International Journal of Supply and Operations Management

IJSOM

May 2014, Volume 1, Issue 1, pp. 54-68
ISSN-Print: 2383-1359
ISSN-Online: 2383-2525
www.ijsom.com

No-idle time Scheduling of Open shops: Modeling and Meta-heuristic Solution Methods

Bahman Naderi[a*], Vahid Roshanaei[b]

[a]*Department of Industrial Engineering, Kharazmi University, Tehran, Iran*
[b]*Department of Industrial Engineering, University of Toronto, Toronto, Canada*

Abstract

In some industries as foundries, it is not technically feasible to interrupt a processor between jobs. This restriction gives rise to a scheduling problem called no-idle scheduling. This paper deals with scheduling of no-idle open shops to minimize maximum completion time of jobs, called makespan. The problem is first mathematically formulated by three different mixed integer linear programming models. Since open shop scheduling problems are NP-hard, only small instances can be solved to optimality using these models. Thus, to solve large instances, two meta-heuristics based on simulated annealing and genetic algorithms are developed. A complete numerical experiment is conducted and the developed models and algorithms are compared. The results show that genetic algorithm outperforms simulated annealing.

Keywords: Scheduling, Open shop, No idle time, mixed integer linear programming, Simulated annealing, Genetic algorithm.

1. Introduction

The problem of shop scheduling can be described as follows. There are a set of n jobs and a set of m machines. The jobs are going to be processed by each of the machines. The processing route of jobs on machines can be either fixed beforehand or determined by decision makers. If the processing route is fixed and identical for all jobs, the problem turns into a flow shop scheduling. If it is fixed yet unique for each job, the problem is job shop scheduling. If it is not fixed, the problem becomes open shop scheduling. In other words, unlike the flow and job shops, the jobs in the open shop can visit machines in any order. The

* Corresponding email address: bahman_naderi62@yahoo.com

decision dimension of the flow and job shops is to sequence jobs on machines. Yet, in the open shop, the decision of planner is not only to sequence jobs on machines but also to sequence machines in the processing route of each job.

In the literature, researchers commonly extend the open shop so as to be more fit for a practical situation. Goldansaz et al. (2013) consider multi-processor open shop scheduling problems with independent setup time and sequence dependent removal time. Chernykh et al. (2013) investigate the routing open shop problem which is a generalization of two problems of the open shop and the metric traveling salesman problems. Chen et al. (2013) study the parallel open shop scheduling problem where each job has two independent non-preemptive operations. Dong et al. (2013) consider the problem of scheduling batch and delivery coordination. Jobs are processed in a two-machine open shop, and then are delivered to a common customer by only one vehicle. Abdelmaguid et al. (2014) study the multiprocessor open shop scheduling problem where there is a set of processing centers each of which has one or more parallel identical machines.

In production industries with capital-intensive machines, it is not desirable to keep such machines idle. Even, in some industries with less expensive machines, it might not be demanding to stop machines between jobs. For example, one can refer the reader to the furnace in the fiberglass industry. Since heating the furnace up to the necessary temperature is both time-consuming and expensive, it always stays on when it starts working. Among other applications of this behavior (i.e., as it might be technically infeasible or uneconomical to interrupt a machine in between jobs) are foundries, production of integrated circuits and the steel making industry (Pan and Ruiz, 2014). These practical situations raise a scheduling environment, called no-idle scheduling. In no-idle scheduling, idle time on a machine is not allowed. In other words, each machine must continuously process jobs from the start of processing the first job to the end of the last job. To fulfill this restriction, the start of processing the first job on a given machine might be delayed.

The literature of no-idle scheduling mainly focuses on flow shop problems. Kalczynski and Kamburowski (2005) deal with no-idle flow shops. They propose a constructive heuristic for solving it that significantly outperforms heuristics. Deng and Gu (2012) present a hybrid discrete differential evolution algorithm for the no-idle permutation flowshop scheduling problem with makespan criterion.

Tasgetiren et al. (2013) study the no-idle permutation flowshop scheduling problem with the total tardiness criterion and propose a discrete artificial bee colony algorithm for this problem. Tasgetiren et al. (2013) study the same problem with Tasgetiren et al., (2013) and present a variable iterated greedy algorithm with differential evolution. The traditional iterated greedy and a variable iterated greedy from the literature are also re-implemented. Pan and Wang (2008) consider the no-idle permutation flow shop scheduling problems with the criterion to minimize makespan. They propose two simple approaches to calculate the makespan, a speed-up method and a discrete particle swarm optimization algorithm which is superior to two heuristics of Tasgetiren et al. (2013) and Kalczynski and Kamburowski (2005).

El Houda Saadani et al. (2005) investigate no-idle flowshop problems with the objective to minimize the makespan. Based on the idea that this problem can be modeled as a travelling salesman problem, an adaptation of the well-known nearest insertion rule is proposed to solve the problem. Baraz and Mosheiov (2008) study makespan minimization on no idle flowshops. They also introduce a greedy algorithm. Goncharov and Sevastyanov (2009) propose several

polynomial time heuristics based on a geometrical approach for the general case and for special cases of 3 and 4 machines. Moreover, they present a comprehensive review of relevant results.

Pan and Ruiz (2014) study mixed no-idle flow shops where not all machines require no-idle restriction. They first mathematically formulate the problem and propose an iterated greedy algorithm enhanced by a speed-up feature. As reviewed, all the papers in the literature of no-idle scheduling consider the flow shop problem.

In this paper, we first formulate the problem by three mixed integer linear programming models. Using these models, the small instances of the problem are solved to optimality. Since the problem under consideration is NP-hard, the best solution method is metaheuristic. There are two different metaheuristic types, single-individual and multi-individual (population-based) ones. To find out what type of metaheuristic performs well for this problem, we have decided to develop one single- and one multi-individual metaheuristic. Among different single-individual alternatives, simulated annealing has shown high performance in different scheduling problems (Andersen et al. 2008). And also, among different multi-individual alternatives, genetic algorithm seems the best one regarding the literature (Dai et al., 2013). Thus, two metaheuristics, based on simulated annealing and genetic algorithm are also developed to solve large instances. Two numerical experiments are conducted to evaluate and compare the models and algorithms.

The rest of the paper is organized as follows. Section 2 formulates the problem by three different mathematical models. Section 3 develops two metaheuristics. Section 4 evaluates both the models and metaheuristics. Section 5 concludes the paper.

2. Problem formulation

To illustrate the problem, we first present a numerical example. Consider a problem with $n = 4$ and $m = 3$. The processing times are shown in Table 1. One possible solution for this problem is presented by Figure 1. As can be seen, there is no interruption when a machine starts processing jobs from the first to the last. The value of makespan becomes 21.

Table 1. Processing times of the example

Jobs	Machines		
	1	2	3
1	2	6	6
2	4	5	4
3	4	4	2
4	5	3	2

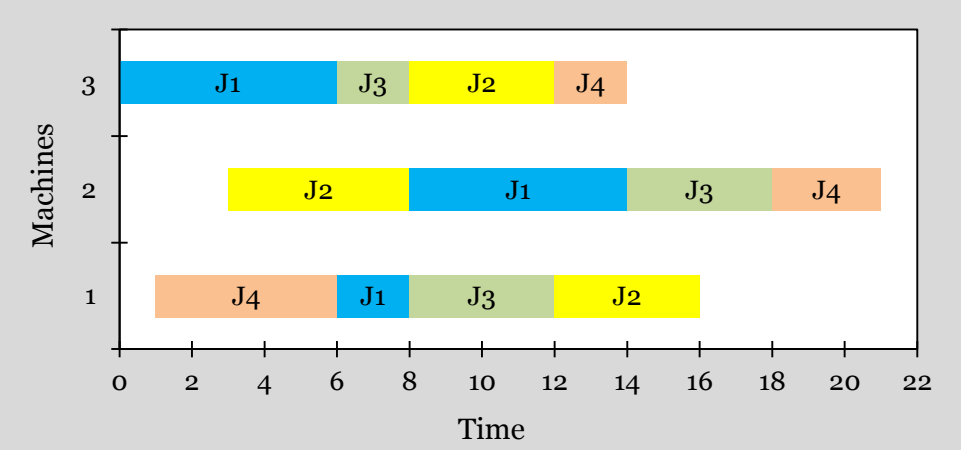

Figure 1. Gantt chart of a feasible solution for the example.

Scheduling problems are commonly formulated as mixed integer linear programming models. The problem under consideration is modeled by three different mathematical models. Before presenting the models, we establish the following parameters and sets.

Parameters:

n	The number of jobs
k, j, e	The index for jobs $\{1,2, ndn\}$
m	The number of machines
i, l, r	The index for jobs $\{1,2, ndm\}$
M	A large positive number

Sets

$p_{j,i}$	The processing time of job j on machine i

For the first model, we have

Decision variables:

$X_{j,i,l}$	Binary variable taking value 1 if O_{ji} is processed immediately after O_{jl}, and 0 otherwise. l ai
$Y_{j,i,k}$	Binary variable taking value 1 if O_{ji} is processed immediately after O_{ki}, and 0 otherwise. k aj
$C_{j,i}$	Continuous variable for the completion time of job j on machine i.
S_i	Continuous variable for the starting time of machine i.
F_i	Continuous variable for the finishing time of machine i.

The first model, called Model 1, is as follows.

Min C_{max}

Subject to:

$$\sum_{k=0,k\neq j}^{n} Y_{j,i,k} = 1 \qquad \forall_{j,i} \qquad (1)$$

$$\sum_{l=0,l\neq i}^{m} X_{j,i,l} = 1 \qquad \forall_{j,i} \qquad (2)$$

$$\sum_{j=1,j\neq k}^{n} Y_{j,i,k} \leq 1 \qquad \forall_{i,k\in\{1,2,..,n\}} \qquad (3)$$

$$\sum_{i=1,i\neq l}^{m} X_{j,i,l} \leq 1 \qquad \forall_{j,l\in\{1,2,..,m\}} \qquad (4)$$

$$\sum_{j=1}^{n} Y_{j,i,0} = 1 \qquad \forall_{i} \qquad (5)$$

$$\sum_{i=1}^{m} X_{j,i,0} = 1 \qquad \forall_{j} \qquad (6)$$

$$F_i = S_i + \sum_{j=1}^{n} P_{j,i} \qquad \forall_{i} \qquad (7)$$

$$C_{j,i} \geq S_i + P_{j,i} \qquad \forall_{j,i} \qquad (8)$$

$$C_{j,i} \leq F_i \qquad \forall_{j,i} \qquad (9)$$

$$C_{j,i} \geq C_{j,l} + P_{j,i} - \left(1 - X_{j,i,l}\right) \times M \qquad \forall_{j,i,l\neq i} \qquad (10)$$

$$C_{j,i} \geq C_{k,i} + P_{j,i} - \left(1 - Y_{j,i,k}\right) \times M \qquad \forall_{j,i,k\neq j} \qquad (11)$$

$$C_{max} \geq C_{j,i} \qquad \forall_{j,i} \qquad (12)$$

$$X_{j,i,l} \in \{0,1\} \qquad \forall_{j,i,l\neq i} \qquad (13)$$

$$Y_{j,i,k} \in \{0,1\} \qquad \forall_{j,i,k\neq j} \qquad (14)$$

Where $C_{j,0} = C_{0,i} = 0$.
Constraint sets (1) and (2) specifies that each operation is performed once. Constraint sets (3) and (4) assure that each operation have at most one succeeding operation. Constraint sets (5) and (6) state that dummy job 0 and machine 0 have exactly one successor. Constraint sets (7), (8) and (9) ensure that no-idle restrictions are met. Constraint set (10) ensures each job can be processed by at most one machine at a time. Constraint set (11) specifies each machine can process at most one job at a time. Constraint set (12) calculates makespan. Finally, Constraint sets (13) and (14) define the decision variables.

In the second model, the following decision variables are defined.
Decision variables:

- $X_{j,i,k}$ Binary variable taking value 1 if O_{ji} occupies *k*-th position in job *j*'s sequence, and 0 otherwise.
- $Y_{j,i,l}$ Binary variable taking value 1 if job *j* occupies *l*-th position of machine *i*, and 0 otherwise.
- $C_{j,i}$ Continuous variable for the completion time of O_{ji}.
- S_i Continuous variable for the starting time of machine *i*.
- F_i Continuous variable for the finishing time of machine *i*.

The second model, called Model 2, becomes as such.

Min C_{max}
Subject to:

$$\sum_{k=1}^{m} X_{j,i,k} = 1 \qquad \forall_{j,i} \qquad (15)$$

$$\sum_{l=1}^{n} Y_{j,i,l} = 1 \qquad \forall_{j,i} \qquad (16)$$

$$\sum_{i=1}^{m} X_{j,i,k} = 1 \qquad \forall_{j,k} \qquad (17)$$

$$\sum_{j=1}^{n} Y_{j,i,l} = 1 \qquad \forall_{i,l} \qquad (18)$$

$$F_i = S_i + \sum_{j=1}^{n} P_{j,i} \qquad \forall_i \qquad (19)$$

$$C_{j,i} \geq S_i + P_{j,i} \qquad \forall_{j,i} \qquad (20)$$

$$C_{j,i} \leq F_i \qquad \forall_{j,i} \qquad (21)$$

$$C_{j,i} \geq C_{j,r} + P_{j,i} - M\left(1 - X_{j,i,k}\right) - M\left(1 - \sum_{t=1}^{k-1} X_{j,r,t}\right) \qquad \forall_{j,i,r,k>1} \qquad (22)$$

$$C_{j,i} \geq C_{e,i} + P_{j,i} - M\left(1 - Y_{j,i,l}\right) - M\left(1 - \sum_{t=1}^{l-1} Y_{e,i,t}\right) \qquad \forall_{j,i,e,l>1} \qquad (23)$$

$$C_{max} \geq C_{j,i} \qquad \forall_{i,j} \qquad (24)$$

$$X_{j,i,k} \in \{0, 1\} \qquad \forall_{j,i,k} \qquad (25)$$

$$Y_{j,i,l} \in \{0, 1\} \qquad \forall_{j,i,l} \qquad (26)$$

Constraint sets (15) and (16) assure that each operation occupy exactly one position. Constraint set (17) and (18) are to ensure that each position of processing route of each job and each position of job sequence of each machine is assigned once. Constraint sets (19), (20) and (21) are to meet no-idle restriction. Constraint set (22) assures that each job cannot be processed by more than one machine and Constraint set (23) is to ensure that each machine cannot process more than one job at a time. Constraint set (24) computes makespan, and Constraint sets (25) and (26) define the decision variables.

The third model includes the following decision variables.
Decision variables:

$X_{j,i,l}$	Binary variable for taking value 1 if O_{ji} is processed after O_{jl}, and 0 otherwise. $i \in \{1, 2, \ldots, m-1\}$, $l > i$
$Y_{j,i,k}$	Binary variable for taking value 1 if O_{ji} is processed after O_{ki}, and 0 otherwise. $j \in \{1, 2, \ldots, n-1\}$, $k > j$
$C_{j,i}$	Continuous variable for the completion time of job j on machine i
S_i	Continuous variable for the starting time of machine i.
F_i	Continuous variable for the finishing time of machine i.

The third model, Model 3, is as follows.

Min C_{max}
Subject to:

$$F_i = S_i + \sum_{j=1}^{n} p_{j,i} \qquad \forall_i \qquad (27)$$

$$C_{j,i} \geq S_i + p_{j,i} \qquad \forall_{j,i} \qquad (28)$$

$$C_{j,i} \leq F_i \qquad \forall_{j,i} \qquad (29)$$

$$C_{j,i} \geq C_{j,l} + p_{j,i} - M\left(1 - X_{j,i,l}\right) \qquad \forall_{j,i \in \{1,2,\text{tom}m-1\},l>i} \qquad (30)$$

$$C_{j,l} \geq C_{j,i} + p_{j,l} - MX_{j,i,l} \qquad \forall_{j,i \in \{1,2,\text{tom}m-1\},l>i} \qquad (31)$$

$$C_{j,i} \geq C_{k,i} + p_{j,i} - M\left(1 - Y_{j,i,k}\right) \qquad \forall_{i,j\in \{1,2,\text{tom}n-1\},k>j} \qquad (32)$$

$$C_{k,i} \geq C_{j,i} + p_{k,i} - MY_{j,i,k} \qquad \forall_{i,j\in \{1,2,\text{tom}n-1\},k>j} \qquad (33)$$

$$C_{max} \geq C_{j,i} \qquad \forall_{j,i} \qquad (34)$$

$$X_{j,i,l} \in \{0,1\} \qquad \forall_{j,i \in \{1,2,\text{tom}m-1\},l>i} \qquad (35)$$

$$Y_{j,i,k} \in \{0,1\} \qquad \forall_{i,j \in \{1,2,\text{tom}n-1\},k>j} \qquad (36)$$

Constraint sets (27), (28) and (29) specify the no-idle restriction. Constraint sets (30) and (31) together assure that each job can be processed at most one machine at a time. While Constraint sets (32) and (33) together ensure that each machine can process no more than one job at a time. Constraint set (34) calculates makespan. Constraint sets (35) and (36) define the decision variables.

3. Developed metaheuristics

In this paper, we develop two metaheuristics of simulated annealing and genetic algorithm to solve large instances of the problem. The proposed algorithms are based on permutation encoding scheme. The permutation scheme is a sorted list of all the operations being processed. By considering the permutation from left to right, the relative sequence of operations is determined. For the sake of simplicity, let us describe the permutation scheme by an example. Suppose we have a problem with $n = 3$ and $m = 2$, summing up to 6 operations. In the case of permutation list, $\{O_{22}, O_{11}, O_{32}, O_{21}, O_{11}, O_{31}\}$ is one possible permutation. An encoded solution is decoded so as to end up with a feasible solution regarding no-idle restriction. Later on, the simulated annealing and genetic algorithm are described in detail.

3.1. Simulated annealing

The simulated annealing (SA) is a metaheuristic known as a local search based one. It is inspired from the annealing process (1999). SA is equipped with a mechanism, called acceptance criterion, which enables it to partially avoid getting trapped in local optima. The acceptance mechanism is to decide if a new generated solution is accepted or rejected. In this mechanism, even worse solutions have a chance to be accepted. Among different successful applications of SA, papers (Naderi et al., 2009a; Zhang et al. 2010) can be pointed out. Naderi et al. (2009a) develop a SA to solve hybrid flow shop scheduling problems. Zhang et al. (2010) propose a SA to solve a high school course timetabling problem.

3.1.1. The structure and acceptance criterion

The simulated annealing starts from an initial solution. It iteratively makes a series of moves until a stopping criterion is fulfilled. The basic principle of SAs is to produce a new solution *s* by a mechanism, called move operator, from the neighborhood of the current solution *x*. This new solution can be either accepted or rejected by another mechanism, known as acceptance mechanism. A parameter *t*, called the temperature, controls the acceptance rule. The variation between objective values of two candidate solutions is computed $\Delta = fit(s) - fit(x)$. If $\Delta \leq 0$, the new solution *s* is accepted, and the search continues from it. Otherwise, the new solution *s* is accepted with probability equal to $exp(\Delta/t_i)$. In other words, the better new solution is always accepted while the worse new solution might be accepted. The probability of acceptance depends on both parameter t and the inferiority size.

At each temperature t_i, a fixed number of neighborhood moves are made. Then, temperature is gradually decreased. We use exponential cooling schedule, $t_i = \alpha \, . t_{i-1}$ (where $\alpha \in (0, 1)$ is temperature decrease rate). After having an initial experiment, the initial temperature is set to be 50 and $\alpha = 0.97$.

The procedure: the proposed SA
Initialization mechanism
While the stopping criterion is not met **do**
    Move operator
    Acceptance mechanism
    Temperature reduction
Endwhile

Figure 2. The outline of the proposed SA

3.1.2. The move operator

In this research, to generate a new solution from the current solution the following procedure is used. To generate a neighbor solution v from the current solution x, this paper considers shift operator which is designed based on insertion operator (Naderi et al. 2009b). It works as follows: one randomly selected operation is randomly relocated into a new position in the permutation.

For example consider a problem with $n = 3$ and $m = 2$. Therefore, there are 6 operations. One of the possible permutations is

$$\{O_{22}, O_{11}, O_{32}, O_{21}, O_{12}, O_{31}\}$$

To apply shift operator, suppose the randomly selected operation becomes operation O_{21} and randomly selected position becomes position 2. In this case, the neighbor solution is

$$\{O_{22}, O_{21}, O_{11}, O_{32}, O_{12}, O_{31}\}.$$

3.2.Genetic algorithm

Genetic algorithm (GA) is designed to deal with some problems of industry that were difficult to solve with conventional methods. Todays, GA is a well-known population based evolutionary algorithm tackling both discrete and continuous optimization problems. The idea behind GA comes from Darwin's ''survival of the fittest'' concept, meaning that good parents produce better offsprings. Many hard optimization problems have been successfully solved by

GA (Wang, 2002; Toledo et al., 2013; Balakrishnan et al., 2003). Wang (2002) solves teacher assignment problems by GA. Toledo et al. (2013) develops a GA to tackle lot sizing problems. Balakrishnan et al. (2003) also solve dynamic layout problem by GA.

3.2.1. General structure

GA searches a solution space with a population of chromosomes each of which represents an encoded solution. A fitness value is assigned to each chromosome according to its performance. The better the chromosome is, the higher this value becomes. The population evolves by a set of operators until some stopping criterion is visited. A typical iteration of a GA, generation, proceeds as follows. The best chromosomes of current population are directly copied to next generation (reproduction). A selection mechanism chooses chromosomes of the current population so as to give higher chance to chromosomes with the higher fitness value. The selected chromosomes are crossed to generate new offspring. After crossing process, each offspring might mutate by another mechanism called mutation. Afterwards, the new population is evaluated again and the whole process is repeated. The outline of the proposed GA is shown in Figure 3.

The procedure: the proposed GA
Initialization mechanism
While the stopping criterion is not met **do**
    Selection mechanism
    Crossover mechanism
    Mutation mechanism
Endwhile

Figure 3. The outline of the proposed GA

3.2.2. The initialization and selection mechanisms

GA starts with a number of chromosomes each of which represents a possible solution. The number of chromosomes is the population size indicated by *pop*, set to 50. The initial chromosomes are randomly generated from the feasible solutions.

After initializing the algorithms, each chromosome is evaluated and its fitness (i.e., objective function) is determined. The chance of chromosome k to be selected for crossover mechanism is as follows.

$$p_k = \frac{fit(k)}{\sum_{h=1}^{pop} fit(h)}$$

where $fit(k)$ is the fitness of chromosome k.

3.2.3. Crossover and mutation mechanisms

New solutions are produced by crossing two other solutions already selected by selection mechanism. These two solutions are called parents. The operator of combining parent are called crossover. The purpose of this combining is to generate better offsprings. To move the search towards better areas, we define a new solution that inherits from both parents. In fact, we combine two parents to form a new solution. In this research, this is done through an operator with the following steps.

Two randomly cut points are selected. Then, the operations between these cut points from Parent 1 are copied to offspring in the same positions. The remaining operations are put into the empty positions of the offspring from Parent 2. The order of the remaining operations is determined by their relative order in Parent 2. For example consider a problem with n = 3 and m = 2. Suppose two parents are

$$\text{Parent 1: } \{O_{22}, O_{11}, O_{32}, O_{21}, O_{12}, O_{31}\}$$
$$\text{Parent 2: } \{O_{32}, O_{12}, O_{31}, O_{21}, O_{22}, O_{11}\}$$

Suppose the two randomly selected cut points are 2 and 4. In this case, the operations from position 2 to position 4 are copied into the same position in the offspring.

$$\text{offespring: } \{-, O_{11}, O_{32}, O_{21}, -, -\}$$

The remaining operations are O_{22}, O_{12} and O_{31}. These operations are copied into offspring according to Parent 2. Thus, the complete offspring becomes

$$\text{offespring: } \{O_{12}, O_{11}, O_{32}, O_{21}, O_{31}, O_{22}\}$$

After crossover, each solution is changed by the mutation operator. The main purpose of applying mutation is to avoid convergence to a local optimum and diversify the population. We use the swap mutation operator which works as follows. Two positions are randomly selected and the operations of these two positions are swapped. For example consider a problem with n = 3 and m = 2. Suppose the encoded solution is

$$\{O_{22}, O_{11}, O_{32}, O_{21}, O_{12}, O_{31}\}$$

The two randomly selected positions are 3 and 6. By swapping the corresponding operations, we have

$$\{O_{22}, O_{11}, O_{31}, O_{21}, O_{12}, O_{32}\}$$

4. Computational evaluation

First, the efficiency of the proposed MILP models is evaluated on a computational experiment including small-sized instances. Afterwards, we assess the general performance of the proposed metaheuristics (i.e., GA and SA) against the optimal solutions obtained by the models. We use a performance measure named relative percentage deviation (RPD) obtained by the following formula:

$$RPD = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} \cdot 100$$

where Min_{sol} and Alg_{sol} are is the lowest C_{max} for a given instance obtained by any of algorithms and the solution obtained by a given algorithm. We implement the MILP models in LINGO 10 and the other algorithms in Borland C++ and run on a PC with 2.0 GHz Intel Core 2 Duo and 2 GB of RAM memory. The stopping criterion used when testing all instances with the metaheuristics is set to a computational time limit fixed to $n \times m \times 0.5$ seconds. This stopping criterion permits for more time as the number of jobs or machines increases.

4.1.Evaluation on small-sized instances

This subsection first compares the efficiency of the MILP models to solve the problem under consideration. We generate a set of different instances as follows. We have 6 problem sizes ranging from ($n \times m$)= (3×3), up to (5×4). The processing times are randomly distributed over (1, 99). For each problem size, we generate 2 instances. Therefore, it sums up to 12 instances. The MILP models are allowed a maximum of 1000 seconds of computational time.

Table 2 shows the results obtained by different models. It shows the average computational time required by the corresponding model in each size. For example, the first instance of problem size (4×3) is solved by Model 1 in 5.07 seconds. The performance of Models 1 and 2 are similar to each other and solve the problem up to (4×4). It might be interesting to state that Model 1 is slightly faster than Model 2 on average. Obviously, the best performing MILP is Model 3 which solves the instances up to size (5×4).

Table 2. Models' results (computational time in seconds)

$n \times m$	Models		
	Model 1	Model 2	Model 3
3×3	0.05	0.04	0.02
3×4	1.21	3.86	0.07
4×3	2.75	5.07	0.11
4×4	230.51	563.24	1.01
5×3	-	-	13.73
5×4	-	-	58.19

We are going to evaluate the algorithms (i.e. GA and SA) against the optimal solutions obtained by the models in the previous small instances. Table 3 shows the results. The best performing algorithm is GA by optimally solving 11 instances out of 12 instances. SA finds optimal solutions of 9 instances.

4.2.Evaluation on large-sized instances

After having investigated the general performance of the metaheuristics, we intend to further compare the proposed algorithms against a set of large instances. We consider the following combinations of n and m.

$$\{(5,5), (10,10), (15,15), (20,20)\}$$

For each combination, we generate 10 random instances by producing random processing times from a uniform distribution over (1, 99). We use the RPD measure to compare the algorithms. Table 4 summarizes the results of the experiments averaged for each combination of n and m. GA still is the best performing algorithm with RPD of 1.58%. SA yields average RPD of 2.86%.

Table 3. Algorithm' results on small instances

$n \times m$	Algorithm (gap)	
	GA	SA
3×3	0.00	0.00
3×4	0.00	0.00
4×3	0.00	0.00
4×4	0.00	0.96
5×3	1.04	3.41
5×4	0.00	2.42
Average	0.17	1.13

Table 4. Algorithm' results on small instances

$n \times m$	Algorithm (gap)	
	GA	SA
5×5	0.81	1.37
10×10	1.96	2.73
15×15	2.18	3.61
20×20	1.39	4.05
Average	1.58	2.94

To further statistically analyze the results, we carry out an analysis of variance test or ANOVA. The results demonstrate that there are significant differences between the algorithms with *p*-value very close to 0. Figure 4 shows the means plot and least significant difference or LSD intervals at 95% confidence level for the different algorithms.

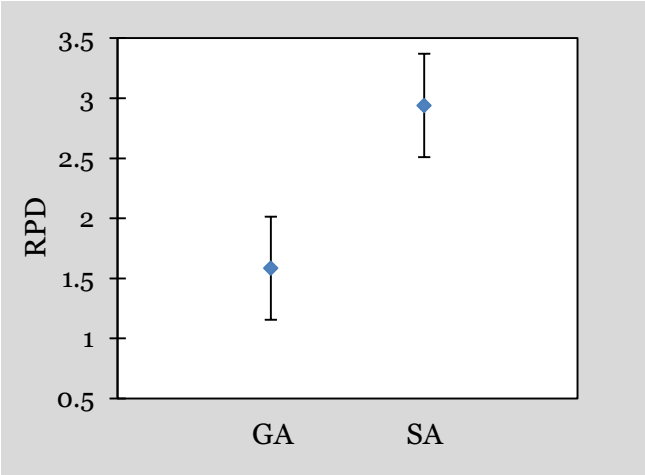

Figure 4. The average RPD and LSD interval of the algorithms

It is also interesting to plot the performance of the algorithms versus the problem size. Figure 5 shows the means obtained by the algorithms in the different problem sizes. In all the four problem sizes, GA outperforms SA. The performance of SA and GA are almost the same in the smallest size of $n = m = 5$. There is a clear trend that GA works better in larger sizes.

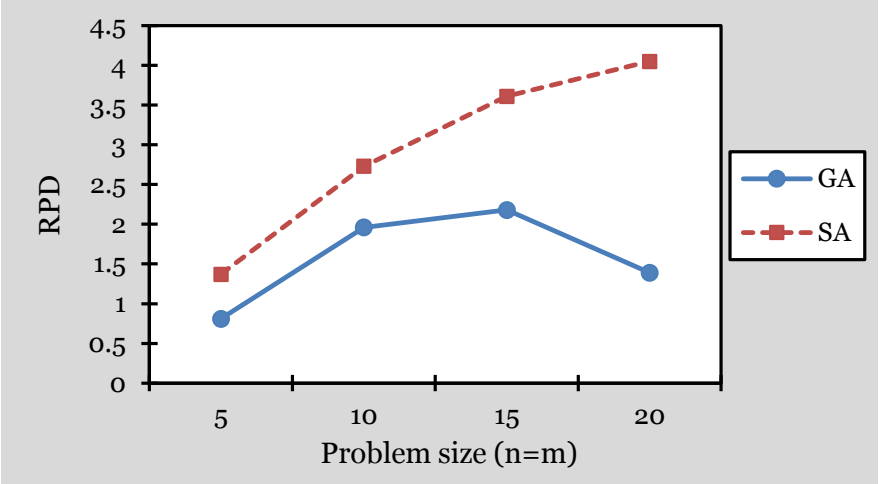

Figure 5. The average RPD of the algorithms versus the problem size.

5. Conclusion and future study

This paper considered no-idle open shop problems. Under no-idle restriction, no idle time on machine is allowed. Therefore, when a machine starts its processing, it has to process job continuously from the first to the last. Since this problem was not studied in the literature, there is no mathematical model for the problem. We first formulated the problem by three

mathematical models. The models were in the form of mixed integer linear programs. The problems up to $n = 5$ and $m = 4$ are solved to optimality.

Since the problem is NP-hard, larger problems cannot be solved by the models. In this case, we developed two metaheuristics based on simulated annealing and genetic algorithm. Two computational experiments were conducted to evaluate the performances of models and metaheuristics. In the first experiment, we used small instances by which we compared the mathematical models and evaluated general performance of the proposed metaheuristics. In the second experiment, two metaheuristics were compared on large instances. All the results supported that the models and metaheuristics effectively solve the no-idle open shop problem. The results showed that genetic algorithm outperformed simulated annealing.

One interesting future research direction is to develop other metaheuristics for this problem. In real cases, some machines, not all, have the no-idle constraint. Hence, it is also worthy to consider mixed no idle open shops. Moreover, it is impressive to study the multi-objective version of the problem under consideration.

References

Abdelmaguid, T.F., Shalaby, M.A., Awwad, M.A., (2014). A tabu search approach forproportionate multiprocessor open shop scheduling. *Computational Optimization and Applications*, Vol. 58(1), pp. 187-203.

Andresen, M., Bräsel, H., Mörig, M., Tusch, J., Werner, F., Willenius, P., (2008). Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. *Mathematical and Computer Modelling*, Vol. 48, pp. 1279-1293.

Baraz, D., Mosheiov, G., (2008). A note on a greedy heuristic for flow-shop makespan minimization with no machine idle-time. *European Journal of Operational Research*, Vol. 184, pp. 810–813.

Balakrishnan, J., Cheng, C.H., Conway, D.G., Lau, C.M., (2003). A hybrid genetic algorithm for the dynamic plant layout problem. *International Journal of Production Economics*, Vol. 86, pp. 107–20.

Chen, Y., Zhang, A., Chen, G., Dong, J., (2013). Approximation algorithms for parallel open shop scheduling. *Information Processing Letters*, Vol. 113(7), pp. 220-224.

Chernykh, I., Kononova, A., Sevastyanova, S., (2013). Efficient approximation algorithms for the routing open shop problem. *Computers and Operations Research*, Vol. 40(3), pp. 841-847.

Dai, M., Tang, D., Giret, A., Salido, M.A., Lid, W.D., (2013). Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing*, Vol. 29(5), pp. 418-429.

DengG., Gu, X., (2012). A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion. *Computers and Operations Research*, Vol. 39, pp. 2152–2160.

Dong, J., Zhang, A., Chen, Y., Yang, Q., (2013). Approximation algorithms for two-machine open shop scheduling with batch and delivery coordination. *Theoretical Computer Science*, Vol. 491, pp. 94-102.

El Houda Saadani, N., Guinet, A., Moall, M., (2005). A travelling salesman approach to solve the F/no-idle/Cmax problem. *European Journal of Operational Research*, Vol. 161, pp. 11–20.

Fatih Tasgetiren, M., Pan, Q.K., Suganthan, P.N., Buyukdagli, O., (2013). A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. *Computers and Operations Research*, Vol. 40, pp. 1729-1743.

Fatih Tasgetiren, M., Pan, Q.K., Suganthan, P.N., Oner, A., (2003). A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Applied Mathematical Modelling*, Vol. 37, pp. 6758–6779.

Goldansaz, S.M., Jolai, F., Zahedi Anaraki, A.H., (2013). A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop. *Applied Mathematical Modelling*, Vol. 37(23), pp. 9603-9616.

Goncharov, Y., Sevastyanov, S., (2009). The flow shop problem with no-idle constraints: A review and approximation. *European Journal of Operational Research*, Vol. 196, pp. 450–456.

Kalczynski, P.J., Kamburowski, J., (2005). A heuristic for minimizing the makespan in no-idle permutation flow shops. *Computers and Industrial Engineering*, Vol. 49, pp. 146–154.

Kolon, M., (1999). Some new results on simulated annealing applied to the job shop scheduling problem. *European Journal of Operational Research*, Vol. 113, pp. 123–136.

Naderi, B., Zandieh, M., Balagh, A.K.G., Roshanaei, V., (2009a). An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion time and total tardiness. *Expert Systems with Applications*, Vol. 36, pp. 9625–9633.

Naderi, B., Zandieh, M., Fatemi Ghomi, S.M.T., (2009b). A study on integrating sequence dependent setup time flexible flow lines and preventive maintenance scheduling. *Journal of Intelligent Manufacturing*, Vol. 20, pp. 683–694.

Pan, Q.K., Ruiz, R., (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, Vol. 44, pp. 41-50.

Pan, Q.K., Wang, L., (2008). No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm. *International Journal of Advanced Manufacturing Technology*, Vol. 39, pp. 796–807.

Toledo, C.F.M., Oliveira, R.R.R., Franca, P.M., (2013). A hybrid multi-population genetic algorithm applied to solve the multi-level capacitated lot sizing problem with backlogging. *Computers and Operations Research*, Vol. 40, pp. 910-919.

Wang, Y.Z., (2002). An application of genetic algorithm methods for teacher assignment problems. *Expert Systems with Applications*, Vol. 22, pp. 295–302.

Zhang, D., Liu, Y., M'Hallah, R., Leung, S.C.H., (2010). A simulated annealing with a new neighborhood structure based algorithm for high school timetabling problems. *European Journal of Operational Research*, Vol. 203, pp. 550–558.