THE FLOW SHOP SCHEDULING PROBLEM MODELED BY MEANS OF TIMED PLACE PETRI NETS*

J. MEDINA-MARIN†, J. C. SECK-TUOH-MORA‡, N. HERNANDEZ-ROMERO§, A. KARELIN¶ and F. NUÑEZ-PIÑA‖

Engineering Department, Autonomous University of Hidalgo State
Pachuca, Hidalgo 42000, Mexico
†*jmedina@uaeh.edu.mx,*
‡*jseck@uaeh.edu.mx,*
§*nhromero@uaeh.edu.mx,*
¶*karelin@uaeh.edu.mx,*
‖*fede30net@gmail.com*
www.uaeh.edu.mx

D. GRADIŠAR

Department of System and Control, Jožef Stefan Institute
Ljubljana, 1000, Slovenia
dejan.gradisar@ijs.si

The Flow Shop Scheduling Problem (FSSP) is a problem that is commonly found by master production scheduling planners in Flexible Manufacturing Systems (FMS). The planner should find the optimal scheduling to carry out a set of jobs in order to satisfy the predefined objective (e.g., makespan). All the jobs are processed in a production line composed of a set of shared machines. Furthermore, the jobs are processed in the same sequence. In order to be able to analyze this problem in a better way, this problem needs to be represented adequately for understanding the relationship among the operations that are carried out. Thus, an FMS presenting the FSSP can be modeled by Petri nets (PNs), which are a powerful tool that has been used to model and analyze discrete event systems. Then, the makespan can be obtained by simulating the PN through the token game animation. In this work, we propose a new way to calculate the makespan of FSSP based on timed place PNs.

Keywords: Flow shop scheduling problem; makespan; Petri nets.

1. Introduction

Flexible Manufacturing Systems (FMSs) are very important in advancing factory automation due to the ability to adjust to customers' preferences and the

*This work was financially supported by the "Consejo Nacional de Ciencia y Tecnología - CONACYT-Mexico" within the Project Grant CB-2014-01-236818.

speed to reconfigure the system. A FMS is a discrete event dynamic system composed of jobs and shared resources [1]. When a manufacturer is designing the master production schedule in a FMS with shared resources, it is common that s/he has to face the decision about the best sequence of jobs in the FMS in order to carry all operations out in the minimum time [2], [3].

This problem is called the Flow Shop Scheduling Problem (FSSP), which is a combinatorial problem classified as NP-hard [4]. The makespan is the time that all the jobs are processed in the FMS, and it depends on the order that all the tasks are performed.

There have been published several research papers about finding the minimum value of makespan in the FSSP. For instance, a D.S. Palmer proposed a method to find an acceptable sequence in less time than exhaustive search [5]. Another algorithm based on heuristic strategies to find suitable solutions was proposed in reference [6]. Dannenbring performed a similar work, where he proposed eleven heuristics to solve the FSSP [7]. Nawas proposed an algorithm based on the assumption that jobs with higher processing time must be treated first; his algorithm is applied to static and dynamic sequencing environment [8]. In reference [9], Taillard applied taboo search to solve FSSP; moreover, he implemented a parallel version of taboo search to improve the algorithm execution time. Framinan and Leisten proposed a heuristic taking into account the optimization of partial schedules; instead of optimize the whole schedule [10]. Later, Framinan, Leisten and Ruiz-Usano proposed two multi-objective heuristics, whose objectives to solve are makespan and flowtime minimization [11].

Several metaheuristics have been used to find the minimum value for the makespan, such as Simulated Annealing [12],[13]; Taboo Search [14], [15]; Genetic Algorithms [16]–[18]; Ant Colony Optimization [19], [20]; Iterated Local Search [21]; and Particle Swarm Optimization Algorithms [22], [23], [27]. These proposals can find reasonable results in less time than exact methods. The main outcome of these methods is that the global minimum could not be found; however, good approximations are obtained in a short time. Thus, all of them need a way to represent the FSSP in order to calculate the makespan. FSSP modeling should be understandable and able to calculate the makespan of a job operations sequence.

FMSs have been modeled via Petri Nets (PNs) in order to simulate and analyze them. PN theory is adequate to represent in a graphical and mathematical way Discrete Event Systems (DES) such as FMSs, because their dynamic behavior based on event occurrence can be modeled by PN elements (places and transitions) [24]. Moreover, PN theory offers analytical and graphical tools to study the modeled systems, based on the relationship among the FMS resources denoted as PN elements. In [29], a timed Petri net is applied to model and simulate a production system, which is generated algorithmically.

One important point in search methods is the calculus of the makespan, taking into account a certain processing order of the tasks. In this paper, we propose the use of an timed PN to calculate the makespan taking into account the PN transition firing.

2. Flow Shop Scheduling Problem

Scheduling tasks in a FMS is a typical combinatorial problem where it is needed to organize the processing of a set of jobs divided in operations, and each operation is carried out in a shared resource [25], [26].

In the FSSP, given the processing times p_{jk} for each job j on every machine k, and a job sequence $S = (s_1, s_2, \ldots, s_n)$ where n jobs ($j = 1, 2, \ldots, n$) will be processed by m machines ($k = 1, 2, \ldots, m$), so the aim of FSSP is to find a sequence order for operation processing with the minimum value for the makespan.

For instance, Table 1 shows a FMS with three machines, four jobs, and each job has three serial operations, one for each machine.

Table 1. Operation times in a FMS.

Machines	Jobs			
	J_1	J_2	J_3	J_4
M_1	96	74	13	71
M_2	90	57	5	23
M_3	35	91	7	38

Note: Every value is denoted in a time unit.

3. Petri Nets Concepts

A PN is a graphical and mathematical tool that has been used to model concurrent, asynchronous, distributed, parallel, non-deterministic, and/or stochastic systems.

The graph of a PN is directed, with weights in their arcs, and bipartite, whose nodes are of two types: places and transitions. Graphically, places are depicted as circles and transition as boxes or bars. PN arcs connect places to transitions or transition to places; it is not permissible to connect nodes of the same type. The state of the system is denoted in PN by the use of tokens, which are assigned to place nodes.

A formal definition of a PN is presented as follows [24].

A Petri net is a 5-tuple, $PN = (P, T, F, W, M_0)$ where:
$P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,
$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,
$F \subseteq \{P \cdot T\} \cup \{T \cdot P\}$ is a set of arcs,
$W = F \rightarrow \{1, 2, 3, \ldots\}$ is a weight function,
$M_0 = P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking,
$P \cap T = \varnothing$ and $P \cup T \neq \varnothing$.

The set of places that are connected to a transition is known as input places, which is denoted as •t. On the other hand, the places connected from a transition are known as output places, and the set of output places are represented by t•.

The token movement through the PN represents the dynamical behavior of the system. In order to change the token position, the following transition firing rule is used [24]:

1. A transition $t \in T$ is enabled if every input place $p \in P$ of t has *w(p,t)* tokens or more. *w(p,t)* is the weight of the arc from *p* to *t*.
2. An enabled transition *t* will fire if the event represented by *t* takes place.
3. When an enabled transition *t* fires, *w(p,t)* tokens are removed from every input place *p* of *t* and *w(t,p)* tokens are added to every output place *p* of *t*. *w(t,p)* is the weight of the arc from *t* to *p*.

3.1. *Timed Place Petri Nets*

PN transitions, places, and arcs can be assigned with a time unit, meaning a time delay defined according to a FMS that considers time in its operations.

A Timed Place Petri Net (TPPN) is an extended PN, where a new element is added. It is a six-tuple $TPPN = \{P, T, F, W, M_0, D)$, where the first fifth elements are similar to PN definition presented above, and $D = \{d_1, d_2, \ldots, d_m\}$ denotes the time-delay for each place $p_j \in P$ [28]. Output transitions t_i for each p_j will be enabled once the time indicated in p_j is reached.

3.2. *Analysis Methods of Petri Nets*

In this chapter, we are applying the matrix equation approach as the analytical method of PN theory in order to calculate the makespan of the FMS modeled.

3.2.1. *Incidence Matrix and State Equation*

A *PN* with *n* transitions and *m* places can be expressed mathematically as an $n \cdot m$ matrix of integers $A = [a_{ij}]$. The values for each element of the matrix are given by: $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$, where a_{ij}^{+} is the weight of the arc from t_i to p_j, and a_{ij}^{-} is the weight of the arc from p_j to t_i.

The state equation is used to determine the marking of a PN after a transition firing, and it can be written as follows:

$$M_k = M_{k\text{-}1} + A^T U_k, \text{ k=1,2,…} \tag{1}$$

where U_k is a $n \times 1$ column vector of n - 1 zeros and one nonzero entries, which represents the transition t_j that will fire. The nonzero entry is located in the position j of U_k. A^T is the transpose of incidence matrix. M_{k-1} is the marking before the firing of t_j. And M_k is the reached marking after the firing of t_j denoted in U_k.

3.2.2. *Reachability Analysis Method*

All the possible states that a system can reach from the initial state of a PN can be derived by using the reachability tree or graph [1].

There are two ways to generate the reachability tree of a Petri net from an initial marking: depth-first and breadth-first. In the depth-first strategy, all the enabled transitions are identified and one of them is fired, creating a new marking. If it is a marking that has been created previously or a marking that has no enabled transitions, stop exploring it, and return to the preceding marking; next, continue with the unexplored transitions. Otherwise, from the new marking, identify the enabled transitions and fire one of them. Continue with these steps until all the transitions have been fired and all markings have been generated if the number of markings is finite.

In the second strategy, all the enabled transitions are identified and fired, generating new markings. For every new marking, that is not old or end marking, again identify and fire all the enabled transitions of the same level. Then, repeat the steps described above to the following levels of the reachability tree until all the levels have been explored.

4. FSSP modeled by a Timed Place Petri Net

In this work we are proposing a different way to obtain the makespan by using timed place PNs. The main idea is to denote every flow shop operation by a simple PN structure composed of one place denoting the operation time, one input transition t_j to place p_i, and one output transition t_{j+1} from place p_i. (Figure 1).

Thus, the processing time τ is stored in the place between the transitions, and it corresponds to the operation time defined in the FSSP. For each operation of job J_i performed in machine M_i there is a processing time τ_{ij}. (Table 2).

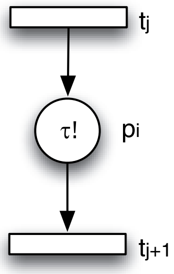

Fig. 1. PN structure denoting one single operation of a job, which is processed in a shared machine during τ time units.

Table 2. Matrix for operation times in a FMS.

Machines	Jobs				
	J_1	J_2	J_3	J_4	…
M_1	τ_{11}	τ_{12}	τ_{13}	τ_{14}	…
M_2	τ_{21}	τ_{22}	τ_{23}	τ_{24}	…
M_3	τ_{31}	τ_{32}	τ_{33}	τ_{34}	…
…	…	…	…	…	…

The first operation of the first job has no dependencies from another operations, and it starts immediately; however, remaining operations depend on the previous operation in the same machine, the previous operation of the same job, or both (Figure 2). Indeed, PN modeling allows setting dependencies among operations and it is taken into advantage in order to define the operations sequence of the FSSP.

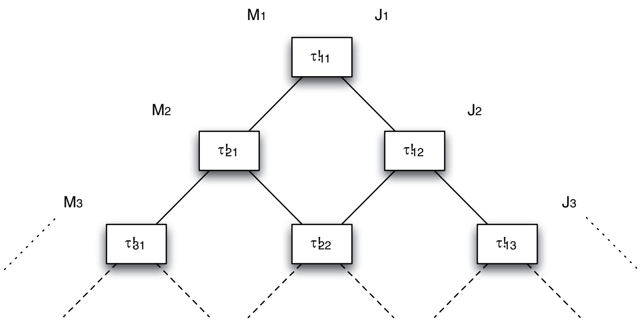

Fig. 2. Job operations dependency from previous operations of the same job, and/or previous operations performed in the same machine.

As we mentioned above, the first job is processed instantly in every machine as soon as the previous machine finishes. Similarly, the first machine processes all the operations as soon as the previous operation job is processed. In these cases, the operations have only one dependency and the accumulative time is the sum of the processing time for all the operations of the first job, or the sum for the operations processed in the first machine. However, the other operations take into account the maximum time from two previous operations, the previous operation in the same machine and the previous operation of the same job.

Once the job operations have been denoted as PN structures, it is important to link them in order to create a unique PN model representing the whole flexible manufacturing system.

Linking lines in figure 2 are represented as places connecting PN transitions for each job operation. Hence, any FMS with a number of shared machines and a number of jobs can be modeled by PNs as shown in figure 3.

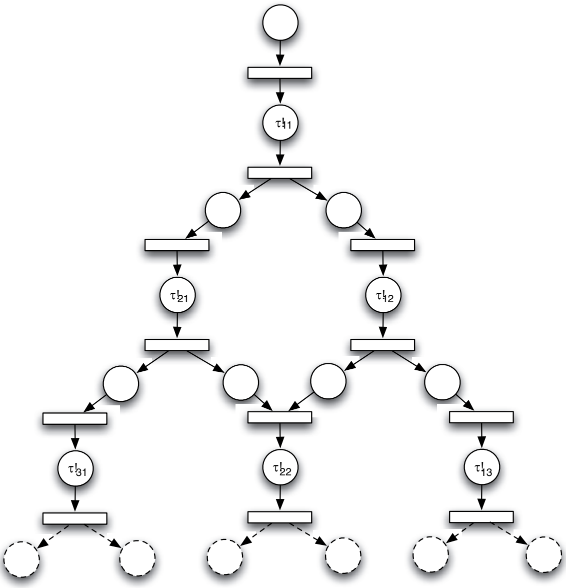

Fig. 3. PN model representing the job operations of a FMS (figure 2), where processing time τ is stored in PN places.

For instance, Table 1 shows the processing times needed for three machines that will process four jobs. All jobs are processed in the same order in all machines. Every value represents the processing time τ needed by an operation O_{ij}. which belongs to a job J_j and it is carried out in a machine M_i.

The PN model for this example is shown in figure 4. It contains twelve PN structures representing every job operation, an input place, an output place, and seventeen connecting places. Thus, the initial marking M_0 is a vector with 31 elements, 30 of them are zero and the 13th element is equal to 1.

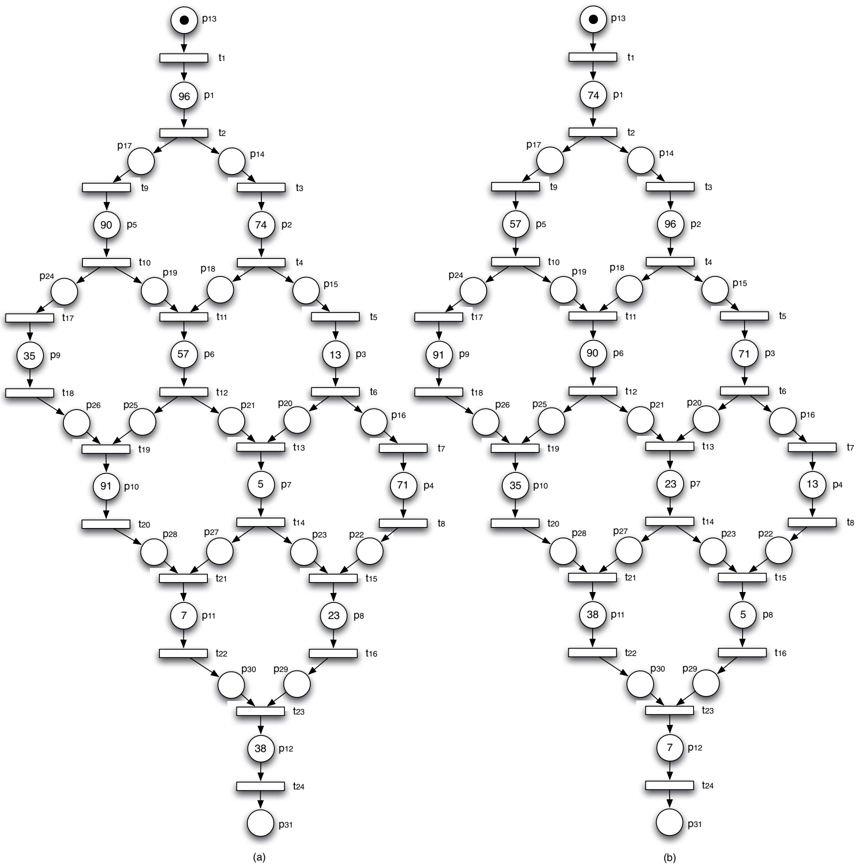

Fig. 4. PN model representing a FMS (figure 2), where processing time τ is stored in PN places. (a) TPPN model for the job sequence [1 2 3 4]. (b) TPPN model for the job sequence [2 1 4 3].

The TPPN model generated by the algorithm `create_TPPN` taking into account the job sequence order is shown in figure 4(a). For a different sequence order the PN structure is the same; however, the processing time values are assigned to different places, according to the desired order. For instance, for the job sequence order [2 1 4 3] the TPPN obtained is denoted in figure 4(b).

5. Algorithms

5.1. *Algorithm utilized to create the PN model*

In this algorithm the PN model is created from the processing time data for every job processed in each machine. The output is the PN model that represents the FMS.

Algorithm `create_TPPN`
Input: τ_{ij}, `OS`
Output: `PN`

```
1. Initialize variables
place = 1
trans = 1
2. For i = 1 to NumberOfMachines
For j = 1 to NumberOfJobs
        Aout(trans,place) = 1
        trans = trans+1
        Ain(trans,place) = 1
        trans = trans+1
        place = place+1
End For
End For
3. For i = 1 to NumberOfMachines
     For j = 1 to NumberOfJobs
       if(i==1 && j==1)
         Ain(1,place) = 1
         pos = place;
         place = place+1
       elseif i==1
         Aout(2*(j-1),place) = 1
         Ain(2*j-1,place) = 1
         place = place+1
       elseif j==1
         Aout(2*(i-2)*nj+2*j,place) = 1
         Ain(2*(i-1)*nj+1,place) = 1
         place = place+1
       else
         Aout(2*(i-2)*nj + 2*j,place) = 1
         Ain(2*(i-1)*nj+2*j-1,place) = 1
         place = place+1
         Aout(2*nj*(i-1)+2*(j-1),place) = 1
         Ain(2*nj*(i-1)+2*j-1,place) = 1
         place = place+1
       End If
    End For
End For
4. Aout(nt,place) = 1
5. PN = Aout - Ain
```

In Step 1 the variables `place` and `trans` are initialized to 1. In Step 2, the PN structure for every operation O_{ij} is created, it contains one place and two transitions. Next, in Step 3 the places to link the PN structures obtained in Step 2 are added to the PN model. In Step 4 the last place is connected to the PN. Finally, in Step 5, the PN is obtained from the subtraction of output arcs (`Aout`) minus input arcs (`Ain`).

5.2. *Algorithm applied to calculate the makespan*

Once the PN model is obtained, we are able to compute the makespan of the FMS according to a job operation sequence. The algorithm to perform this calculus is the following.

Algorithm `getMakespan`
Input: `PN, S, M0, D`
Output: `Makespan`

```
1. Initialize variables
Dacum = [0 0 0 … 0]
2. For i = 1 to NumberOfTransitions
// input places to ti
    ip = •ti
    // output places from ti
    op = ti•
    max = -∞;
    For j=1 to |ip|
        If Dacum(ip(j)) > max
            max = Dacum(ip(j));
        End If
    End For
    For k=1 to |op|
        Dacum(op(k)) = max + D(op(k));
    End For
  End For
3. Makespan = Dacum(NumberOfPlaces)
```

In Step 1, the variable Dacum is initialized to a vector with zero values. This variable is utilized to accumulate the total time needed to perform all the operations. In Step 2, every enabled transition is fired, and the maximum accumulated time from its input places is taken and placed in the output place plus the corresponding time τ. In Step 3 the accumulated time is assigned to the variable `Makespan`.

5.3. *Example*

For the job sequence [1 2 3 4], whose TPPN shown in figure 4(a), the calculus of the makespan is as follows. Vector D contains the processing times for every machine in the system (figure 5a). At the beginning, `Dacum` has only zero values, and this vector has 31 elements. Starting the PN simulation by firing the enabled transition t_1, the value 96 is assigned to `Dacum` in the position of place p_1. Next, the enabled transition t_2 is fired and its firing assigns the value 96 to `Dacum` in the position of its output places p_{14} and p_{17}. After that, enabled transition t_3 is fired, and the accumulative value in its input place p_{14} (96) plus the processing time in D in the position of its output place p_2 (τ=74) is assigned to `Dacum` in the position of p_2 (96+74=170).

All the transitions are fired in the same way, but in the case of two input places, the maximum accumulative time from both input places is considered for the sum of the total time.

In this example, the total time needed to process all the jobs is 379 time units, as shown in figure 5(b).

(a)

Vector D

p1	p2	p3	p4	p5	p6	p7	p8	p9	p10	p11	p12	p13	p14	p15	p16	p17	p18	p19	p20	p21	p22	p23	p24	p25	p26	p27	p28	p29	p30	p31
96	74	13	71	90	57	5	23	35	91	7	38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)

Vector Dacum

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t1	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t2	96	0	0	0	0	0	0	0	0	0	0	0	0	96	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t3, t9	96	170	0	0	186	0	0	0	0	0	0	0	0	96	0	0	96	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t4, t10	96	170	0	0	186	0	0	0	0	0	0	0	0	96	170	0	96	170	186	0	0	0	0	186	0	0	0	0	0	0	0
t5, t11, t17	96	170	183	0	186	243	0	0	221	0	0	0	0	96	170	0	96	170	186	0	0	0	0	186	0	0	0	0	0	0	0
t6, t12, t18	96	170	183	0	186	243	0	0	221	0	0	0	0	96	170	183	96	170	186	183	243	0	0	186	243	221	0	0	0	0	0
t7, t13, t19	96	170	183	254	186	243	248	0	221	334	0	0	0	96	170	183	96	170	186	183	243	0	0	186	243	221	0	0	0	0	0
t8, t14, t20	96	170	183	254	186	243	248	0	221	334	0	0	0	96	170	183	96	170	186	183	243	254	248	186	243	221	248	334	0	0	0
t15, t21	96	170	183	254	186	243	248	277	221	334	341	0	0	96	170	183	96	170	186	183	243	254	248	186	243	221	248	334	0	0	0
t16, t22	96	170	183	254	186	243	248	277	221	334	341	0	0	96	170	183	96	170	186	183	243	254	248	186	243	221	248	334	277	341	0
t23	96	170	183	254	186	243	248	277	221	334	341	379	0	96	170	183	96	170	186	183	243	254	248	186	243	221	248	334	277	341	0
t24	96	170	183	254	186	243	248	277	221	334	341	379	0	96	170	183	96	170	186	183	243	254	248	186	243	221	248	334	277	341	**379**

Fig. 5. (a) Vector D stores the time needed for every machine M_i to process each job J_j. (b) Reachability tree denoting the evolution of vector Dacum depending on the fired transitions.

On the other hand, applying the algorithm `getMakespan` to the TPPN of figure 4(b) representing the job sequence [2 1 4 3], the total time required to finish all the jobs is 340 time units.

6. Conclusion

The nature of flexible manufacturing systems allows the use of shared resources; however, this versatility produces complications when the manufacturers think about the best sequence to process all the jobs. One of these is know as Flow Shop Scheduling Problem, which is a NP-hard problem that have been analyzed applying different kinds of techniques, such as exact models and heuristics strategies. One important calculus in the FSSP is the makespan value, which depends on the sequence of operations for each job and the order of machine utilization.

In this work we propose a timed place PN model to represent the processing times in a FMS with a number of jobs ready to be processed, and a number of machines utilized to process the jobs. Moreover, two algorithms are described. The first one is used to create the PN model from the time processing needed for each job operation. And the second algorithm obtains the total time required to finish all the jobs in a defined job sequence.

As further work, we are applying this PN representation in a heuristic method in order to find an acceptable job sequence to find a minimum makespan.

References

1. M.C. Zhou, and K. Venkatesh, *Modeling, Simulation, and Control of Flexible Manufacturing Systems*. (World Scientific, NY, 1999).
2. M.L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, Fourth Edition, (Springer, NY, 2012).
3. J.K. Lenstra, A.H.G. Kan, and P. Brucker, Complexity of machine scheduling problem, *Annals of Discrete Mathematics*, vol. 1, pp. 343-362. (1977).
4. A.H.G. Rinnooy Kan, Machine Scheduling Problems: Classification, *Complexity and Computations*, Nojhoff, The Hague. (1976).
5. D.S. Palmer, Sequencing jobs through a multistage process in the minimum total time: A quick method of obtaining a near-optimum, *Operational Research Quarterly*, vol. 16, pp. 101-107. (1965).
6. H.G. Campbell, R.A. Dudek, and M.L. Smith, A heuristic algorithm for the n job, m machine sequencing problem, *Management Science*, vol. 16, no. 10, pp. B630-B637. (1970).
7. D.G. Dannenbring, An evaluation of flow shop sequencing heuristics, *Management Science*, vol. 23, no. 11, pp. 1174-1182. (1977).

8. M. Nawaz, E.E. Enscore Jr., and I. Ham, A heuristic algorithm for the m-machine, n-job flow shop sequencing problem, *Omega*, vol. 11, no. 1, pp. 91-95. (1983).
9. E. Taillard, Some efficient heuristic methods for the flowshop sequencing problems, *European Journal of Operational Research*, vol. 47, pp. 65-74. (1990).
10. J .M. Framinan, and R. Leisten, An efficient constructive heuristic for flowtime minimisation in permutation flow shops, *Omega*, vol. 31, pp. 311-317. (2003).
11. J.M. Framinan, R. Leisten, and R. Ruiz-Usano, Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimisation, *European Journal of Operational Research*, vol. 141, pp. 559-569. (2002).
12. I.H. Osman, and C. Potts, Simulated annealing for permutation flow shop scheduling, *Omega*, vol. 17, no. 6, pp. 551-557. (1989).
13. F. Ogbu, and D. Smith, The application of the simulated annealing algorithm to the solution of the n/m/Cmax flowshop problem, *Computers and Operations Research*, vol. 17, no. 3, pp. 243-253. (1990).
14. J. Grabowski, and M. Wodecki, A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion, *Computers and Operations Research*, vol. 31, no. 11, pp. 1891-1909. (2004).
15. E. Nowicki, and C. Smutnicki, A fast tabu search algorithm for the permutation flowshop problem, *European Journal of Operational Research*, vol. 91, pp. 160-175. (1996).
16. T. Aldowaisan, and A. Allahverdi, New heuristics for no-wait f1owshops to minimize makespan, *Computers and Operations Research*, vol. 30, no. 8, pp. 1219-1231. (2003).
17. T. Murata, H. Ishibuchi, and H. Tallaka, Genetic algorithms for f1owshop scheduling problems, *Computers and Industrial Engineering*, vol. 30, no. 4, pp. 1601-1071. (1996).
18. R. Ruiz, C. Maroto, and J. Alcaraz, Two new robust genetic algorithms for the flowshop scheduling problems, *Omega*, vol. 34, pp. 461-476. (2006).
19. C. Rajendran, and H. Ziegler, Ant-colony algorithms for permutation f1owshop scheduling to minimize makespan/total flowtime of jobs, *European Journal of Operational Research*, vol. 155, no. 2, pp. 426-438. (2004).
20. T. Stutzle, An ant approach to the f1owshop problem, In: *Proceedings of the 6th European Congress on Intelligent Techniques and Soft Computing (EUFIT'98)*, (Verlag Mainz, Aachen, Germany, 1998).

21. T. Stutzle, Applying iterated local search to the permutation f1owshop problem, Technical Report, AIDA-98-04, Darmstad University of Technology, Computer Science Department, Intellctics Group, Darmstad, Germany. (1998).
22. M.F. Tasgetiren, M. Sevkli, Y.C. Liang, and G. Gencyilmaz, Particle swarm optimization algorithm for permutation flowshop sequencing problem, In: *Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence (ANTS2004)*, LNCS 3172. (Brussels, Belgium, 2004).
23. M.F.Tasgetiren, Y.C. Liang, M. Sevkli, and G. Gencyilmaz, Particle swarm optimization algorithm for makespan and total f1owtime minimization in the permutation f1owshop sequencing problem, *European Journal of Operational Research*. (2006).
24. T. Murata, Petri Nets: Properties, Analysis and Applications, *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541-580. (1989).
25. M.A. Gonzalez-Hernandez, Metaheuristics solutions for Job-Shop Scheduling Problem with sequence-dependent setup times, PhD Thesis. University of Oviedo. (2011).
26. R. Qing-dao-er-ji, and Wang, Y. A new hybrid genetic algorithm for job shop scheduling problem, *Computers and Operations Research*, vol. 39. (2012).
27. Q.K. Pan, M.F. Tasgetiren, and Y.C. Liang, A Discrete Particle Swarm Optimization Algorithm for the Permutation Flowshop Sequencing Problem with Makespan Criterion, *Research and Development in Intelligent Systems XXIII*. (Springer London, 2007).
28. Z. Zhao, G. Zhang, and Z. Bing, Scheduling Optimization for FMS Based on Petri Net Modeling and GA, Proceedings of the *IEEE International Conference on Automation and Logistics.* (Chongqing, China, 2011).
29. D. Gradišar, and G. Mušic, Production-process modeling based on production-management data: a Petri-net approach, *International Journal of Computer Integrated Manufacturing*, vol. 20, Issue 8. (2007).
30. J. Medina-Marin, D. Gradisar, J. C. Seck-Tuoh-Mora, N. Hernandez-Romero, and F. Nunez-Piña, "A Petri Net Model to obtain the Makespan in the Flow Shop Scheduling Problem," Lecture Notes in Engineering and Computer Science: Proceedings of *The World Congress on Engineering and Computer Science 2016*, 19-21 October, 2016, San Francisco, USA, pp. 788-792.