پانزدهمین کنفرانس بین المللی مهندسی صنایع
15th Iran International Industrial Engineering Conference
Jan 23-24, 2019
۳ و ۴ بهمن ۱۳۹۷
Yazd University
Iran Institute of Industrial Engineering

Solving a single-server parallel machine problem with preventive maintenances using a hyper-heuristic algorithm

Zahra Izadi, , Homa Amirian, Rashed Sahraeian*
Department of Industrial Engineering, Shahed University, Tehran, Iran

* Corresponding author: Rashed Sahraeian, sahraeian@shahed.ac.ir

Abstract
This paper considers a parallel machine problem subject to a single server, machine unavailability due to preventive maintenances, setup times and release dates. The goal is to minimize the makespan and the total idle times, simultaneously. Since this special case of parallel machine is known to be NP-hard, a hyper-heuristic method is proposed to solve the model. The hyper-heuristic has two levels; on the lower level, a set of well-known algorithms in scheduling is defined. The selection policy is presented in the upper level where, considering the convergence and diversity of the current solutions, the most appropriate algorithm is chosen from the toolbox. The advantage of a hyper-heuristic lies in its flexibility in switching between different methods to improve the exploration and exploitation of the overall procedure. The proposed hyper-heuristic is validated and tested on different problem scales and the results emphasize its high precision and rapid convergence.

Keywords: Scheduling; parallel machines; single server; maintenance activity; hyper-heuristics

1. Introduction

Scheduling, as a decision making process plays an important role in production systems [1]. Generally, the goal of a scheduling problem is to find the best possible sequence of jobs/machines to optimize one or multiple objectives [2]. The current paper studies the parallel machine environment with the goal of minimizing the makespan and the total idle times. In parallel machine systems, jobs can be processed by any of the available uniform machines which are similar in performance and speed. The key in finding the optimal schedule in parallel machine environment is balance. With this rationale, in an optimal sequence, the workload is distributed between the available machines. For simplicity, most studies on scheduling assume that the machines are always available, however in reality, this assumption may not be the true [3]. Generally, there are two main causes which lead to machine unavailability [4] incidents (e.g. machine breakdown), and b) preventive measures (e.g. periodical repairs and system upgrades). Ergo, maintenance activities in production systems can be in turn carried out in two ways: reactive or proactive. In reactive mode, the maintenance activities are administered in response to a failure in the system. The duration it takes to fix the problem (known as the down time) leads to productivity loss. On the other hand, in proactive mode, the maintenances are planned and executed periodically to prevent the occurrence of a major failure in the system. Initially, proactive measures cost more than their reactive counterparts, however in the long term, proactive planning has been proved to result in more profitable plans and reliable systems [4]. In regard to proactive mode, [6] studied parallel machines by considering the repair and maintenance periods with intervention. Furthermore, [7] studied the two parallel machines problem with offline/online scheduling to determine unavailability periods. Some researchers examined the unrelated parallel machine problem with aging effects and multi-maintenance activities [8]. They assume that the output of each machine after repairing is similar to its original output. Similarly, the present paper focuses on the proactive/preventive maintenance activities to improve the reliability and long-term productivity of the system.

With the increase in automation across many industries, human involvement in production lines are reduced. In many cases, an operator is merely used for un/loading the machines or to supervise the process. Ergo, usually a single operator is assigned to several machines, who should then alternate between overseeing the jobs on different machines. The current study focuses on the case of a single server (operator) getting assigned to two parallel machines who has the responsibility of setting up each job on one of the machines. During the setup time of a job on a machine, the simultaneous demands for the presence of the server lead to an idle time on the other machine [98, 10 and 110].

Since the single-server parallel machine systems with setup times, maintenance activities and the objective of minimizing the makespan are among NP-hard problems [12], heuristics and meta-heuristics are among common methods used to tackle the problem. For instance, [13] use multi-objective ant colony algorithm to determine the best combination of task-machine assignments as well as preventive maintenance periods. [14] Tackle the preventive

maintenance in scheduling systems using six adaptations of existing heuristics and metaheuristics to optimize the system availability and the makespan. [15] Propose several heuristics based on branch-and-bound method for identical parallel machine scheduling problem with planned maintenances to minimize the sum of completion times. Similarly, [16] use an innovative genetic algorithm to minimize the total tardiness of a parallel machine system. [10] Consider the problem of two parallel machines with the goal of minimizing the makespan. In their problem, each job has a setup time which needs to be carried out by a single server. They propose a simulated annealing and a genetic algorithm to tackle the problem. However, instead of a meta-heuristic or a heuristic algorithm, the present paper proposes a hyper heuristic method which uses the advantages of both in solving a problem. Hyper-heuristic algorithms are classified among algorithm configuration methods and automatic algorithm selection techniques [17]. In designing hyper-heuristics, there are two main levels: the upper level (decision-making stage) and the lower level (processing stage). In each iteration, based on the current status of the achieved solution, the first level chooses the best solution technique among the algorithms of the second level. For instance, if the hyper-heuristic is trapped in the local optima, the first level decides to continue the algorithm with a solution technique that increases the diversity of the solutions. Ergo, the combined set of the algorithms defined in the second level becomes important. In other words, these solution techniques should be chosen in a way to cover each other's weaknesses and increase the chance of rapid convergence and diverse solutions. As a result, a hyper-heuristic algorithm can include several heuristics, meta-heuristics or even exact methods [18].

The rest of this paper is organized as follows. Section 2 proposes the mathematical formulation to model the problem. Section 3 describes the proposed hyper-heuristic in details. The validation tests and discussions on the performance of this algorithm are explained in section 4. Finally, the paper is concluded in section 5.

2. Problem definition

The problem examined in this study can be explained as follows. There are two parallel machines which are overseen by a single server. Initially, the server sets up a job on the first machine, the duration of which is called the setup time. During the setup time of a job on the first machine, the second machine is idle since the single server is busy on the first machine. Once the server is done with loading the first machine, the machine starts processing the job and the server is free to load the next job on the second machine. To equalize the workloads between the machines, the server alternates between the first and second machines so that jobs with odd numbers (job 1, job 3, job 5, etc.) are carried out on the first machine and the jobs with even numbers (job 2, job 4, job 6, etc.) are processed on the second machine. The objectives are the simultaneous minimization of the makespan ( ) and the total forced idle times on the machines due to the absence of a server ( ). In parallel machine environment, the reduction of the idle times decreases the makespan, however, the reverse is not necessarily true since the decrease in the makespan can increase/decrease or have no effects on the total idle times. Ergo the total idle times and the makespan do not necessarily have a direct relation and so can be considered simultaneously in a multi-objective problem.

The following assumptions are considered in modelling the problem:

- Machines are the same in terms of capability and speed.
- At any given point in time, each machine can process one job or maintenance activity, and vice versa, each job/maintenance activity can be scheduled on only one machine.
- Pre-emption is not allowed. In other words, once a job starts, it has to continue uninterrupted until it is completed.
- Machines are unavailable during the maintenance activities.
- Jobs have similar priorities, with their own release dates, setup times and processing times.
- Setup times are considered independent of the sequence and the processing times.
- During the loading time, both the machine and the server are busy.
- The number and the duration of maintenance activities are known in advance but their optimal positions (i.e. the point in time when they are best to be carried out) are decided by the model.
- If a maintenance activity is scheduled after a job, the completion time of that job is assumed to be the summation of its finish time and the maintenance duration scheduled after it.

2.1. Mathematical formulation

In this section, the problem is formulated based on the original model proposed by [9]. The proposed model modifies their work by adding the maintenance activities and the release dates to the formulation as well as considering multiple objectives in the problem. The remaining of this section models the problem using the notations described in Table 1 as follows.

Table 1. Notations used in the proposed model

Parameters and coefficients	
$j = 1,...,n$	Job index
$i = 1,...,n$	Position index
$m = 1,...,M$	Maintenance index
s_j	Setup time of job j
p_j	Processing time of job j
r_j	Release time of job j (Earliest possible start time)
d_{im}	Duration of the m^{th} maintenance activity if it is scheduled after the job in the i^{th} position
α $(0 \le \alpha \le 1)$	Objective weights
Decision variables	
x_{ij}	If job j is scheduled in the i^{th} position, it equals to 1, and otherwise 0
y_{im}	If the m^{th} maintenance activity is performed after the completion of the job in the i^{th} position, it equals to 1 and otherwise 0
ST_i	Start of the setup time of the job in i^{th} position

CL_i	End of the setup time of the job in i^{th} position
CT_i	Completion time of the job in i^{th} position - If a maintenance activity is scheduled after the job in i^{th} position, this variable equals to the sum of its setup time, the processing time and the maintenance duration.
IT_i	Forced idle time occurred between processing the jobs in positions i and $i-1$ while the single server is busy

$$minimize \quad Z = \alpha f_1 + (1-\alpha) f_2, \quad \begin{cases} f_1 = C_{max} \\ f_2 = \sum_{i=1}^{n-1} IT_i \end{cases} \tag{1}$$

$s.t.$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad \forall j \tag{2}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i \tag{3}$$

$$\sum_{i=1}^{n} y_{im} = 1 \qquad \forall m \tag{4}$$

$$\sum_{m=1}^{M} y_{im} \le 1 \qquad \forall i \tag{5}$$

$$\sum_{j=1}^{n} x_{ij} (s_j + p_j) = CT_i - ST_i - \sum_{m=1}^{M} y_{im} d_{im} \qquad \forall i \tag{6}$$

$$\sum_{j=1}^{n} x_{ij} s_j = CL_i - ST_i \qquad \forall i \tag{7}$$

$$ST_i \ge CL_{i-1} \qquad \forall i = 2,...,n \tag{8}$$

$$ST_i \ge CT_{i-2} \qquad \forall i = 3,...,n \tag{9}$$

$$IT_1 = CL_1 - ST_1 \tag{10}$$

$$IT_i = ST_{i+1} - CT_{i-1} \qquad \forall i = 2,...,n-1 \tag{11}$$

$$ST_i \geq r_i \qquad \forall i \tag{12}$$

$$ST_i, CT_i, CL_i, IT_i \geq 0 \qquad \forall i \tag{13}$$

$$x_{ij} \in \{0,1\}; \forall i, j \quad y_{im} \in \{0,1\}; \forall i, m \tag{14}$$

The objective functions f_1 and f_2 in Equation (1), minimize the makespan (i.e. the maximum completion time of all jobs) and the total idle times, respectively. Since the objectives are both a function of time to be minimized, simple additive weighting technique is used to combine the two objectives into a single function (Z) via the weights α and $1-\alpha$. Equations (2) and (3) ensure that each job is assigned to only one position and each position is filled by only one job, respectively. Equation (4) assigns each maintenance activity to a single position, i.e. each maintenance activity starts only once. Also, maintenance activities cannot follow each other consecutively. Ergo, constraint (5) ensures that after the completion of a job, either no maintenance activity is performed or only one maintenance activity is scheduled to be carried out. Equation (6) formulates the completion time of the job in the i^{th} position, considering the maintenance periods. Equation (7) determines the start and finish times of the setup procedure for the job in the i^{th} position. Constraint (8) ensures that the single server can perform only one job at a time. Constraint (9) balances the load on the two parallel machines so that the jobs are processed in a staggered order; i.e. jobs with odd and even numbers are respectively processed on the first and second machines. Equation (10) calculates the forced idle time on the second machine while the server is busy processing the first job on the first machine. Equation (11) determines the idle times of the remaining jobs on each machine. Constraint (12) ensures that the initialization of the setup time of a job in the i^{th} position is later than or equal to its release date. Finally, constraints (13) and (14) define the non-negative and binary decision variables, respectively. The proposed model has $13n+M-4$ constraints and $n(4+n+M)$ decision variables.

2.2. An illustrative example

In this section, the proposed model is validated by a small numerical example for 10 jobs, 4 maintenance activities, 2 parallel machines and a single server. The remaining parameters of this example are summarized in tables 2 and 3. For simplicity, the release dates are set as zero. The results found by solving the model using IBM ILOG CPLEX software are shown in tables 4 and 5.

Table 2. Setup times and durations

Jobs	1	2	3	4	5	6	7	8	9	10
Setup times	33	2	43	47	34	38	38	20	33	9
Durations	41	33	36	38	14	34	33	9	6	49

Table 3. Maintenance times for different positions

Positions	1	2	3	4	5	6	7	8	9	10
Maintenance 1	5	15	20	10	5	15	20	10	5	15
Maintenance 2	15	20	10	5	15	20	10	5	15	15
Maintenance 3	15	20	10	5	15	20	10	5	15	15
Maintenance 4	15	20	10	5	15	20	10	5	15	15

Table 4. Results of solving the model for the numerical instance

Positions	1	2	3	4	5	6	7	8	9	10

Jobs	2	6	5	8	10	4	7	1	3	9
Machine	1	2	1	2	1	2	1	2	1	2
Start of setup times	0	2	40	74	98	109	156	194	227	273
End of setup times	2	40	74	94	107	156	194	227	270	306
Completion times	40	74	98	108	156	194	227	273	306	312
Maintenance scheduled	1	-	2	4	-	-	-	3	-	-
Idle time on machine	2	-	-	-	2	-	-	-	-	-
Forced idle times	2	-	-	-	1	-	-	-	-	-

Assuming $\alpha = 0.5$, the results found from table 4 lead to $Z = 157.5$, $C_{\max} = 312$ and $\sum IT = 3$ for the optimal sequence of $2 \to 6 \to 5 \to 8 \to 10 \to 4 \to 7 \to 1 \to 3 \to 9$.

This solution corresponds to the following decision variables and the maintenance activities; $x_{12} = x_{26} = x_{35} = x_{48} = x_{510} = x_{64} = x_{77} = x_{81} = x_{93} = x_{109} = 1$, $y_{11} = y_{32} = y_{83} = y_{44} = 1$. The remaining decision variables for other jobs, positions or maintenances get the value of zero. This schedule is also illustrated in Figure 1 where notations 'S', 'P' and 'M' indicate the setup times, durations and maintenance activities, respectively. In this example, the idle times are occurred on machine 2, in time periods 0 to 2 and 108 to 109.

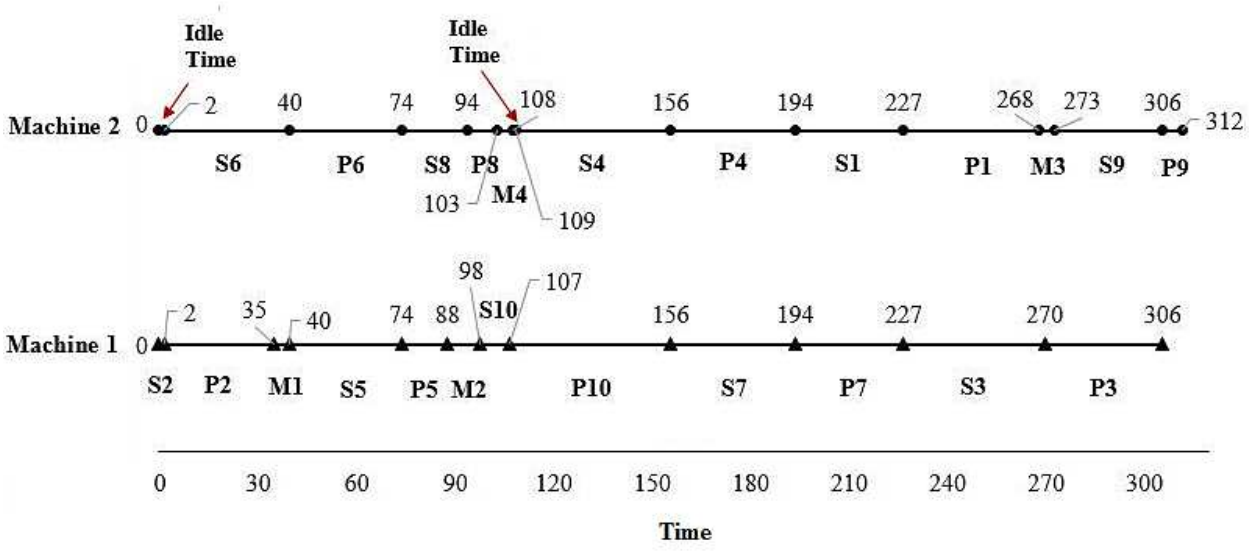

Figure 1. Resultant schedule of the numerical example

2.3. Testing on the problem scale

In order to find the largest problem scale that can be solved optimally with the proposed model, examples with one maintenance activity and up to 50 jobs are randomly chosen and solved using a PC with 4 GB of RAM, 1.7 GHz on a Win 7. The processing times and the relative gaps from the optimal solutions for different problem scales are summarized in table 5. As can be seen in table 5, this model can be solved optimally for up to 28 jobs, 2 parallel machines and a single server.

Table 5. Results of solving the model for different problem scales

Problem scale	Gap	Run time (sec)	Status
50 jobs	2.23%	2740.33	Out of memory

40 jobs	1.73%	8540.37	Out of memory
35 jobs	1.67%	2806.99	Out of memory
30 jobs	1.73%	8540.37	Out of memory
28 jobs	0.75%	3244.71	Optimal solution found

2.4. Testing on different objective weights

To see the effects of different weights on the objective values, $f_1 = C_{\max}$, $f_2 = \sum IT$ and Z, the model is solved via different α values, ranging from 0 to 1. Figure 2 shows the corresponding objectives for each α value. For simplicity, the normal logarithm of the objectives are calculated so that all the values can be shown in one figure. As can be seen in figure 2, with the increase of α, the makespan undergoes marginal reduction while the total tardiness values are increased. The combined effects of these objectives also increase the value of the final objective Z. In this figure, the optimal Pareto front is also pointed out. For instance points *A* and *B* in figure 2 indicate two non-dominated solutions which correspond to alpha values of 0.2 and 0.3, respectively. With the goal of minimizing both objectives, point *A* has lower tardiness values but higher makespan, while point *B* has higher tardiness but lower makespan. In other words, points *A* and *B* show two solutions which are as good as one another, and hence they are included in the Pareto front.

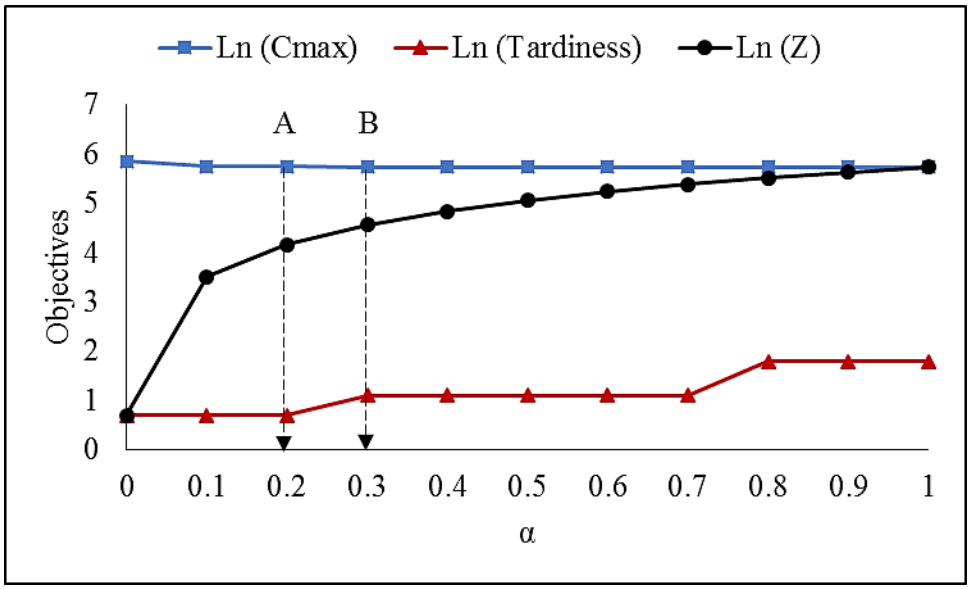

Figure 2. Objective values corresponding to different weights

3. The proposed hyper-heuristic algorithm

The proposed model in section 2 can be solved optimally for up to 28 jobs. However, for large scale problems, the optimal method fails to find a solution in an acceptable time. Therefore, this section proposes a hyper-heuristic method to tackle the large-scale problems. This method has the following features:

1. The sets of algorithms chosen for the lower level of the hyper-heuristic are the solution techniques proven to have high efficiency in solving scheduling problems [19]. These include the local search operators such as insertion, swap, reversion, shift-to-left (circular shift) as well as meta-heuristic algorithms such as tabu search and simulated annealing. Moreover, to prevent the hyper-heuristic from getting trapped in the local optima, the randomization operator is also added to the solution techniques of this level. Note that the contribution of this paper lies in the proper selection of these algorithms based on the current condition of the problem.

2. The conditional learning rationale of the upper level can be defined as follows. For the first 10 iterations, the proposed hyper-heuristic is allowed to randomly choose one of the algorithms of the lower level. Each algorithm is assigned a score which will be increased with a constant number if the algorithm is chosen. Upon the selection of one

of these algorithms from the lower level, the scores of the remaining algorithms decrease via the same constant number. After the first 10 iterations, the algorithm with the highest score is chosen for processing in each run.

3. The hyper-heuristic conducts the search algorithm both on the sequences of the jobs and the maintenance activities. Ergo in each iteration, using the old sequences, two new sequences are created for jobs and maintenances. Therefore three combinations of solutions are tested against the old sequences of jobs and maintenances: *a*) new job sequence and old maintenance sequence *b*) old job sequence and new maintenance sequence *c*) new job sequence and new maintenance sequence. Either of these three solutions can replace the old sequences provided that they improve the overall performance.

The overview of the proposed hyper-heuristic can be explained as follows.

Stage 1- Problem definition

- Set the number of maximum iterations ($maxit$)
- Define the problem (i.e. parameters of the jobs and the maintenances)

Stage 2- Initialization

- Create a random initial sequence for the jobs
- Create a random sequence for scheduling the maintenance activities
- Evaluate the initial sequences for the jobs and the maintenances via Equation (1)

Stage 3- Main loop

For each iteration $it = 1 : maxit$

- *if* $it < 10$

I. A random score is assigned to each of the six algorithms of the second level

- *else*

II. Choose the algorithm with the highest score among the first to sixth algorithms of the second level. Let a be the number of the selected algorithm.

  - If $a = 1$ select insertion operator
  - If $a = 2$ select reverse operator
  - If $a = 3$ select swap operator
  - If $a = 4$ select shift-to-left operator
  - If $a = 5$ select randomization operator
  - If $a = 6$ select the tabu search algorithm
  - If $a = 7$ select the simulated annealing algorithm

- Create two new sequences for the jobs and the maintenances by solving the problem with the selected algorithm
- Increase the score of the selected algorithm by $1/maxit$
- Decrease the scores of unselected algorithms by $1/maxit$
- Evaluate the three new solutions (new job sequence - old maintenance sequence; old job sequence - new maintenance sequence; new job sequence - new maintenance sequence) against the old solution (old job sequence - old maintenance sequence)
- Replace the old solution with one of the new solutions if the objective value is improved, otherwise continue with the old solution

End of the main loop

Stage 4- If the termination criteria is met, stop the algorithm and report the objective values

4. Results and discussions

In this section, small, medium and large scale instances are randomly generated and used to test the performance of the proposed hyper-heuristic against the exact method. The categorization is based on the number of jobs multiplied by the number of maintenances. If the multiplied result is less than 30, then the problem is a small scale problem. If the multiplied value is between 30 and 100, the problem is categorized as medium scale and if it is larger than 100, the problem is among large scale problems. The maximum iteration is used as the termination criteria and is experimentally set to 10000. In tables 6 and 7, the performance of the hyper-heuristic on the model is reported for different problem scales. The exact method is coded in CPLEX software, which uses the combination of branch and bound (B&B) and

branch & cut (B&C) approaches to tackle the problem. Assuming that any solution with a relative gap of lower than 2% is an optimal solution, in the small scale problems, both the exact and the hyper-heuristic methods find the optimal solutions in less than 3 seconds. For instance, in table 6, for the problem of 8 jobs and 3 maintenances, the exact method and the hyper-heuristic, both return the errors of zero for the *best* achieved solutions. However, the error of the *average* value for the exact method is zero, while this value for the hyper-heuristic is 1.26% which is an indicator of the randomness and the higher variance of the hyper-heuristic. For the medium scale problem, the hyper heuristic finds the *near* optimal solutions with minimum and maximum gaps of about 1% and 3%, respectively. As is expected, the hyper heuristic method returns higher variances with the worst variance found to be 0.1. For the large scale problems in table 7, CPLEX solver runs out of memory prior to reaching the last stages of branching. Ergo, the exact method returns the estimated solutions with about 6% to 8% gaps. Moreover, the exact method requires 36 to 77 minutes to achieve this relative solution. On the other hand, the proposed hyper-heuristic takes 2 to 3.5 minutes to achieve the solutions with lower gaps of approximately 2% to 8%.

In summary, the proposed hyper-heuristic has similar precision to the exact method for the small to medium scale problems. However, in the large scale problems, it achieves better results under less computational time. The convergence of the proposed algorithm is investigated in figure 3 for the problem of 50 jobs and 10 maintenance activities. As can be seen in figure 3, this convergence is rapid at first and the algorithm stalls between 5000 to 10000 iterations as it has achieved the best possible solution.

Table 6. Comparison of the hyper-heuristic and the exact method for small/medium problems*

Scale	Problem	Method	Run time (sec)	Errors%		Variance%	Status
				Best	Average		
Small	5J-2M**	Exact	1.38	0	0	0	Optimal solution
		Heuristic	1.52	0	1.87	0.057	
	8J-3M	Exact	2.75	0	0	0	
		Heuristic	2.72	0	1.26	0.025	
Medium	10J-4M	Exact	2.47	0	0	0	Optimal solution
		Heuristic	2.78	2.17	2.98	0.008	Near optimal
	15J-6M	Exact	33.50	0	0	0	Optimal solution
		Heuristic	34.53	0.9	2.87	0.105	Near optimal

* The solutions are the average of 10 runs.
** The notations 'J' and 'M' indicate the numbers of jobs and maintenances, respectively.

Table 7. Comparison of the hyper-heuristic and the exact method on large problems*

Problem	Method	Run time (sec)	Errors%	Variance%	Status
29J-4M**	Exact	2608.66	5.49	0	Out of memory
	Heuristic	116.67	6.17	1.328	Near optimal solution
30J-5M	Exact	2147.77	4.85	0	Out of memory
	Heuristic	182.95	1.80	0.315	Near optimal solution
40J-8M	Exact	4623.03	6.41	0	Out of memory
	Heuristic	209.90	3.40	0.125	Near optimal solution
50J-10M	Exact	3183.82	8.05	0	Out of memory
	Heuristic	164.01	7.97	16.385	Near optimal solution

* The solutions are the average of 10 runs.
** The notations 'J' and 'M' indicate the numbers of jobs and maintenances, respectively.

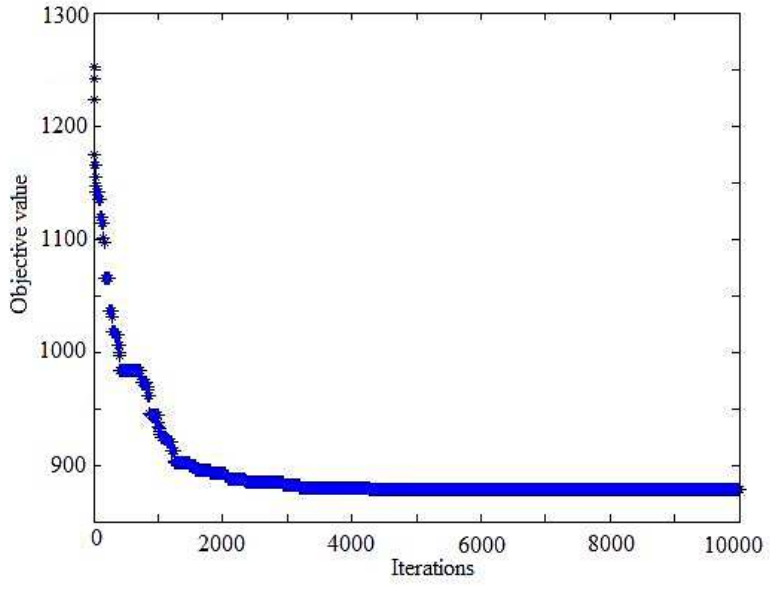

Figure 3. The convergence of the proposed hyper-heuristic

Figure 4 compares the computational time of the hyper-heuristic and the exact method for large scale problems. In this figure, it can be clearly concluded that the hyper-heuristic is significantly faster than the exact method for all the large scale problems.

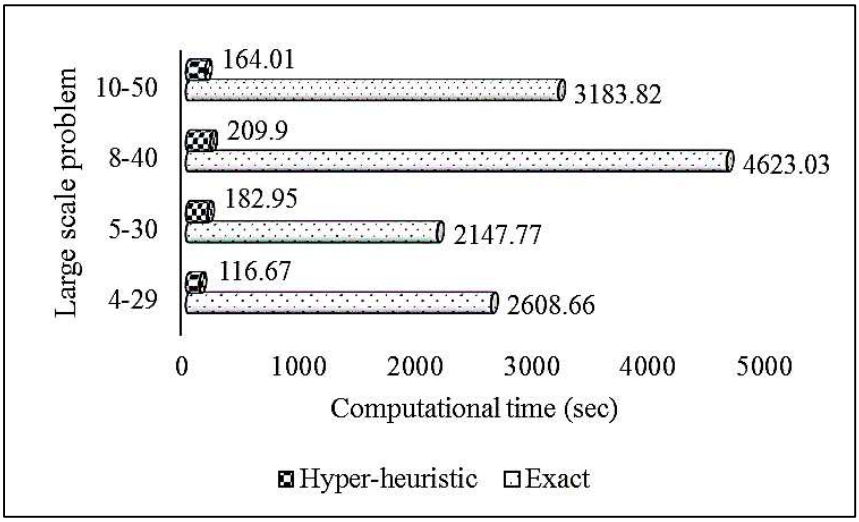

Figure 4. Computational times of the hyper-heuristic and the exa

5. Conclusion

Scheduling is one of the most prominent requirements of any production system. The present paper studies a special case of the parallel machine scheduling systems subject to the availability of a single server, maintenance periods, the setup times and the release dates. The problem is modelled via mixed integer programming with the goal of the simultaneous minimization of the maximum completion times and the total idle times. Next, a hyper-heuristic solution technique is proposed which uses a set of well-known algorithms in scheduling to tackle this problem. The hyper-

heuristic performs as well as the exact method in small to medium scale problems. In large scale problems, it achieves more precise results in less computational times than the exact method. This paper can be extended both in modelling and in the solution technique. The model can be extended by the addition of random breakdowns, different objective functions and consideration of more than two machines. The hyper-heuristic can be modified to accommodate new algorithms in the lower level and probability-based selection policies in the upper level.

6. References

[1] Pinedo, M (2010). Scheduling: Theory, Algorithms and Systems. New York: Springer-Verlag.
[2] Eshlaghi, A and S Sheibatolhamdy (2011). Scheduling in flexible job shop manufacturing system by improved tabu search. African Journal of Business Management, 5, 4863-4872.
[3] Ma, Y, Chu, C and C Zuo (2010). A Survey of scheduling with deterministic machine availability constraints. Computers & Industrial Engineering, 58, 199-211.
[4] Dalfard, V and G Mohammadi (2012). Two meta-heuristic algorithms for solving multi-objective flexible job shop scheduling with parallel machine and maintenance constraints. Computers & Mathematics with Applications, 64, 2111-2117.
[5] Wang, S and J Yu (2010). An effective heuristic for job shop scheduling problem with maintenance activities. Computers & Industrial Engineering, 59, 436-447.
[6] Xu, D, Sun, K and H Li (2008). Parallel machine scheduling with almost periodic maintenance and non-preemptive jobs to minimize makespan. Computers & Operations Research, 35, 1344-1349.
[7] Xu, D, Cheng, Z, Yin, Y and H Li (2009). Makespan minimization for two parallel- machines scheduling with a periodic availability constraint. Computers & Operations Research, 36, 1809-1812.
[8] Yang, D, Cheng, T C E, Yang, S and C Hsu (2012). Unrelated parallel- machine scheduling with aging effects and multi-maintenance activities. Computers & Operations Research, 39, 1458-1464.
[9] Hasani, K, Kravchenko, S and F Werner (2014a). Minimizing interference for scheduling two parallel machines with a single server. International Journal of Production Research, 52, 7148-7158.
[10] Hasani, K, Kravchenko, S and F Werner (2014b). Simulated annealing and genetic algorithms for the two-machine scheduling problem with a single server. International Journal of Production Research, 52, 3778-3792.
[11] Hasani, K, Kravchenko, S and F Werner (2014c). Minimizing total weighted completion time approximately for the parallel machine problem with a single server. Information Processing Letters, 114, 500-503.
[12] Glass, C, Shafransky, Y and V Strusevich (2000). Scheduling for parallel dedicated machines with a single server. Naval Research Logistics, 47, 304-328.
[13] Berrichi, A, Yalaoui, F, Amodeo, L and M Mezghiche (2010). Bi-objective ant colony optimization approach to optimize production and maintenance scheduling. Computers and Operations Research, 37, 1584-1596.
[14] Ruiz, R, Carcia-Diaz, J and C Maroto (2007). Considering scheduling and preventive maintenance in the flow shop sequencing problem. Computers & Operations Research, 34, 3314-3330.
[15] Mellouli, R, Sadfi, C, Chu, C and I Kacem (2009). Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. European Journal of Operational Research, 197, 1150-1165.
[16] Lee, W, Wang, J and L Lee (2015). A hybrid genetic algorithm for an identical parallel-machine problem with maintenance activity. Journal of the Operational Research Society, 66, 1906-1918.
[17] Messelis, T and P Causmaecker (2014). An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. European Journal of Operational Research, 233, 511–528.
[18] Misir, M (2012). Intelligent hyper-heuristics: A tool for solving generic optimization problems. Ph.D. Thesis, KU Leuven.
[19] Almeida, B, Correia, I and F Saldanha-da-Gama (2016). Priority-based heuristics for the multi-skill resource constrained project scheduling problem. Expert Systems with Applications, 57, 91–103.