A parallel machine scheduling problem with two-agent and tool change activities: an efficient hybrid metaheuristic...
A parallel machine scheduling problem with two-agent and tool change activities: an efficient hybrid metaheuristic algorithm

Maziar Yazdani, Seyed Mohammad Khalili & Fariborz Jolai


To link to this article: http://dx.doi.org/10.1080/0951192X.2015.1130261

Published online: 11 Jan 2016.
A parallel machine scheduling problem with two-agent and tool change activities: an efficient hybrid metaheuristic algorithm

Maziar Yazdani*, Seyed Mohammad Khalili and Fariborz Jolai

School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran
(Received 2 December 2014; accepted 11 July 2015)

Scheduling with multiple agents has been widely studied. However, a little work has been done on multi-agent scheduling with availability constraints. This paper addresses a two-agent parallel machine scheduling problem with tool change activities. The aim is to minimise the total completion time of all jobs while keeping the maximum makespan of agent two's jobs below a fixed level. A mathematical model is presented to solve the problem optimally in small-sized instances. Then, an imperialist competitive algorithm (ICA) is developed to solve large-sized instances of the problem. For further enhancement, the proposed ICA is hybridised with a simple but efficient local search algorithm. A set of experimental instances are carried out to evaluate the algorithm. The proposed algorithm is carefully evaluated for its performance against an available algorithm. The results of computational experiments show the desirable performance of the proposed algorithm.

Keywords: two-agent; parallel machines; tool change activity; imperialist competitive algorithm; local search

1. Introduction

In traditional scheduling problems, researchers assumed that the machines are continuously available over the planning horizon. However, this assumption may not be true in many practical situations. For instance, a machine may not be available during the planning horizon due to tool changes, periodic services or maintenance activities (Azadeh et al. 2013, 2014). It is clear that the maintenance and service activities are important to improve the quality of the products or the production efficiency of the machines, thus many researchers have considered this assumption in their studies (Ghodratnama et al. 2010; Chen 2013; Ramezanian, Saida-Mehrabad, and Fattahi 2013).

On the other hand, to the best of our knowledge, most of the papers in scheduling problems with maintenance activities considered that all jobs belong to a single agent. But many practical situations revealed this assumption is not applicable in many real-life conditions and jobs might come from different agents (Lee and Wang 2014; Choi and Chung 2014; Cheng 2014). There might be two or several agents who compete on the same resources, and each agent has its own goal. Basically, products of the each agent have their own quality level in real production systems, such as high quality or regular quality. Generally, high-quality products need strict manufacturing setting while regular-quality products only need normal (standard) manufacturing setting. In this paper, job quality is considered in a two-agent scheduling problem where the jobs belong to two agents. The jobs of the first agent (AG1) need to be processed in strict manufacturing setting due to their high-quality requirements while the jobs of the second agent (AG2) have regular quality. This paper presents a variation of the two-agent scheduling problem where maintenance activities and job quality are considered simultaneously.

Metaheuristics are efficient algorithm concepts to generate approximate solutions to non-deterministic polynomial-time hard (NP-hard) optimisation problems. Over the past years, many researchers have successfully tried to apply metaheuristics to scheduling problems (Tayebi Araghi, Jolai, and Rabiee 2014). Due to complexity of the considered problem, in this study, an efficient hybrid metaheuristic algorithm is proposed to tackle the problem.

Contributions of the present work are summarised as follows:

(1) To the best of our knowledge, no researchers have considered scheduling problems with multiple agents and tool change activities simultaneously.
(2) However, few researchers in scheduling problems have considered scheduling problems with job quality consideration.
(3) To solve the proposed problem optimally, a new mathematical model is presented.
(4) Since the problem is strongly NP-hard, to finding the near-optimal solution for large-sized problems...
within reasonable times, a hybrid metaheuristic algorithm is developed. This algorithm is hybridised with three simple but efficient local search where one of these local search is developed based on some proposed dominance properties.

The rest of this paper is organised as follows. In Section 2, the literature review is provided. The problem formulation is described in Section 3. The proposed metaheuristics that are used to solve the problem are presented in Section 4. The analysis of computational experiments is provided in Section 5. Finally, conclusions are presented in Section 6.

2. Literature review

In this section, relevant literature is reviewed in two separate but complementary categories: scheduling problems with multiple agents and scheduling problems with tool change activity, respectively.

Two-agent scheduling problems have been investigated comprehensively in the past decades. Agnetis et al. (2004) and Baker and Smith (2003) studied single-machine scheduling problems with multiple agents in which their objective functions include the total weighted completion time, the number of tardy jobs and the maximum of regular non-decreasing functions of the job completion times. Yuan, Shang and Feng (2005) revised the dynamic programming recursion formulae in Baker and Smith (2003) and proposed a polynomial-time algorithm for the same problem. Ng, Cheng and Yuan (2006) considered a single-machine two-agent problem in which one agent minimises the total completion time of its jobs with a bound on the number of the other agent’s tardy jobs. Cheng, Ng, and Yuan (2006) studied the problem with the objective of minimising the number of tardy jobs for each agent. They proved that the problem is strongly NP-hard. Agnetis, Pacciarelli and Paciﬁci (2007) studied the same scheduling problems in which several agents, each owning a set of non-preemptive jobs, compete to perform their respective jobs on one shared processing resource and the cost function depends on the job completion times only. Cheng, Ng, and Yuan (2008) focused on multi-agent scheduling on a single machine in which the agents’ objective functions are of the max-form. Liu and Tang (2008) studied the two-agent scheduling problems with the consideration of a simple linear deterioration effect on a single machine. Lee et al. (2009) provided fully polynomial approximation schemes and an efficient approximation algorithm with a reasonable worst-case bound for multi-agent scheduling problems where the objective function is total weighted completion time. Agnetis, de Pascale and Pacciarelli (2009) developed branch-and-bound algorithms for several NP-hard single-machine scheduling problems with two competing agents. Lee, Chen and Wu (2010) considered a two-agent scheduling problem on a two-machine permutation flowshop problem where the objective function is to minimise the total tardiness of the first agent’s jobs while no tardy job is allowed for the second agent. Recently, Leung, Pinedo and Wan (2010) generalised the single-machine scheduling problems with multiple agents based on problems proposed by Agnetis et al. (2004) to the case of multiple identical parallel machines. Liu, Zhou and Tang (2010) studied a two-agent single-machine problem with linear ageing or learning effects. Their objective was minimising the total completion time jobs of first agent, while the maximum cost of the second agent cannot exceed an upper bound (UB). They proposed a branch-and-bound and a simulated annealing algorithm to derive the optimal and near-optimal solutions for the problem, respectively. Oron, Shabtay and Steiner (2015) studied two-agent scheduling problems on a single machine with equal job processing times, and proposed pseudo-polynomial time algorithms for those problems. Reisi-Nafchi and Moslehi (2015) addressed order acceptance and two-agent scheduling problem simultaneously. They proposed hybrid linear programming and genetic algorithm (GA) method for considered problem. Choi (2015) considered a two-agent single-machine scheduling problem with linear position-based ageing effects and job-dependent ageing ratios where the objective was minimising the total weighted completion time of all jobs for two agents, where the makespan for one agent was constrained under an UB. Yin, Cheng, Wan, et al. (2015) addressed several two-agent single-machine scheduling problems with deteriorating jobs where the actual processing time of any job of the two agents was an increasing linear function of its starting time. Yin, Cheng, Yang, et al. (2015) addressed two scheduling problems with two agents (agents A and B), where the due dates of agent A’s jobs were decision variables. The objective was to determine the optimal due dates for agent A’s jobs and the job sequence for both agents’ jobs simultaneously to minimise the total cost associated with the due date assignment and weighted number of tardy jobs of agent A, while keeping the maximum of regular functions (associated with each B-job) or the number of tardy jobs of agent B below or at a fixed threshold.


Tool change activities can be considered as flexible maintenance activities where the tools may be changed at any time within their life times and the machine cannot process or set up any job during tool change activities. Akturk, Ghosh and Gunes (2003, 2004) studied single-machine scheduling problems where the maintenance activities need to be scheduled jointly with jobs. Akturk, Ghosh and Kayan (2007) addressed the problem of scheduling a computer numerically controlled machine subject to tool changes and controllable machining conditions. Their objective was minimising the total completion time. Chen (2008) considered a single-machine scheduling problem subject to tool wear, given the allowed maximum continuous working time of the machine. In his research, job processing and tool changes are scheduled simultaneously. He developed two models to optimally solve this problem. Qi (2007) studied two scheduling problems with maintenance activity and analysed the worst-case performance of the classical shortest process time (SPT) and earliest due date algorithms, respectively. Sbihi and Varnier (2008) considered a single-machine scheduling problem with several maintenance periods under two different scenarios and proposed some dominance properties and an efficient heuristic. Sun and Li (2010) addressed a scheduling problem in which the machines need to be maintained regularly and the largest consecutive working time for each machine cannot exceed an upper limit. Recently, Xu et al. (2013) considered a single-machine scheduling problem with tool changes. To solve the scheduling problem with small and medium size, they proposed two models. To solve the scheduling problem with large size, they proposed three sets of algorithms and focused on the performance of six algorithms based on the studies of a new bin packing problem.

3. Problem description and mathematical modelling

The considered problem in this paper schedules $n$ non-preemptive jobs which are processed on parallel machines and the tools of the machines have to be maintained or changed during the production process. The machines are available at the beginning of the scheduling. Jobs are independent and there are no precedence relations between them. Hence, all jobs are available at the beginning of the scheduling. In the considered problem, jobs come from two agents. Based on these two agents, jobs are divided into two groups, special jobs ($J^1_{AG1}$) and ordinary jobs ($J^1_{AG2}$) which come from the first and second agents, respectively. It is assumed that the latest time at which the machine must start its tool change activity is $v$ which is measured in relation to the completion of the latest previous tool change activity. Special jobs must be processed only at certain time intervals, where the start time of each of these intervals coincides with the completion of tool change activity and the length of each of them is at most $u$, where $0 \leq u \leq v$. Also, $w$ is the time of performing each tool change activity (M). The objective is to determine the start time of each tool change activity and schedule all jobs while minimising the total completion time of all jobs as well as keeping the maximum makespan of other agent’s jobs below a fixed level. Figure 1 illustrates the representation of the problem on two parallel machines.

3.1. Assumptions

Since different assumptions and constraints can result in different scheduling problems, the following assumptions are considered in this study:

1. Pre-emption is not allowed.
2. There is no set-up time before a job is processed.
3. All jobs are available at the beginning of planning horizon.
4. The processing time of jobs which belong to AG1 are $p^1_{AG1} < u$.

![Figure 1](image-url)  
**Figure 1.** Illustration of the considered problem.
(5) The processing time of jobs which belong to AG2 are \( p_{i}^{ AG2 } < v \).

(6) A dummy tool change activity has just been finished at the beginning of planning horizon.

Indices

\( i \) Index for jobs, \( i = 1, \ldots, N_{1} + N_{2} \), where \( N_{1} \) and \( N_{2} \) are number of jobs from AG1 and AG2 requiring processing at time zero, respectively.

\( k \) Index for positions, \( b = 1, \ldots, k + 1 \).

\( m \) Index for machines, \( m = 1, \ldots, M \).

Parameters

\( p_{im} \) Processing time of job \( i \) on machine \( m \).

\( UB \) Maximum makespan of other AG2’s jobs.

\( w \) The time needed to perform each tool replacement.

\( v \) Certain time intervals, where the start time of each of these intervals coincides with the completion of a tool replacement.

\( L \) The latest time at which the machine starts its tool change activity.

\( M \) A very large number.

Decision variables

\( c_{k} \) Completion time of job scheduled in position \( k \).

\( x_{ikm} \) A binary variable that is equal to 1 if job \( i \) is scheduled in position \( k \) on machine \( m \), otherwise 0.

\( r_{km} \) A binary variable that is equal to 1 if tool replacement is scheduled after position \( k \) on machine \( m \) and 0 otherwise 0.

\( e_{km} \) Elapsed time between the completion time of the preceding tool replacement and the completion time of the job in position \( k \) on machine \( m \) and otherwise 0.

\[ \begin{align*}
\text{(5)} & \quad \sum_{i=1}^{N_{1} + N_{2}} x_{ikm} \leq \sum_{j=1}^{N_{1} + N_{2}} x_{ijk-1,m}; \quad k = 2, \ldots, N_{1} + N_{2} \\
& \quad m = 1, 2, \ldots, M \\
\text{(6)} & \quad c_{j} + L(2 - x_{ijkm} - x_{ijk-1,m}) \geq c_{i} + p_{im} + r_{k-1,m}, \quad w; \\
& \quad k = 2, \ldots, N_{1} + N_{2} \quad m = 1, 2, \ldots, M \\
& \quad i, j = 1, 2, \ldots, N_{1} + N_{2} \quad i \neq j
\end{align*} \]

\[ \begin{align*}
\text{(7)} & \quad c_{i} \geq \sum_{k=1}^{N_{1} + N_{2}} p_{im} x_{ikm}; \quad i = 1, 2, \ldots, N_{1} + N_{2} \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(8)} & \quad e_{km} \leq \sum_{i=1}^{N_{1} + N_{2}} p_{i} x_{ikm}; \quad k = 1, 2, \ldots, N_{1} + N_{2}, \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(9)} & \quad e_{km} \leq v; \quad k = 1, 2, \ldots, N_{1} + N_{2}, \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(10)} & \quad e_{k-1,m} + \sum_{i=1}^{N_{1} + N_{2}} p_{i} x_{ikm} \leq e_{km} + (r_{k-1,m} \times v) \\
& \quad + L \times (1 - \sum_{i=1}^{N_{1} + N_{2}} x_{ikm}); \quad i = 2, \ldots, N_{1} + N_{2}, \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(11)} & \quad e_{k,m} \leq L \times \sum_{i=1}^{N_{1} + N_{2}} x_{ikm}; \quad k = 1, \ldots, N_{1} + N_{2}, \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(12)} & \quad e_{k,m} \leq (v \times \sum_{i=N_{1} + 1}^{N_{2}} x_{ikm}) + w; \\
& \quad k = 1, 2, \ldots, N_{1} + N_{2}, \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(13)} & \quad r_{k-1,m} \leq \sum_{i=1}^{N_{1} + N_{2}} x_{ikm}; \quad k = 2, \ldots, N_{1} + N_{2}, \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(14)} & \quad x_{ikm} \in \{0, 1\}; \quad i = 1, 2, \ldots, N_{1} + N_{2}, \\
& \quad k = 2, \ldots, N_{1} + N_{2}, \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(15)} & \quad k_{km} \in \{0, 1\}; \quad k = 1, \ldots, N_{1} + N_{2}, \\
& \quad m = 1, 2, \ldots, M
\end{align*} \]

\[ \begin{align*}
\text{(16)} & \quad c_{i} \geq 0; \quad i = N_{1} + 1, \ldots, N_{2}
\end{align*} \]
The objective function is to minimise the total completion time of all jobs that is presented in Equation (1). Constraint (2) specifies that the makespan of agent AG2’s jobs does not exceed a given UB. Constraint (3) ensures that each job is assigned to one of the existing positions on the machines. Constraint (4) guarantees that on each existing positions, at most one job can be assigned. Constraint (5) ensures that until one position on a machine is empty, jobs are not assigned to the subsequent positions. Constraint (6) ensures that the completion time of each job in a sequence on a machine is at least equal to the sum of the completion time of the preceding job, tool change activity (if any) and the processing time of the present job. Constraint (7) measures completion time for each work on each machine. Constraints (8)–(11) meet the requirement that the values of \( e_{km} \) are set properly. Constraint (12) ensures that each job which belongs to the AG2 is processed according to the requirements. Constraint (13) ensures that a tool change is done only if after that tool change a job has to be processed. Constraints (14)–(17) set the ranges of the variables.

4. Proposed algorithm

For the problem in hand, due to its complexity, finding optimum solution is challenging. Metaheuristics can often find good solutions with less computational effort (Yazdani and Jolai 2015). With this motivation, in this section, a hybrid imperialist competitive algorithm (HICA) is proposed to solve the considered problem. Imperialist competitive algorithm (ICA) is a metaheuristic optimisation algorithm introduced by Atashpaz-Gargari and Lucas (2007) and has shown good capability for problems with continuous search spaces (Azadeh et al. 2015). The general form of ICA and the steps of the proposed solution method are described in Sections 4.1 and 4.2.

4.1. Original imperialist competitive algorithm

Unlike the most of metaheuristic algorithms, such as GA and simulated annealing (SA), which are inspired by modelling the natural processes, ICA uses sociopolitical evolution of human as a source of inspiration (Atashpaz-Gargari and Lucas 2007). Atashpaz-Gargari and Lucas (2007) explained the ICA as ‘like other evolutionary ones, this algorithm starts with an initial population. In this algorithm, any individual of the population is called a country. Some of the best countries (in optimisation terminology, countries with the least cost) are selected to be the imperialist states and the rest form the colonies of these imperialist states. All the remaining colonies of initial countries are divided among the mentioned imperialists based on their power. The power of each country is calculated from the objective function of the proposed model.

After dividing all colonies among imperialists and creating the initial empires, these colonies start moving towards their relevant imperialist country. This movement is a simple model of assimilation policy which was pursued by some of the imperialist states. The total power of an empire depends on both the power of the imperialist country and the power of its colonies. For this fact, we consider the total power of an empire as the sum of power of imperialist country and a percentage of mean power of its colonies. Then the imperialistic competition begins among all the empires. Any empire that is not able to succeed in this competition and increase its power (or at least prevent decreasing its power) will be eliminated from the competition. The imperialistic competition will gradually result in an increase in the power of powerful empires and a decrease in the power of weaker ones. Weak empires will lose their power and ultimately they will collapse. The movement of colonies towards their relevant imperialists along with competition among empires and also the collapse mechanism will hopefully cause all the countries to converge to a state in which there exists just one empire in the world and all the other countries are colonies of that empire. In this ideal new world, colonies have the same position and power as the imperialist’.

4.2. Proposed imperialist competitive algorithm

Since original ICA is designed for problems with continuous space, this paper adopts this solution method for the considered discrete problem. Also, the original ICA is improved by proposing a new local search. Next, the steps of the proposed ICA are presented.

Figure 2. Sample of the two-part sequence encoding for a problem with nine jobs and three machines.
4.2.1. Representation of the solution and initial countries

Solution representation should be easy to decode to reduce the cost of the algorithm. Thus, a two-part sequence solution representation is used which is shown in Figure 2. The length of the vector is equal to the number of job (n) plus the number of machines (m). The two-part sequence technique, as the name implies, divides the individuals into two parts. The first part with length of n represents a permutation of n jobs and the second part with length of m gives the number of jobs assigned to each machine. Therefore, the total length of a solution is n + m in this representation. The sum of values in the second part must be equal to the number of jobs to represent a valid solution.

The first step in the proposed ICA, like other population-based metaheuristic, is forming initial population. Each individual of the population is called a country which corresponds to chromosome in GA. In the proposed algorithm, initial solutions are generated according to the presented pseudo-code in Figure 3.

4.2.2. Evaluation function

The evaluation function consists of calculating the value of the objective function for the solution represented by each individual.

4.2.3. Imperialism

For producing the initial imperialist, some of the countries with minimum cost are selected as emperors. So, the remaining countries (N_col) are considered as colonies which are partitioned among the mentioned imperialists based on their power. Power of an imperialist is obtained by computing the normalised cost for each emperor as follows:

$$C_{on} = c_{on} - \max\{c_{o}\}$$  \hspace{1cm} (18)

where \(c_{on}\) is the cost of nth emperor and \(C_{on}\) is the normalised cost for nth emperor. Now a proportional power for each emperor is computed as follows:

$$P_{n} = \frac{C_{on}}{\sum_{j=1}^{N_{col}} C_{oj}}$$  \hspace{1cm} (19)

Finally, the number of colonies that are dedicated to nth emperor is calculated as follows:

$$N \cdot C_{on} = \text{round}\{P_{n} \cdot (N_{col})\}$$  \hspace{1cm} (20)

After allocating the colonies to emperors, the imperialism competition begins and each emperor tries to seize other imperialists’ countries.

4.2.4. Assimilation methods

After forming initial empires, each of their colonies begins progresses towards their related imperialist country. This movement in this algorithm is called assimilation policy. A new crossover operator called two-part chromosome crossover (Yuan et al. 2013) is used as assimilation operator in this paper. In the rest of this section, an example is presented below to illustrate the process of using this assimilation to move a country towards its relevant imperialist country.

Figure 3. Generating initial solutions.

Figure 4. Step 1.

Figure 5. Step 2.

Figure 6. Step 3.

Figure 7. Step 4.

Figure 8. Step 5.
Step 1: Consider two sequences; imperialist sequence and colony sequence (see Figure 4).
Step 2: Randomly select blocks (see Figure 5).
Step 3: Shuffle block positions according to the first part of colony’s sequence (see Figure 6).
Step 4: Add blocks for each machine (see Figure 7).
Step 5: Construct the new colony’s two-part sequence (see Figure 8).

As shown in the example, five basic steps are implemented. Firstly, a pair of sequences (imperialist sequence and colony sequence) are considered. Secondly, some blocks are selected randomly from the first part of imperialist’s sequence. In this case, the selected jobs are 6 and 9 for machine 1, 2 and 3 for machine 2, and 5 for machine 3. Hence, the number of randomly selected blocks are (2, 2 and 1) as shown in Step 2. Next, the order of the remaining blocks is sorted according to the positions in the first part of colony’s sequence. In this example, the remaining jobs in the first part of imperialist’s sequence are 1, 7, 8, 4 and the order of these blocks is shuffled to 4, 1, 8, 7 according to the first part of colony’s sequence. Next, based on the number of unselected blocks, a uniform random number, between 1 and the current value of unselected blocks, is computed to determine how many new blocks will be added for each machine in the new sequence. Here, the unselected blocks are 4, 1, 8, 7 and if, for example, the integer sequence (2, 1, 1) of uniform random numbers between 1 and the current number of unselected blocks is generated iteratively, then in the new sequence of colony, machine 1 gets jobs 4 and 1, machine 2 gets 8 and machine 3 gets 7. Therefore, for the first part of the new colony’s sequence, machine 1 would have 6, 9, 4, 1, machine 2 would have 3, 2, 8 and machine 3 would have 5, 7. Lastly, the two-part sequence for the country is constructed by updating the information in the second part of its sequence. In this example, 4, 3, 2 is generated by summation of (2, 2, 1) + (2, 1, 1) = (4, 3, 2).

4.2.5. Revolution policy
Revolution policy is like as the mutation operator of the GA. For this purpose, two jobs are randomly selected and swapped (see Figure 9).

4.2.6. Cost of an imperialism
The total cost of the nth imperialism is obtained as follows:

\[ T \cdot C_n = \text{Cost( emperor }_n) \]
\[ \quad + \gamma \cdot \text{mean}\{\text{cost of colonies of emperor }_n\} \]  \hspace{1cm} (21)

where \( \gamma \) is a positive ratio between 0 and 1.

4.2.7. Imperialist competition
As all empires try to take the possession of the colonies of the other empires and seizing them, the imperialistic competition gradually brings about a decrease in the power of weaker empires and an increase in the power of more powerful ones (Atashpaz-Gargari and Lucas 2007). The imperialistic competition is modelled by just picking one (some) of the weakest colonies of the weakest empires and making a competition among all empires to possess this (these) colony (ies). Within this competition each emperor who cannot develop their colonies will be omitted and will be seized with stronger imperialism. This process will be continued until stop condition is satisfied. The best objective function value (OFV) of each iteration is recorded and compared to that obtained so far.

4.2.8. The hybridisation of the proposed algorithm
Many researchers concluded that hybrid algorithms for scheduling problems could end up with high-quality results (Naderi, Khalili, and Tavakkoli-Moghaddam 2009). The purpose of the hybridization in this research is to overcome shortcomings of ICA and avoiding trapping in local optimum. Thus, a local search is employed for the hybridisation. This local search algorithm employs three advanced local search structures. Our strategy to hybridise ICA with the proposed local search is as follows. In each iteration, the proposed local search is applied to the emperors. In the following subsection, our proposed local search is explained briefly.

(I) NS-I: A job without repetition is removed randomly from the current sequence, and then

\begin{verbatim}
 Procedure: LS - 1
 while \( j = \sqrt{\text{sqrt}(N)} \) or \( i = < N \) (\( N \) = total number of the jobs) remove job \( k \) at random and without repetition from the current sequence \( \pi \) \( \pi = \text{remove job } k \) into a new random position in current sequence \( \pi \) if objective function (\( \pi \)) < objective function (\( \epsilon \)) \( \pi = \epsilon \) else \( j = j + 1 \) endif \( i = i + 1 \) endfor
\end{verbatim}

Figure 9. An example of revolution policy.

Figure 10. Procedure of local search type 1.
relocated into another position randomly. All jobs are selected one after another without repetition in a random order if no improvement is obtained through inserting jobs into a new randomly selected position for \( \sqrt{NJ} \) times, where \( NJ \) is equal to number of jobs and \( \sqrt{\cdot} \) returns the square root of element. The procedure of this local search is shown in Figure 10.

(II) NS-II: Three dominance properties of the considered problem are stated, and next, using these properties, second local search is developed. To prove these properties, a definition is presented.

**Definition 1.** Let \( S' = S(i \leftrightarrow j) \) be the sequence \( S' \) obtained from sequence \( S \) by a pairwise interchange of jobs \( i \) and \( j \) in sequence \( S \).

**Property 1.** Let \( i \) and \( j \) be adjacent jobs from the same agent in \( S \) which are scheduled on same machine without any tool change between them. If \( p_j < p_i \), then \( S \) will be dominated by \( S' \).

**Proof.** The result follows immediately from the SPT rule.

**Property 2.** Let \( i \) and \( j \) be adjacent jobs from different agents in \( S \) which are scheduled on the same machine without any tool change between them. If \( j \) belongs to agent one and if \( p_j < p_i \) then \( S \) will be dominated by \( S' \).

**Proof.** The result follows immediately from the SPT rule.

**Property 3.** Let \( i \) and \( j \) be adjacent jobs from different agents in \( S \) which are scheduled on same machine without any tool change between them. If \( j \) belongs to agent two and if \( p_j < p_i \), then \( S \) can be dominated by \( S' \).

**Proof.** In some conditions, \( S \) is definitely dominated by \( S' \), for example: After pairwise interchange of jobs \( i \) and \( j \) if there is no need for tool change activity on machine, then \( S \) will be dominated by \( S' \).

III. NS-III: In the third local search, three jobs are removed randomly relocated into three other positions which these positions are selected randomly. Generating a new sequence by this local search is done \( \lambda \) times, where \( \lambda \) is square root of number of jobs. The general outline of this local search is shown in Figure 12.

Concluding, a new local search is proposed with three types of local search. The general outline of the proposed local search is shown in Figure 13.
The main steps of the proposed HICA are summarised in the pseudo-code shown in Figure 14.

5. Computational results and discussion

The literature review specifies that there is no study on scheduling problems on two-agent scheduling and maintenance activities simultaneously. Therefore, the present paper seems to be the first study in this field. So the proposed algorithm is compared with the ICA proposed by Behnamian and Zandieh (2011) for solving scheduling problems. Furthermore, for the validation of the proposed model and to find the optimal solutions for small-sized instances, the mathematical model is solved by GAMS. Metaheuristic algorithms are coded in MATLAB language. Solutions are provided on a personal computer with 2.2-GHz Intel Core 2 Duo CPU and 2-GB RAM memory under a Microsoft Windows 7 environment. In order to tune the parameters of the proposed algorithm, a full factorial design (Montgomery 2008) is applied. Table 1 shows the proposed values for the parameters.

Table 1. Tuning the parameters for the proposed algorithm.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Small-sized instances</th>
<th>Medium-sized instances</th>
<th>Large-sized instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of countries</td>
<td>100</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Number of initial imperialist</td>
<td>10</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>$\gamma$ revolution probability</td>
<td>0.02</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>Revolution probability</td>
<td>0.02</td>
<td>0.02</td>
<td>0.05</td>
</tr>
</tbody>
</table>

5.1. Data generation

To present the efficiency of the proposed solution method, problems with different sizes from 6 to 150 jobs are considered. The processing times are generated from the discrete uniform distribution $[30, 100]$, and UB for makespan of the AG2's jobs is considered as $\sum p_i$/number of machines. Levels of other factors are shown in Table 2.

Table 2. Problems factor levels.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of machines</td>
<td>2, 3</td>
</tr>
<tr>
<td>Number of jobs (small size)</td>
<td>6, 7, 8, 9, 10</td>
</tr>
<tr>
<td>Number of jobs (medium size)</td>
<td>15, 20, 30, 40, 50</td>
</tr>
<tr>
<td>Number of jobs (large size)</td>
<td>70, 80, 90, 100, 150</td>
</tr>
<tr>
<td>Number of jobs from AG1</td>
<td>Number of jobs/2</td>
</tr>
<tr>
<td>Number of jobs from AG2</td>
<td>Number of jobs - Number of jobs/2</td>
</tr>
<tr>
<td>$v$</td>
<td>100</td>
</tr>
<tr>
<td>$w$</td>
<td>5</td>
</tr>
<tr>
<td>Processing times AG1</td>
<td>$[1,v]$</td>
</tr>
<tr>
<td>Processing times AG2</td>
<td>$[1,w]$</td>
</tr>
</tbody>
</table>

The optimum results, calculation time to solve instances by the proposed solution methods and the produced errors are reported in Table 3.

Table 3. Experimental results.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Small-sized instances</th>
<th>Medium-sized instances</th>
<th>Large-sized instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of countries</td>
<td>100</td>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>Number of initial imperialist</td>
<td>10</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>0.02</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>Revolution probability</td>
<td>0.02</td>
<td>0.02</td>
<td>0.05</td>
</tr>
</tbody>
</table>

PRD is obtained from the formula:

$$PRD = \frac{\text{Alg}(OF) - \min(OF)}{\min(OF)} \times 100$$

where $\text{Alg}(OF)$ is the best OF obtained by an algorithm and $\min(OF)$ is the best OF obtained by GAMS or metaheuristics.
where $\text{Alg}(OF)$ is the objective function value obtained for a given algorithm and $\text{min}(OF)$ is the best solution obtained for each instance by any of the two algorithms in five runs or by GAMS. Also, percentage of the error for metaheuristic outputs is calculated by:

$$ \text{Error} = \frac{|\text{Alg}(OF) - \text{min}(OF)|}{\text{min}(OF)} \times 100\% $$

The algorithm stops, if the $OF$ has not been improved for the last predefined iterations, where in present study this number is 15, 30 and 60 iterations, respectively, for small-, medium- and large-sized instances.

### Table 3. Comparison of results ($n$ is number of jobs and $m$ is number of machines).

<table>
<thead>
<tr>
<th>Size</th>
<th>$n \times m$</th>
<th>GAMS</th>
<th>ICA</th>
<th>Proposed HICA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Time (sec)</td>
<td>Best solution</td>
<td>PRD (%)</td>
</tr>
<tr>
<td>Small</td>
<td>$6 \times 2$</td>
<td>337</td>
<td>7.7</td>
<td>337</td>
</tr>
<tr>
<td></td>
<td>$6 \times 3$</td>
<td>270</td>
<td>27.9</td>
<td>270</td>
</tr>
<tr>
<td></td>
<td>$7 \times 2$</td>
<td>592</td>
<td>86.6</td>
<td>592</td>
</tr>
<tr>
<td></td>
<td>$7 \times 3$</td>
<td>488</td>
<td>265.2</td>
<td>488</td>
</tr>
<tr>
<td></td>
<td>$8 \times 2$</td>
<td>886</td>
<td>774.3</td>
<td>886</td>
</tr>
<tr>
<td></td>
<td>$8 \times 3$</td>
<td>319</td>
<td>1164.7</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>$9 \times 2$</td>
<td>745</td>
<td>1174.9</td>
<td>745</td>
</tr>
<tr>
<td></td>
<td>$9 \times 3$</td>
<td>395</td>
<td>3600.0</td>
<td>395</td>
</tr>
<tr>
<td></td>
<td>$10 \times 2$</td>
<td>1074</td>
<td>3600.0</td>
<td>1074</td>
</tr>
<tr>
<td></td>
<td>$10 \times 3$</td>
<td>809</td>
<td>3600.0</td>
<td>815</td>
</tr>
</tbody>
</table>

| Medium  | $15 \times 2$ | – | – | 1666 | 5.02 | 2.73 | 2.7 | 1622 | 0.67 | 2.16 | 0.0 |
|         | $15 \times 3$ | – | – | 1098 | 3.97 | 3.22 | 2.9 | 1067 | 0.95 | 2.65 | 0.0 |
|         | $20 \times 2$ | – | – | 2785 | 5.49 | 3.59 | 2.8 | 2710 | 0.85 | 3.05 | 0.0 |
|         | $20 \times 3$ | – | – | 2139 | 5.26 | 4.21 | 3.2 | 2072 | 0.73 | 3.27 | 0.0 |
|         | $30 \times 2$ | – | – | 6536 | 2.78 | 7.41 | 2.8 | 6355 | 0.44 | 6.0 | 0.0 |
|         | $30 \times 3$ | – | – | 4308 | 6.26 | 7.51 | 3.2% | 4174 | 0.73 | 7.1 | 0.0 |
|         | $40 \times 2$ | – | – | 12487 | 6.37 | 7.93 | 3.1 | 12105 | 0.73 | 7.46 | 0.0 |
|         | $40 \times 3$ | – | – | 9176 | 3.64 | 9.62 | 3.2 | 8891 | 0.47 | 6.36 | 0.0 |
|         | $50 \times 2$ | – | – | 16967 | 3.22 | 10.92 | 2.8 | 16498 | 0.29 | 9.61 | 0.0 |
|         | $50 \times 3$ | – | – | 11192 | 3.33 | 12.81 | 3.3 | 10830 | 1.03 | 10.17 | 0.0 |

| Average |              | 1430.1 | 1.04 | 1.084 | 0.2 | 0.00 | 0.993 | 0.0 |

| Large   | $70 \times 2$ | – | – | 34769 | 6.30 | 43.17 | 3.4 | 33629 | 0.71 | 30.96 | 0.0 |
|         | $70 \times 3$ | – | – | 24527 | 6.76 | 48.17 | 4.4 | 23489 | 0.36 | 33.15 | 0.0 |
|         | $80 \times 2$ | – | – | 38180 | 4.55 | 51.85 | 3.1 | 37028 | 0.49 | 39.32 | 0.0 |
|         | $80 \times 3$ | – | – | 29351 | 5.72 | 54.93 | 3.8 | 28282 | 0.71 | 41.78 | 0.0 |
|         | $90 \times 2$ | – | – | 49626 | 5.53 | 44.46 | 2.6 | 48359 | 0.34 | 35.15 | 0.0 |
|         | $90 \times 3$ | – | – | 33731 | 5.95 | 52.63 | 4.3 | 32350 | 0.89 | 36.3 | 0.0 |
|         | $100 \times 2$ | – | – | 66895 | 3.74 | 54.57 | 3.7 | 64533 | 0.83 | 37.18 | 0.0 |
|         | $100 \times 3$ | – | – | 40174 | 3.68 | 66.03 | 4.2 | 38543 | 0.61 | 47.5 | 0.0 |
|         | $150 \times 2$ | – | – | 155854 | 3.50 | 124.72 | 2.7 | 151800 | 0.27 | 86.26 | 0.0 |
|         | $150 \times 3$ | – | – | 102408 | 4.14 | 130.76 | 3.3 | 99155 | 1.11 | 89.85 | 0.0 |

| Average |              | 4.99 | 67.129 | 3.5 | 0.63% | 47.745 | 0.0 |
|         | Average      | 3.52 | 25.0693 | 2.6 | 0.44 | 18.1737 | 0.0 |

Figure 15. Comparison the computational time of the GAMS and the metaheuristic algorithms for small-sized instances.
medium- and large-sized instances. According to Table 3, the results show the superiority of proposed algorithm to ICA in all instances regarding the quality of the solutions and computational time. Figure 15 shows the comparison of computational time of the GAMS and metaheuristic algorithms for small-sized instances. As this figure shows, the GAMS is not able to solve the instances with more than nine jobs in reasonable times. Also, computation times of the proposed algorithm are less than the computation times of the ICA. The computation times of the proposed HICA and ICA, for medium- and large-sized instances, are represented in Figures 16 and 17, respectively. As it can be seen, the proposed metaheuristics have a superior performance than the ICA. Also, PRD plot for the interaction between algorithm and instances are depicted in Figure 18.

6. Conclusion and future research
Over the past decade, the multi-agent scheduling problem has been becoming an attentively studied field, as is being used in lots of industrial areas. A lot of research has been done and many assumptions have been considered in literature. This study considered the expansion of the multi-agent scheduling problem where tool change activity is considered. For this problem, a mathematical model is
presented and this model is solved for small-sized instances exactly by GAMS. For medium- and large-sized instances, an efficient hybrid algorithm based on ICA is developed to solve the problem. In order to verify the performance of the proposed algorithm, problems with different sizes are solved. Computational results showed that the proposed algorithm is capable to obtain appropriate solutions for practical size problems in a reasonable computational time.

Developing new exact or metaheuristic algorithms for the considered problem and also applying proposed ICA for other scheduling problems under job deterioration assumption are interesting future research directions.

Acknowledgement

We are grateful for the valuable comments and suggestion from the respected reviewers. Their valuable comments and suggestions have enhanced the strength and significance of our paper.

Disclosure statement

No potential conflict of interest was reported by the authors.

References


