Int J Adv Manuf Technol (2015) 76:1705–1718
DOI 10.1007/s00170-014-6390-6

ORIGINAL ARTICLE

Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times

Oliver Avalos-Rosales · Francisco Angel-Bello · Ada Alvarez

Received: 2 September 2013 / Accepted: 11 September 2014 / Published online: 25 September 2014

Abstract In this paper, an unrelated parallel machine scheduling problem with sequence and machine-dependent setup times and makespan minimization is studied. A new makespan linearization and several mixed integer formulations are proposed for this problem. These formulations outperform the previously published formulations regarding size of instances and computational time to reach optimal solutions. Using these models, it is possible to solve instances six times larger than what was previously solved and to obtain optimal solutions on instances of the same size up to four orders of magnitude faster. A metaheuristic algorithm based on multi-start algorithm and variable neighbourhood descent metaheuristic is proposed. Composite movements were defined for the improvement phase of the proposed metaheuristic algorithm that considerably improved the performance of the algorithm providing small deviations from optimal solutions in medium-sized instances and outperforming the best known solutions for large instances.

Keywords Scheduling · Unrelated parallel machines · Dependent setup times · Makespan · Multi-start · VND

O. Avalos-Rosales · A. Alvarez
Universidad Autónoma de Nuevo León, Av. Universidad s/n, San Nicolás de los Garza, NL, Mexico

O. Avalos-Rosales
e-mail: oliver@yalma.fime.uanl.mx

A. Alvarez
e-mail: ada.alvarezs@uanl.mx

F. Angel-Bello (✉)
Tecnológico de Monterrey, Campus Monterrey, Av. Eugenio Garza Sada 2501, Monterrey, NL, Mexico
e-mail: fangel@itesm.mx

1 Introduction

In this work, a problem is addressed in scheduling n jobs on m unrelated parallel machines with the objective of minimizing the makespan, taking into consideration setup times that depend both on the machine and the sequence.

The parallel machine scheduling problem has been extensively studied due to its practical applications in various manufacturing systems such as printed circuit board manufacturing, group technology cells, semiconductor manufacturing, painting and plastic industries, injection moulding processes, remanufacturing, etc. [11]. An interesting survey on parallel machines can be found in [17]. However, most of the literature address identical or uniform machines, where the processing time of a job is the same regardless of the machine where it is processed or is proportional to the speed of the machine respectively. A recent work on non-identical parallel machines addressing fuzzy processing times is presented by Balin [3].

Less research has been conducted for studying the case where the processing time of each job depends on the machine on which it is processed, that is, the machines are unrelated [5]. This is a common situation in several applications where there are parallel machines with different capabilities.

From studies dealing with unrelated parallel machines, only a few addresses the problem considering setup times. Most assume that there are no setup costs or they are independent of job sequence. Recently, Lin and Yang-Kuei and Chi-Wei [14, 24] studied scheduling problems on unrelated parallel machines with release dates and without setup times. Lin [14] proposed a Particle Swarm Optimization algorithm for minimizing the makespan, and Yang-Kuei and Chi-Wei [24] proposed several dispatching rules considering three performance measures.

However, this situation may not always be true in practice. A setup is a set of operations that should be performed after processing a job on a machine to prepare it for processing the next job. In various real-world industrial/service environments, these times are sequence-dependent, that is, depending not only on the job that will be processed but also on the job processed just before [12].

The number of works addressing unrelated parallel machine scheduling problems with sequence-dependent and machine-dependent setups are even fewer, although this situation appears, for example, in the textile, printed circuit boards and chemical industries [19] as well as in the injection moulding process and in LCD manufacturing processes [4]. All that has been previously mentioned gave the motivation to focus on this kind of problem.

There are several performance criteria to measure the quality of a scheduling. Recent results using metaheuristic algorithms related to minimize the total tardiness on parallel machines with sequence and machine-dependent setups times can be found in [4, 11, 13, 15].

One of the performance criteria most broadly used is the minimization of the maximum completion time of the schedule, which is known as makespan ($C_{\max}$). It is a very important measure of performance since it gives the total time elapsed in processing all jobs under consideration [6]. The makespan is relevant in situations when a received batch of jobs should be completed as soon as possible [1]. This kind of situation is especially common in server farms, data centers and compute cloud (e.g. the Amazon Elastic Compute Cloud) [21].

The problem of minimizing the makespan considering two identical machines is a NP-hard [8]. Indeed, a problem with unrelated machines and sequence-dependent setups is also NP-hard. This, coupled with the fact that re-schedules are often required, usually prevents the use of exact algorithms, and it is not surprising that many of the methodologies that have been developed are based on heuristics.

Regarding the unrelated parallel machine scheduling problem (UPMSP) with sequence and machine-dependent setup times and makespan minimization objective, some heuristic algorithms have been developed recently. Helal et al. [10] proposed a Tabu Search algorithm; Rabadi et al. [19] introduced a new metaheuristic, MetaRaSP, which incorporates randomness within priority rules to construct a feasible solution. More recently, Arnaout et al. [2] proposed an ant colony algorithm, Ying et al. [25] developed a simulated annealing approach which incorporates a restricted search strategy, Fleszar et al. [7] proposed a multi-start variable neighbourhood descent metaheuristic hybridized with mathematical programming and Vallada and Ruiz [23] presented a genetic algorithm which exhibits a new crossover operator including a local search procedure.

Only a few studies have developed exact methods to solve the problem addressed in this paper. Tran and Beck [22] proposed a Benders decomposition-based method for minimizing the makespan, while Rocha et al. [20] presented a branch and bound approach for minimizing the makespan plus the weighted tardiness.

In this study, an unrelated parallel machine scheduling problem with sequence and machine-dependent setup times and makespan minimization is studied. From the literature review, it is clear that this problem has not been sufficiently studied in the available literature.

The main contributions of this paper can be summarized as:

- New formulations for the problem and a new linearization for the makespan are proposed, which in conjunction outperform the previously published formulations regarding size of instances and computational time to reach optimal solutions
- A multi-start algorithm based on the variable neighbourhood descent metaheuristic was designed and implemented, which outperforms the best known results from literature.
- Composite movements were defined for the improvement phase of the proposed metaheuristic algorithm that considerably improved the performance of the algorithm.

2 Problem formulation

The following assumptions and notations are used to describe the problem:

- There is a set M of m parallel machines.
- Machines are continuously available, and each machine can handle one job at a time without preemption, that is, once the processing of a job has started, it cannot be interrupted.
- There is a set N of n jobs to be scheduled.
- All the jobs are available in time 0. No precedence constraints among jobs are imposed.
- Each job j has associated a processing time p_{ij} in each machine i.
- There is a machine-dependent setup time s_{ijk} for processing job k just after job j. In general, $s_{ijk} \neq s_{ikj}$.
- The objective is to minimize the makespan $C_{\max}$.

Figure 1a illustrates that setup times are asymmetric, while Fig. 1b illustrates that processing times and setup times are machine dependent.

Finding a solution to the problem means to determine the allocation of jobs to machines and the order in which each machine will process the assigned jobs.

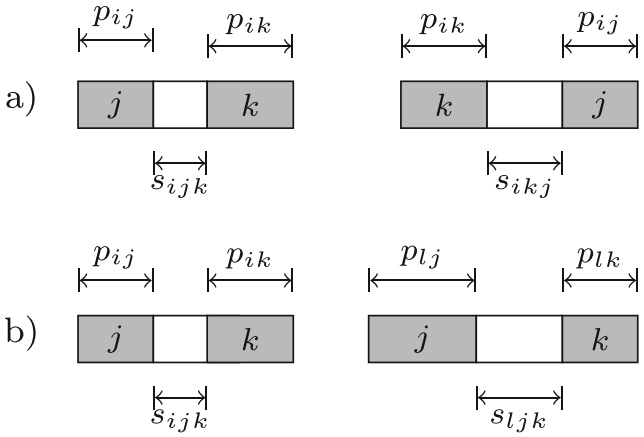

Fig. 1 Graphical representation of processing times and setup times

Figure 2 shows a graphical representation of a solution to the addressed problem with 17 jobs and 3 machines. In the figure, blank blocks represent setup times and grey blocks represent processing times.

First, two mixed integer formulations, referred to as Model 1 and Model 2, are proposed to formalize the problem. Model 1 is a modification of formulations presented in [19, 23]. Model 2 is a generalization of the one proposed by Tran and Beck [22] for an alternative resource scheduling that was used in a Benders decomposition process. From this, several variations of these models have been obtained including a new linearization for the makespan.

In order to establish these formulations, let us introduce the following variables.

$$X_{ijk} = \begin{cases} 1, & \text{if job } j \text{ is scheduled before job } k \text{ in} \\ & \text{machine } i, \\ 0, & \text{otherwise.} \end{cases}$$

C_j : completion time of job j.

The set N plus a dummy job 0 will be denoted by N_0 and V will denote a very large constant. Variables X_{i0k} and X_{ij0} are used to specify which jobs k and j will be processed first and at the end of each machine i, respectively. The processing times and setup times associated to the dummy job are considered 0 ($p_{i0} = 0$, $s_{i0k} = 0$ and $s_{ij0} = 0$).

Model 1 can be stated as

$$\min C_{\max}. \tag{1}$$

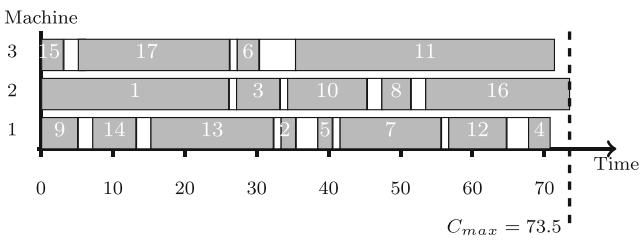

Fig. 2 Graphical representation of a solution

Subject to:

$$\sum_{i\in M}\sum_{\substack{j\in N_0\\ j\neq k}} X_{ijk} = 1, \qquad \forall k \in N, \tag{2}$$

$$\sum_{i\in M}\sum_{\substack{k\in N_0\\ j\neq k}} X_{ijk} = 1, \qquad \forall j \in N, \tag{3}$$

$$\sum_{\substack{k\in N_0\\ k\neq j}} X_{ijk} - \sum_{\substack{h\in N_0\\ h\neq j}} X_{ihj} = 0, \quad \forall j \in N, \quad \forall i \in M, \tag{4}$$

$$\sum_{k\in N} X_{i0k} \leq 1, \qquad \forall i \in M, \tag{5}$$

$$C_k - C_j + V(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \quad \forall j \in N_0, \forall k \in N, j \neq k, \quad \forall i \in M, \tag{6}$$

$$C_0 = 0, \tag{7}$$

$$C_j \leq C_{\max}, \qquad \forall j \in N, \tag{8}$$

$$X_{ijk} \in \{0, 1\}, \quad \forall j \in N_0, \forall k \in N_0, j \neq k, \quad \forall i \in M,$$

$$C_j \geq 0, \qquad \forall j \in N. \tag{9}$$

Objective (1) minimizes the makespan of the solution. Constraints (2) establish that every job has exactly one predecessor, while constraints (3) establish that every job has exactly one successor. Constraints (4) are the so-called "flow conservation constraints". They ensure that if a job is scheduled in a machine, then it has a predecessor and a successor in that machine. Constraints (5) ensure that at most, one job is scheduled as the first job on each machine. Constraints (6) provide a right processing order, avoiding loops. Basically, they establish that if $X_{ijk} = 1$, then $C_k \geq C_j + s_{ijk} + p_{ik}$. If $X_{ijk} = 0$, the constraint becomes redundant. Constraint (7) sets the completion time of the dummy job to 0. Constraints (8) linearize the objective function. Finally, constraints (9) define the nature of the variables.

For the second formulation, in addition to the above defined variables, let us introduce the following ones.

$$Y_{ik} = \begin{cases} 1, & \text{if job } k \text{ is assigned to machine } i, \\ 0, & \text{otherwise.} \end{cases}$$

Model 2 can be obtained from Model 1 replacing constraints Eqs. 2, 3 and 4 by

$$\sum_{i\in M} Y_{ik} = 1, \qquad \forall k \in N, \tag{10}$$

$$Y_{ik} = \sum_{j\in N_0, j\neq k} X_{ijk}, \quad \forall k \in N, \qquad \forall i \in M, \tag{11}$$

$$Y_{ij} = \sum_{k\in N_0, k\neq j} X_{ijk}, \quad \forall j \in N, \qquad \forall i \in M, \tag{12}$$

and adding $Y_{ik} \geq 0, \quad \forall k \in N, \forall i \in M$ to constraints (9).

Constraints (10) ensure that each job is assigned exactly to one machine. Constraints (11) establish that every job

has exactly one predecessor and both are assigned to the same machine. Constraints (12) guarantee that every job has exactly one successor and both are assigned to the same machine.

Model 1 has $n^2m + nm$ binary variables, $n + 1$ continuous variables and $n^2m + nm + 3n + m + 1$ constraints, while Model 2 has $n^2m + nm$ binary variables, $nm + n + 1$ continuous variables and $n^2m+2nm+2n+m+1$ constraints.

Note that in both formulations, the makespan $C_{\max} = \max_{j\in N}\{C_j\}$ has been linearized as

$$C_j \leq C_{\max}, \qquad \forall j \in N.$$

When Model 1 and Model 2 were solved using the Branch and Bound algorithm (B&B), it was observed that the lower bounds yielded by the linear relaxations were very weak.

This gave the motivation to propose a different representation for the objective function. This representation was obtained from the analysis of the structure of the solutions to the linear relaxations for Model 1 and Model 2.

Specifically, when solving the linear relaxations for different data instances, we realized that the values of C_j were near 0. With these values for these variables, also the objective function value was near to 0.

For obtaining a new linearization for the makespan, the concept of machine span will be used. The span O_i of machine i is defined as the completion time of the last job scheduled in machine i. Using variables X_{ijk}, the span O_i can be calculated as

$$O_i = \sum_{\substack{j\in N_0\\ j\neq k}} \sum_{k\in N} (s_{ijk} + p_{ik})X_{ijk}, \qquad \forall i \in M,$$

and then the makespan can be linearized as

$$O_i \leq C_{\max}, \qquad \forall i \in M,$$

or equivalently as

$$\sum_{\substack{j\in N_0\\ j\neq k}} \sum_{k\in N} (s_{ijk} + p_{ik})X_{ijk} \leq C_{\max}, \qquad \forall i \in M. \tag{13}$$

In addition, using variables Y_{ik} linearization (13) can be rewritten as

$$\sum_{\substack{j\in N_0\\ j\neq k}} \sum_{k\in N} s_{ijk} * X_{ijk} + \sum_{k\in N} p_{ik} * Y_{ik} \leq C_{\max}, \qquad \forall i \in M. \tag{14}$$

It is not difficult to see that any feasible solution satisfies these constraints, which means that constraints Eqs. 13 and 14 are valid inequalities to Model 1 and Model 2, respectively. On the other hand, the left side of these constraints easily compute the time at which each machine processes its last job (O_i), as the sum of the setup time for processing each job plus the processing time of that job. In addition, they do not depend on the large constant V and force $C_{\max}$ to take a positive value if any of the variables X_{ijk} or Y_{ik} are positive.

In order to investigate which combination of defined variables with makespan linearization produces better results, two alternatives have been considered for each formulation (Model 1 and Model 2):

(a) To change the linearization of the makespan. It means to replace constraints (8) by constraints (13) in Model 1 and by constraints (14) in Model 2, obtaining Model 1a and Model 2a, respectively.
b) To include both linearizations. It means to add constraints (13) to Model 1 and constraints (14) to Model 2, obtaining Model 1b and Model 2b, respectively.

In short, the formulations considered in this work have the following characteristics:

	x_{ijk}	y_{ij}	$C_j \leq C_{\max}$	$O_i \leq C_{\max}$
Model 1	✓		✓	
Model 2	✓	✓	✓	
Model 1a	✓			✓
Model 2a	✓	✓		✓
Model 1b	✓		✓	✓
Model 2b	✓	✓	✓	✓

Note than Model 1a and Model 2a have $(n - m)$ less constraints than Model 1 and Model 2, respectively, while Model 1b and Model 2b have (m) more constrains than their corresponding formulations Model 1 and Model 2.

In the computational experiment section, it will be shown that the new makespan linearization allows to solve larger instances and accelerates the solution process based on branch and bound. Also, a detailed analysis of the behaviour of previous formulations will be given.

3 Multi-start algorithm

When designing a solution method for scheduling problems, one should consider that usually, it is hard in this context to define neighbourhoods that preserve feasibility, but constructing a feasible solution may be a more simple process. On the other hand, diversification should be an important component of the method for not being stuck in small areas of the solution space. Multi-start methods offer a framework that all together provides diversification and take advantage of the ease of constructing solutions.

A multi-start method is one that executes multiple times from different initial points in the solution space. The earliest works in multi-start methods were developed for nonlinear unconstrained optimization problems and consisted of the evaluation of the objective function at randomly generated points. In metaheuristic optimization, multi-start

algorithms are composed, generally, of two phases: (1) diversification generator and (2) improvement method. A multi-start algorithm iterates between these two phases while saving the best solution found throughout the improvement method.

The function of the diversification generator is to provide starting solutions to the improvement phase, while ensuring a certain degree of difference between them. In order to endow diversification, randomness or some kinds of memory are commonly included in methods that are used to provide starting solutions. Another way to include diversity is to generate new solutions by perturbing the previously generated solutions. Regarding the improvement methods, local search procedures are, in general, simple and powerful. In these procedures, the search for a new solution is conducted with respect to a neighbourhood structure. Usually, the neighbours of a given solution are feasible solutions that can be obtained through simple movements. The search begins at an initial feasible solution, explores the neighbourhood and moves to a neighbouring solution that improves the value of the objective function if this improved solution exists. When a new solution is found, the search is restarted. The search terminates when for the current solution, any neighbouring improved solution does not exist, that is, when the search reaches a local minimum. For more information about multi-start methods, see the work of Marti et al.[16] .

Figure 3 shows a general pseudo-code for a generic multi-start algorithm.

Next, the designed constructive and improvement procedures for the addressed problem are described.

3.1 Constructive procedure

The most commonly used solution representation for the parallel machine scheduling problem is an array of jobs S_i for each machine i that represents the processing order of the jobs assigned to that machine, that is, $x = \{S_1, S_2, \ldots, S_m\}$, where S_1 represents the processing order of jobs assigned to the machine 1 and S_2 the same for machine 2, and so on.

```
   Data: Instance of the problem.
1  x* ← ∅;
2  while stop criterion is not satisfied do
3  |   x ← Constructive();
4  |   x ← Improvement(x);
5  |   if x is better than x* then
6  |   |   x* ← x;
7  |   end
8  end
   Result: Solution x*.
```

Fig. 3 Pseudo-code for a generic multi-start algorithm

The procedure works as follows: First, jobs are sorted in a non-increasing order according to its average processing time over all machines, that is, $\bar{p_j} = \sum_{i \in M} p_{ij}/m$, then a candidate list is formed with the s first jobs (s is a parameter) from which one is randomly selected. For the selected job j, it is determined the best insertion point in each sequence S_i, that is, the insertion point where the machine span is less increased. Let us denote by Δ_i the increase on the span O_i of machine i for the best insertion point of job j in that machine. Then, the job j will be inserted in machine $i^* = \text{argmin}\{O_i + \Delta_i\}$ in the best insertion point in machine i^*. The candidate list is then updated, and the process is repeated until all jobs have been assigned to machines. A pseudo-code for this procedure is shown in Fig. 4.

With the purpose of illustration, let us consider the following example.

Example Let $n = 8$, $m = 2$ and the following processing times and setup times.

$$p_{ij} = \begin{bmatrix} 10 & 79 & 71 & 95 & 51 & 42 & 78 & 46 \\ 10 & 75 & 28 & 96 & 37 & 61 & 95 & 64 \end{bmatrix}$$

$$S_{1jk} = \begin{bmatrix} 0 & 80 & 54 & 9 & 10 & 65 & 83 & 36 \\ 46 & 0 & 53 & 65 & 73 & 11 & 46 & 34 \\ 35 & 42 & 0 & 24 & 10 & 46 & 46 & 21 \\ 58 & 31 & 24 & 0 & 7 & 50 & 26 & 97 \\ 79 & 35 & 6 & 9 & 0 & 62 & 24 & 33 \\ 47 & 28 & 63 & 88 & 51 & 0 & 23 & 46 \\ 34 & 91 & 51 & 69 & 2 & 24 & 0 & 41 \\ 69 & 46 & 61 & 7 & 66 & 71 & 23 & 0 \end{bmatrix};$$

```
   Data: Instance of the problem.
1  foreach i ∈ M do  S_i ← Empty sequence: O_i ← 0;
2  L ← List of jobs j ∈ N ordered non-increasingly by p̄_j;
3  LC ← Candidate list with the s first jobs;
4  while LC ≠ ∅ do
5  |   j ← Select randomly a job from LC;
6  |   foreach i ∈ M do
7  |   |   Find the best position q_i to insert j in S_i;
8  |   |   Save q_i and Δ_i;
9  |   end
10 |   i* ← arg min_i{O_i + Δ_i}
11 |   Insert j in the sequence S_i* at the position q_i*
12 |   Remove j from L and update LC;
13 end
   Result: Solution x = (S_1, S_2, ..., S_m).
```

Fig. 4 Pseudo-code for procedure Constructive()

$$S_{2jk} = \begin{bmatrix} 0 & 74 & 5 & 25 & 45 & 46 & 81 & 35 \\ 64 & 0 & 1 & 91 & 23 & 19 & 74 & 27 \\ 16 & 99 & 0 & 62 & 25 & 44 & 86 & 32 \\ 35 & 95 & 23 & 0 & 45 & 49 & 70 & 21 \\ 18 & 44 & 50 & 1 & 0 & 84 & 90 & 71 \\ 15 & 31 & 48 & 79 & 95 & 0 & 9 & 61 \\ 93 & 37 & 59 & 96 & 13 & 1 & 0 & 6 \\ 14 & 37 & 94 & 84 & 49 & 98 & 81 & 0 \end{bmatrix};$$

The processing times and setup times associated to the dummy job are considered 0 ($p_{i0} = 0$, $s_{i0k} = 0$ and $s_{ij0} = 0$).

Following the pseudo-code in Fig. 4, a solution $x = \{-; -\}$ is initialized with empty sequences S_1, S_2 and $O = \{0, 0\}$. Then, the average processing time is calculated for each job

$$\bar{p} = [\,10,\ 77,\ 49.5,\ 95.5,\ 44,\ 51.5,\ 86.5,\ 55\,],$$

and the list of jobs $L = [\,4,\ 7,\ 2,\ 8,\ 6,\ 3,\ 5,\ 1\,]$ sorted in non-increasingly order by the $\bar{p}_j$ values is formed (step 2). The candidate list $LC = [\,4,\ 7,\ 2\,]$ is created with size $s = 3$ for this example (step 3) and a job is randomly selected from it. Suppose that job 2 has been selected (step 5). In order to find its best insertion point (steps 6–10), how the O_i of each machine is increased should be evaluated. There is only one position for inserting this job in each machine with the following increments: $\Delta_1 = s_{102} + p_{12} + s_{120} - s_{100} = 0 + 79 + 0 - 0 = 79$ and $\Delta_2 = s_{202} + p_{22} + s_{220} - s_{200} = 0 + 75 + 0 - 0 = 75$ respectively. Since $\min\{O_1 + \Delta_1, O_2 + \Delta_2\} = \min\{0 + 79, 0 + 75\} = 75$ and $i* = \arg\min_i\{O_i + \Delta_i\} = 2$, the best insertion point is in S_2. Inserting the job 2 in S_2, the following partial solution $x = \{-; 2\}$ is obtained with $O = \{0, 95\}$ (step 11). The updated candidate list is $LC = [\,4,\ 7,\ 8\,]$ (step 12) and while $LC \neq \emptyset$, the process is repeated from step 5.

Let us suppose that in the next iterations, jobs 4, 6, 3, 7 and 5 were chosen from the respective candidate lists obtaining the following partial solution $x = \{4, 5, 3; 2, 6, 7\}$ with $O = \{230, 259\}$. At this moment, only two jobs remain non-assigned, so both are in the candidate list, $LC = [\,8,\ 1\,]$. Suppose that job 1 is chosen. There are four possible insertion points in each machine. Calculating $\Delta_1 = \min\{s_{101} + p_{11} + s_{114} - s_{104}, s_{141} + p_{11} + s_{115} - s_{145}, s_{151} + p_{11} + s_{113} - s_{153}, s_{131} + p_{11} + s_{110} - s_{130}\} = \min\{0 + 10 + 9 - 0, 58 + 10 + 10 - 7, 79 + 10 + 54 - 6, 35 + 10 + 0 - 0\} = \min\{19, 71, 137, 45\} = 19$ and $\Delta_2 = \min\{s_{201} + p_{21} + s_{212} - s_{202}, s_{221} + p_{21} + s_{216} - s_{226}, s_{261} + p_{21} + s_{217} - s_{267}, s_{271} + p_{21} + s_{210} - s_{270}\} = \min\{0 + 10 + 74 - 0, 64 + 10 + 46 - 19, 15 + 10 + 81 - 9, 93 + 10 + 0 - 0\} = \min\{84, 101, 97, 103\} = 84$, obtaining $\min\{O_1 + \Delta_1, O_2 + \Delta_2\} = \min\{230 + 19, 259 + 84\} = 249$ and $i* = \arg\min_i\{O_i + \Delta_i\} = 1$.

The above calculations indicate that the best insertion point is at the first position of S_1, obtaining $x = \{1, 4, 5, 3; 2, 6, 7\}$ with $O = \{249, 259\}$. Repeating the process for the last non-assigned job (job 8), $x = \{1, 4, 5, 3, 8; 2, 6, 7\}$ with $O = \{316, 259\}$ is obtained.

3.2 Improvement procedure

The improvement procedure consists of two stages. In both stages, intermachine and intramachine movements are used, but they are applied in a different way in each stage.

In the first stage, local searches based on intermachine movements are firstly applied, and then, intramachine movements based on Or-opt [18] are applied to each machine. In the second stage, the composed movements are defined. Each composed movement uses one intermachine movement and then applies Or-opt to the involved machines in order to improve each sequence.

In both stages, strategies borrowed from variable neighbourhood descent (VND) metaheuristic [9] have been developed which rely on the following facts: (i) a local minimum with respect to one neighbourhood structure is not necessarily so with another and (ii) a global minimum is a local minimum with respect to all possible neighbourhood structures. An initial solution is improved by exploring several neighbourhood structures $N_g(g = 1, \ldots, g_{\max})$ starting from N_1. If an improvement using N_g is made, the algorithm restarts exploration from N_1, otherwise it moves to N_{g+1}. When g exceeds $g_{\max}$, the process is terminated. A generic pseudo-code of VND is shown in Fig. 5.

3.2.1 Improvement procedure: Stage I

In order to explore job assignments to machines, two neighbourhoods have been included, while to improve the job sequence at each machine, one more neighbourhood has been considered. The neighbourhoods defined to deal with the assignment problem are the following:

Data: Initial Solution x.
1 $g \leftarrow 1$;
2 **while** $g \leq g_{\max}$ **do**
3 $\quad x' \leftarrow$ The best solution from $N_g(x)$;
4 $\quad$ **if** x' *is better than* x **then** $x \leftarrow x'$; $g \leftarrow 1$;
5 $\quad$ **else** $g \leftarrow g + 1$;
6 **end**
Result: Solution x.

Fig. 5 Pseudo-code for a generic VND

– The insertion neighbourhood, where each job is extracted from its currently assigned machine and is inserted in all possible positions of all other machines.
– The interchange neighbourhood, where each job of each machine is interchanged with each job assigned to any other machine.

Both insertion neighbourhood and interchange neighbourhood have been divided into smaller sub-neighbourhoods and a VND search has been devoted to each type of movement. This is accomplished because insertion and interchange neighbourhoods have a great number of solutions and because the span of the machines has been considered to guide the search.

Specifically, each neighbourhood has been divided into smaller sub-neighbourhoods, where the first to be explored is the one that involves the most busy machine. That is, let L denote the list of index of the machines sorted in a non-increasing way of their spans and let $[i]$ and $[l]$ represent the machines that occupy position i and position l in L, respectively. Then,

– $N^i_{\text{ins}}(x)$: Insertion sub-neighbourhood, where each job is extracted from machine $[i]$ and inserted in all possible positions of all other machines.
– $N^i_{\text{int}}(x)$: Interchange sub-neighbourhood, where each job of machine $[i]$ is interchanged with each job assigned to any other machine $[l]$, $i < l$.

The VND procedure that uses sub-neighbourhoods $N^i_{\text{ins}}(x)$ will be referred to as *ins*-VND, while the other that uses sub-neighbourhoods $N^i_{\text{int}}(x)$ will be referred to as *int*-VND.

Figure 6 shows a pseudo-code for the Improvement procedure: Stage I.

First, *ins*-VND is applied (step 2). When no more improvements are found using these sub-neighbourhoods, *int*-VND is applied (step 3). When no more improvements are found using these sub-neighbourhoods, a local search procedure based on Or-opt moves is applied to each machine trying to decrease the machine span (step 4). Steps 2 to 4 are repeated until two consecutive procedures fail to improve their input solutions.

Data: Solution $x = (S_1, S_2, \ldots, S_m)$.
1 **repeat**
2 $\quad x \leftarrow$ Find local optimum using *ins*-VND(x);
3 $\quad x \leftarrow$ Find local optimum using *int*-VND(x);
4 $\quad x \leftarrow$ **foreach** $i \in M$ **do** $S_i \leftarrow$Or_opt(S_i);
5 **until** *two consecutive procedures fail to improve the solution*
Result: Solution $x = (S_1, S_2, \ldots, S_m)$.

Fig. 6 Pseudo-code for procedure Improvement: Stage I (x)

Details about implementation of VND will be discussed in Section 3.2.3.

3.2.2 Improvement procedure: Stage II

The general idea here is similar to the one sketched in Fig. 6, but now, composite moves are used to define the neighbourhoods. The composite insertion (interchange) consists of a simple insertion (interchange) movement plus the optimization of the sequence in both involved machines.

In this case, for each VND, the neighbourhood is divided in the following way:

– $N^i_{\text{com\_ins}}(x)$: The composite insertion sub-neighbourhood where each job is extracted from machine $[i]$ and inserted at the end of any other machine $[l]$. Both sequences $S_{[i]}$ and $S_{[l]}$ are optimized using Or-opt.
– $N^i_{\text{com\_int}}(x)$: The composite interchange sub-neighbourhood where each job of machine $[i]$ is interchanged with each job assigned to any other machine $[l]$, $i < l$, and both sequences $S_{[i]}$ and $S_{[l]}$ are optimized using Or-opt.

The VND procedure that uses sub-neighbourhoods $N^i_{\text{com\_ins}}(x)$ will be referred to as *com_ins*-VND, while the other that uses sub-neighbourhoods $N^i_{\text{com\_int}}(x)$ will be referred as *com_int*-VND.

Figure 7 shows a pseudo-code for the Improvement procedure: Stage II.

First, *com_ins*-VND is applied (step 2). When no more improvements are found, *com_int*-VND is applied (step 3). Steps 3 and 4 are repeated until one of these procedures fails to improve its input solution.

3.2.3 Implementation details

A generic pseudo-code for our VND procedures is showed in Fig. 8. In this pseudo-code, the word *type* refers to the neighbourhood's type that is used in each VND procedure.

Throughout the search, movements are accepted regarding the criterion explained next.

Consider a movement involving machines i and l. Let O_i, O_l, O^{mov}_i and O^{mov}_l denote the span of machine i and machine l before and after the movement mov is executed. The movement is accepted if the makespan, restricted to the

Data: Solution $x = (S_1, S_2, \ldots, S_m)$.
1 **repeat**
2 $\quad x \leftarrow$ Find a local optimum using *com_ins*-VND(x);
3 $\quad x \leftarrow$ Find a local optimum using *com_int*-VND(x);
4 **until** *composite interchange fails to improve the solution*
Result: Solution $x = (S_1, S_2, \ldots, S_m)$.

Fig. 7 Pseudo-code for procedure Improvement: Stage II (x)

Data: Solution $x = (S_1, S_2, \ldots, S_m)$.
$Lm \leftarrow$ Machine list non-increasingly ordered by Span;
$i \leftarrow 1$;
while $i \leq m$ **do**
  $[i] \leftarrow$ The i'th machine in Lm
  best_mov $\leftarrow$ Find the best movement from type *type* in sub-neighborhood $N^i_{type}(x)$.
  if *value*(*best_mov*) > 0 **then**
    $x \leftarrow$ Apply *best_mov* to x;
    Update Lm; $i \leftarrow 1$;
  end
  else $i \leftarrow i + 1$;
end
Result: Solution $x = (S_1, S_2, \ldots, S_m)$.

Fig. 8 Pseudo-code for procedure *type*-VND(x)

involved machines, is decreased, that is,

$$\max\{O_i^{\text{mov}}, O_l^{\text{mov}}\} < \max\{O_i, O_l\}. \quad (15)$$

Among several tested criteria, this acceptance criterion showed the best results in preliminary experiments. Note that it implicitly ensures that an acceptable movement does not worsen the makespan. Additionally, it may be easily generalized to movements involving more than two machines.

Let us define the value of a movement mov as

$$\text{value(mov)} = \max\{O_i, O_l\} - \max\{O_i^{\text{mov}}, O_l^{\text{mov}}\},$$

or equivalently as

$$\text{value(mov)} = \min\{\ \max\{O_i, O_l\} - O_i^{\text{mov}},\ \max\{O_i, O_l\} - O_l^{\text{mov}}\}.$$

A movement is considered acceptable if its value is positive. Moreover, mov2 is considered better than mov1 if value(mov2) > value(mov1).

So as not to explore the whole neighbourhood and to reduce the number of calculations, first, the expression value(best_mov) is set to 0.

For evaluating a neighbour, the following inequality is verified

$$a = \max\{O_i, O_l\} - O_i^{\text{mov}} > \text{value(best\_mov)}.$$

In case if it is not fulfilled, the neighbour is discarded without more calculations, but if it is fulfilled, the following second inequality is verified

$$b = \max\{O_i, O_l\} - O_l^{\text{mov}} > \text{value(best\_mov)}.$$

If this second inequality is not satisfied, the neighbouring is discarded. Otherwise, the value of best_mov is updated as value(best_mov) $= \min\{a, b\}$ and a new neighbour is evaluated.

To further reduce the calculations, the computation of the span of machines i and l is performed through the computation of the changes carried out by the movement mov. That is,

$$O_i^{\text{mov}} = O_i + \delta_i$$

and

$$O_l^{\text{mov}} = O_l + \delta_l,$$

where δ_i, δ_l denote how much the span of machines i and l would change if a movement was performed. Recall that movements were defined related to the neighbourhoods established in each stage.

In summary, the proposed multi-start algorithm can be outlined through the pseudo-code presented in Fig. 9.

Example (continued) Following the pseudo-code in Fig. 9, once the solution $x = \{1, 4, 5, 3, 8; 2, 6, 7\}$ with $O = \{316, 259\}$ has been created using Constructive(), the procedure Improvement: Stage I(x) is applied to this solution. Then, according to the pseudo-code of this procedure, the local optimum using *ins*-VND(x) should be found (step 2 in Fig. 6).

To do that, the pseudo-code shown in Fig. 8 is followed. Since $L_m = [\,1\ 2\,]$ (step 1), the insertion of each job of machine 1 (the machine in first position of L_m) in each position of machine 2 is analyzed ($N^1_{\text{ins}}(x)$). Of all these possible movements, the one with the best value consists of the removal of the job 5 from machine 1 and insert it after job 7 in machine 2. For this movement, $\delta_1 = s_{143} - s_{145} - s_{153} - p_{15} = 24 - 7 - 6 - 51 = -40$ and $\delta_2 = s_{275} + s_{250} - s_{270} + p_{25} = 13 + 0 - 0 + 37 = 50$, then value(mov) $= \max\{316, 259\} - \max\{316 - 40, 259 + 50\} = 316 - 309 = 7$, which is positive and greater than the value

Data: Instance of the problem.
$x^* \leftarrow \emptyset$;
while *stop criterion is not satisfied* **do**
  $x \leftarrow$ Constructive();
  $x \leftarrow$ Improvement: Stage I(x);
  $x \leftarrow$ Improvement: Stage II(x);
  if x *is better than* x^* **then**
    $x^* \leftarrow x$;
  end
end
Result: Solution x^*.

Fig. 9 Pseudo-code for the proposed multi-start algorithm

of any other movement. Performing this movement (step 7), the solution $x = \{1, 4, 3, 8; 2, 6, 7, 5\}$ with $O = \{276, 309\}$ is obtained.

Now, $L_m = [\,2\ 1\,]$, $i = 1$ (step 8) and N^1_{ins} consists of exploring the insertion of each job of machine 2 in each position of machine 1, but no movement with positive value was found. Then, $i \leftarrow 2$ (step 10) and N^2_{ins} is explored. Since in this neighbourhood any movement with positive value does not exist, *ins*-VND stops.

Now, according to the pseudo-code of the procedure Improvement: Stage I(x), the local optimum using *int*-VND(x) should be found (step 3 in Fig. 6), starting from solution $x = \{1, 4, 3, 8; 2, 6, 7, 5\}$ with $O = \{276, 309\}$.

Following the pseudo-code shown in Fig. 8, $L_m = [\,2\ 1\,]$. When exploring N^1_{int}, i.e the interchange of each job of machine 2 with each job of machine 1, the movement with the best value is the one that interchanges the job 2 of machine 2 with job 3 of machine 1. For this movement, $\delta_1 = -s_{143} - s_{138} - p_{13} + s_{142} + s_{128} + p_{12} = -24 - 21 - 71 + 31 + 34 + 79 = 28$ and $\delta_2 = -s_{202} - s_{226} - p_{22} + s_{203} + s_{236} + p_{23} = -0 - 19 - 75 + 0 + 44 + 28 = -22$, then value(mov) $= \max\{276, 309\} - \max\{279 + 28, 309 - 22\} = 309 - 307 = 2$, which is positive and greater than the value of any other movement. Performing this movement (step 7), the solution $x = \{1, 4, 2, 8; 3, 6, 7, 5\}$ with $O = \{304, 278\}$ is obtained. The machine list $L_m = [\,1\ 2\,]$ is updated, $i \leftarrow 1$ (step 8) and the new neighbourhood N^1_{int} is explored (step 5). Since in this neighbourhood any movement with positive value does not exist and when dealing with two machines there is just one such neighbourhood, the procedure *int*-VND stops.

Returning to the pseudo-code of the procedure Improvement: Stage I(x), a local search procedure based on Or-opt movements is applied to each machine (step 4 in Fig.6) in order to decrease the span of each machine. The procedures *ins*-VND(x), *int*-VND(x) and Or_opt(S_i) are repeated until two consecutive procedures fail to improve the solution (step 5 in Fig. 6).

Now, following the pseudo-code in Fig. 9, the procedure Improvement: Stage II(x) is applied to the solution $x = \{6, 7, 5, 4; 2, 8, 1, 3\}$ with $O = \{300, 223\}$, which is the solution obtained by Improvement: Stage I(x). Then, according to the pseudo-code of the procedure Improvement: Stage II(x), the local optimum using *com_ins*-VND(x) should be found (step 2 in Fig. 7).

Since $L_m = [\,1\ 2\,]$ (step 1 in Fig. 8), the neighbourhood $N^1_{\text{com\_ins}}$ is explored, that is, the insertion of each job of machine 1 (the machine in first position of L_m) at the end of machine 2 and the optimization of both sequences using Or_opt. The movement with the best value consists of the shifting of job 5 from machine 1 to machine 2. For this movement, $\delta_1 = s_{174} - s_{175} - s_{154} - p_{15} = 69 - 2 - 9 - 51 = 7$ and $\delta_2 = s_{235} + s_{250} - s_{230} + p_{25} = 25 + 0 - 0 + 37 = 62$, then value(mov) $= \max\{300, 223\} - \max\{300 + 7, 223 + 62\} = 300 - 307 = -7$, which is negative, but when the Or_opt is applied to both sequences, value(mov) $= \max\{300, 223\} - \max\{300 + 7 - 42, 223 + 62 + 0\} = 300 - \max\{265, 285\} = 15$, which is positive and greater than the value of any other movement, obtaining the solution $x = \{4, 7, 6; 2, 8, 1, 3, 5\}$ with $O = \{265, 285\}$. The machine list $L_m = [\,2\ 1\,]$ is updated and the procedure *com_ins*-VND(x) continues its execution.

When it is finished, *com_int*-VND(x) is applied (step 3 Fig. 7). The procedures *com_ins*-VND(x) and *com_int*-VND(x) are repeated until one of them fails to improve the solution (step 4 in Fig. 7), finishing the procedure Improvement: Stage II(x).

According to the pseudo-code of the multi-start algorithm, the incumbent is updated (steps 6–8 in Fig. 9) and the algorithm continues until the stop criterion is reached.

4 Computational results

In this section, firstly, the results obtained when comparing the proposed formulations and their variations are shown. Secondly, the experiments carried out to evaluate the performance of the proposed algorithm are described.

The experiments were conducted on a Pentium Dual Core PC with a 2.00 GHz and 3 GB RAM processor under Ubuntu 11.1.

4.1 Instances

Three instance sizes were used: small, medium and large. Small and large instances were taken from [23] and are available at http://soa.iti.es. Additionally, medium-sized instances were generated for conducting the experiments in this research.

For the set of small instances, the following combinations of number of jobs (n) and number of machines (m) were considered: $n \in \{6, 8, 10, 12\}$, $m \in \{2, 3, 4, 5\}$. For the set of large instances, the following values were tested: $n \in \{50, 100, 150, 200, 250\}$, $m \in \{10, 15, 20, 25, 30\}$. Setup times were uniformly distributed in four ranges: 1–9, 1–49, 1–99 and 1–124. Processing times were uniformly distributed between 1 and 99. There are 10 replicates for each possible combination of number of machines, number of jobs and range of setup times, obtaining a total of 640 small instances and 1000 large instances.

The medium-sized instances were generated in a similar way. The following combinations of number of jobs and number of machines were considered: $n \in \{20, 30, 40, 50, 60\}$ and $m \in \{2, 3, 4, 5\}$. The setup times were uniformly distributed in three ranges: 1–49, 1–99 and

1–124. The processing times were uniformly distributed between 1 and 99. For each combination, ten replicates were generated, obtaining a total of 600 medium-sized instances. These instances were generated in order to evaluate the scope of the proposed formulations and to assess the performance of the metaheuristic algorithm in instances of medium size. They are available for interested readers upon request.

4.2 Comparing the formulations

All formulations were implemented using the concert technology of CPLEX 12.4. The solver was allowed to run for 1 h. If the solver was unable to reach the optimal solution within this time, the best integer solution found is reported.

Instances were grouped by number of machines and number of jobs. Therefore, results have been averaged over all instances belonging to each group, that is, 40 for small-sized groups and 30 for medium-sized groups.

Table 1 shows results comparing formulations using different linearizations for the makespan, that is, Model 1, Model 2, Model 1a, Model 2a, Model 1b and Model 2b. Columns 1 and 2 refer to the size of the instances. Entries in columns 3 and 6 (#Uns) exhibit how many instances were unsolved to optimality for each group; entries in columns 4 and 7 display the average percentage gap (%gap). For each instance, this percentage gap has been computed as

$$\%\text{gap} = 100 \times \frac{\text{best\_obj\_int-best\_low\_bound}}{\text{best\_obj\_int}},$$

where best_obj_int is the objective value of the best feasible solution found and the best_low_bound is the best lower bound found. Entries in columns 5 and 8 (time) show the average elapsed CPU time (in seconds), for Model 1 and Model 2 respectively. Since Model 1a, Model 2a, Model 1b and Model 2b solved to optimality all instances, only the average elapsed CPU times (in seconds) to reach optimal solutions are reported for them in columns 9, 10, 11 and 12, respectively.

Note that using the typical linearization for $C_{\max}$, Model 1 and Model 2 were not capable of solving most of the instances with 12 jobs, showing a large CPU time and gap. On the other hand, models that involve the proposed linearization solved to optimality all instances consuming less than 3 s. In particular, the one that exhibits the best performance is Model 2b, which includes assignments variables and uses both linearization for the makespan.

Table 2 shows the performance of Model 2b on medium-sized instances. As before, columns 1 and 2 refer to the size of the instances, entries in column 3 (#Uns) exhibit how

Table 1 Performance of proposed formulations for small instances

Number	m	Model 1			Model 2			Model 1a	Model 2a	Model 1b	Model 2b
		#Uns	%gap	Time	#Uns	%gap	Time	Time	Time	Time	Time
6	2	0	0	0.52	0	0	0.41	0.10	0.07	0.06	*0.05*
	3	0	0	0.28	0	0	0.32	0.21	0.16	0.10	*0.08*
	4	0	0	0.29	0	0	0.30	0.21	0.21	0.11	*0.09*
	5	0	0	0.31	0	0	0.28	0.19	0.23	0.09	*0.08*
8	2	0	0	13.47	0	0	10.45	0.24	0.12	0.16	*0.10*
	3	0	0	5.87	0	0	5.38	0.38	0.26	0.30	*0.17*
	4	0	0	1.71	0	0	2.05	0.51	0.39	0.29	*0.18*
	5	0	0	1.14	0	0	1.23	0.45	0.44	0.18	*0.17*
10	2	7	4.25	1499.81	3	1.36	1124.72	0.46	0.23	0.37	*0.22*
	3	0	0	167.00	0	0	133.34	0.77	0.39	0.48	*0.38*
	4	0	0	26.89	0	0	31.94	1.00	0.62	0.63	*0.44*
	5	0	0	7.8	0	0	11.1	0.87	0.81	0.46	*0.36*
12	2	38	49.45	3549.48	38	47.10	3532.12	0.84	*0.35*	1.09	0.36
	3	29	18.29	3094.16	28	17.87	2990.70	1.37	0.57	1.00	*0.52*
	4	3	1.83	796.61	4	1.12	799.23	2.68	1.01	2.61	*0.87*
	5	0	0	91.45	0	0	112.00	2.18	1.22	1.76	*0.71*

Values in italics indicate the lowest CPU times

Table 2 Performance of Model 2b for medium instances

Number	m	Model 2b		
		#Uns	%gap	Time
20	2	0	0	1.25
	3	0	0	3.25
	4	0	0	10.96
	5	0	0	43.3
30	2	0	0	4.19
	3	0	0	19.07
	4	0	0	165.15
	5	0	0	460.14
40	2	0	0	12.74
	3	0	0	79.4
	4	0	0	589.9
	5	3	0.2	1730.5
50	2	0	0	44.11
	3	0	0	332.87
	4	4	0.17	1925.3
	5	20	2.27	3187.95
60	2	0	0	111.7
	3	1	0.02	1171.5
	4	9	0.49	2765.61
	5	28	3.58	3580.46

many instances were unsolved to optimality for each group, while column 4 (%gap) shows the average percentage gap. Column 5 (time) reports the average elapsed computational time in seconds.

As can be observed, Model 2b is capable of solving to optimality all the 20-job and 30-job instances, 97.5 % of the 40-job instances, 80 % of the 50-job instances and 68.33 % of the 60-job instances.

As expected, the computational time grows as the size of the instance increases. It is worthy to mention that the unsolved instances consume the allowed computational time (3600 s), which increases the average CPU time for the group of instances.

4.3 Evaluating the performance of the solution method

In this subsection, the experiments carried out to evaluate the performance of the proposed metaheuristic algorithm are reported. In order to evaluate the contribution of each stage of the improvement procedure on the quality of obtained solutions, three versions of the algorithm will be tested:

- MAI: only the first stage of the improvement procedure is applied.
- MAII: only the second stage of the improvement procedure is applied.
- MAIII: both stages of the improvement procedure are applied.

For conducting this experiment the best known solutions for medium and large instances will be used: the optimal or the best known solutions for medium-size instances obtained by Model2b and the best known solutions reported in [23] for large instances. Results regarding small instances are not reported because the three tested versions of the algorithm obtained the optimal solutions for all instances, consuming,in average, less than 1 ms per instance.

The algorithm was implemented in C++ programming language. The stopping criterion of the algorithm consists in allowing a maximum CPU time. After preliminary calibration, it was set equal to $200 \times n$ ms. The stopping criterion was determined in a similar way than [23] and the three versions were allowed to run for the same computational time for a fair comparison.

Table 3 shows, for medium-sized instances, the comparison between the three versions of the multi-start algorithm with optimal or best found solutions obtained using Model 2b. As before, columns 1 and 2 refer to the size of instance. Columns 3, 4, 5 and 6 report the average relative deviation for Model 2b, MAI, MAII and MAIII, respectively, from the best known solution. For each instance, the relative deviation (%RD) is computed as $\%\mathrm{RD} = 100 * (C_{\max} - C^*_{\max})/C^*_{\max}$, where $C_{\max}$ is the makespan obtained with the specific tested method and $C^*_{\max}$ is the makespan of the best known solution.

From Table 3, it can be noticed that the two versions that include the stage II (composite movements) obtain better values. This indicates that it is more convenient to include the stage II than to assign all the allowed computational time to stage I. Moreover, as the instance size grows, the version that includes both stages reaches better results. This can be observed more clearly in Fig. 10.

Table 4 shows, for the large instances, the comparison between the three versions of the multi-start algorithm with the best known solutions so far obtained by the genetic algorithm proposed by [23]. In columns 3 to 6, the average relative deviations of GA, MAI, MAII and MAIII are displayed, where relative deviation (%RD) for each instance is calculated as before. However, $C^*_{\max}$ is the makespan of the new best known solution. The time limits for GA are shown in column 7, while time limits for MAI, MAII and MAIII are shown in column 8 (MAs) for each combination of m and n.

It can be observed that MAIII performs better than GA in 83.5 % of the instances and has the same behaviour in 6.2 %; only 10.3 % of the instances obtained worse results.

Table 3 Relative deviation between Model 2b and three versions of multi-star algorithm on medium instances

Number	m	Relative deviation			
		Model 2b	MAI	MAII	MAIII
20	2	0	0	0	0
	3	0	*0*	0.11	0.12
	4	0	0.17	0.04	*0.02*
	5	0	0.11	*0*	*0*
30	2	0	0.56	*0.27*	0.32
	3	0	0.71	0.5	*0.4*
	4	0	1.06	0.89	*0.58*
	5	0	2.19	0.77	*0.39*
40	2	0	1.9	*1.27*	1.31
	3	0	2.58	*1.14*	1.45
	4	0	3.7	*1.62*	1.78
	5	0	4.3	2.5	*1.74*
50	2	0	3.25	1.9	*1.67*
	3	0	4.63	2.61	*2.46*
	4	0	5.51	3.16	*2.61*
	5	0	5.69	2.86	*2.56*
60	2	0	4.49	2.87	*2.65*
	3	0	5.86	2.98	*2.64*
	4	0	6.52	3.38	*3.27*
	5	0.12	6.74	3.05	*2.57*

Values in italics in columns 4, 5 and 6 indicate the lowest average relative deviations

Moreover, with these tested variants of our algorithm, 871 new best solutions were found for the 1000 large instances generated by Vallada and Ruiz [23], which are available for interested readers upon request.

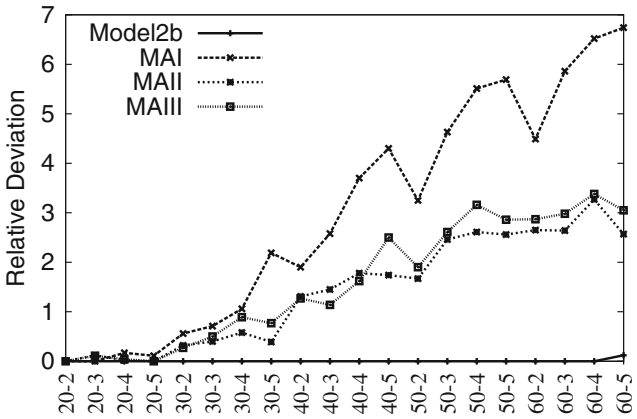

Fig. 10 Relative deviation of Model 2b, MAI, MAII and MAIII versus number of jobs and machines ($n - m$)

Table 4 Comparison between three versions of multi-start algorithm and published results on large instances

Number	m	Relative deviation				Time (s)	
		GA	MAI	MAII	MAIII	GA	MAs
50	10	1.96	3.10	2.62	*1.66*	12.5	10
	15	4.39	1.70	1.41	*0.83*	18.8	10
	20	6.70	2.29	2.65	*1.17*	25.0	10
	25	7.54	2.07	2.29	*1.96*	31.3	10
	30	6.51	2.07	2.76	*1.28*	37.5	10
100	10	2.78	6.86	1.92	*1.37*	25.0	20
	15	4.29	6.03	1.63	*0.98*	37.5	20
	20	5.45	4.65	2.00	*1.04*	50.0	20
	25	8.06	3.79	1.52	*1.16*	62.5	20
	30	8.80	3.23	1.81	*0.94*	75.0	20
150	10	4.20	7.83	1.70	*0.71*	37.5	30
	15	5.72	8.60	1.82	*1.01*	56.3	30
	20	6.96	7.06	1.78	*0.96*	75.0	30
	25	9.45	6.27	1.78	*0.95*	93.8	30
	30	9.51	4.03	1.58	*0.29*	112.5	30
200	10	2.61	5.18	3.97	*0.96*	50.0	40
	15	6.89	8.47	2.95	*0.69*	75.0	40
	20	9.61	8.98	1.77	*0.93*	100.0	40
	25	11.60	8.06	2.13	*1.17*	125.0	40
	30	11.97	7.41	2.42	*0.86*	150.0	40
250	10	*1.16*	3.75	17.54	2.65	62.5	50
	15	5.96	6.85	10.71	*0.74*	93.8	50
	20	9.23	8.79	4.89	*0.50*	125.0	50
	25	13.11	9.71	2.22	*0.86*	156.3	50
	30	13.20	7.97	2.52	*0.90*	187.5	50
All		7.11	5.79	3.22	*1.06*	75.0	30

Values in italics in columns 3 and 6 indicate the lowest average relative deviations

Given that the computer used by Vallada and Ruiz [23] to implement GA and the one used in this experiment have similar features, the average CPU time spent by GA and MA to reach the solutions are shown in Table 4. From this, it is clear that the computational time used by the variants of the multi-start algorithm is always less than the CPU time used by GA. The results obtained in this experiment indicate (as for the medium-sized instances) the convenience of including the stage II instead of assigning all the allowed computational time to stage I.

Results in Table 4 and Fig. 11 show that MAIII (the version including both stages in the improvement procedure) reaches better results for larger instances. It can be observed

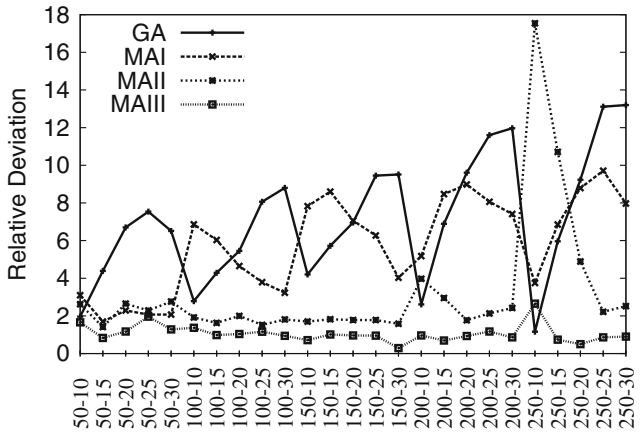

Fig. 11 Relative Deviation of GA, MAI, MAII and MAIII versus number of jobs and machines ($n - m$)

that even though good results were obtained when using only the stage II, they are still improved when this stage starts from a good initial solution provided by the stage I.

To confirm the previous observations, some statistical tests were performed, specifically the Friedman test for paired samples and the Wilcoxon signed rank test.

The Friedman test has been applied to the relative deviations obtained by each method for large instances (columns 3–6 in Table 4), and the resulting p value of 0.000 clearly indicates that there are statistically significant differences among the four methods tested.

The Wilcoxon test was applied to each pair of algorithms with a significance level of $\alpha = 0.05$. For this test, first, the difference scores between the relative deviations of the two selected algorithms are calculated and then hypothesis H0: MD = 0 and H1: MD > 0 are stated, where MD denotes the population median difference. The results, shown in Table 5, confirm that MAIII outperforms GA, MAI and MAII with p values of 0.000, MAII outperforms GA and MAI with p values of 0.001 and MAI outperforms GA with a p value of 0.028.

Table 5 Results of the Wilcoxon signed rank test to each pair of algorithms

Tested algorithms	N for Test	Wilcoxon Statistic	p values	Estimated media
GA-MAIII	25	322	0	6.04
MAI-MAIII	25	325	0	4.695
MAII-MAIII	25	325	0	1.175
GA-MAII	25	283	0.001	4.52
MAI-MAII	25	280	0.001	3.18
GA-MAI	25	234	0.028	1.335

In summary, it can be concluded that there is statistical evidence to affirm that MAIII is significantly better than the other algorithms and the three versions of the proposed algorithm outperform the best results from the literature.

5 Concluding remarks

Several formulations have been proposed in this paper for a scheduling problem with sequence and machine–dependent setup times. These formulations involve linearizing the makespan as the maximum of the completion times of the machines, which provides improved dual bounds and speeds up the solution process when using a branch-and-bound commercial solver. Particularly, the formulation that includes arc variables and assignment variables as well as the typical and the new proposed linearization of the makespan reached the best results.

In order to solve larger instances, an efficient multistart algorithm was designed and implemented, whose main feature lies in the improvement phase. Two stages were defined for this procedure where the inter- and intramachine movements, defined to explore the neighbourhood, were used in a different way in each stage. The algorithm takes advantage of the speed of the search in the first stage and uses composite movements in the second stage to intensify the search. On the other hand, the proposed acceptance criterion allows to define the value of a movement, guides the search in an efficient way and reduces the CPU time required to explore a neighbourhood. Computational results show the superiority of the proposed algorithm over the best known results from literature. A future interesting work may include to investigate the use of both linearization of the makespan in formulations that consider other performance measures like the total completion time or the total tardiness. From a methodological point of view, it is worthy to explore the combination of the simple and composite movements in other scheduling problems. The generalization of this study in environments where the machines present unavailability periods is also an interesting research avenue.

Acknowledgments This work has been partly supported by the Research Chair in Industrial Engineering of Tecnológico de Monterrey (ITESM Research Fund CAT128) and the Mexican National Council of Science and Technology (grant CB 167019).

References

1. Allahverdi A (2000) Minimizing mean flowtime in a two-machine flowshop with sequence-independent setup times. Comput Oper Res 27(2):111–127

2. Arnaout J, Rabadi G, Musa R (2010) A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times. J Intell Manuf 21(6):693–701
3. Balin S (2012) Non-identical parallel machine scheduling with fuzzy processing times using robust genetic algorithm and simulation. Int J Innov Comput Inf Control 8(1 B):727–745
4. Chen JF (2009) Scheduling on unrelated parallel machines with sequence-and machine-dependent setup times and due-date constraints. Int J Adv Manuf Technol 44(11–12):1204–1212
5. Cheng T, Ding Q, Lin B (2004) A concise survey of scheduling with time-dependent processing times. Eur J Oper Res 152(1):1–13
6. De P, Morton T (1980) Scheduling to minimize makespan on unequal parallel processors. Decis Sci 11(4):586–602
7. Fleszar K, Charalambous C, Hindi K (2012) A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times. J Intell Manuf 23(5):1949–1958
8. Garey R, Johnson D (1979) Computers and intractability: a guide to the theory of NP-completeness. Series of Books in the Mathematical Sciences. W. H. Freeman
9. Hansen P, Mladenović N, Pérez JAM (2010) Variable neighbourhood search: methods and applications. Ann Oper Res 175(1):367–407
10. Helal M, Rabadi G, Al-Salem A (2006) A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times. Int J Oper Res 3(3):182–192
11. Lee JH, Yu JM, Lee DH (2013) A tabu search algorithm for unrelated parallel machine scheduling with sequence-and machine-dependent setups: minimizing total tardiness. Int J Adv Manuf Technol:1–9
12. Lee Y, Pinedo M (1997) Scheduling jobs on parallel machines with sequence-dependent setup times. Eur J Oper Res 100(3):464–474
13. Lin SW, Lu CC, Ying KC (2011) Minimization of total tardiness on unrelated parallel machines with sequence-and machine-dependent setup times under due date constraints. Int J Adv Manuf Technol 53(1–4):353–361
14. Lin YK (2013) Particle swarm optimization algorithm for unrelated parallel machine scheduling with release dates. Math Probl Eng. doi:10.1155/2013/409486
15. Lin YK, Hsieh FY (2014) Unrelated parallel machine scheduling with setup times and ready times. Int J Prod Res 52(4):1200–1214
16. Martí R, Moreno-Vega JM, Duarte A (2010) Advanced multi-start methods. In: Gendreau M, Potvin JY (eds) Handbook of Metaheuristics, vol 146. Springer, pp 265–281
17. Mokotoff E (2001) Parallel machine scheduling problems: a survey. Asia Pac J Oper Res 18(2):193–242
18. Or I (1976) Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. PhD thesis. Northwestern University, Evanston
19. Rabadi G, Moraga R, Al-Salem A (2006) Heuristics for the unrelated parallel machine scheduling problem with setup times. J Intell Manuf 17(1):85–97
20. Rocha P, Ravetti M, Mateus G, Pardalos P (2008) Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. Comput Oper Res 35(4):1250–1264
21. Tian K, Jiang Y, Shen X, Mao W (2010) Makespan minimization for job co-scheduling on chip multiprocessors. Tech. Rep. WM-CS-2010-08, Department of Computer Science, College of William & Mary
22. Tran T, Beck J (2012) Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In: 20th European Conference on Artificial Intelligence, pp 774–780
23. Vallada E, Ruiz R (2011) A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur J Oper Res 211:612–622
24. Yang-Kuei L, Chi-Wei L (2013) Dispatching rules for unrelated parallel machine scheduling with release dates. Int J Adv Manuf Technol 67(1–4):269–279
25. Ying K, Lee Z, Lin S (2012) Makespan minimization for scheduling unrelated parallel machines with setup times. J Intell Manuf 23(5):1795–1803