RAIRO-Oper. Res. 53 (2019) 111–128
https://doi.org/10.1051/ro/2017086

RAIRO Operations Research
www.rairo-ro.org

THE CYCLE SHOP SCHEDULING: MATHEMATICAL MODEL, PROPERTIES AND SOLUTION ALGORITHMS

Bahman Naderia[1,2,*] and Sheida Goharib[3]

Abstract. Conventionally, in scheduling problems it is assumed that each job visits each machine once. This paper studies a novel shop scheduling called cycle shop problems where jobs might return to each machine more than once. The problem is first formulated by two mixed integer linear programming models. The characteristics of the problem are analyzed, and it is realized that the problem suffers from a shortcoming called redundancy, *i.e.*, several sequences represents the same schedule. In this regard, some properties are introduced by which the redundant sequences can be recognized before scheduling. Three constructive heuristics are developed. They are based on the shortest processing time first, insertion neighborhood search and non-delay schedules. Then, a metaheuristic based on scatter search is proposed. The algorithms are equipped with the redundancy prevention properties that greatly reduce the computational time of the algorithms. Two sets of experiments are conducted. The proposed model and algorithms are evaluated. The results show the high performance of model and algorithms.

Mathematics Subject Classification. 90C11, 90C59.

Received February 4, 2017. Accepted November 3, 2017.

1. Introduction

Scheduling involves utilizing of some limited resources to perform some jobs. Traditionally it is assumed that the resources are some machines that can process only one job at a time. Each job needs some operations for completion and each machine performs one operation type. If all jobs have the same processing route among the machines, the production system is called flowshops. If each job has its own processing route among the machines, the production system is job shops. If the processing route of jobs is not fixed in advance and can be determined by the scheduler, the system is open shops [18]. For more information, the reader is referred to [1, 2].

Note that in all the three abovementioned shop scheduling systems, each job needs a set of operations for completion and for each operation, there is one dedicated machine. Thus, each job visits each machine once. This paper considers a special class of scheduling problems where jobs might have to return to the same machine one

Keywords and phrases: Scheduling, cycle shop, mixed integer linear programming model, metaheuristic algorithm; constructive heuristics.

[1] Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran.
[2] Department of Mechanical, Automotive, and Materials, Faculty of Engineering, University of Windsor, Windsor, ON, Canada.
[3] Department of Systems Science and Industrial Engineering, State University of New York at Binghamton, New York, USA.

* Corresponding author: bahman.naderi@khu.ac.ir

or more times before completion [4, 21]. For example, assume one job with 6 operations and this processing route:

$$\{1\ 2\ 4\ 3\ 2\ 4\}$$

That is, this job requires operation type 1 once, operation type 2 twice, operation type 3 once and operation type 4 twice. We assume machines are dedicated. That is, each machine can process only one operation type. Therefore, this job visit machine 1 once, machine 2 twice and so on. The applications of such shops can be found in any industry that requires capital-intensive machines. In such an industry, rather than employing a machine to a particular operation, the manufacturer logically prefer that jobs make multiple visits to a machine for multiple-operation processing. One can refer the reader to a semiconductor manufacturing and printed circuit boards [20].

Let us describe the difference between the problem under consideration and two other well-known problems where jobs may visit each machine more than once (flexible job shops and reentrant flow shop). In the flexible job shops, machines are flexible; that is, a machine can carry our different operation types. Hence, one job might visit a machine more than once (*i.e.*, the same machine to process another operation type). This problem also differs from the reentrant flowshop in which jobs might visit machine more than once; yet, the processing route of jobs are all the same.

In the literature with emphasizing on multiple visits of a machine by a job, the problem of the reentrant flowshop scheduling is considered by Kang *et al.* [11], Hekmatfar *et al.* [10], Chen *et al.* [5], Cho *et al.* [7] and Dugardin *et al.* [8]. The reentrant job shop scheduling is also studied by Topaloglu and Kilincli [20] and Xie *et al.* [22].

As just reviewed, there is almost no paper studying the cycle shop. The first systematic step to study an operations research problem is to formulate it by the mathematical programming. Thus, this paper first develops two mathematical models in form of mixed integer linear programs. Using the model and the branch and bound algorithm of specified software CPLEX, the small-sized instances of the problem are solved to optimality. Yet, this approach is not effective for all instance sizes. Since the reentrant flow shop, a special case of the problem under consideration, is NP-hard [6], we can conclude this problem is also NP-hard. In this case, a metaheuristic based on scatter search is developed. This algorithm includes five mechanisms of diversification generation, improvement, reference set update, subset generation and solution combination. To design this algorithm, we use permutation encoding scheme of operations. We analyze this scheme to overcome its shortcoming of redundancy (different permutations represent the same solution). In the diversification generation mechanism, an insertion heuristic is developed to initiate the algorithm from quality solutions. The subset generation mechanism is designed so as to have both diversification and intensification capabilities. A fast and simple, but effective local search is applied as the improvement mechanism.

The rest of the paper is organized as follows. Section 2 develops the mathematical model of the problem. Section 3 introduces the properties of the problem. Section 4 proposes solution algorithms. Section 5 conducts the experiment to tune and evaluate the model and algorithms for performance. Section 7 concludes the paper and suggests some leads for future research.

2. Problem definition and formulations

In the cycle shop, there is a set of n jobs each of which requires n_j operations for completion. On the other hand, there is a set of m machines each of which performs one operation type. The different operations of a job might be the same operation type. Thus, the job might need to be processed by the same machine more than once. Each job has its own processing route (*i.e.*, the sequence in which a job visits machines) that can be different from the others. The objective is to sequence jobs on machines so as to minimize the completion time of the last job, or makespan.

To better illustrate the problem, one numerical example is presented. Consider an instance with $n = 4$ and $m = 3$ where $n_j = \{4, 3, 5, 5\}$. Table 1 shows the processing routes and processing times of the jobs. It means that job 1 must visit machine 2, and machine 3, then machine 1. After that it is again processed by machine 3.

TABLE 1. The processing route of the jobs.

Job	Processing route	Processing times
1	2-3-1-3	4-6-3-5
2	1-2-1	5-7-3
3	2-3-2-1-2	3-2-4-8-2
4	1-3-2-3-2	7-5-6-3-4

TABLE 2. The sequence of the jobs on machines.

Machine	Sequence
1	4-2-3-1-2
2	3-1-3-4-2-4-3
3	3-4-1-4-1

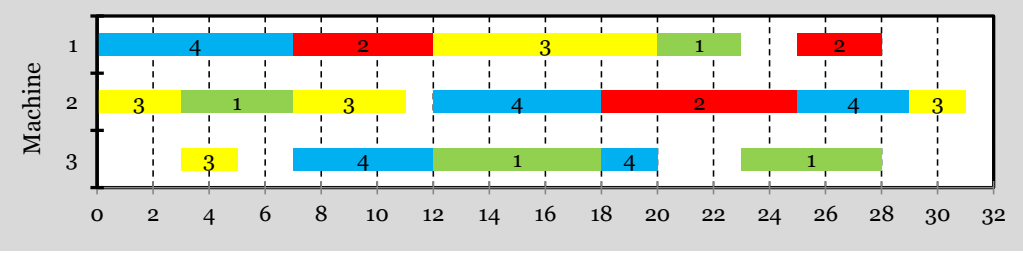

FIGURE 1. The Gantt chart of schedule of the solution. (Color online.)

Table 2 presents one possible solution to this instance. In this solution, machine 1 processes jobs 4, 2, 3, 1 and 2, respectively. Figure 1 shows the Gantt chart of the schedule obtained by this solution. The completion of jobs 1, 2, 3 and 4 are 28, 28, 31 and 29, respectively. Therefore, makespan of this schedule becomes 31. As it could be seen, job 3 visits machine 2 three times while it visits machines 1 and 3 once.

The cycle shop problem is now mathematically formulated. The construction of mathematical models for an operations research problem is the first key step. This is so because the mathematical model can express the all aspects of a problem explicitly [17]. It also can be used as an input for many solution algorithms like branch and bound [13]. More effective the model is constructed, more efficient the algorithm becomes. Moreover, regarding the great advances obtained in computers' capacity and specified software, the model can be used to solve the small or even medium size problems. The mathematical model of scheduling problem is commonly in form of mixed integer linear programming (MILP). For this problem, two MILP models are developed.

The following parameters and indices are used in the model.

n Number of jobs.
j, k Indices for jobs where $\{1, 2, \ldots, n\}$.
m Number of machines.
n_j Number of operations of job j.
$l,\ i$ Indices for operations of job j $\{1, 2, \ldots, n_j\}$.
$O_{j,l}$ lth operation of job j.
$p_{j,l}$ Processing time of $O_{j,l}$.

The following sets are also defined.

$E_{j,l}$ The set including all the operations of the other jobs being processed on the same machine with $O_{j,l}$.

$O_{j,l'}$ The last operation before operation $O_{j,l}$ in processing route of job j being processed on the same machine.
$O'_{j,l'}$ The first operation after operation $O_{j,l}$ in processing route of job j being processed on the same machine.
F_i The set of the first operations of all jobs if this operation is processed on machine i.

Note that we define a dummy job 0. It has m operations where each one is processed by one machine (*i.e.*, $O_{0,i}$ is processed on machine i). These operations are the first operation being processed on its corresponding machine. Each set $E_{j,l}$ includes the dummy operation of this job. Note that for the first operation of each job on each machine, there is no $O_{j,l'}$; and also for the last operation of each job on each machine, there is no $O'_{j,l'}$. The decision variables of the model are as such.

$X_{j,l,k,i}$ Binary variable taking value 1 if $O_{j,l}$ is processed immediately after $O_{k,i}$, and 0 otherwise (where $O_{k,i} \in E_{j,l}$).
$Y_{j,l}$ Binary variable taking value 1 if $O_{j,l}$ is processed immediately after $O_{j,l'}$, and 0 otherwise.
$Z_{j,l}$ Binary variable taking value 1 if $O'_{j,l'}$ is processed immediately after $O_{j,l}$, and 0 otherwise.
$C_{j,l}$ The completion time of $O_{j,l}$.

Let us remind that for the first operation of each job on each machine, we have no $Y_{j,l}$ and also for the last operation of each job on each machine, we have no $Z_{j,l}$.

The MILP model is as follows.

$$\text{Min } C_{\max} \tag{2.1}$$

Subject to:

$$\sum_{O_{k,i} \in E_{j,l}} X_{j,l,k,i} + Y_{j,i} = 1 \qquad \forall_{O_{j,l}} \tag{2.2}$$

$$\sum_{O_{j,l} \in E_{k.i} \ | j>0} X_{j,l,k,i} + Z_{k,l} \leq 1 \qquad \forall_{O_{k,i}} \tag{2.3}$$

$$\sum_{O_{j,l} \in F_i} X_{j,l,0,i} = 1 \qquad \forall_{i=\{1,2,\ldots,m\}} \tag{2.4}$$

$$C_{j,l} \geq C_{j,l-1} + p_{j,l} \qquad \forall_{O_{j,l}} \tag{2.5}$$

$$C_{j,l} \geq C_{k,i} + p_{j,l} - M\,(1 - X_{j,l,k,i}) \qquad \forall_{O_{j,l}, O_{k,i} \in E_{j,l}} \tag{2.6}$$

$$C_{\max} \geq C_{j,n_j} \qquad \forall_j \tag{2.7}$$

$$C_{j,l} \geq 0 \qquad \forall_{O_{j,l}} \tag{2.8}$$

$$X_{j,l,k,i} \in \{0,\ 1\} \qquad \forall_{O_{j,l}, O_{k,i} \in E_{j,l}} \tag{2.9}$$

$$Y_{j,l}, Z_{j,l} \in \{0,\ 1\} \qquad \forall_{O_{j,l}} \tag{2.10}$$

Equation (2.1) is the objective function. Constraint set (2.2) ensures that each operation is performed. Constraint set (2.3) assures that each operation has at most one succeeding operation. Constraint set (2.4) specifies that each operation of the dummy job 0 has one succeeding operation (since it is the first operation on the corresponding machine). Constraint set (2.5) ensures that a job cannot be processed by more than one machine at a time. Constraint set (2.6) specifies that a machine cannot process more than one operation at a time. Constraint set (2.7) calculates the makespan. Constraint sets (2.8), (2.9) and (2.10) define the decision variables.

The second model views the problem as series of sequencing decisions, not necessarily immediately. In this case, the dummy job 0 is not required. The decision variables of the second model are as follows.

$X_{j,l,k,i}$ Binary variable taking value 1 if $O_{j,l}$ is processed after $O_{k,i}$, and 0 otherwise (where $O_{k,i} \in E_{j,l}$ and $j<n, k>j$).

$C_{j,l}$ The completion time of $O_{j,l}$.

The second MILP model is as follows.

$$\text{Min } C_{\max} \tag{2.11}$$

Subject to:

$$C_{j,l} \geq C_{j,l-1} + p_{j,l} \qquad \forall_{O_{j,l}} \tag{2.12}$$

$$C_{j,l} \geq C_{k,i} + p_{j,l} - M\,(1 - X_{j,l,k,i}) \qquad \forall_{O_{j,l},O_{k,i} \in E_{j,l}, j<n,k>j} \tag{2.13}$$

$$C_{k,i} \geq C_{j,l} + p_{k,i} - M\,(X_{j,l,k,i}) \qquad \forall_{O_{j,l},O_{k,i} \in E_{j,l}, j<n,k>j} \tag{2.14}$$

$$C_{\max} \geq C_{j,n_j} \qquad \forall_j \tag{2.15}$$

$$C_{j,l} \geq 0 \qquad \forall_{O_{j,l}} \tag{2.16}$$

$$X_{j,l,k,i} \in \{0,\ 1\} \qquad \forall_{O_{j,l},O_{k,i} \in E_{j,l}} \tag{2.17}$$

where $C_{j,0} = 0$.

Equation (2.11) is the objective function. Constraint set (2.12) assures that a job cannot be processed by more than one machine at a time. Constraint sets (2.13) and (2.14) ensure that a machine cannot process more than one operation at a time. Constraint set (2.15) calculates the makespan. Constraint sets (2.16) and (2.17) define the decision variables.

3. The encoding scheme and its properties

The first step to solve a problem is to encode the problem, *i.e.*, making a solution recognizable for algorithms. The scheme to encode the problem remarkably influences the performance of any algorithm [14]. The most commonly used encoding scheme in scheduling problems is the permutation representation.

As discussed earlier, the objective of this problem is to schedule the operations of all jobs. Thus, a solution includes a permutation of operations. The encoding scheme used here can be described as follows. There are a set of n jobs and each job j has a set of n_j operations that must be processed on m machines. Each job number appears in the permutation as the number of its operations. By scanning the permutation from left to right, the k-th appearance of a job number represents the k-th operation of that job. As long as a job number repeats as the number of its operations, the solution is always feasible.

For example, consider a problem with 3 jobs where jobs 1, 2 and 3 have 3, 5 and 4 operations, respectively. The job number 2 repeats five times. For this problem, there are 12 operations including {1 1 1 2 2 2 2 2 3 3 3 3}. One possible solution is as follows.

$$\{3\ 2\ 1\ 3\ 2\ 1\ 3\ 2\ 3\ 1\ 2\ 3\}$$

According to this solution, the first operation of job 3 is scheduled. Then the first operation of job 2 is scheduled at the earliest possible time and so on.

This type of encoding scheme suffers from a noticeable shortcoming, called redundancy. It means that the same solution might be obtained despite the change in the positions of the operations in the permutation. In other words, different permutations might represent the same solution. The paper studies the redundancy in permutation list to overcome this serious disadvantage. For example, the algorithm exchanges two operations of the same job, no new permutation is generated since we use job numbers in the permutation.

Theorem 3.1. *Let S' be a permutation produced by the swap of two adjacent operations in permutation S. Two permutations S' and S are redundant if and only if the same machine does not process the swapped operations.*

Proof. Let two permutations S and S' be

$$S = \{A, O_{j,l}, O_{k,i}, B\} \quad S' = \{A, O_{k,i}, O_{j,l}, B\}$$

A and B are the sets of all the other operations and the same for both S and S'. If the jobs of two operations $O_{j,l}$ and $O_{k,i}$ are the same (*i.e.*, $j = k$), the two permutations are the same. Suppose the case the jobs are different (*i.e.*, $j \neq k$). Considering the encoded versions, we have

$$S = \{A, j, k, B\} \quad S' = \{A, k, j, B\}$$

Therefore, $S \neq S'$ and we have two different permutations (*i.e.*, encoded solutions). We now show that they both represent the same schedule. Without loss of generality, assume that machines l and i process operations $O_{j,l}$ and $O_{k,i}$, respectively. Thus, we have two cases. Assume $O_{j,l}$ and $O_{k,i}$ are processed by machines : l' and i'.
Case 1: $l' \neq i'$

Let us suppose after scheduling the operations of Part A, the completion time of machines (cm) and jobs (ct) are

$$\begin{aligned} cm &: (cm_1, \ldots, cm_{l'}, \ldots, cm_{i'}, \ldots, cm_m) \\ ct &: \ (ct_1, \ldots, ct_j, \ldots, ct_k, \ldots, ct_n) \end{aligned}$$

Note that cm is a vector showing the time that machines are available to process the next operatio and ct is a vector showing when jobs are available for next operation. It is clear that cm and ct of S and S' are the same. After scheduling $O_{j,l}$ and $O_{k,i}$ according to both S and S', we have the same

$$\begin{aligned} cm &: (cm_1, \ldots, cm_{l'} + p_{j,l}, \ldots, cm_{i'} + p_{k,i}, \cdots, cm_m) \\ ct &: \ (ct_1, \ldots, ct_j + p_{j,l}, \ldots, ct_k + p_{k,i}, \ldots, ct_n) \end{aligned}$$

Since partial permutation B is the same for both S and S', the remaining operations yield the same schedule.
Case 2: $l = i$

After scheduling $O_{j,l}$ and $O_{k,i}$ according to S, we have

$$\begin{aligned} cm &: (cm_1, \ldots, cm_{l'} + p_{j,l} + p_{k,l}, \ldots, cm_m) \\ ct &: \ (ct_1, \ldots, ct_j + p_{j,l}, \ldots, ct_k + p_{j,l} + p_{k,l}, \ldots, ct_n) \end{aligned}$$

according to S', we have

$$\begin{aligned} cm &: (cm_1, \ldots, cm_{l'} + p_{k,l} + p_{j,l}, \ldots, cm_m) \\ ct &: \ (ct_1, \ldots, ct_j + p_{k,l} + p_{j,l}, \ldots, ct_k + p_{k,l}, \ldots, ct_n)\,. \end{aligned}$$

Therefore, they end up with different cm and c. As a result, S and S' are two different schedule. Finally, two permutations S and S' are redundant if and only if the same machine does not process the swapped operations. This completes the proof. □

Theorem 3.2. *Let S' be a permutation produced by the swap of two non-adjacent operations in permutation S. Two permutations S' and S are redundant if it holds:*

1. *The same machine does not process the two operations.*
2. *The intermediate operations neither belong to the two jobs of the operations nor the two machines process them.*

Proof. It is clear that if the two swapped operation belong to the same job, no new permutation is generated. Hence, we only consider the case where the jobs are different. The proof of condition 1 is similar to Theorem 3.1. Therefore, only the second condition needs to be proved. Due to the more conceptual simplicity and easier comprehensibility, we prove this theorem for the case of two intermediate operations. The proof can be generalized to multiple intermediate operations as well.

Consider two different permutations S and S'

$$S = \{A, O_{j,l}, O_{z,h}, O_{x,y}, O_{k,i}, B\} \quad S' = \{A, O_{k,i}, O_{z,h}, O_{x,y}, O_{j,l}, B\}$$

The positions of two non-adjacent operations $O_{j,l}$ and $O_{k,i}$ are exchanged, while two operations $O_{z,h}$ and $O_{x,y}$ are intermediate operations. We know that jobs j and k are the same with job z and x. That is,

$$j \neq k, \quad z \neq j, \quad z \neq k, \quad x \neq j, \quad x \neq k$$

Without loss of generality, assume machines l', h', y' and i' process operations $O_{j,l}, O_{z,h}, O_{x,y}$ and $O_{k,i}$, respectively. For machines of these operations, we know that machines l' and $i^{'}$ are not the same with machine h' and y'. That is,

$$l' \neq i', \quad h' \neq l', \quad h' \neq i', \quad y' \neq l', \quad y' \neq i'$$

Note that we might also have either $z = x$ or $h = y$. After scheduling the operations in Part A, we have

$$cm : (cm_1, \ldots, cm_{l'}, cm_{h'}, cm_{y'}, cm_{i'}, \ldots, \ cm_m)$$
$$ct : \ (ct_1, \ldots, ct_j, ct_z, ct_x, ct_k, \ldots ct_n)$$

Now, if four operations of $O_{j,l}, O_{z,h}, O_{x,y}, O_{k,i}$ are processed according to both S and S', we have the same results as follows.

$$cm : (cm_1, \ldots, cm_{l'} + p_{j,l}, cm_{h'} + p_{z,h}, cm_{y'} + p_{x,y}, cm_{i'} + p_{k,i}, \ldots, \ cm_m)$$
$$ct : \ (ct_1, \ldots, ct_j + p_{j,l}, ct_z + p_{z,h}, ct_x + p_{x,y}, ct_k + p_{k,i}, \ldots ct_n)$$

Since the permutation of the operations in Part B are the same for both S and S', they become the same schedule. These results indicate that if conditions 1 and 2 are met; a redundant solution is obtained. This completes the proof for two intermediate operations. This procedure can be similarly generalized to the case of multiple intermediate operations. □

4. The proposed scatter search

Scatter search is a well-known evolutionary metaheuristic including effective features for both search diversification and intensification. Comparing with other evolutionary metaheuristics, SS more systematically generate new solutions. SS is inspired from this fact that combination of desirable solutions with diverse solutions can search the solution space more effectively. This strategy of combination obtains better empirical results. SS has been successfully applied to solve different combinatorial optimization problems [12].

Scatter search works with a set of solutions, called the reference set, and explores the solution space through this set. The reference set is, indeed, the collection of the quality solutions obtained during the search. Notice that the quality of a solution depends on not only its objective function value but also its diversification (*i.e.*, its difference with other solutions in the reference set). The solutions in the reference set are updated by new solutions generated by combining the solutions previously available in this set.

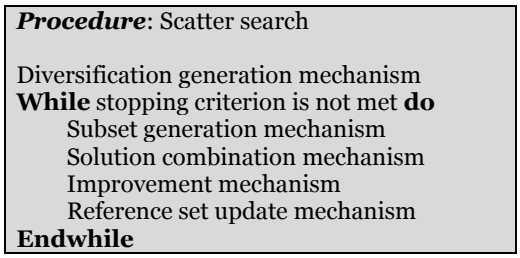

FIGURE 2. The general outline of scatter search.

The basic structure of SS consists of five mechanisms, and the novelty of a SS is related to the way these mechanisms are designed and implemented. The mechanisms are diversification generation, improvement, reference set update, subset generation and solution combination. The diversification generation mechanism is to generate both good and diverse solutions. The purpose of the improvement mechanism is to enhance the solutions of the reference set. The reference set update mechanism is to manage the solutions that enter and exit from the reference set to keep both good and diverse solutions. The subset generation mechanism is to define a subset of solutions as a basis for combining solutions. Finally, the solution combination mechanism to combine the solutions determined in previous mechanism. Figure 2 shows the general outline of basic SS.

Since the above-mentioned mechanisms are not restricted to a single uniform design, the structure of SS can be highly flexible and there are a variety of SSs designed in different way in the literature.

4.1. Diversification generation mechanism

In this mechanism, the initial solutions of the reference set are generated. It should be done in such a way to have both good and diverse solutions. To this end, we develop a heuristic, called iterative insertion heuristic (IH) as the generator of good solutions. IH is based on the well-known heuristic proposed by Nawaz *et al.* [15] for flowshop problems. In IH, there are two sets O and S including unscheduled operations and scheduled operations. IH can be explained as follows.

Step 1. Sort operations according to the longest processing time first in a set O.
Step 2. Replace each operation with the job number it belongs to.
Step 3. Take the first job number from set O into set S and remove it from set O.
Step 4. Take the first job number in set O and insert it into all possible positions in set S. To insert kth job number there are k possible positions among the scheduled operations. Consider lth occurrence of each job number as its lth operation and schedule operations according to k possible operation sequences of the previous step.
Step 5. Select the sequence with the lowest makespan and update set S accordingly. Remove the job number from set O.
Step 6. If $|O| = 0$, then stop; otherwise, go to Step 4.

TABLE 3. The data of the example with $n = 3$ and $m = 3$.

Job	The number of operations	Processing route	Processing times
1	3	1-3-2	3-8-6
2	4	1-2-3-2	6-2-3-5
3	3	2-3-1	2-4-7

The following example is provided to numerically clarify the procedure. Consider a problem with 3 jobs and 3 machines. Table 3 shows the data of this example including the number of operations, processing route and processing times.

The procedure:

Step 1. $O = \{O_{12}, O_{33}, O_{21}, O_{13}, O_{24}, O_{32}, O_{23}, O_{11}, O_{31}, O_{22}\}$
Step 2. $O = \{1, 3, 2, 1, 2, 3, 2, 1, 3, 2\}$
Step 3. $O = \{3, 2, 1, 2, 3, 2, 1, 3, 2\}$ and $S = \{1\}$

Iteration 1.

Step 4. Job = 3, $S'_1 = \{3, 1\}$ and $S'_2 = \{1, 3\}$
Step 5. $\begin{cases} C_{\max}(S'_1) = 3 \\ C_{\max}(S'_2) = 3 \end{cases} \rightarrow S = S'_1$ and $O = \{2, 1, 2, 3, 2, 1, 3, 2\}$
Step 6. $|O| \neq 0$

Iteration 2.

Step 4. Job = 2, $S'_1 = \{2, 3, 1\}$, $S'_2 = \{3, 2, 1\}$ and $S'_3 = \{3, 1, 2\}$
Step 5. $\begin{cases} C_{\max}(S'_1) = 9 \\ C_{\max}(S'_2) = 9 \\ C_{\max}(S'_3) = 9 \end{cases} \rightarrow S = S'_1$ $S = S'_1$ and $O = \{1, 2, 3, 2, 1, 3, 2\}$
Step 6. $|O| \neq 0$

Iteration 3.

Step 4. Job = 1, $S'_1 = \{1, 2, 3, 1\}$, $S'_2 = \{3, 1, 2, 1\}$, $S'_3 = S'_4 = \{3, 2, 1, 1\}$
Step 5. $\begin{cases} C_{\max}(S'_1) = 11 \\ C_{\max}(S'_2) = 17 \\ C_{\max}(S'_3) = 17 \\ C_{\max}(S'_4) = 17 \end{cases} \rightarrow S = S'_1$ and $O = \{2, 3, 2, 1, 3, 2\}$
Step 6. $|O| \neq 0$

Iteration 4.
$\vdots$
Iteration 9.

Step 4. Job = 2, $S'_1 = \{2, 3, 1, 3, 1, 2, 3, 2, 1, 2\}$, $S'_2 = \{3, 2, 1, 3, 1, 2, 3, 2, 1, 2\}$, $S'_3 = \{3, 1, 2, 3, 1, 2, 3, 2, 1, 2\}$, $S'_4 = \{3, 1, 3, 2, 1, 2, 3, 2, 1, 2\}$, $S'_5 = S'_6 = \{3, 1, 3, 1, 2, 2, 3, 2, 1, 2\}$, $S'_7 = S'_8 = \{3, 1, 3, 1, 2, 3, 2, 2, 1, 2\}$, $S'_9 = S'_{10} = \{3, 1, 3, 1, 2, 3, 2, 1, 2, 2\}$
Step 5. $\begin{cases} C_{\max}(S'_1) = 28 \\ C_{\max}(S'_2) = 28 \\ C_{\max}(S'_3) = 25 \\ C_{\max}(S'_4) = 25 \\ C_{\max}(S'_5) = 25 \\ C_{\max}(S'_7) = 25 \\ C_{\max}(S'_9) = 25 \end{cases} \rightarrow S = S'_3$ and $O = \emptyset$
Step 6. $|O| \neq 0$ and Stop.

The final solution becomes {3,1,2,3,1,2,3,2,1,2}. Figure 3 shows the Gantt chart of the final schedule.

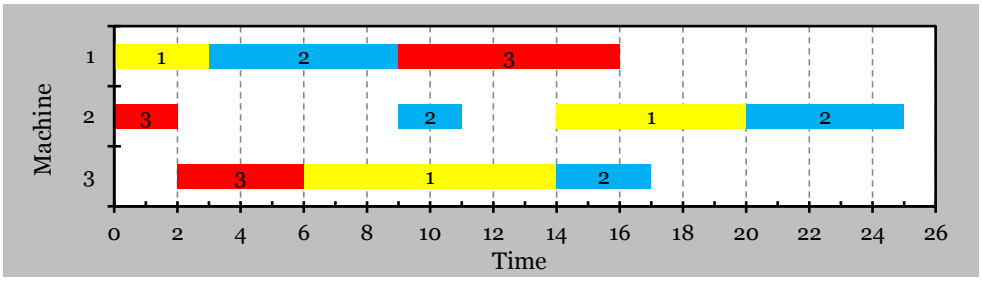

FIGURE 3. The resultant schedule of the example with 5 jobs and 3 machines. (Color online.)

As it can be seen in this example, one important shortcoming is the redundancy of sequences produced. By use of the research findings introduced in the previous section, the redundant sequences can be easily recognized before scheduling and then prevented. The research findings are utilized to diagnose the conditions under which the redundancy occurs. They enable algorithms to prevent form moving aimlessly from one sequence to another redundant sequence.

In the above mentioned example, in the last iteration 10 sequences are obtained by inserting job number 2 into the partial sequence. All the 10 sequences belong to two clusters of redundant schedules. $\{S'_1, S'_2\}$ and $\{S'_3, S'_4, \ldots, S'_{10}\}$ are redundant schedule. One can easily see trend from the makespan obtained in iteration 9. Therefore, only two sequences need to be checked. S'_1 and S'_3 are only evaluated using the redundancy prevention.

The half of solutions in the reference set is generated by this heuristic. The other half are generated randomly. Different initial solutions can be generated by this heuristic by sorting operations randomly in Step 1. The remaining solutions are generated randomly from feasible space.

4.2. Subset generation mechanism

In this mechanism, a basis for combining solutions of reference set is generated. That is, solutions are divided into some subsets, and solutions in each subset are combined. To implement this idea, we develop a procedure as follows. At each iteration, the solutions are partitioned into k groups with the same number of solutions. To make sure that each group includes at least a good solution, the k best solutions in the reference set are selected as the head for each group. Then, the remaining solutions are partitioned among the groups randomly. The following subsets are then considered for combining in each group.

- The first two best solutions
- The best and worst solutions
- The second best and worst solutions
- The combined solution of the first two best and the worst solution
- One randomly selected solution and the best solution (O_1 times)
- One randomly selected solution and the best second solution (O_2 times)

Therefore, in each group, a total of $2 * (4 + O_1 + O_2)$ new solutions are generated regarding abovementioned subsets.

4.3. Solution combination mechanism

In this mechanism, two solutions of each subset defined by the subset generation mechanism are combined. To this end, two-point crossover is used. This mechanism works as follows.

Step 1. Two cut points are randomly selected.
Step 2. The permutation of the better solution between these two cut points are copied into the same positions of the new solution.

Step 3. The empty positions are filled with remaining job numbers of the other solution.

Let us further describe the procedure by an example. Consider a problem with 3 jobs and 3 machines. Consider the following two solutions.

$$\{3,1,1,3,2,2,3,1,2,1\} \quad \text{and} \quad \{1,2,3,1,3,3,2,2,1,1\}$$

The first two best are combined. Suppose the cut points are 3 and 7. In this case, the resultant solution becomes

$$\{1,3,1,3,2,2,3,2,1,1\}$$

4.4. Improvement mechanism

The new solutions, generated by the solution combination mechanism, undergo an improvement mechanism. In this mechanism, a neighboring solution is generated by swapping the positions of two randomly selected operations. If this new neighboring solution is better that the incumbent solution is accepted. Otherwise, it is rejected. This procedure repeats vb times over each new solution.

4.5. Reference set update mechanism

All the solutions including both newly generated solutions and the incumbent solutions, grouped into k clusters, are considered. There already exist *popsize* solutions in the incumbent population. Moreover, there are $2*(4+O_1+O_2)$ newly generated solutions. Among these $popsize+2*(4+O_1+O_2)$ solutions, the first *popsize* best solutions constitute the individuals of the next population, and they are divided again into k groups. The remaining solutions are also deleted.

5. The numerical experiments

This section evaluates the performance of the proposed models and algorithms (*i.e.*, IH, SS and SSR). Note that SS is SSR equipped with redundant prevention rules. In this regard, the proposed algorithms are compared with adaptations of two available algorithms in the relevant literature. The algorithms are tabu search (TS) proposed by Bruker and Kampmeyer [3] and genetic algorithm (GA) proposed by Nose *et al.* [16]. The algorithms are coded in Borland C++. The stopping criterion is set to a time limit of 0.2 nm seconds. It is noteworthy to indicate that all the algorithms are equipped with redundancy prevention. That is, according to the introduced properties they can recognize redundant sequences before scheduling; hence they can avoid checking them. The efficiency of this improvement is evaluated later.

At first, an experiment is designed based on Taguchi method to tune the parameters and operators of the proposed metaheuristic algorithm. Then, a set of small instances are generated and the model is evaluated for its computational time complexity. On the same small instance, the general performances of algorithms are compared with the optimal solution of the model. Another set of large instances is generated and the performances of the algorithms are compared.

6. The parameter tuning

Since appropriate designing of parameters has significant impact on performance of algorithms, the proposed algorithm is evaluated using different parameters. This section elaborates the parameter setting of SS algorithm. There are many techniques to statistically design an experimental investigation. Although a full factorial experiment is the most frequently used, this approach is not effective when the number of factors significantly increases. Taguchi method is, a type of a fractional factorial experiment, proper for experiments with several factors. The performance measure is the signal to-noise (S/N) ratio and the purpose is to maximize the signal-to noise ratio [19]. The term “signal” represents the mean response variable and “noise” represents the standard

TABLE 4. The levels of parameters of PSO algorithms.

Level	Parameter			
	popsize	O_1	O_2	*vb*
1	200	*popsize*/2	*popsize*/5	10
2	250	*popsize*/4	*popsize*/6	15
3	300	*popsize*/6	*popsize*/7	20

TABLE 5. The modified orthogonal array L_9.

Trial	Levels of control factors			
	A	B	C	D
1	$A(1)$	$B(1)$	$C(1)$	$D(1)$
2	$A(1)$	$B(2)$	$C(3)$	$D(2)$
3	$A(1)$	$B(3)$	$C(2)$	$D(3)$
4	$A(2)$	$B(1)$	$C(3)$	$D(3)$
5	$A(2)$	$B(2)$	$C(2)$	$D(1)$
6	$A(2)$	$B(3)$	$C(1)$	$D(2)$
7	$A(3)$	$B(1)$	$C(2)$	$D(2)$
8	$A(3)$	$B(2)$	$C(1)$	$D(3)$
9	$A(3)$	$B(3)$	$C(3)$	$D(1)$

deviation. This ratio is calculated as follows.

$$\text{S/N ratio} = -10\log_{10}\frac{1}{n}\left[\sum_{i=1}^{n}\text{RPD}_i^2\right]$$

where RPD (*i.e.*, relative percentage deviation) is

$$\text{RPD}_i = \frac{\text{OF}_i - \text{Min}_i}{\text{Min}_i} * 100$$

where OF_{ij} is the objective function value achieved by the algorithm and Min_i is the best solution obtained for test problem i.

The proposed SS has four three-level parameters of *popsize*, O_1, O_2 and *vb*. The levels of parameters are represented in Table 4. Thus, our experiment is 3^4 one. Regarding to these levels, Taguchi design suggests L_9 as the fittest orthogonal array since it is designed for such an experiment with four three-level factors. Table 5 shows the design of experiment proposed by L_9. In the full factorial design, there are 81 different treatments; yet, in Taguchi design, we only test 9 treatments, shown in Table 5, out of 81 ones.

30 different instances generated in different sizes are used. Thus, we have a total of 9*30 observations. Minitab 14 is used to analyze the results shown in Figure 4. The selected values are *popsize* of 250, $O_1$of popsize/4, O_2 of popsize/6 and vb of 15.

6.1. Evaluation on small sized instances

Using a set of small instances, the performance of the models and the algorithms are evaluated. We consider $n = \{4, 6, 8, 10\}$, $m = \{3, 4\}$, $n_j = \{U(2, 5)\}$ and $p_{j,i} = \{U(130)\}$. All the combinations sum up to 8 different sizes. For each size, we generate two instances. These instances are solved by the models and algorithms. Both models solve instances up to 6 jobs within 1000 s. Comparing the performance the two models, Model 1 would

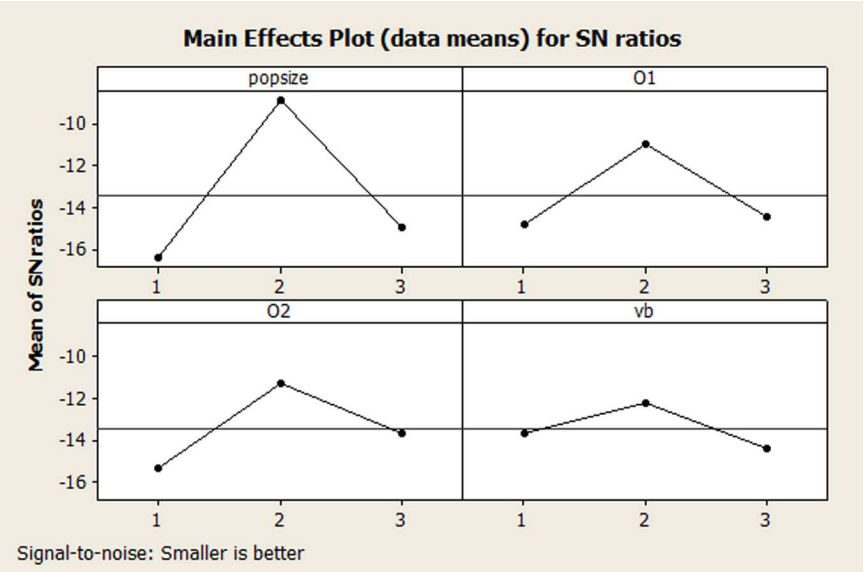

FIGURE 4. The S/N of controlled parameters value. (Color online.)

TABLE 6. The results on the small instances.

Instance		$C^*_{\max}$	Com. time (model)		$C_{\max}$ (algorithms)			
n	m		Model 1	Model 2	SS	SSR	TS	GA
4	3	118	0.30	0.08	118	118	118	124
4	3	128	0.33	0.08	128	128	128	128
4	4	129	0.42	0.09	129	129	129	129
4	4	111	0.11	0.03	111	111	111	114
6	3	92	5.11	0.62	92	92	92	101
6	3	87	3.63	2.20	88	94	90	90
6	4	125	201.00	103.29	125	125	125	125
6	4	145	368.15	189.00	145	145	145	145
8	3	193*	1000	1000	193	196	193	200
8	3	217*	1000	1000	217	217	217	217
8	4	181*	1000	1000	181	181	181	181
8	4	120*	1000	1000	120	120	120	120
10	3	224*	1000	1000	224	224	224	224
10	3	222*	1000	1000	222	222	222	222
10	4	202*	1000	1000	202	202	202	204
10	4	194*	1000	1000	194	195	194	196

*Upper bound of the mathematical models.

obtain the optimal solutions in lower computational times. Regarding the algorithms, SS performs the best among all with obtaining 15 optimal solutions. Table 6 shows the results.

6.2. Evaluation on large sized instances

The section proceeds with the numerical comparison of the tested algorithms (*i.e.*, IH, SS, SSR, TS and GA) on large instances generated as follows. We consider $n = \{20, 40, 60, 80, 100\}$ and $m = \{2, 3, 6\}$. The number of operations for jobs are randomly generated from a uniform distribution between $[m-3, m+5]$, and the

TABLE 7. The results of the algorithms on the large instances.

n	m	Algorithms				
		GA	SSR	SS	TS	IH
20	2	1.12	0	0	0	1.13
	3	1.05	0.30	0	0	1.86
	6	3.49	0.81	0.29	0	3.58
40	2	0.38	0	0	0	1.11
	3	0.97	0	0.39	0	1.58
	6	3.33	3.56	1.99	1.08	7.43
60	2	0.82	0.19	0.09	0.77	1.77
	3	3.64	1.80	0	2.11	4.19
	6	2.60	0.73	0.02	1.75	4.55
80	2	2.05	0.05	0.02	2.19	2.45
	3	1.86	0.93	0	0.61	2.16
	6	3.87	0.30	0.25	5.82	5.72
100	2	0.88	0.05	0	1.06	1.51
	3	3.32	0.88	0	4.29	3.73
	6	3.46	1.94	0	5.10	5.66
Average		2.19	0.77	0.20	1.65	3.23

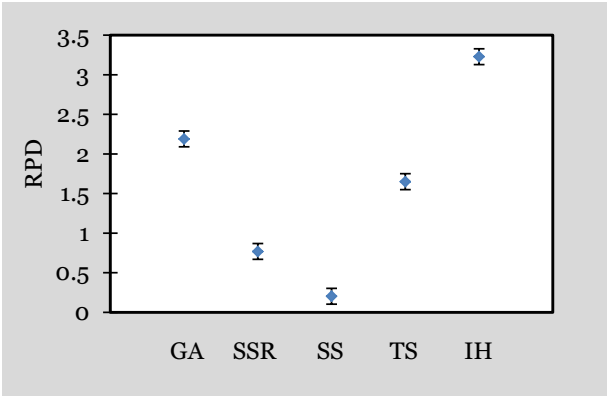

FIGURE 5. Means plot with LSD intervals for the different algorithms. (Color online.)

processing times from a uniform distribution between $U[1, 20]$. There are 5 instances for each of 15 combination sizes. It sums up to 75 instances. To compare the algorithms, RPD measure (Eq. (2.12)) is used.

Table 7 shows the results, averaged RPD for each combination of n and m. The best performing algorithm is SS with average RPD of 0.20%. SSR obtains the average RPD of 0.77%. After the proposed algorithms, TS achieves the average RPD of 1.65%. The worst performing algorithm is also GA with average RPD of 2.19%.

To statistically analyze the results, an ANOVA is conducted where the algorithm type is the single controlled factor. The results show that there are statistically significant differences between various algorithms with a p-value very close to zero. Figure 5 shows the means plot with least significant difference (LSD) intervals at the 95% confidence level for the different algorithms. As it can be seen, the proposed provides statistically better results among the tested algorithms.

To further analyze the results, the performance of the algorithms *versus* the problem sizes is evaluated. Figure 6 plots the average RPD of algorithms in different sizes of n and m.

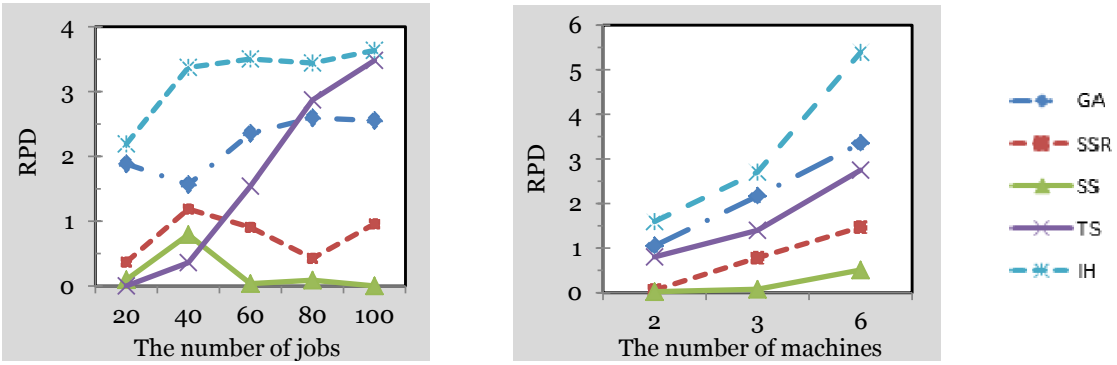

FIGURE 6. Means plot of algorithms *versus* the number of jobs/machines. (Color online.)

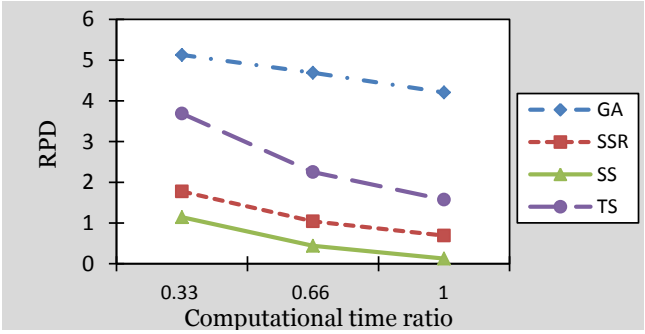

FIGURE 7. Means plot of algorithms *versus* computational time ratio. (Color online.)

TABLE 8. The impact of the redundancy prevention theorems.

n	m	Percentage of sequences		Computational time (s)		
		Discarded	Checked	IH(E)	IH(N)	Reduction
10	5	78.13	21.87	0.0048	0.0109	55.96%
	10	88.21	11.79	0.0093	0.0905	89.72%
20	5	79.58	20.42	0.0188	0.0826	77.23%
	10	89.07	10.92	0.1154	0.7629	84.87%
30	5	79.67	20.33	0.1031	0.2823	63.47%
	10	89.49	10.51	0.5101	3.5787	85.74%
40	5	79.69	20.31	0.3244	1.2979	75.01%
	10	89.65	10.35	1.5069	7.6441	80.28%
50	5	79.59	20.42	0.3261	1.883	82.68%
	10	89.70	10.30	2.0499	19.719	89.60%
Average		84.28	15.72			78.46%

To further analyze the results, the performance of the algorithms *versus* the problem sizes is evaluated. Figure 7 plots the average RPD of algorithms in different sizes of n and m.

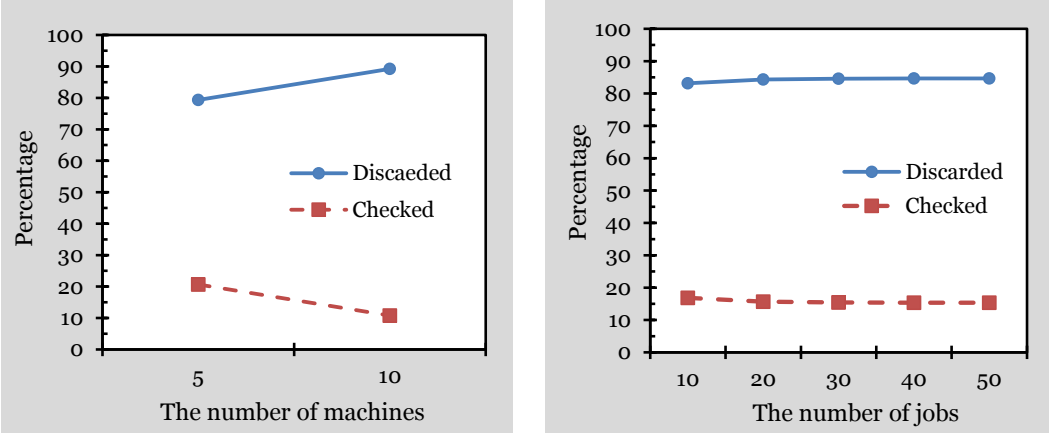

FIGURE 8. The percentage of discarded/checked sequences *versus* the number of jobs/machines. (Color online.)

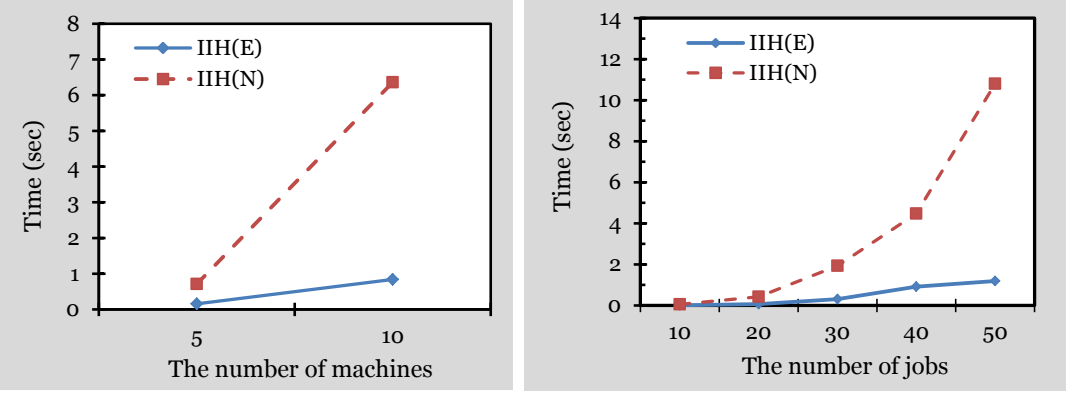

FIGURE 9. The computational time (s) of IH(E) and IH(N) *versus* the number of jobs/machines. (Color online.)

6.3. The impact of redundancy prevention theorems

The redundancy prevention theorems are used to recognize the redundancy before it takes place. To evaluate the efficiency of these research findings, we analyze how many redundant schedules are avoided while implementing the simple applications of these two theorems. We compare the IH heuristic before and after the implementation of above theorems. IH that excludes redundant solutions is called IH(E), while IH that includes redundant solutions is named IH(N).

The large instances are solved to analyze the reduction in computational time originated by the application of redundancy prevention theorems. Table 8 shows the percentage of sequences discarded and the computational time in seconds of IH(E) and IH(N). On average, the theorems discard 85% of the sequences while it reduces the computational time by 78%. In the case of (50,10), IH(E) solves the problems in around 19 s before applying the concepts of above theorems while after applying them, this computational time plummets from 19 s to 2 s. Figures 8 and 9 plot the percentage of discarded/checked sequences and the computational time of the two algorithms *versus* the problem size. The almost same percentage of sequences is redundant in the different number of jobs, while in larger number of machines, more sequences are redundant.

7. Conclusion and future research

This paper studied the problem of scheduling the cycle shop. In this problem, jobs visited machines in a cyclic manner. That is, they might be processed by a machine more than once. The problem was first formulated by two mixed integer linear programs. Through this model, small instances were solved to optimality. The encoding scheme that fits to this problem is permutation of job numbers. This scheme suffered from the serious shortcoming of redundancy. Some novel properties were introduced by which the redundancy was recognized. Enhanced by these properties, we developed a scatter search with five mechanisms of diversification generation, improvement, reference set update, subset generation and solution combination. In the diversification generation mechanism, an insertion heuristic is developed to initiate the algorithm from quality solutions. The subset generation mechanism is designed so as to have both diversification and intensification capabilities. A fast and simple, but effective local search is applied as the improvement mechanism.

Using Taguchi method, the choice of parameters and operators were comprehensively discussed, and the algorithm was perfectly tuned. The efficiency of introduced theorems on the performance of the algorithms was analyzed. The results showed over 80% of sequences are redundant schedule and the algorithms equipped with implementation of those theorems could easily identify and consequently avoid checking them. Two sets of small and large instances were generated and the algorithms were evaluated on them. The results showed that the effectiveness of the algorithms. As an interesting future research, the problem can be extended to multi-objective case. Moreover, it can be developed to consider some realistic assumptions such as setup times. It is also a line for future research to evaluate the performance of other algorithms such as particle swarm optimization.

Acknowledgements. The authors thank Research Deputy of Kharazmi University to support this research.

References

[1] K.J. Baker, Introduction to Sequencing and Scheduling. Wiley, NewYork, USA (1974).
[2] J. Błazewicz, K.H. Ecker, E. Pesch, G. Schmidt and J. Weglarz, Handbook of Scheduling: From Theory to Applications. Springer, Germany (2007).
[3] P. Bruker and T. Kampmeyer, Tabu search algorithms for cyclic machine scheduling problems, in The Fifth Metaheuristics International Conference, Japan (2003).
[4] A. Che and C. Chu, A polynomial algorithm for no-wait cyclic hoist scheduling in an extended electroplating line. *Oper. Res. Lett*. **33** (2005) 274–284.
[5] J.S. Chen, J.C.H. Pan and C.K. Wu, Hybrid tabu search for re-entrant permutation flow-shop scheduling problem. *Expert Syst. Appl*. **34** (2008) 1924–1930.
[6] J.S. Chen, J.C.H. Pan and C.M. Lin, A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. *Expert Syst. Appl.* **34** (2008) 570–577.
[7] H.M. Cho, S.J. Bae, J. Kim and I.J. Jeong, Bi-objective scheduling for reentrant hybrid flow shop using Pareto genetic algorithm. *Comput. Ind. Eng.* **61** (2011) 529–541.
[8] F. Dugardin, F. Yalaoui and L. Amodeo, New multi-objective method to solve reentrant hybrid flow shop scheduling problem. *Eur. J. Oper. Res.* **203** (2010) 22–31.
[9] J. Gao, L. Sun and M. Gen, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Comput. Oper. Res.* **35** (2008) 2892–2907.
[10] M. Hekmatfar, S.M.T. FatemiGhomi and B. Karimi, Two stage reentrant hybrid flow shop with setup times and the criterion of minimizing makespan. *Appl. Soft Comput.* **11** (2011) 4530–4539.
[11] Y.H. Kang, S.S. Kim and H.J. Shin, A scheduling algorithm for the reentrant shop: an application in semiconductor manufacture. *Int. J. Adv. Manuf. Technol.* **35** (2007) 566–574.
[12] B. Naderi and R. Ruiz, A scatter search algorithm for the distributed flowshop scheduling problem. *Eur. J. Oper. Res.* **239** (2014) 323–334.
[13] B. Naderi and N. Salmasi, Permutation flowshops in group scheduling with sequence-dependent setup times. *Eur. J. Ind. Eng.* **6** (2012) 177–198.
[14] B. Naderi, R. Ruiz and M. Zandieh, Algorithms for a realistic variant of flowshop scheduling. *Comput. Oper. Res.* **37** (2010) 236–246.
[15] M. Nawaz, E.E. Enscore Jr. and I. Ham, A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *Omega* **11** (1983) 91–95.
[16] K. Nose, A. Hiramatsu and M. Konishi, Using genetic algorithm for job-shop scheduling problems with reentrant product flows, in 7th IEEE International Conference on Emerging Technologies and Factory Automation, Spain (1999).

[17] C.H. Pan, A study of integer programming formulations for scheduling problems. *Int. J. Syst. Sci.* **28** (1997) 33–41.
[18] M. Pinedo, Scheduling Theory, Algorithms, and Systems, 3rd edn. LLC, Springer, New York (2008).
[19] R.J. Ross, Taguchi Techniques for Quality Engineering. McGraw-Hill, USA (1989).
[20] S. Topaloglu and G. Kilincli, A modified shifting bottleneck heuristic for the reentrant job shop scheduling problem with makespan minimization. *Int. J. Adv. Manuf. Technol.* (2009) **44** 781–794.
[21] R. Uzsoy, C.Y. Lee and L.A. Martin-Vega, A review of production planning and scheduling models in the semiconductor industry, part 1: system characteristics, performance evaluation and production planning. *IIE Trans.* **24** (1992) 47–61.
[22] X. Xie, L. Tang and Y.Li, Scheduling of a hub reentrant job shop to minimize makespan. *Int. J. Adv. Manuf. Technol.* **56** (2011) 743–753.