Contents lists available at ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

Modeling and scheduling no-wait open shop problems

B. Naderi[a], M. Zandieh[b,*]

[a] Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Tehran, Iran
[b] Department of Industrial Management, Accounting and Management Faculty, Shahid Beheshti University, G.C., Tehran, Iran

ARTICLE INFO

Article history:
Received 21 September 2012
Accepted 4 January 2014
Available online 4 July 2014

Keywords:
Scheduling
Open shop
No-wait
Mixed integer linear programming
Genetic algorithm
Variable neighborhood search

ABSTRACT

This paper studies the problem of scheduling open shops with no intermediate buffer, called no-wait open shops under makespan minimization. No-wait scheduling problems arise in many realistic production environments such as hot metal rolling, the plastic molding, chemical and pharmaceutical, food processing and several other industries. To tackle such problems, we first develop three different mathematical models, mixed integer linear programs, by which we can solve the problem to optimality. Besides the models, we propose novel metaheuristics based on genetic and variable neighborhood search algorithms to solve the large-sized problems in an acceptable computational time. The key point in any scheduling solver is the procedure of encoding and decoding schemes. In this paper, we propose a simple yet effective tailor-made procedure of encoding and decoding schemes for no-wait open shop problems. The operators of the proposed metaheuristics are designed so as to consider the specific encoding scheme of the problem. To evaluate the performance of models and metaheuristics, we conduct two computational experiments. The first includes small-sized instances by which we compare the mathematical models and assess general performance of the proposed metaheuristics. In the second experiment, we further evaluate the potential of metaheuristics on solving some benchmarks in the literature of pure open shops. The results show that the models and metaheuristics are effective to deal with the no-wait open shop problems.

1. Introduction

In general, scheduling problems can be described by a set of n jobs that need to be processed by a set of m machines. The problem is called a flow shop if all the jobs follow the same processing route through the machines, i.e., they are first processed on machine 1, then on machine 2, and so on until machine m. If each job has its own processing route to go through machines, the problem is a job shop. Now, if there are no predetermined processing routes for the jobs, the problem becomes an open shop. In other words, in open shops, there are two decisions to make, the determination of the processing routes of the jobs as well as job sequence on each machine (Pinedo, 2008). The following assumptions are usually characterized to open shops. All the jobs are independent and available for their process at time 0. All machines are continuously available. Each machine can at most process one job at a time. The process of a job on a machine cannot be interrupted. There are infinite buffers between all machines. There is no transportation time between machines. The objective function is the minimization of makespan.

Some of the common assumptions in open shop scheduling impede many of the possible practical applications of this important problem. For example, the existence of an infinite storage capacity between machines is one of these assumptions (Pang, 2013). This is, jobs can unlimitedly wait for machines to be available. In many scheduling environments, however, due to characteristics of the jobs or the processing technology, the operations of a job must be performed without any interruption between machines, which is known as no-wait restriction. Typical application of no-wait scheduling problems arises in hot metal rolling industries, where the heated metal has to undergo a set of operations at continuously high temperatures before it is cooled in order to prevent defects. Similarly, in the plastic molding and silverware production industries, a set of operations must be performed to immediately follow one another to prevent degradation. Other examples include chemical and pharmaceutical industries, food processing industries, and advanced manufacturing environments. For a detailed explanation of the applications on no-wait scheduling problems, the papers by Goyal and Sriskandarajah (1988), Hall and Sriskandarajah (1996) could be good references. In spite of its practical applications and

* Corresponding author. Tel.: +98 21 22431842.
E-mail address: m_zandieh@sbu.ac.ir (M. Zandieh).

http://dx.doi.org/10.1016/j.ijpe.2014.06.011
0925-5273/© 2014 Elsevier B.V. All rights reserved.

theoretical issues, as shown in the survey paper by Hall and Sriskandarajah (1996), no-wait open shop scheduling (NW-OSS) has given far less attention than the other scheduling problems such as no-wait flow shops. According to three folds notation of Graham et al. (1979), the problem of no-wait open shop scheduling to minimize makespan can be classified as $O/nwt/C_{max}$.

Adiri and Amit (1984) consider NW-OSS where all operations have equal processing times and present a dispatching rule to minimize total completion time. Sidney and Sriskandarajah (1999) study two-machine NW-OSS and introduce a heuristic to solve the problem. As cited by Liaw et al. (2005), Yao and Soewandi (2000) and Yao and Lai (2002) address the problem of two-machine NW-OSS and propose a heuristic and a genetic algorithm, respectively. Liaw et al. (2005) also consider the same problem of two-machine NW-OSS. They propose a branch-and-bound armed with some dominance rules as well as a two-phase heuristic. Lin et al. (2008) study NW-OSS with movable dedicated machines. The objective is the minimization of total occupation time of all the machines. They introduce a mixed integer program, however unfortunately non-linear, to formulate the problem. They also propose a two-phase heuristic whose first phase constructs an initial sequence and the second phase improves that sequence. As far as we reviewed and as summarized by Table 1, almost none of the existing papers consider classical multi-machine NW-OSS. There is almost no attempt to linearly model the problem. There is no systematic encoding and decoding schemes for NW-OSS. Only one paper presents a metaheuristic for a relevant problem, not similar one.

It is known that the problem of two-machine no-wait open shop scheduling is strongly NP-hard (Sahni and Cho, 1979). Since this problem is a specific case of NW-OSS where $m=2$, we can conclude general multi machines NW-OSS is also NP-hard. Two common approaches to tackle the scheduling problems are the utilization of mathematical programming and heuristic approaches (Stafford Jr. et al., 2005). Due to the great advance recently obtained in capacity of computers and creation of fast optimization software, presentation of MILP models is becoming more and more interesting among the researchers. We develop a total of three different MILP models to formulate the no-wait open shops. We also carry out an experiment to analyze and compare the performance of the proposed models.

As we earlier mentioned, the second common way to solve such a problem is heuristic approaches which can be divided into two main groups: constructive and improvement heuristics. Constructive heuristics are those build a sequence quickly by a fixed predetermined rule. Therefore, they always yield the same results for a given instance. Improvement heuristics (so-called metaheuristics) are those iteratively improve a (or some) sequence(*s*) produced by random or by the constructive heuristics. Besides the presentation of models, we aim at providing tools to solve the large-sized problems. To do so, we need to establish a procedure to encode and decode solutions. The encoding and decoding schemes are almost the key point in the success of any algorithm, and the most problem-specific feature of any algorithm in the production scheduling problems. It should be designed in such a way that it has high adaptability to any operator and great simplicity to code. It also should avoid infeasible solutions in order to save the algorithms' computational time by searching only feasible space. No-wait scheduling problems need more meticulous care than pure problems due to high possibility of generating infeasible solutions. The situation becomes even more vital in open shop problems in comparison with the other scheduling problems. This paper establishes a tailor-made procedure of encoding and decoding schemes for no-wait open shop problems that keeps all above-mentioned characteristics. Afterwards, we propose three high performing metaheuristics based on variable neighborhood search (VNS) and genetic algorithm (GA). VNS is known to be a powerful, yet simple to understand and code metaheuristic. In this paper, we develop two different VNSs centered on curtailed and greedy fashions. Besides the VNSs, we apply an GA incorporating with some powerful operators. Model's efficiency and metaheuristics' capability to solve the problem studied here are investigated on two computational evaluations including small- and large-sized instances.

The rest of paper is organized as follows. Section 2 develops three different mixed integer linear programs. Section 3 presents a novel procedure of the encoding and decoding schemes. Section 4 introduces the proposed metaheuristics. Section 5 describes the experimental design to evaluate the posed methods including the mathematical models and algorithms. Finally, Section 6 gives some interesting conclusions and future studies.

2. Problem formulation

Even though MILP models might not be an efficient solution method for all problem sizes, they are a natural way to attack scheduling problems (Stafford Jr. et al., 2005). This section presents three different MILP models to formulate no-wait open shop problems based on three different variables. We analyze differences between the models. For example, we compare the number of binary and continuous variables as well as the number of constraints required for each model to formulate the same sized problem; also, we study their possible impact on performance of the models. We have the following similar notations in all the three models:

Parameters

n	The number of jobs
m	The number of machines
$O_{j,i}$	The operation of job j on machine i
$P_{j,i}$	The processing time of $O_{j,i}$

Table 1
Papers in literature of no-wait open shops.

Author	Year	Problem description	Model	Algorithm			Algorithm description
				Exact	Heuristic	Metaheuristic	
Adiri and Amit	1984	The same processing times for all jobs			✓		Dispatching rule
Sidney and Sriskandarajah	1999	Two-machine			✓		Approximation algorithm
Yao and Soewandi	2000	Two-machine			✓		–
Yao and Lai	2002	Two-machine				✓	Genetic algorithm
Liaw et al.	2005	Two-machine		✓			Branch and bound
Lin et al.	2007	Movable dedicated machines and minimization of total occupation time of all machines	✓			✓	An iterative improvement

M A large positive number

Variables

$C_{j,i}$ The completion time of $O_{j,i}$ (1)

S_j The starting time of job j (2)

2.1. Model 1

Model 1 uses binary variables (BV) that show the relative sequence of different operations of a job as well as the relative order of the jobs on each machine. This type of BV is first proposed by Rios-Mercado and Bard (1998) for flow shop problems. Due to the identity of these BVs, Model 1 is called sequence-based model. Note for every machine we introduce a dummy job 0 which precedes the first job on that machine; and for every job, a dummy machine 0 that precedes the first operation of that job. It is necessary to indicate that the model requires the big M. The following notations are established:

Indices

k	Job index	$\{0, 1, 2, \ldots, n\}$
j	Job index	$\{1, 2, \ldots, n\}$
l	Machine index	$\{0, 1, 2, \ldots, m\}$
i	Machine index	$\{1, 2, \ldots, m\}$

Variables

$X_{j,i,l}$ Binary variable that takes value 1 if $O_{j,i}$ is processed immediately after $O_{j,l}$, and 0 otherwise. $l \neq i$

$Y_{j,i,k}$ Binary variable that takes value 1 if $O_{j,i}$ is processed immediately after $O_{k,i}$, and 0 otherwise. $k \neq j$

Variables in (1) and (2)

Model 1 formulates NW-OSS as follows:

Minimize C_{max}

Subject to:

$$\sum_{k=0,k\neq j}^{n} Y_{j,i,k} = \sum_{l=0,l\neq i}^{m} X_{j,i,l} = 1 \quad \forall_{j,i} \tag{3}$$

$$\sum_{j=1,j\neq k}^{n} Y_{j,i,k} \leq 1 \quad \forall_{i,k\epsilon\{1,2,\ldots,n\}} \tag{4}$$

$$\sum_{i=1,i\neq l}^{m} X_{j,i,l} \leq 1 \quad \forall_{j,l\epsilon\{1,2,\ldots,m\}} \tag{5}$$

$$\sum_{j=1}^{n} Y_{j,i,0} = 1 \quad \forall_{i} \tag{6}$$

$$\sum_{i=1}^{m} X_{j,i,0} = 1 \quad \forall_{j} \tag{7}$$

$$Y_{j,i,k} + Y_{k,i,j} \leq 1 \quad \forall_{i,j,k>j} \tag{8}$$

$$X_{j,i,l} + X_{j,l,i} \leq 1 \quad \forall_{j,i,l>i} \tag{9}$$

$$C_{j,i} \leq S_j + \sum_{l=1}^{m} P_{j,i} \quad \forall_{j,i} \tag{10}$$

$$C_{j,i} \geq S_j + P_{j,i} \quad \forall_{j,i} \tag{11}$$

$$C_{j,i} \geq C_{j,l} + P_{j,i} - (1 - X_{j,i,l}) \times M \quad \forall_{j,i,l\neq i} \tag{12}$$

$$C_{j,i} \geq C_{k,i} + P_{j,i} - (1 - Y_{j,i,k}) \times M \quad \forall_{j,i,k\neq j} \tag{13}$$

$$C_{max} \geq C_{j,i} \quad \forall_{j,i} \tag{14}$$

$$X_{j,i,l} \in \{0, 1\} \quad \forall_{j,i,l\neq i} \tag{15}$$

$$Y_{j,i,k} \in \{0, 1\} \quad \forall_{j,i,k\neq j} \tag{16}$$

where

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

Constraint set (3) states that every operation is scheduled once. Constraint sets (4) and (5) ensure that every operation must have at most one succeeding operation in processing route of the jobs and in job order of every machine. Constraint sets (6) and (7) enforce that dummy job 0 and machine 0 must have exactly one successor. Constraint sets (8) and (9) state that an operation cannot be both predecessor and successor of another operation at the same time. Constraint sets (10) and (11) are to assure that no-wait restrictions are held. Constraint set (12) ensures that $O_{j,i}$ cannot start before $O_{j,l}$ completes if $O_{j,l}$ precedes $O_{j,i}$. Constraint set (13) specifies that if $O_{j,i}$ is processed immediately after $O_{k,i}$, it cannot begin before $O_{k,i}$ completes. Constraint set (14) calculates makespan. Ultimately, Constraint sets (15) and (16) define the decision variables.

2.2. Model 2

In this model, BVs specify the operations' position in both processing route of a job and job order of a machine; so, it is called position-based model. This type of the BV is first proposed by Wagner (1959) for flow shop problems. It does not need to define dummy job 0. For model 2, we establish the following notations:

Indices

j, e	Job index	$\{1, 2, \ldots, n\}$
$k,$	Position index for jobs	$\{1, 2, \ldots, n\}$
i, r	Machine index	$\{1, 2, \ldots, m\}$
l	Position index for machines	$\{1, 2, \ldots, m\}$

Variables

$X_{j,i,k}$ Binary variable that takes value 1 if $O_{j,i}$ occupies k-th position in the processing route of job j, and 0 otherwise.

$Y_{j,i,l}$ Binary variable that takes value 1 if job j occupies l-th position in the job order of machine i, and 0 otherwise.

Variables in (1) and (2)

Model 2 characterizes NW-OSS as follows:

Minimize C_{max}

Subject to:

$$\sum_{k=1}^{m} X_{j,i,k} = \sum_{l=1}^{n} Y_{j,i,l} = 1 \quad \forall_{j,i} \tag{17}$$

$$\sum_{i=1}^{m} X_{j,i,k} = 1 \quad \forall_{j,k} \tag{18}$$

$$\sum_{j=1}^{n} Y_{j,i,l} = 1 \quad \forall_{i,l} \tag{19}$$

$$C_{j,i} \geq S_j + P_{j,i} \quad \forall_{j,i} \tag{20}$$

$$C_{j,i} \leq S_j + \sum_{l=1}^{m} P_{j,i} \quad \forall_{j,i} \tag{21}$$

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

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

$$C_{max} \geq C_{j,i} \quad \forall_{i,j} \tag{24}$$

$$X_{j,i,k} \in \{0,\ 1\} \quad \forall_{j,i,k} \tag{25}$$

$$Y_{j,i,l} \in \{0,\ 1\} \quad \forall_{j,i,l} \tag{26}$$

Constraint set (17) assures that every operation must exactly occupy one position in the processing route of every job and job order of every machine. Constraint set (18) ensures that n positions of every job must be occupied once, and Constraint set (19) states that m positions of every machine must be assigned once. Constraint sets (20) and (21) hold no-wait restriction. Constraint set (22) assures that $O_{j,i}$ begins only after $O_{j,r}$ finishes if $O_{j,i}$ occupies a position after $O_{j,r}$. Similarity, Constraint set (23) is to ensure that $O_{j,i}$ is processed after the completion of $O_{e,i}$. Constraint set (24) computes makespan, and Constraint sets (25) and (26) define the decision variables.

2.3. Model 3

We develop another sequence-based model with the following specifications. In the processing route of each job j, for each pair of $O_{j,i}$ and $O_{j,r}$, there is a variable which shows whether $O_{j,i}$ is processed after $O_{j,r}$ or not (notice, not necessarily immediately). This type of the BV is first proposed by Manne (1960) for job shop problems. In Model 3, the notations are as follows:

Indices

j,k	Job index	$\{1, 2, \ldots, n\}$
i,l	Machine index	$\{1, 2, \ldots, m\}$

Variables

$X_{j,i,l}$ Binary variable that takes value 1 if $O_{j,i}$ is processed after $O_{j,l}$, and 0 otherwise. $i \in \{1, 2, \ldots, m-1\}$, $l > i$

$Y_{j,i,k}$ Binary variable that takes value 1 if $O_{j,i}$ is processed after $O_{k,i}$, and 0 otherwise. $j \in \{1, 2, \ldots, n-1\}$, $k > j$

Variables in (1) and (2)

NW-OSS is again formulated as follows:

Minimize C_{max}

Subject to:

$$C_{j,i} \geq S_j + P_{j,i} \quad \forall_{j,i} \tag{27}$$

$$C_{j,i} \leq S_j + \sum_{l=1}^{m} P_{j,i} \quad \forall_{j,i} \tag{28}$$

$$C_{j,i} \geq C_{j,l} + P_{j,i} - M \bullet (1 - X_{j,i,l}) \quad \forall_{j,i\ \in\ \{1,2,\ldots,m-1\}, l>i} \tag{29}$$

$$C_{j,l} \geq C_{j,i} + P_{j,l} - M \bullet X_{j,i,l} \quad \forall_{j,i\ \in\ \{1,2,\ldots,m-1\}, l>i} \tag{30}$$

$$C_{j,i} \geq C_{k,i} + P_{j,i} - M \bullet (1 - Y_{j,i,k}) \quad \forall_{i,j\ \in\ \{1,2,\ldots,n-1\}, k>j} \tag{31}$$

$$C_{k,i} \geq C_{j,i} + P_{k,i} - M \bullet Y_{j,i,k} \quad \forall_{i,j\ \in\ \{1,2,\ldots,n-1\}, k>j} \tag{32}$$

$$C_{max} \geq C_{j,i} \quad \forall_{j,i} \tag{33}$$

$$X_{j,i,l} \in \{0,\ 1\} \quad \forall_{j,i\ \in\ \{1,2,\ldots,m-1\},\ l\ >\ i} \tag{34}$$

$$Y_{j,i,k} \in \{0,\ 1\} \quad \forall_{i,j\ \in\ \{1,2,\ldots,n-1\},\ k\ >\ j} \tag{35}$$

Constraint sets (27) and (28) ensure the no-wait restriction. Constraint sets (29) and (30) are the dichotomous pairs of constraints relating each pair of operations in processing route of every job; and similarity, Constraint sets (31) and (32) are the dichotomous pairs of constraints for each pair of jobs in job order of every machine. Constraint set (33) calculates makespan. Constraint sets (34) and (35) define the decision variables.

Table 2
Comparison of models' characteristics.

Model	No. of BVs	No. of CVs	Number of constraints
1	$nm(n+m)$	$n+nm$	$nm(4+\frac{3n}{2}+\frac{3m}{2})+n+m$
2	$nm(n+m)$	$n+nm$	$nm(5+2nm-n-m)+n^2+m^2$
3	$\frac{nm}{2}(n+m-2)$	$n+nm$	$nm(1+n+m)$

Now, we intend to compare the models with respect to the required numbers of binary variables, continuous variables and constraints for a same sized problem with n and m. Table 2 summarizes the results. Model 3 has the least number of BVs where Models 1 and 2 have the same number of BVs. Relative proportion of BVs required by Model 3 against the other models is $((n+m-2)/2(n+m))$; that is, its number of BVs is even less than the half of those of Models 1 and 2. All the models have the same number of continuous variables. Model 1 has fewer constraints as Model 2 does. Model 3 again formulate the problems by the least number of constraints.

3. The procedure of encoding and decoding schemes

The encoding scheme is the procedure of making a solution recognizable for algorithms and plays an important role in the effectiveness of the algorithms. The decoding scheme is a complementary procedure that turns an encoded solution into a schedule. The effective procedures should have simplicity to code and high adaptability to any operators. Another necessary feature, even most important, for any procedure is to avoid infeasible solutions. In this case, algorithms are capable of saving their computational time by searching only feasible solution space. No-wait scheduling problems need more meticulous care than pure problems due to high possibility of generating infeasible solutions. The situation becomes even more vital in open shop problems in comparison with the other scheduling problems such as no-wait flow shops. In this paper, we establish encoding and decoding schemes for no-wait open shop problems that have all above-mentioned characteristics. We introduce the proposed encoding and decoding schemes.

Two commonly used encoding schemes in the literature of the open shop are permutation list and rank matrix (Liaw, 2000; Bräsel et al., 2008; Andresen et al., 2008). Permutation list is an ordered list of all the operations. By scanning the permutation from left to right, the relative starting times of operations in which they are processed are merely expressed. As a result, the process of each operation must not begin before its previous operations on the permutation start. The rank matrix is a matrix in which each row shows the operations order of one job on different machines, and each column characterizes the operations order of all jobs on one machine. Similar to the permutation list, an operation cannot begin before the previous operations in the operation order of the corresponding machine and job complete. In both cases, the relative order of jobs on machines is exactly and strictly determined, while in NW-OSS, very often this leads to the generation of infeasible solutions.

For the sake of simplicity, let us describe it by an example. Suppose we have a problem with $n=3$ and $m=2$, summing up to 6 operations. Table 3 shows the processing times of the operations. In the case of permutation list, $\{O_{22}, O_{11},\ O_{32}, O_{21}, O_{11}, O_{31}\}$ is one of those permutations ending up with an infeasible solution. That is, Job 1 cannot be the first job on Machine 1 while on Machine 2, Job 1 is processed after Jobs 2 and 3 because no-wait restrictions are not kept.

In the case of the rank matrix, the following matrix can be stated as one of those matrices generating the infeasible solutions. In other words, it is impossible to process Job 2 as a first job on Machine 1 and as third job on Machine 2; and at the same time, the no-wait restrictions are held.

$$\begin{array}{cc} & \begin{array}{cc} M1 & M2 \end{array} \\ \begin{array}{c} J1 \\ J2 \\ J3 \end{array} & \begin{bmatrix} 2 & 3 \\ 1 & 4 \\ 3 & 1 \end{bmatrix} \end{array}$$

As it could be seen in the previous example, the commonly used encoding schemes of regular open shops are not effective for the case of NW-OSS. NW-OSS needs a specific encoding scheme adapted to no-wait constraints. In NW-OSS, besides the processing route of each job, one just needs to determine the starting time of each job j (S_j) instead of the operations, because the starting times of the all the operations of job j can be calculated through S_j. In both permutation list and rank matrix, an encoded solution becomes infeasible if the starting times of the operations cannot be equal to the starting times of those operations considering no-wait constraints. Therefore, many encoded solutions are likely to end up infeasible.

We have tested several procedures from simple to complicated ones. Finally, it turned out that the following procedure could possess all it takes to be an effective encoding scheme. The proposed encoding scheme has two main parts:

Part (1)- A permutation showing relative order of the jobs rather than the operations.

Part (2)- A matrix whose rows determine the relative sequence of operations of each job. Unlike the rank matrix, each column does not show the job order on the machines.

Table 3
The processing times of the example with $n=3$ and $m=2$.

Job	Machine	
	1	2
1	7	5
2	6	9
3	8	12

In this case, we just determine the relative order of the jobs and the processing route of each job. Therefore, by a proper decoding procedure, it always ends up with a feasible solution. All the possible solutions of NW-OSS could be covered by this type of encoding scheme. Considering previous examples, one of the solutions is

$$\begin{array}{lcc} \text{Part (1)} - \text{Job order}: & & \begin{bmatrix} 2 & 1 & 3 \end{bmatrix} \\ \text{Part (2)} - \text{Machine order for each job}: & \begin{array}{c} J1 \\ J2 \\ J3 \end{array} & \begin{bmatrix} 2 & 1 \\ 1 & 2 \\ 1 & 2 \end{bmatrix} \end{array} \tag{36}$$

Part 1 of this encoded solution means that $S_2 \le S_1 \le S_3$. Part 2 specifies the processing routes. Job 1 is processed first on Machine 2, then Machine 1, Job 2 on Machine 1 then Machine 2 and Job 3 on Machine 2 then Machine 1.

We need a decoding scheme to turn an encoded solution into a schedule. The procedure must clarify the least possible starting times of the jobs in such a way that no-wait constraints are met. To this end, we set the starting time of the first job in Part 1 as 0. Then, to find the earliest possible starting time of subsequent job, we first set the starting time to be 0. The first operation of that job is scheduled. If any overlap happens between this operation and previously scheduled operations on the same machine, the starting time is set to as the first possible starting time that avoids any overlap. Then, the second operation in processing route of the job is scheduled. Again, if any overlap occurs between this operation and previously scheduled operations on that machine, we have to restart from the first operation in order to hold no-wait constraints. The procedure repeats until all the operations of the job are scheduled. Fig. 1 shows the pseudo code of the decoding scheme.

Let us numerically illustrate the procedure by decoding the solution in Eq (36). Job 2 is the first job to process. The completion times of its operations become: $O_{21}=6$ and $O_{22}=15$ (Fig. 2, Step 1). The next job is Job 1. Its first operation (O_{12}) finishes on 5 time units. It has no overlap with O_{22} (Fig. 2, Step 2); therefore, we go for its subsequent operation, i.e. O_{11}. If we schedule O_{11} in no-wait manner, there is the overlap between O_{11} and O_{21}. So, the earliest possible starting time of $O_{11}$is 6; consequently, $S_1=1$. We schedule O_{12} and O_{11} with this updated starting time (Fig. 2, Step 3). The same procedure is applied to schedule Job 3 (Fig. 2, Step 4).

4. The proposed metaheuristics

This paper proposes two metaheuristics to tackle the problem. The first one is a novel variable neighborhood search algorithm

```
Procedure Dncoding_scheme

P1 = Permutation in Part 1 of encoding scheme;          %P1 is a matrix $n \times 1$
P2 = Matrix in Part 2 of encoding scheme;               %P2 is a matrix $n \times m$

for $j = 1$ to $n$ do
    $i = 1$;
    $S_{P1[j]} = 0$;                    % $P1[j]$ is the job in $j$-th position of P1
    while $i \le m$ do
        Schedule operation $P2[P1[j], i]$   % $P2[P1[j], i]$ is $i$-th operation of Job $P1[j]$ in its processing route
        if any overlap happens do           %Overlap between $P2[P1[j], i]$ and previously scheduled operations on
                                              the same machine
            Update $S_{P1[j]}$              %To be equal to the earliest time $P2[P1[j], i]$can start without any overlap
            $i = 1$;
        elseif
            $i = i + 1$;
        endif
    endwhile
endfor
```

Fig. 1. The pseudo code of the decoding scheme.

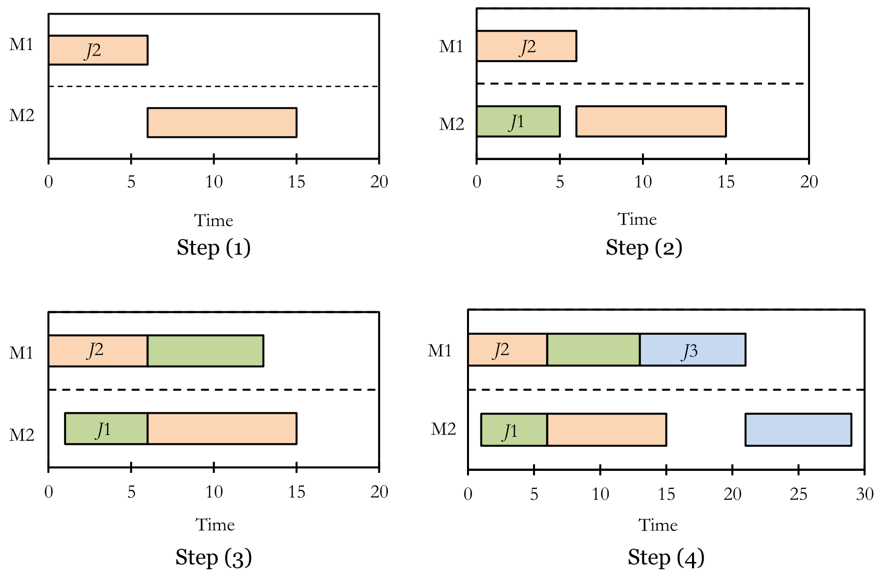

Fig. 2. The Gantt chart of the example decoded by the proposed procedure.

with very simple structure. The second one is a genetic algorithm in which we adapt some high performing operators to the type of the proposed encoding scheme. Our goal is to develop very simple yet effective algorithms for the problem. The following subsections describe in details the two metaheuristics.

4.1. *Variable neighborhood search algorithm*

One of well-known metaheuristics applied to combinatorial optimization problems is variable neighborhood search (VNS). The VNS is regarded as a local search based metaheuristic. Unlike iterated local search (ILS) algorithms iterating over one constant type of the local search approach, VNS explores solution spaces by more than one type of local search approach. The central observations of VNS are: (1) A local minimum one neighborhood structure is not necessarily the locally minimal with respect to another neighborhood structure. (2) A global optimum is the locally optimal with respect to all neighborhood structures. VNS has been applied with success to other scheduling problems including single machine (Liao and Cheng, 2007), flowshops (Naderi et al., 2009), job shops (Roshanaei et al., 2009) and the other problems (Divsalar et al., 2013).

The proposed encoding scheme has two separate parts; i.e. job order and processing route of each job. To generate a neighborhood of the current solution, we have two alternatives: (1) To introduce an operator changing an encoded solutions in both parts and (2) to propose one separate operator for each part of an encoded solution.

In the case of alternative 1, it seems that the length of neighborhoods is too large; therefore, the algorithm essentially performs a random search through search space with next possible state chosen practically uniformly over search space. Consequently, we select the second alternative and introduce two different types of local searches to explore the search space. In this case, the proposed algorithm is classified as an VNS. We introduce two types of VNSs whose local searches incorporating two different principles: greedy and curtailed insertion fashions. In greedy fashion, many neighbors are generated from the current solution and the best one is selected. In curtailed fashion, one single neighbor is produced and then it is accepted or rejected by an acceptance criterion. On one hand, the algorithms using curtailed local searches (CLS) are certainly faster than those using greedy local searches (GLS). On the other hand, algorithms using GLSs are likely more precise and systematic than those using CLSs. To evaluate the performances of both greedy and curtailed fashions, we develop two VNSs each of which applies one fashion along with a same framework. The VNS based on CLS is called VNS (CLS), and the VNS based on GLS is called VNS(GLS).

4.1.1. *Variable neighborhoods search (curtailed fashion)*

In VNS(CLS), we construct a VNS employing two types of local searches based on the curtailed fashion. The two local searches are:

Local search 1 (LS1): The first local search is centered on the first part of an encoded solution. That is, it changes the job sequence as follows: one randomly selected job is relocated into a new random position in the sequence.

Local search 2 (LS2): The second local search focuses on the second part of an encoded solution i.e. it reinvestigates the processing routes of jobs as follows: for each job *j*, chosen at random without repetition, the following procedure is applied until no improvement is obtained in three consecutive iterations: one randomly selected machine is reinserted into a new random position in the processing route of job *j*.

Framework: VNS(CLS) starts from a randomly generated solution, and explores the solution space by applying two above-mentioned local searches in a specific framework. Let the current solution be β. An iteration VNS could be described as follows: current solution β undergoes LS1. Therefore, a new candidate solution β', however inferior, is generated. Then, LS2 is applied to the β'. The nature of LS2 operates in such a way that only superior solutions are accepted. As a result, never does it worsen the β'. Despite possible improvement made by LS2, the β' might be still inferior to the β. After completion of LS2, we must decide whether or not the β' replaces the β as current solution of next iteration. The β' is accepted only if it ameliorates the β (i.e. it improves C_{max} value), it is prone to stagnation situations of the search in consequence of inadequate diversification capability. Hence, the inferior β' is even accepted if the following acceptance criterion is satisfied: a counter increases by one unit if β' is worse than β. If the counter shows a value less than 10 and any

```
Procedure VNS(CLS)

$\beta$ = Initialization;
$\beta_{Best} = \beta$;
counter1 = 1;
while stopping criterion is not met do
    $\acute{\beta}$ = local search 1 on ($\beta$);
    for $j = 1$ to $n$ do
        counter2 = 1;
        while counte2 $\leq 3$ do
            $\hat{\beta}$ = local search 2 on the processing route of a job $j$ at random without repetition ($\acute{\beta}$);
            if $C_{max}(\hat{\beta}) < C_{max}(\acute{\beta})$ do
                $\acute{\beta} = \hat{\beta}$;
                counter2 = 1;
            elseif
                counter2 = counter2+ 1;
            endif
        endwhile
    endfor
    if $C_{max}(\acute{\beta}) < C_{max}(\beta)$do
        $\beta = \acute{\beta}$;
        Update $\beta_{Best}$;
        counter1 = 1;
    elseif
        counter1 = counter1 + 1;
        if (counter1 $\geq 10$ and $rand \leq \left[\rho - \left\{\left(C_{max}(\acute{\beta}) - C_{max}(\beta)\right)/C_{max}(\beta)\right\}\right]$ do
            couner1 = 1;
            $\beta = \acute{\beta}$;
        endif
    endif
endwhile
```

Fig. 3. Outline of the proposed VNS(CLS).

improvement is made (i.e. $C_{max}(\beta') < C_{max}(\beta)$), the counter restarts form 0. When the counter takes value 10, it means no better solution from neighborhood solutions of β is found after 10 consecutive iterations of VNS(CLS). Therefore, very likely the algorithm gets stuck in a local optimum. From now on, any inferior solution β' is accepted if

$$rand \leq \left[\rho - \left(\frac{C_{max}(\beta') - C_{max}(\beta)}{C_{max}(\beta)}\right)\right]$$

where *rand* is a random number uniformly distributed over (0, 1), and ρ is an adjustable parameter that specifies the probability of acceptance of inferior solutions. If any solution is accepted, the counter is again reset. The stopping criterion is set to as a fixed computational time of $n \times m \times 0.5$ s. Fig. 3 outlines the procedure of the VNS(CLS).

4.1.2. Variable neighborhood search (greedy fashion)

VNS(GLS) is a bi-local search algorithm based on the greedy fashion. The specifications of VNS(GLS) could be described as follows:

Local search 3 (LS3): Like LS1, the third local search focuses on the first part of an encoded solution. In LS3, one randomly selected job is relocated into all the possible positions in the job order.

Local search 4 (LS4): Like LS2, the forth local search is centered on the second part of an encoded solution. Its procedure is as follows: for each job *j*, selected at random without repetition, all the machines, one by one again chosen at random with no duplication, are reinserted into all the possible positions in the processing route of job *j*. For each job *j*, the procedure iterates until no progress is obtained by relocation of one machine in the processing routes of the jobs.

Framework: In VNS(GLS), a typical iteration is performed as follows: current solution β suffers from LS3 i.e. a job is removed and reinserted into all the possible positions. The best location for that job is selected. Then, the β undergoes LS4, i.e. the processing route of each job *j* is re-investigated. If no improvement is made in a predetermined number of consecutive iterations, an operator called shaking is applied to extricate the algorithm from the probable local optimum. In this shaking operator, the positions of three jobs are randomly regenerated. To reduce the risk of jumping into a further area, we produce 10 neighbors by shaking operator and choose the best one. When shaking operator is applied the counter restarts. We have the same stopping criterion as that of VND(CLS) (Fig. 4).

4.2. Genetic algorithm

Genetic algorithm (GA) is one of the well-known and historical population based metaheuristics. There is almost no NP-hard problem not being attacked by GAs in the literature. This is so because GAs' framework is very adaptable to any operators and problems. Many papers (Bennell et al., 2013; Ruiz et al., 2006; Zhang et al., 2013) report the effectiveness of GA. This paper develops a simple GA having powerful operators. They are high performing operators of the literature that are adapted to the proposed encoding scheme. GA begins from a set of solutions, called *population*. A typical iteration of GA, called *generation*, is as follows. Some of the solutions of current population, called *parents*, are chosen by a random procedure, called *selection mechanism*. An operator, called *crossover*, combines the parents and produces new solutions, called *offspring*. The offspring probably undergoes another operator, called *mutation*, for further changes. In each generation, some of the best solutions of previous

```
Procedure VNS(GLS)

$\beta$ = Initialization;
$\beta_{Best} = \beta$;
counter = 1;
while stopping criterion is not met do
    Local search 3 on ($\beta$);
    for $j = 1$ to $n$ do
        Local search 2 on the processing route of a job $j$ at random without repetition ($\beta$);
    endfor
    if improvement is made do
        counter = 1;
        Update $\beta_{Best}$;
    elseif
        counter = counter + 1;
        if counter $\geq 4$ do
            counter = 1;
            Shaking operator on ($\beta$);
        endif
    endif
endwhile
```

Fig. 4. Pseudo code of the proposed VNS(GLS).

```
Procedure MGA

Initialization;
while stopping criterion is not met do
    $\beta_1, \beta_2$= Perform elite strategy;        %two best solutions are transferred into next population
    for $v = 3$ to $popsize$ do
        for $d = 1$ to 2 do
            Select Parent $d$-th using tournament mechanism
        endfor
        $\beta_v$= Cross the parents using SJ2OX;
        Perform LS2 on the produced offspring $\beta_v$;
    endfor
    for $v = 2$ to $popsize$ do
        if mutation criterion for offspring $\beta_v$ is satisfied do
            Mutate offspring $\beta_v$ using LS1;
        endfor
    endfor
    Reproduction;                    %Construct the next population using $\beta$s
Endwhile
```

Fig. 5. The pseudo code of the proposed MGA.

generations are directly brought into the current generation, called *elite strategy*. GA evolves until a stopping criterion is fulfilled.

We now describe the specifications of the proposed GA. Initial solutions (*popsize*) are randomly generated. Two of the best solutions of the current population are directly copied into the next population. Selection mechanism is tournament method which could be stated as follows: to select a parent two solutions are randomly selected, then the better chosen solution is accepted as a parent. Since crossovers usually need two parents, the procedure is executed twice. The crossover is "similar job 2-point order" or SJ2OX first proposed by Ruiz et al. (2006) for the permutation flowshops. By crossover, we intend to improve the job order (i.e. part1 of an encoded solution). Since the part1 is the permutation of the jobs, we apply the SJ2OX on it. SJ2OX has three steps:

1) Two parents are compared on a position-by-position basis. If we have identical jobs in the same positions, they are copied over to both offspring.
2) Two random cut points are taken and the jobs between these two points are directly copied to the offspring.
3) The missing jobs of each offspring are copied in the relative order of the other parent. After crossover, we should select an operator working on part2 of an encoded solution because the algorithm takes care of part1 by crossover. To this end, we make use of the above-mentioned local search 2 employed by VNS(CLS). That is, each offspring generated by crossover undergoes LS2.

Mutation procedure is as follows: each offspring suffers LS1 (i.e. one job in part1 of encoding scheme is shifted to a new random position) if a predefined mutation criterion is satisfied. Each offspring undergoes the mutation: (1) if it is not a unique solution or (2) if a randomly generated number, uniformly distributed between (0, 1), is less than 0.2. By employing a mutation operator, we expect the GA to diversify the solutions in each population and avoid convergence to local optima. It seems the great choice of the mutation can greatly influence GA's performance. In this regard, we develop two GAs: one without the mutation, called GA, and the other incorporating the mutation, called MGA. We again use the same stopping criterion as we used for VNSs. The pseudo code of the proposed MGA is shown in Fig. 5.

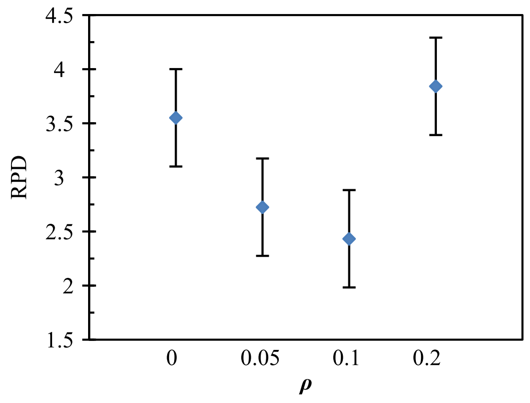

Fig. 6. The means plot and LSD intervals for the different levels of ρ.

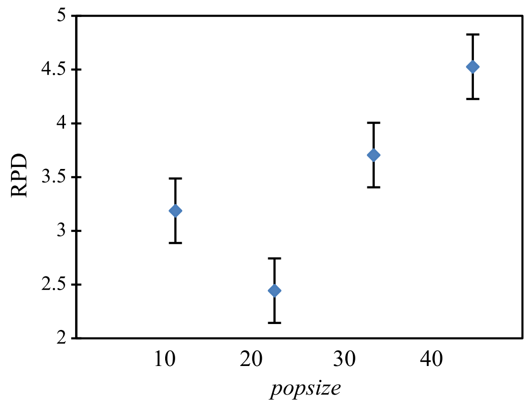

Fig. 7. The means plot and LSD intervals for the different levels of *popsize*.

5. Computational evaluation

This section first tunes the parameters of the proposed metaheuristics, then the efficiency of the proposed MILP models is evaluated on a computational experiment including small-sized instances. Afterwards, we investigate the general performance of the proposed metaheuristics (i.e. VNS(CLS), VNS(GLS), GA and MGA) against the optimal points obtained by the models. Moreover, we further evaluate the performance of the algorithms against benchmarks in the literature of pure open shops. Since existing optimal solutions of the benchmarks are for pure open shops, they cannot be used for the evaluation. In this case, we use a performance measure named Relative Percentage Deviation (RPD) obtained by the following formula:

$$\mathrm{RPD} = \frac{Alg_{sol} - Min_{sol}}{Min_{sol}} 100 \tag{37}$$

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 CPLEX 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 $n \times m \times 0.5$ s. This stopping criterion permits for more time as the number of jobs or machines increases.

5.1. Parameter tuning of the proposed metaheuristics

To set the parameter of the algorithms, we generate a set of 30 instances. We have (n, m): (5, 5), (10, 10), (20, 20). The processing times are uniformly distributed on [1, 99]. There are 10 instances for each combination of n and m. Since each of the proposed algorithms has at most one parameter that needs to be tuned, we utilize the analysis of variance (ANOVA) to conduct the experiment.

VNS(CLS) has only one parameter, ρ, that is used in acceptance criterion of VNS(CLS). We examine the values of 0, 0.2, 0.4 and 0.6. By considering value 0 (meaning that only better solution is accepted), we can evaluate the presence of systematic acceptance criterion. Fig. 6 shows the results of the experiment (means and least significant difference or LSD) for ρ. As it could be seen, ρ of 10 works better than the other values.

MGA (and GA) also has one parameter: number of solutions in the population (*popsize*). We consider *popsize*=10, 20, 30 and 40. Fig. 7 plots the means and LSD intervals. *popsize* of 20 outperforms the other levels.

5.2. Evaluation of the MILP models on small-sized instances

This subsection compares the efficiency of the MILP models to solve NW-OSS problems. We generated a set of different instances as follows. We have 10 problem sizes ranging from ($n \times$ m)= (3×3), up to (6×6). The processing times are randomly distributed over (1, 99). For each problem size, we generate 2 instances. The MILP models are allowed a maximum of 900 s (15 min) of computational time.

Table 4 shows the results obtained by different models. If an instance is solved to optimality by one of the models, in the corresponding interchange in Table 4, makespan and computational time (in second) of the model is reported; otherwise, it is

Table 4
The results obtained by the models for NW-OSS.

$n \times m$	Instances	Model 1		Model 2		Model 3	
		C_{max}	Time	C_{max}	Time	C_{max}	Time
3×3	1	125	0.05	125	0.05	125	0.01
	2	244	0.05	244	0.03	244	0.03
3×4	1	275	1.04	275	5.52	275	0.08
	2	239	0.28	239	1.79	239	0.06
4×3	1	189	1.53	189	0.0.81	189	0.06
	2	280	20.30	280	9.75	280	0.19
4×4	1	218	290.41	218	642.91	218	0.20
	2	322	229.02	322	548.69	322	0.44
4×5	1	–	–	–	–	316	1.64
	2	–	–	–	–	337	2.73
5×4	1	–	–	–	–	307	28.30
	2	–	–	–	–	309	17.89
5×5	1	–	–	–	–	369	137.80
	2	–	–	–	–	305	4.43
5×6	1	–	–	–	–	430	695.90
	2	–	–	–	–	326	108.64

Table 5
The RPDs obtained by the algorithms on the small-sized instances.

$n \times m$	Algorithms			
	VNS(CLS)	VNS(GLS)	GA	MGA
3×3	2	2	2	2
3×4	2	2	2	2
4×3	2	2	1	2
4×4	2	2	1	2
4×5	2	1	1	2
5×4	2	1	0	1
5×5	1	1	0	0
5×6	0	0	0	0
Ave.	88%	69%	44%	69%

marked with "–", i.e. the corresponding model does not optimally solve the given instances. For example, the first instance of problem size (5×5) is only solved by Model 3 in 10.9 s. The performance of Models 1 and 2 are similar to each other. The both solve the problem sizes up to (4×4). It might be interesting to state that Model 1 is slightly faster than Model 2 on average. Obviously, the highest performing MILP is Model 3 which manages to solve the instances up to (5×6).

5.3. Evaluation of the metaheuristics on small-sized instances

We are going to evaluate the algorithms (i.e. VNS(CLS), VNS (GLS), GA and MGA) against the optimal solution obtained by the models in the previous subsection. Table 5 shows the results. The best performing algorithm is VNS(CLS) by optimally solving 13 instances out of 16 instances (almost 88% of the instances). The second best are obtained by both MGA and VNS(GLS) by finding optimal solutions of 11 instances (almost 69% of the instances). GA only provides 6 optimal solutions (almost 44% of the instances); therefore, it is the worst performing algorithm.

5.4. Evaluation of the proposed algorithms on large-sized instances

After having investigated the general performance of the metaheuristics, we intend to further evaluate the proposed algorithms against standard benchmarks in the literature, i.e. Taillard (1993), Brucker et al. (1997) and Guéret and Print (1998). Taillard's benchmark includes 50 instances ranging from $(n=m=4)$ to $(n=m=20)$. These instances are known to be easy to solve due to uniform distribution of the processing times. The Brucker et al.'s benchmark has 52 harder instances sized from $(n=m=3)$ up to $(n=m=8)$. In these instances, total processing times of each job j (i.e. $\sum_{i=1}^{m} P_{j,i}$) and machine loads (i.e. $\sum_{j=1}^{n} P_{j,i}$) are either equal to 1000 time units or very close to this value. Guéret and Print's benchmark consists of 80 instances systematically generated from $(n=m=3)$ up to $(n=m=10)$. It is said that these instances are even harder than those previously generated by Brucker et al. (1997).

Table 6
The average RPD obtained by each of the algorithms against three well-known benchmarks.

benchmark	VNS(CLS)	VNS(GLS)	GA	MGA	TPSM
Taillard					
tail_4 × 4	0.00	2.26	3.53	0.00	3.89
tail_5 × 5	1.30	3.51	4.49	2.48	5.54
tail_7 × 7	1.67	3.17	4.76	3.11	6.08
tail_10 × 10	2.66	4.46	3.68	0.13	6.35
tail_20 × 20	4.08	5.64	3.46	1.66	8.46
Guéret and Prins					
gp03	0.00	2.97	2.41	0.00	3.21
gp04	0.00	3.98	3.05	0.00	4.80
gp05	0.48	3.90	4.50	1.89	5.06
gp06	1.56	3.49	4.93	2.34	5.18
gp07	2.15	3.10	5.33	2.49	4.73
gp08	1.26	3.77	4.91	1.53	5.04
gp09	2.32	4.54	4.83	1.06	5.10
gp010	3.05	5.69	5.43	2.80	6.82
Brucker et al.					
j3	0.00	0.00	1.34	0.00	2.69
j4	0.00	2.24	3.08	1.24	3.35
j5	0.89	3.70	4.72	2.77	5.93
j6	1.61	3.87	4.46	3.33	6.46
j7	1.86	3.21	3.58	1.49	4.58
j8	1.90	4.04	4.38	2.50	5.55
Average	1.44	3.61	4.08	1.61	5.24

In this subsection, we also test and compare our proposed algorithms against the adaptation of a relevant algorithm in the close literature. The most of the methods in the literature, shown by Table 1, are not proper to be in the comparison. The algorithm proposed by Adiri and Amit (1984) is a dispatching rule for a case where all the processing times are equal. The methods of Sidney and Sriskandarajah (1999) and Liaw et al. (2005) are approximation algorithm and a branch and bound method, respectively; and they cannot be brought into the comparison. The algorithms of Yao and Soewandi (2000) and Yao and Lai (2002) are for the case of two-machine NW-OSS and do not seem extendable to the case of multi-machine NW-OSS. Hence, the only remaining choice is the Two-Phase Solution Method (or TPSM) of Lin et al. (2008). Of course, this method is developed for the minimization of the total occupation time of all the machines. In this case, we adapt the solution method so as to minimize makespan.

We use RPD (Eq. 37) to compare the algorithms. Table 6 summarizes the results of the experiments averaged for each combination of n and m of the three benchmarks. VNS(CLS) is the best performing algorithm with RPD of 1.44%. MGA provides the second best results with RPD of 1.61%. TPSM is the worst performing algorithm with RPD of 5.22%. Comparing MGA and GA shows the importance of the mutation operator to solve the problem studied here. Comparing VNS(CLS) and VNS(GLS) concludes the algorithms utilizing the curtailed fashion is more effective than those utilizing greedy fashion for such a complex problem.

To further statistically analyze the results, we carry out an "analysis of variance" or ANOVA. Due to clear difference between GA and the other algorithms, it is excluded from the experiment. The results demonstrate that there are significant differences

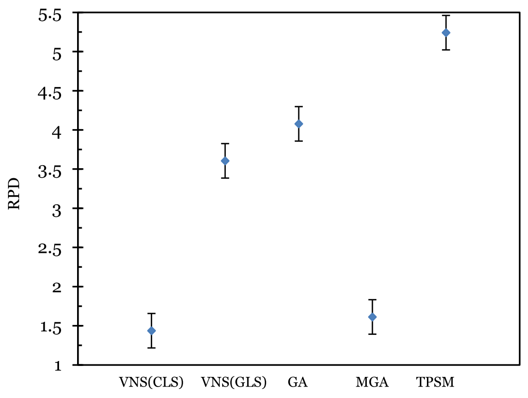

Fig. 8. Means plot and LSD intervals for the metaheuristics.

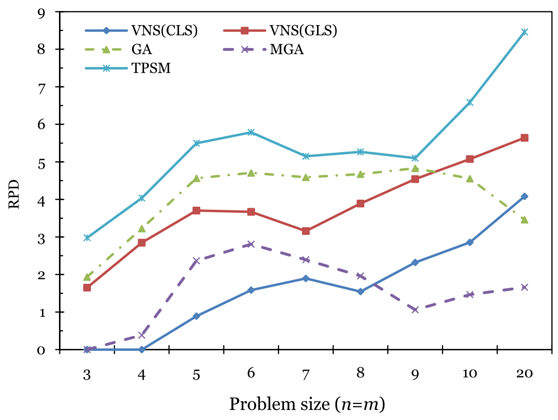

Fig. 9. Means plot for the metaheuristics versus the problem size.

between the algorithms with p-value very close to 0. Fig. 8 shows the means plot and least significant difference or LSD intervals at 95% confidence level for the different algorithms. As could be seen, VNS(CLS) and MGA are statistically similar, while the both outperform VNS(GLS) and GA. It is also interesting to plot the performance of the algorithms versus the problem size. Fig. 9 shows the means obtained by the algorithms in the different problem sizes. In instances up to $n = m = 8$, VNS(CLS) outperforms MGA. In large problem sizes, MGA provides better results than VNS(CLS).

6. Conclusions and future studies

This paper dealt with a specific case of open shop problems, known as no-wait open shop, under makespan minimization. Three mathematical formulations were constructed to solve the problem to optimality. The models were in the form of mixed integer linear programs. We then introduced an effective procedure of the encoding and decoding schemes by which we were enabled to propose novel metaheuristics based on genetic algorithms and variable neighborhood search to solve the large-sized problems in an acceptable computational time. The proposed variable neighborhood search metaheuristics were centered on two different concepts, greedy and curtailed fashions. The proposed genetic algorithms were designed based a simple framework incorporating powerful operators.

Two computational experiments were carried out to evaluate the performances of models and metaheuristics. In the first experiment, we generated small-sized instances by which we compared the mathematical models and evaluated general performance of the proposed metaheuristics. In the second experiment, potential of metaheuristics on solving some benchmarks in the literature of pure open shops were further evaluated. All the results supported that the models and metaheuristics were effective to tackle the no-wait open shop problems. As an interesting future research, one can study multi-objective no-wait open shop problems. Another impressive research is to present a branch-and-bound or any other exact methods.

References

Adiri, I., Amit, N., 1984. Open shop and flow shop scheduling to minimize sum of completion times. Comput. Oper. Res. 11, 275–284.

Andresen, M., Bräsel, H., Morig, M., Tusch, J., Werner, F., Willenius, P, 2008. Simulated annealing and genetic algorithms for minimizing mean flow time in an open shop. Math. Comput. Model. 48 (7–8), 1279–1293.

Bennell, J.A., Lee, L.S., Potts, C.N., 2013. A genetic algorithm for two-dimensional bin packing with due dates. Int. J. Prod. Econ. 145 (2), 547–560.

Bräsel, H., Herms, A., Morig, M., Tautenhahn, T., Tusch, T., Werner, F., 2008. Heuristic constructive algorithms for open shop scheduling to minimize mean flow time. Eur. J. Oper. Res. 189 (3), 856–870.

Brucker, P., Hurink, J., Jurisch, B., Wöstmann, B.A., 1997. A branch and bound algorithm for the open-shop problem. Discret. Appl. Math. 76, 43–59.

Divsalar, A., Vansteenwegen, P., Cattrysse, D., 2013. A variable neighborhood search method for the orienteering problem with hotel selection. Int. J. Prod. Econ. 145 (1), 150–160.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: a survey. Ann. Discret. Math. 5, 287–326.

Goyal, S.K., Sriskandarajah, C., 1988. No-wait shop scheduling: computational complexity and approximate algorithms. Oper. Res. 25, 220–244.

Guéret, C., Prins, C., 1998. Classical and new heuristics for the open shop problem: a computational evaluation. Eur. J. Oper. Res. 107, 306–314.

Hall, N.C., Sriskandarajah, C.A., 1996. Survey of machine scheduling problems with blocking and no-wait in process. Oper. Res. 44, 510–525.

Liao, C.J., Cheng, C.C., 2007. A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. Comput. Ind.l Eng. 52, 404–413.

Liaw, C.F., 2000. A hybrid genetic algorithm for the open shop scheduling problem. Eur. J. Oper. Res. 124, 28–42.

Liaw, C.F., Cheng, C.Y., Chen, M., 2005. Scheduling two-machine no-wait open shops to minimize makespan. Comput. Oper. Res. 32, 901–917.

Lin, H.T., Lee, H.T., Pan, W.J., 2008. Heuristics for scheduling in a no-wait open shop with movable dedicated machines. Int. J. Prod. Econ. 111, 368–377.

Manne, A.S., 1960. On the job shop scheduling problem. Oper. Res. 8, 219–223.

Naderi, B., Zandieh, M., Fatemi Ghomi, S.M.T., 2009. A study on integrating sequence dependent setup time flexible flow lines and preventive maintenance scheduling. J. Intell. Manuf. 20, 683–694.

Pang, K.W., 2013. A genetic algorithm based heuristic for two machine no-wait flowshop scheduling problems with class setup times that minimizes maximum lateness. Int. J. Prod. Econ. 141 (1), 127–136.

Pinedo, M.L., 2008. Scheduling: Theory, Algorithms, and Systems, third ed.. Springer Science+Business Media, New York.

Sahni, S., Cho, Y., 1979. Complexity of scheduling shops with no-wait in process. Math. Oper. Res. 4, 448–457.

Sidney, J.B., Sriskandarajah, C., 1999. A heuristic for the two-machine no-wait open shop scheduling problem. Naval Res. Logist. 46, 129–145.

Stafford Jr. E.F., Tseng, F.T., Gupta, J.N.D., 2005. Comparative evaluation of MILP flowshop models. J. Oper. Res. Soc. 56, 88–101.

Rios-Mercado, R.Z., Bard, J.F., 1998. Computational experience with a branch-and-cut algorithm for flowshop scheduling with setups. Comput. Oper. Res. 25, 351–366.

Roshanaei, V., Naderi, B., Jolai, F., Khalili, M., 2009. A variable neighborhood search for job shop scheduling with setup times to minimize makespan. Futur. Gener. Comput. Syst. 25, 654–661.

Ruiz, R., Maroto, C., Alcaraz, J., 2006. Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34, 461–476.

Taillard, E., 1993. Benchmarks for basic scheduling problems. Eur. J. Oper. Res. 64, 278–285.

Wagner, H.M., 1959. An integer linear programming model for machine scheduling. Naval Res. Logist. Q. 6, 131–140.

Yao, M.J., Lai, C.W., 2002. A genetic algorithm for the two machine open shop scheduling problem with blocking. In: The Second Japanese-Sino Optimization Meeting (JSOM 2002), Kyoto, Japan, pp. 25–27.

Yao, M.J., Soewandi, H., 2000. Simple heuristics for the two machine open shop problem with blocking. J. Chin. Inst. Ind. Eng. 17 (5), 537–547.

Zhang, R., Chang, P.C., Wu, C., 2013. A hybrid genetic algorithm for the job shop scheduling problem with practical considerations for manufacturing costs: investigations motivated by vehicle production. Int. J. Prod. Econ. 145 (1), 38–52.