{\langle \max_i(w_{ij}),\ for\ all\ j \in J_+\rangle, \langle \min_i(w_{ij}),\ for\ all j \in J_-\rangle\}$
    - $I_{worst} = \{\langle \min_i(w_{ij}),\ for\ all\ j \in J_+\rangle, \langle \max_i(w_{ij}),\ for\ all\ j \in J_-\rangle\}$
(5) For each situation, we calculate its L^2-distance to each of the ideal and negative ideal situations:

- $d_{ib} = \sqrt{\sum_j l_{ij}}$ with $l_{ij} = (w_{ij} - C^j_{best})^2$
  where C^j_{best} is the value of the criterion corresponding to column j in I_{best}
- $d_{iw} = \sqrt{\sum_j l_{ij}}$ with $l_{ij} = (w_{ij} - C^j_{worst})^2$
  where C^j_{best} is the value of the criterion corresponding to column j in I_{worst}

(6) We calculate the similarity between each situation i and the ideal one:
- $s_i = \frac{d_{ib}}{d_{ib}+d_{iw}}$, ranging from 0 to 1

(7) Rank the possible situations and choose the best one.

3.2.2. *Use of TOPSIS in our case.* First we decided to use TOPSIS the same way the authors of the original article did, using as criteria the SPT and the LNQ principles:

- SPT stands for Shortest Processing Time. It is considered as an efficient rule of assignment that consists in assigning in priority to the shortest task to process. We can consider that this criterion will have a positive impact on our situation.
- LNQ stands for Longest Number in Queue. This criterion allows us to balance the effect of SPT on queues, as if only short tasks are computed, longer tasks can stack over time and machine queues can get out of control.

3.3. **New TOPSIS method extended to 4 criteria.**

3.3.1. *Our adaptation.* As the original problem aimed at taking the increase of fatigue level over time in consideration, we add criteria directly linked to fatigue to the original TOPSIS method presented in the article.The principal motivation to this comes from the fact that SPT and LNQ criteria do not capture the fatigue increase over time, and that the only way such a phenomenon is indirect and through the real-time recalculation of duration.

3.3.2. *New criteria.* The first "effect" linked to fatigue we try to include into a criterion is the evolution of the levels of fatigue of workers themselves, as we assume we have to minimize the speed of such an increase over time to preserve workers efficiency, and so on to reduce MFT. Thus, this criterion is calculated as followed: $C_3 = \Delta_{ij}(t_i, t) = \left(1 - F_i(t_i)\right)\left(1 - e^{-d_j(t-t_i)}\right)$, with the worker designated by i, the machine by j, and the duration of the potential task by $t - t_i$. As we try to minimize the fatigue of each worker, the amount of fatigue added to the status of a worker is something to reduce. Thus, we consider that this criterion has a negative impact on our situation.

The second effect linked to fatigue we try to include into a criterion is the increase of a task duration. The idea is to create a ratio that illustrates how hard a task gets impacted by the level of fatigue of the workers who is assigned to it. Such a ratio will be calculated following the fatigue model presented above: $C_4 = 1 +_j ln\left(1 + F_i(t_i)\right)$. As we want the worker to be assigned to a task on which he is efficient, we consider that this criterion has a negative impact.

4. Numerical experiments

4.1. **Description of the different parameters.** We follow as closely as possible the experimental settings of *Aicha Ferjani, Achraf Ammar, Henri Pierreval and Sabeur Elkosantini.*

Our problem is based on four different products and eight machines. The four products must be processed in machines according to specific sequences (Figure 1).

	Machine	M1	M2	M3	M4	M5	M6	M7	M8
Product	p_1	1	2	3	4				5
	p_2		1		2			3	
	p_3	3		1		2			
	p_4					1	2	3	4

FIGURE 1. Products sequences

We consider four different skills matrix Q (Figure 2) and the same exponential distributions with time-dependence $\lambda_1(t)$, $\lambda_2(t)$, $\lambda_3(t)$, $\lambda_4(t)$ (Figure 3).

The different processing times for each task follow a uniform law. During the simulation, we consider that these laws are discretes. The parameters of the laws are indicated in Figure 4. The penibility coefficient for each machine are defined in Figure 5.

Three different values of δ are experimented:

- $\delta = 0$, we do not consider fatigue.
- $\delta = 0.3/ln(2)d_j$, the processing time increases of at most about 30%.
- $\delta = 0.7/ln(2)d_j$, the processing time increases of at most about 70%.

To compare our results, we simulated other methods than the ones described previously:

- A method call *Random*, which is simply a random assignment of the new machine in $S_i^{t_i}$.
- A method call *Original TOPSIS* taken from the reference article, which is the TOPSIS with two criteria: SPT (Shortest Processing Time) and LNQ (Longest Number in Queue).
- An *SPT* method.

By a lack of information concerning the unity of time used in the original study, we decided to work in minutes. Basically, a working day is composed of 8 hours of work (480 minutes). Every worker has a level of fatigue equal to zero at the beginning of each working day.

We multiply the time interval of product arrival by 10 to reduce the workload.[1]

We decide to multiply the fatigue parameter by a constant lower than 1. Indeed, if we kept the original formula, the fatigue could explode (Worker 1 goes from zero to maximum fatigue level in one short task on Figure 6). Other workers are all above a level of fatigue of 80% after one task too. We adapted our simulation by dividing the increasing level of fatigue of each task by 5.

Then, we approximate the appropriate weight coefficients for each criteria in *Original TOPSIS*. We work on a sample of 100 days and choose the right pair of coefficient by choosing the best pair with a tolerance level of 10^{-2}. The different pairs are shown in Figure 6.

We also compute the four weights v_1, v_2, v_3, v_4 for the four criteria of *New TOPSIS*. We proceed in the same way as in the *Original TOPSIS*. However, there are much more possibilities and the

[1]This operation is not described in the original article, but we decide to do so in order to keep our results consistent with regards of the way our simulator works.

$$Q = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Skill matrix of test case 1

$$Q = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Skill matrix of test case 2

$$Q = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Skill matrix of test case 3

$$Q = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Skill matrix of test case 4

FIGURE 2. Skill matrix for each test case

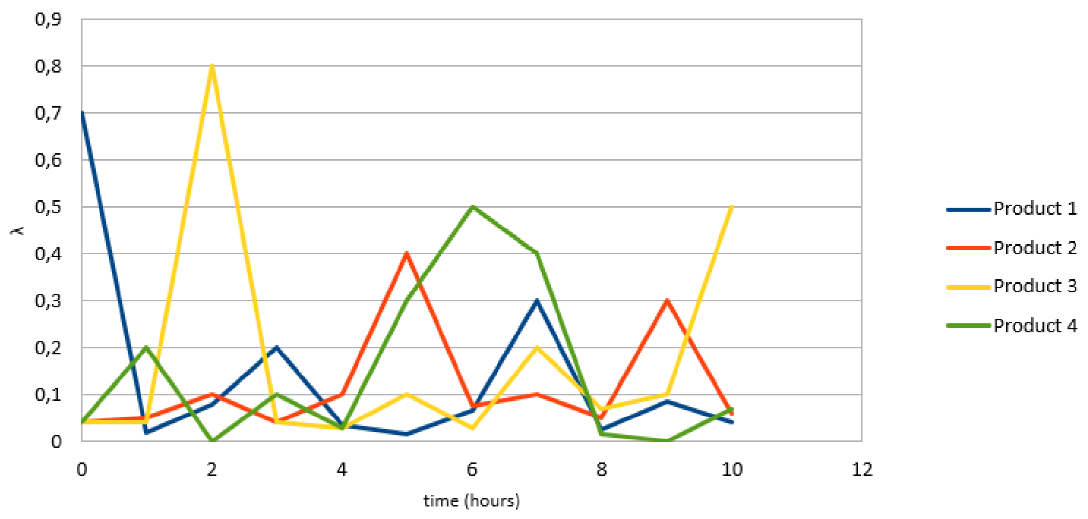

FIGURE 3. Fluctuation of the customer's demand over the time

	Machine	M1	M2	M3	M4	M5	M6	M7	M8
Product	p_1	[14,17]	[5,8]	[28,32]	[2,4]				[30,37]
	p_2		[10,13]		[20,25]			[6,8]	
	p_3	[25,28]		[5,10]		[10,14]			
	p_4					[5,9]	[12,15]	[30,34]	[3,7]

FIGURE 4. Products processing times (minutes)

time taken to calculate the coefficients was lengthened. That is why we reduce the length of the sample to 15 days. Furthermore, the tolerance level was fixed to 10^{-1}. The different set of weights

Machine	M1	M2	M3	M4	M5	M6	M7	M8
d_j	0.8	0.5	0.1	0.3	0.2	0.7	0.1	0.5

FIGURE 5. Penibility coefficient of each machine

Skill Matrix	δ	v_1	v_2
Q1	0	0.08	0.92
	$0.3/ln(2)d_j$	0.42	0.58
	$0.7/ln(2)d_j$	0.4	0.6
Q2	0	0.92	0.08
	$0.3/ln(2)d_j$	0.83	0.17
	$0.7/ln(2)d_j$	0.75	0.25
Q3	0	0.25	0.75
	$0.3/ln(2)d_j$	0.01	0.99
	$0.7/ln(2)d_j$	0.26	0.74
Q4	0	0.95	0.05
	$0.3/ln(2)d_j$	0.98	0.02
	$0.7/ln(2)d_j$	0.99	0.01

FIGURE 6. Pair of coefficient used for *Original TOPSIS*

Skill Matrix	δ	v_1	v_2	v_3	v_4
Q1	0	0.1	0.6	0	0.3
	$0.3/ln(2)d_j$	0.2	0.3	0.3	0.2
	$0.7/ln(2)d_j$	0.2	0.2	0.2	0.4
Q2	0	0	0.2	0.8	0
	$0.3/ln(2)d_j$	0.3	0.3	0.2	0.2
	$0.7/ln(2)d_j$	0.1	0.1	0.3	0.5
Q3	0	0.5	0.1	0.3	0.1
	$0.3/ln(2)d_j$	0.3	0.1	0.1	0.5
	$0.7/ln(2)d_j$	0.5	0	0.2	0.3
Q4	0	0.1	0.1	0.7	0.1
	$0.3/ln(2)d_j$	0.5	0.3	0.1	0.1
	$0.7/ln(2)d_j$	0.1	0.1	0.5	0.3

FIGURE 7. Pair of coefficient used for *New TOPSIS*

are shown in Figure 7.

The model of our simulation depends on many probabilistic laws (exponential distributions for product arrivals, uniform laws for task processing) and that is why the MFT also follows a distribution law and observes some variations (we will bring more details in the next section). We could not used the results of a single day of simulation. So we decided to simulated each case with 5000 days. These days are independent because as we said it, workers have to process all the daily tasks before

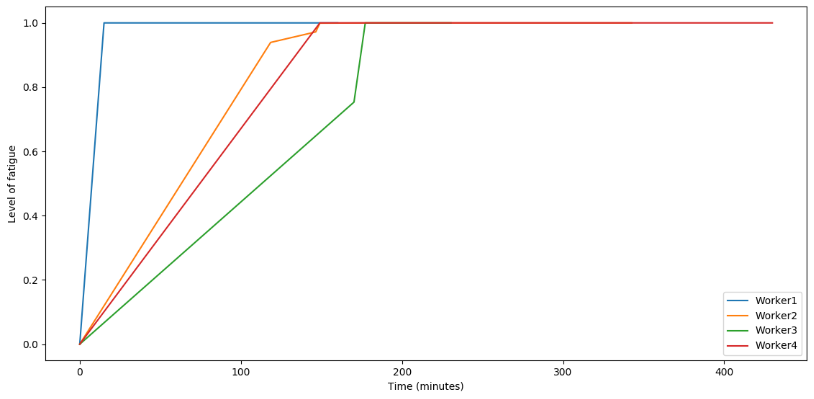

FIGURE 8. Evolution of the fatigue on a single day with the first skill matrix and a method of random assignment $\left(\delta = \frac{0.3}{ln(2)d_j}\right)$

Skill Matrix	δ	v_1	v_2
Q1	0	0.19	0.81
	$0.3/ln(2)d_j$	0.35	0.65
	$0.7/ln(2)d_j$	0.36	0.64

FIGURE 9. Pairs of coefficient used for *Original TOPSIS* in the problem one worker - two machines

δ	v_1	v_2	v_3	v_4
0	0.4	0.3	0.1	0.2
$0.3/ln(2)d_j$	0.4	0	0.3	0.3
$0.7/ln(2)d_j$	0.2	0.4	0.1	0.3

FIGURE 10. Set of coefficient used for *New TOPSIS* in the problem one worker - two machines

leaving and the level of fatigue of each worker is equal to zero at the beginning of a working day. For each simulation we computed 5000 runs and:

- The average MFT.
- The variance of the different MFTs.
- The average time which is necessary to do all the daily tasks.

We add to this study one more case in addition to the first four: a case with only one worker and two machines. The products arrivals are entirely known before the beginning of the day. The worker is going to do 15 tasks, randomly distributed between the two machines. The length of each task follows a uniform integer law with parameters $[20, 25]$. The penibility coefficients are the same than the first two machines of the previous problem.
We compute the weight coefficients for *Original TOPSIS* and *New TOPSIS* with the same method described previously. However we used a sample of 1000 days for *Original TOPSIS* and 30 days for *New TOPSIS*.

4.2. **Results.**

	Method	Average	Standard deviation
$\delta = 0$	Random	2699	60
	SPT	2686	63
	Original TOPSIS	2696	58
	New TOPSIS	2682	61
	Dynamic Programming	2673	58
$\delta = 0.3/ln(2)d_j$	Random	3509	79
	SPT	3489	81
	Original TOPSIS	3507	77
	New TOPSIS	3486	78
	Dynamic Programming	3474	75
$\delta = 0.7/ln(2)d_j$	Random	4589	105
	SPT	4563	105
	Original TOPSIS	4583	101
	New TOPSIS	4584	101
	Dynamic Programming	4544	100

FIGURE 11. Results for a problem with one worker and two machines offline

	Method	Average	Standard deviation	Processing time
$\delta = 0$	Random	2943	2024	618
	SPT	2621	1964	606
	Original TOPSIS	2600	1875	607
	New TOPSIS	2942	2136	617
	Dynamic Programming	2424	1576	605
$\delta = 0.3/ln(2)d_j$	Random	4912	3131	740
	SPT	4183	3211	713
	Original TOPSIS	4165	2977	717
	New TOPSIS	4492	3149	730
	Dynamic Programming	3804	2564	713
$\delta = 0.7/ln(2)d_j$	Random	7123	4729	936
	SPT	6555	4865	888
	Original TOPSIS	6450	4668	894
	New TOPSIS	6789	4642	915
	Dynamic Programming	5882	3905	885

FIGURE 12. Results for the first skill matrix

4.3. **Comments.** We are initially interested in the simplified case with only one worker and two machines. The improvement bring by our heuristics are not very consequential (circa 1% compared to *Random*). In all cases, *Dynamic Programming* reach the lowest MFT which shows the efficiency of the algorithm.

The results on the first skill matrix (Figure 10) are also quite positive ! Each heuristic found a better solution than the average one. In the three cases, *Dynamic Programming* is the most efficient heuristic to minimize the MFT. It is also the heuristic which provides the most stable results (in

	Method	Average	Standard deviation	Processing time
$\delta = 0$	Random	1940	1128	560
	SPT	1961	1185	564
	Original TOPSIS	1976	1197	562
	New TOPSIS	1975	1148	562
	Dynamic Programming	1991	1186	564
$\delta = 0.3/ln(2)d_j$	Random	2965	1887	632
	SPT	3046	1987	634
	Original TOPSIS	2941	1829	630
	New TOPSIS	3057	2016	636
	Dynamic Programming	3038	2000	635
$\delta = 0.7/ln(2)d_j$	Random	4512	2927	747
	SPT	4640	3213	751
	Original TOPSIS	4459	2899	749
	New TOPSIS	4619	3134	750
	Dynamic Programming	4666	3065	755

FIGURE 13. Results for the second skill matrix

	Method	Average	Standard deviation	Processing time
$\delta = 0$	Random	2105	1287	568
	SPT	2031	1216	560
	Original TOPSIS	2065	1256	562
	New TOPSIS	2046	1213	561
	Dynamic Programming	2009	1198	560
$\delta = 0.3/ln(2)d_j$	Random	3154	2000	645
	SPT	3113	2006	631
	Original TOPSIS	3122	2052	635
	New TOPSIS	3093	1967	631
	Dynamic Programming	3118	2015	632
$\delta = 0.7/ln(2)d_j$	Random	4728	2969	770
	SPT	4741	3145	745
	Original TOPSIS	4752	3060	764
	New TOPSIS	4732	3138	745
	Dynamic Programming	4643	2980	739

FIGURE 14. Results for the third skill matrix

each case, the variance is the lowest). For the processing time, it is difficult to have a good analyse because even if *Dynamic Programming* has a shorter processing time than average, some heuristics provides the same processing time for a bigger MFT (for instance *Original TOPSIS* and *SPT* for $\delta = 0$).
As we look at the first skill matrix, we can clearly explain these results (for the MFT). Each worker is only able to work on two machines and there is no other worker who is able to work on it. *Dynamic Programming* is based on a problem with one worker/two machines, this explains its good behavior.

	Method	Average	Standard deviation	Processing time
$\delta = 0$	Random	1827	971	556
	SPT	1816	995	554
	Original TOPSIS	1807	970	555
	New TOPSIS	1785	949	553
	Dynamic Programming	1807	956	555
$\delta = 0.3/ln(2)d_j$	Random	2638	1476	610
	SPT	2634	1463	613
	Original TOPSIS	2641	1479	614
	New TOPSIS	2620	1481	611
	Dynamic Programming	2672	1522	613
$\delta = 0.7/ln(2)d_j$	Random	3854	2224	706
	SPT	3888	2240	707
	Original TOPSIS	3811	2179	705
	New TOPSIS	3861	2185	710
	Dynamic Programming	3899	2294	710

FIGURE 15. Results for the fourth skill matrix

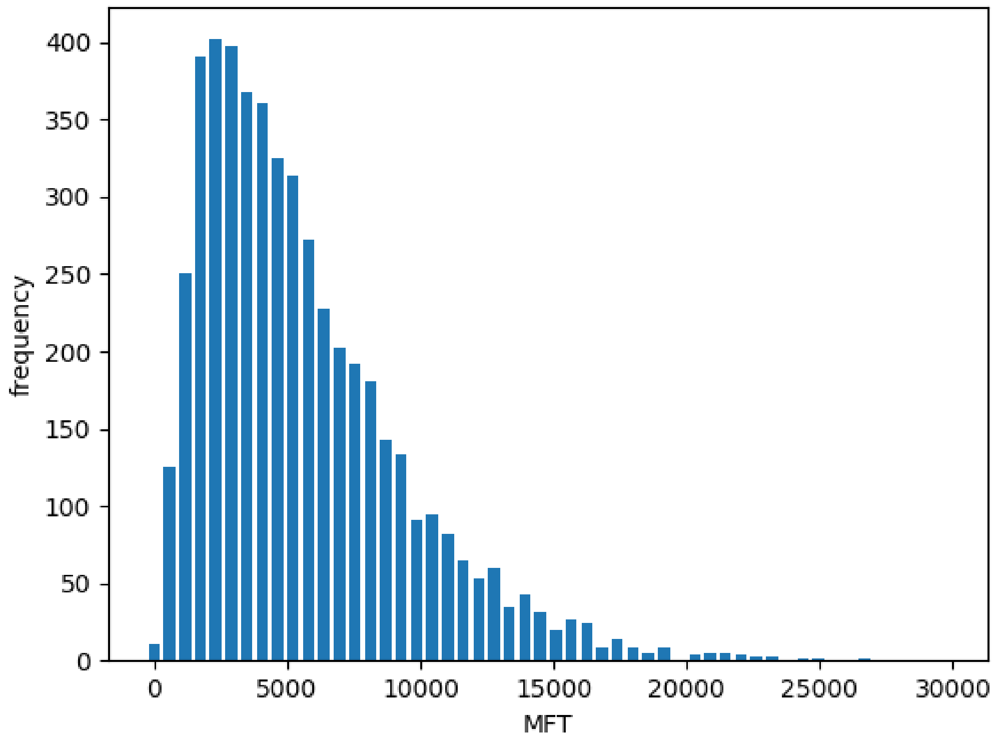

FIGURE 16. Frequency distribution of the MFT for *Dynamic Programming* during 5000 days, first skill matrix, $\delta = 0.7/ln(2)d_j$

However, the results on the second skill matrix are less positive. No method can do better than the *Random* one. The increase in the level of qualification of workers and the emergence of conflicts of interest between workers on certain machines add complexity to the decision-making process. Apparently, none of the proposed method is able to find a better way to solve this problem. There is still a hierarchy between the *Original Topsis* and *SPT* as it was written and described in the original study.
For this case, the decision-making criterion has to be further study.

On the third skill matrix, the results are clearly positive again. However the improvement is less more significant compared to the first skill matrix (lower than 5% compared to *Random* in this case, about 21% in the first test case). *Dynamic Programming* obtained the best results for $\delta = 0$ and $\delta = 0.7/ln(2)d_j$ and *New TOPSIS* is the best heuristic for $\delta = 0.3/ln(2)d_j$.
In this case, there are more workers (six), but they are less qualified compared to the second test case. They are able to work on a maximum of two machines and there are "interferences" between the assignments.

On the fourth test case, we have six workers who are all very qualified.
Results are also positive. *New TOPSIS* is the best heuristic for the first two test cases. *Original TOPSIS* managed to reach the minimum score for the third test case. However *SPT* is the most stable method for the second test case by reaching the lowest standard deviation.

The frequency distribution of the MFT always look the same. On Figure 16, there is an example of a such law. This law seems to be in the form of a Fisher law. Generally speaking, this law is very spread out and the average is in the lower part of the distribution function. In some cases, the MFT explodes and can reach values almost 4 times higher than its average.

5. Conclusion

Excepted for the second test case, results are encouraging. We manage to do better than the original study in most cases. Standard deviation is often linked to the MFT (the best heuristic is also the most stable one). However, our study is just preliminary : the weight parameters of each TOPSIS are roughly optimized. On the other hand, it also means that a method like *New TOPSIS* has a good potential ! Furthermore, *Dynamic Programming* can also be improved by expanding the sub-problem from two machines to n machines.

Heuristics presented do not take in consideration the synergy between the different workers, or the fatigue's distribution between workers. It is also an area for improvement and it can clearly be the subject of further study...