J Intell Manuf
DOI 10.1007/s10845-015-1039-3

An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem

Maroua Nouiri · Abdelghani Bekrar · Abderezak Jemai · Smail Niar · Ahmed Chiheb Ammari

Received: 17 July 2014 / Accepted: 17 January 2015

Abstract Flexible job-shop scheduling problem (FJSP) is very important in many research fields such as production management and combinatorial optimization. The FJSP problems cover two difficulties namely machine assignment problem and operation sequencing problem. In this paper, we apply particle swarm optimization (PSO) algorithm to solve this FJSP problem aiming to minimize the maximum completion time criterion. Various benchmark data taken from literature, varying from Partial FJSP and Total FJSP, are tested. Experimental results proved that the developed PSO is enough effective and efficient to solve the FJSP. Our other objective in this paper, is to study the distribution of the PSO-solving method for future implementation on embedded systems that can make decisions in real time according to the state of resources and any unplanned or unforeseen events. For this aim, two multi-agent based approaches are proposed and compared using different benchmark instances.

M. Nouiri · A. Jemai
LIP2 Laboratory, Faculty of Science of Tunis, Tunis, Tunisia

A. C. Ammari
MMA Laboratory, INSAT Institute,
Carthage University, 1080 Tunis, Tunisia

A. C. Ammari
Renewable Energy Group, Department of Electrical and Computer Engineering, Faculty of Engineering,
King Abdulaziz University, Jeddah 21589, Saudi Arabia

A. Bekrar · S. Niar
UVHC, LAMIH Laboratory, 59313 Valenciennes, France

A. Bekrar (✉) · S. Niar
University of Lille Nord de France, 59000 Lille, France
e-mail: Abdelghani.Bekrar@univ-valenciennes.fr

Keywords Flexible job-shop scheduling problem · Makespan · Particle swarm optimization algorithm · Routing and scheduling · Multi-agent system · Enmbeded system

Introduction

The study of production systems is complex given the large number of integrated entities and their interactions. This complexity is one of the scheduling problem causes, which are generally recognized as very difficult to solve. Particularly, the large number of entities that needs to be managed according to a given schedule with a lot of tasks may involve a lot of constraints. Among these constraints we can mention the precedence, transportation, due date, time and/or resource availability. Scheduling of operations involves the resource allocation over a period of time to perform a collection of tasks. This is one of the most critical issues in the planning and managing of manufacturing processes (Pezzellaa et al. 2008).

Getting the best solutions for the scheduling problems is of a big importance for industry, since the production rate and the expense of any production plant depend on the schedules used for controlling the work in the plant (Motaghedi-larijani et al. 2010). An effective schedule enables the industry to utilize its resources effectively and attain the strategic objectives as reflected in its production plan (Girish and Jawahar 2009).

There are several types of scheduling problems in a production system that differ according to the definition of their range and their organization. One of the most difficult problems in this area is the Job-shop Scheduling Problem (JSP), where a set of jobs must be processed by a set of machines. Each job is formed by a sequence of consecutive opera-

Published online: 05 February 2015

tions and each operation requires exactly one machine. The decision concerns how to sequence the operations on the machines, where a given performance indicator has to be optimized (Pezzellaa et al. 2008).

In this paper we are interested in the flexible job shop scheduling problem (FJSP) which is a generalization of the classical job shop problem. Each operation can be processed on a given machine that is chosen among a finite subset of candidate machines. The FJSP is more difficult than the classical JSP since it adds a new decision level to the sequencing one, i.e., the machine assignment that involves the selection of one machine among the available ones for each operation. Thus, the FSJP flexibility will be considerably increasing the complexity of the problem as it requires an additional level of decisions. The aim is to find an allocation for each operation and to define the sequence of operations on each machine to minimize the maximum competition time called the makespan (Tanga et al. 2011).

For its strongly NP-hard nature, many efficient heuristics and meta-heuristics methods are developed to get nearly optimal solutions which satisfy the constraints and minimize or maximize the objective function (Tanga et al. 2011; Lai and Wu 2013; Gao et al. 2014; Gen and Lin 2014; Fernández and Raupp 2014). Among these methods, we have Genetic Algorithms (GA) (Jalilvand-Nejad and Fattahi 2013), Particle Swarm Optimization (PSO) (Jia et al. 2007; Girish and Jawahar 2009; Jun-jie et al. 2009) and Tabu Search (TS) methods (Li et al. 2010; Meeran and Morshed 2012).

This paper investigates an effective particle swarm optimization algorithm for flexible job-shop scheduling problem (FJSP). Moreover, in a real situation, events like machine breakdowns and urgent client orders may occur which affects the normal and good functioning of the shop floor. In this case, the centralized approaches, like those cited above are generally inefficient. For a better efficiency coping with such situation, we propose for the FJS problem a distributed approach based on multi-agent system mechanism.

The presentation of the paper is organized as follows: First, an overview of relevant literature review is discussed in section "Literature review". Section "Problem formulation" gives the formulation of FJSP. The PSO algorithm, parameters selections, encoding scheme, initialization of the swarm and neighborhoods topologies are presented in section "Particle swarm optimization". Two multi-agent system based approaches are presented in section "Distributed approche using multi-agent system". In section "Experimental results", the developed PSO algorithms are applied to solve common benchmarks selected from literature and the obtained performance results are analyzed and discussed. Concluding remarks and future study directions are given in section "Conclusions".

Literature review

This section gives a literature review for solving the FJSP while focusing only on meta-heuristic approaches such as Tabu Search (TS), Simulated Annealing (SA), Genetic Algorithms (GA) and the recently developed PSO (Pezzellaa et al. 2008). A Tabu Search (TS) algorithm is implemented by Brandimarte to solve the flexible job shop scheduling problem targeting a makespan minimization (Brandimarte 1993).

Genetic Algorithms (GA) which are effective meta-heuristics to solve combinatorial optimization problems have been also successfully adopted to solve the FJSP. Many papers are written on this topic (Kacem et al. 2002; Pezzellaa et al. 2008; Zhang et al. 2011; Zambrano rey et al. 2014). All these studies used different integrated GA approaches with different coding schemes, initial population generation, chromosome selection or offspring generation strategies (Pezzellaa et al. 2008). Kacem et al proposed in (Kacem et al. 2002) a genetic algorithm controlled by the assignment model that was generated by the approach of localization to mono-objective and multi-objective FJSP. Pezzellaa et al proposed a genetic algorithm with several techniques to create population and select the individuals to reproduce new individuals (Pezzellaa et al. 2008). Zhang et al. developed a genetic algorithm and proposed two assignment methods: Global Selection (GS) and Local Selection (LS) to generate high-quality initial population in the initialization stage and accelerate the convergence speed (Zhang et al. 2011). Motaghedi-larijani et al proposed an hybrid GA capable to optimize different objectives functions such: minimizing makespan, minimizing total workload, and minimizing workload of the most loaded machine (Motaghedi-larijani et al. 2010). In this work, a hill climbing approach is also implemented to improve the GA solutions (Motaghedi-larijani et al. 2010).

Rather than GA, PSO has been widely used in many real world applications due to its simplicity and capability to deal efficiently with optimization problems. Recently, PSO is implemented to solve FJSP. In this context, Venter and Sobieszczanski introduced a parallel PSO algorithm to enhance the performance of the approach (Venter and Sobieszczanski-Sobieski 2006). Zhaohong Jia developed an improved PSO to optimize the best overall solution by adding chaotic methods (Jia et al. 2007). Bai Jun-jie et al proposed a PSO algorithm to solve FJSP with split lot (Jun-jie et al. 2009).

Recently the hybridization of PSO with other meta-heuristics to better solve the FJSP have been also successfully adopted. Many papers are written on this topic. Xia and Wu introduced a hybrid algorithm that combines PSO and simulated annealing to solve FJSP while minimizing the makespan (Xia and Wu 2005). Hongbo Liu et al implemented a hybrid metaheuristic called the Variable Neighborhood Particle Swarm Optimization (VNPSO) to solve a multi-objective FJSP (Liu et al. 2007). Zhang et al. proposed an hybrid algorithm between PSO and Tabu search to solve FJSP (Zhang et al. 2009). Jun-qing Li et al proposed an hybrid algorithm that combines PSO, Tabu search and genetic operators (mutation and crossover operators) to solve an FJSP problem (Li et al. 2010). Tanga et al proposed an hybrid algorithm that combines PSO and genetic algorithms (Tanga et al. 2011).

During the last few years, successful results have been achieved using distributed approaches like multi-agent systems to solve complex dynamic flexible job shop scheduling problems targeting the next generations of advanced manufacturing systems that need to incorporate more distributed control; machine flexibility and product flexibility (Emin 2012). For this case, Chen et.al (Chen et al. 2004) proposed an agent-based genetic algorithm for solving a JSP problem. In this work, the creation of the initial population is accelerated and the processing of selection, crossover and mutation are controlled in an intelligent way. Asadzadeh and Zamanifar (Asadzadeh and Zamanifar 2010) implemented an agent-based parallel genetic algorithm for the job shop scheduling problem. In this case, the initial population is created in an agent-based way by using the method proposed in (Chen et al. 2004) and an appropriate migration policy is defined to coordinate the exchange of migrants between the different agents. In (Azzouz et al. 2012) Azzouz and Ennigrou proposed a new promising multi-agent approach to solve the FJSP. The proposed model combines a local optimization approach based on Tabu Search (TS) metaheuristic and a global optimization approach based on genetic algorithm (GA). In this new approach two sorts of agent are used: the first agent is named resource agents and is responsible for the local optimization process using Tabu search. The second is the interface agent who has a global view of the system and is responsible for a global optimization based on Genetic algorithm.

Another new multi-agent scheduling system MASS for solving FJSP is presented by Wei and Dongmei in (Wei and Dongmei 2012). In this work, the MASS is inspired by the structure and negotiation strategies of the human immune system HIS. Many other works can also be cited here. Ennigrou and Ghedira (Ennigrou and Ghédira 2004) propose a multi agent model based on Tabu Search (TS). The authors in (Ennigrou and Ghédira 2008) propose a Multi agent approach based on a new local diversification technique for FJSP. This work has been used by Henchiri and Enngirou in (Henchiri and Enngirou 2013) to investigate a multi-agent model based on the hybridization of TS and PSO. This is called Flexible Job Shop Multi-Agent Tabu Search Particle Swarm Optimization (FJS MATSPSO). The implemented model is composed of Resources agents responsible for a local optimization process based on TS and an Interface agent responsible for a global optimization based on PSO.

Finally, and as it is reported in all the previously cited studies, different meta-heuristics are applied to solve the FJSP problem. The challenge is always to have the appropriate meta-heuristic capable for better solving this problem. A comparative study between these different meta-heuristics has shown that genetic algorithms and particle swarm optimization are the most effective in terms of the results performance quality

Problem formulation

There are two kinds of FJSP, Total FJSP (T-FJSP) and Partial FJSP (P-FJSP). For the T-FJSP, each job can be operated on every machine from the set of machines M (Li et al. 2010) while for the P-FJSP, each operation can be processed on one machine of subset of M (Liu et al. 2007).

We define the flexible job shop scheduling problem formally with the following definitions:

- J $= \{J_1, J_2 \ldots J_n\}$ is a set of n independent jobs to be scheduled. Each job J_i consists of a predetermined sequence of operations. O_{ij} is the operation j of job J_i.
- M $= \{M_1, M_2, \ldots, M_m\}$ is a set of m machines. Each machine can process only one operation at a time. Each operation can be processed without interruption during its performance on one of the set of machines. We denote with P_{ijk} the processing time of operation O_{ij} when executed on machine M_k. All machines are available at time 0.

Constraints

The constraints are rules that limit the possible assignments of the operations. In our case, the following constraints are considered:

- Jobs are independent and no priorities are assigned to any job type.
- Each machine can process only one operation at a time.
- Job pre-emption or cancellation is not allowed.
- All jobs are simultaneously available at time zero.
- The predetermined sequence of operations for each job forces each operation to be scheduled after all predecessor operations (precedence/conjunctive constraint).
- There are no precedence constraints among operations of different jobs.
- After a job is processed on a machine it is transported to the next machine immediately and the transportation time is negligible.
- Breakdowns are not considered.

Objective function

The optimization is expressed by a function that shall minimize or maximize a given criteria. Most of the reported research focused on single objective optimization cases. For these cases, the objective is to find a schedule that requires a minimum time to complete all operations. This minimum time is called the minimum makespan time. We may have to optimize additional objectives, such as earliness and tardiness of jobs, minimizing the total workload, and minimizing the workload of the most loaded machine. These different cited objectives can be combined to implement a multi objective optimization. In the literature, many papers are written on multi-objective optimization for FJSP (Binh and Cing 2008; Motaghedi-larijani et al. 2010). In this paper, we are interested in a single objective optimization case that consists in minimizing the makespan denoted as C_{max}. The makespan value is calculated as follows:

$$C_{max} = \max t_{ij}$$

where t_{ij} is the completion time of operation O_{ij}.

Particle swarm optimization

Method description

The Particle Swarm Optimization algorithm was introduced in 1995 by Kenney and Eberhart. It was inspired from the social behavior of animals living in swarms, such as flocks of birds (Kennedy and Eberhart 1995). The PSO algorithm is initialized with a population of particles; each of the particles represents a candidate solution to a problem and has three main attributes: the position in the search space $\mathrm{x_i(t)}$, the current velocity $\mathrm{v_i(t)}$ and the best position ever found by the particle during the search process $\mathrm{x_i^*(t)}$. The principle of the algorithm is to move these particles to find optimal solution. The search trajectory of a particle is regulated according to the flying experience of its own and its neighboring particles. During the search, each particle updates its velocity and position by the following equations (Li et al. 2010) :

$$v_i(t+1) = w * v_i(t) + c_1 * \left[x_i^*(t) - x_i(t)\right] + c_2 * \left[x_g^*(t) - x_i(t)\right] \quad (1)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (2)$$

where $\mathrm{x_i^*(t)}$ is the best position of each particle which represents the private best objective (fitness) value so far obtained, and $\mathrm{x_g^*(t)}$ is the global best particle which denotes the best position among all particles in the population. w is the inertia weight, which is used to maintain the particle; c_1 and c_2 are random numbers between [0,1].

Parameters selections

The PSO algorithm includes some tuning parameters that greatly influence the algorithm performance. Despite recent research efforts, the selection of the algorithm parameters remains empirical to a large extent. Several typical choices of the algorithm parameters are reported in (Trelea 2003; Clerc and Kennedy 2002). Ioan Cristian Trelea in (Trelea 2003) propose parameter selection guidelines in order to guarantee the optimal convergence. There are many several kinds of coefficient adjustments. The one used in this paper was developed by Kennedy (Kennedy 1999) and involves the constriction factor presented in the equation below:

$$\mathrm{k} = 1 - \frac{1}{\alpha} + \frac{\sqrt{|\alpha^2 - 4*\alpha|}}{2}$$

where $\alpha = c_1 + c_2 > 4;\ c_1 = 2;\ c_2 = 2,1$

So the equation of velocity becomes:

$$\mathrm{v_i(t+1) = k * \Big(v_i(t) + c_1 * \left[x_i^*(t) - x_i(t)\right] + c_2 * \left[x_g^*(t) - x_i(t)\right]\Big)} \quad (3)$$

Encoding particle

In order to successfully apply PSO for solving the FJSP problem, appropriate representation of particles is of a big importance. In this context, (Jia et al. 2007) divided the position of a particle as well as the velocity into two vector parts: *Process [ ]* and *Machine [ ].*

In this improved representation, *Machine [ ]* vector represents the machines assigned to operations and *Process[]* defines the sequence of operations. Each component in *Process[ ]* vector denotes an operation of job j, which is j a natural number less than n. The sequence of the components determines the order of operations for the different jobs. Each component in vector *Machine[]*, is a natural number m that denotes the number of a selected machine that processes the corresponding operation in *Process[]*. The two *Process* and *Machine* vectors have the same size which is equal to the total number of operations.

In Fig. 1 we give a possible encoded particle for a 4*5 FJSP instance.

In Fig. 1, the total number of operations is 12, so the particle is denoted as two vectors, each with 12 dimensionalities. The number in *Process[ ]* denotes the number of a job, and the same job numbers in different positions represent the operation order of this job. The number of occurrence of each number corresponds to the total number of operations of this job (Job 1: 3 operations, Job 2: 3 operations, Job 3: 4 operations, Job 4: 2 operations). So in this representation the feasibility is considered automatically.

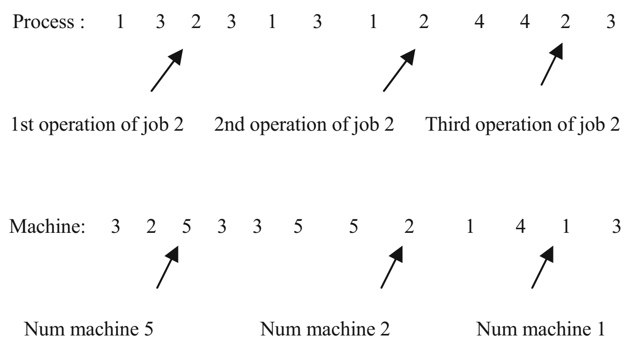

Fig. 1 Encoding particle scheme

In the PSO, the diversification of the population is based in general on the neighborhoods topologies described in the following sub-section. However, to further improve this diversification of the algorithm, we impose a distance between particles positions as defined by Clerc in his paper (Clerc 2004).To diversify and to more explore the research space, we impose that there is a minimum distance between solutions. Let $x1$ and $x2$ be two positions. The distance between these positions is defined by $\mathrm{d(x1, x2)} = \mathrm{x1} - \mathrm{x2}$. If the distance between the current position and the new position is less than a defined small epsilon, this means that it does not move much and it goes exactly at near the same position.

Initialization

The quality of the initial population affects to a certain extent the quality of the solution or the convergence speed of the population. So, how to produce a better initial population quality becomes a critical step in the first section.

The initialization process can be divided into two stages (Li et al. 2010) machine assignment initialization and operation sequence initialization. Several approaches for the initialization of machine assignment and sequence initialization are developed.

In this paper we use three approaches for the selection of the initial population:

- Random rule (Li et al. 2010)**:** which consist to select a random number of jobs and machines for every operation.
- Approaches by localization of Kacem et al. (2002)**:** we select randomly a number of jobs. Then we choose the machine which has a minimum processing time for the first operation to the selected job. Next we apply the approach by localization of Kacem et al (Kacem et al. 2002), which consists in finding the machine with the minimum processing time for each operation, for the rest of operations. Then select the global minimum processing time fixing that assignment, and then to add this time to every subsequent entry in the same column (machine workload update),
- Modified approach “MMkacem”: We select randomly a number of jobs and then we search for the machine that has the minimum processing time in the processing time table.

Neighborhoods topologies

Another important issue that deserves attention, is the communication topology used to spread information inside the swarm. Many topologies have been proposed and improved to accelerate the convergence process (Bastos-Filho et al. 2009). The two most commonly used methods are known as *gbest* and *lbest*.

In the *gbest* or the star topology, particles can share information globally through a fully-connected structure. This topology uses a global neighborhood mechanism where the trajectory of each particle’s search is influenced by the best point found by any member of the entire population (Kennedy 1999).

On the other hand, there are topologies based on a local neighborhood, called ring or lbest (Kennedy and Mendes 2003). In this approach, the particles only share information with their direct neighbors defined based on indexes.

Some others topologies have been proposed to balance the extreme behavior of the gbest and lbest approaches, such as the Von Neumann topology where particles are connected by a grid creating a social structure.

Recently, a dynamic communication topology was proposed to improve the PSO degree of convergence focusing on the distribution of the particles in the search space such us Dynamic ring and Dynamic Clan (Bastos-Filho et al. 2009; Wang and Xiang 2008). We have tested these topologies cited below and we have not found any approach that is better in comparison the others in terms of quality of the solution.

Algorithm description

In a previous study (Nouiri et al. 2013), we proposed a Particle swarm optimization (PSO) to solve deterministic FJSP and the obtained results show that the approach is very efficient in solving these problems. According to the above description, we give the PSO algorithm for FJSP as follows:

Step 1 : initialize all parameters : ***Swarm size***, number of generation ***Max_Iteration***, c_1, c_2, ***wmax, wmin***
Step 2 : Generation 0

- Initialize swarm : the population of particles: 60 % MMKacm, 20 % KacemHammadi, 20 % random,
- evaluate each particle : compute the fitness value C_{max}.

```
    - initialize pBestParticule of each particle by a copy of itself.
    - initialize gBestParticule as the global particle with the minimal fitness value in the current population.
Step 3 : While (generation < Max_Iteration) do
    - generation = generation + 1.
    - Update the velocities and positions for each particle i by Eqs. (2) and (3)
    - update fitness value for each particle.
    - update pbestParticule for each particule.
    - update global solution gbestParticule.
    End while
Step 4: output result gBestParticule (the order of different operations of different jobs, machine assignment, fitness value).
```

Distributed approche using multi-agent system

The high combinatorial complexity of the flexible job shop problem makes it hard to find the optimal solution within a reasonable time in most cases. The multi-agent optimization method can reduce the combinatorial complexity of the problem using task decomposition and real-time distribution (Asadzadeh and Zamanifar 2010).

A multi-agent system is an artificial system composed of a population of autonomous agents, which cooperate with each other to reach common objectives, while simultaneously each agent pursues individual objectives (Chen et al. 2004).

Agents and multi-agent systems have wide application in parallel and distributed systems. One of the important features of agents is their capability in parallel implementing of meta-heuristics. Also, among the important advantages of any multi-agent method is to allow one agent to solve subproblems locally and to propose a global solution as a result of interactions between the different agents. We used this feature in our approach and propose a parallel and distributed model for the flexible job shop scheduling problem. We developed a multi-agent optimization system based on PSO namely Muti-Agent Particle Swarm Optimization MAPSO containing some agents with special actions.

In this section we introduce two agent-based methods and briefly describe their structures.

First multi agent PSO model (MAPSO1)

In order to compare the execution time between PSO and Multi agent PSO, we first propose a simple architecture based on the PSO proposed in this paper. Our experiments are concentrated on evaluating the makespan and the CPU time. In this model, each agent has its own acquaintances (the agents that it knows and with which it can communicate), a local memory composed of its static and dynamic knowledge and a mailbox in which it stores the messages received from the other agents. We can describe them as follows.

Execute agent (EA)

An optimization process based on particle swarm optimization method is located in the Execute Agent. This agent creates a sub-population in the first stage with a specific initialization method and then runs the PSO algorithm. In the second stage, they transmit the best particle found to synchronization agent (SA).

According to the implemented initialization methods, we define three Execute Agent (EA). Each EA create its own population with a specified initialization method and then runs PSO algorithm (see section "Particle swarm optimization").

Synchronization agent (SA)

The acquaintances of a Resource agent are composed of all existing agents in the system. This agent is responsible for triggering the collection phase (receiving the best particle from each Execute agent EA) and then determines the global best particle.

The architecture of our method with the different layers are shown in Fig. 2.

Second multi agent PSO model (MAPSO2)

In the first *MAPSO1* architecture, the Synchronization Agent is not of a big use as it is absent throughout the optimization phase and will be performing only in the collect phase (reception best particle from each EA).In addition, we are targeting the better suitable Multi agent PSO architecture for embedded system in terms of the number of agents, the role of each agent, the effects of migration policy etc. So we proposed a second MAPSO which has a more complex architecture. This multi agent MPPSO model is inspired from (Asadzadeh and Zamanifar 2010).

The layers of the proposed *MAPSO2* agent-based architecture for FJSP are shown in Fig. 3.

In this model, each agent is developed for a special purpose. The description of each agent is as follows:

Boss agent (BA)

The acquaintances of BA are composed of the Synchronization Agent (SA). BA has the responsibility of creating the initial population for the PSO algorithm and of the global optimization process.

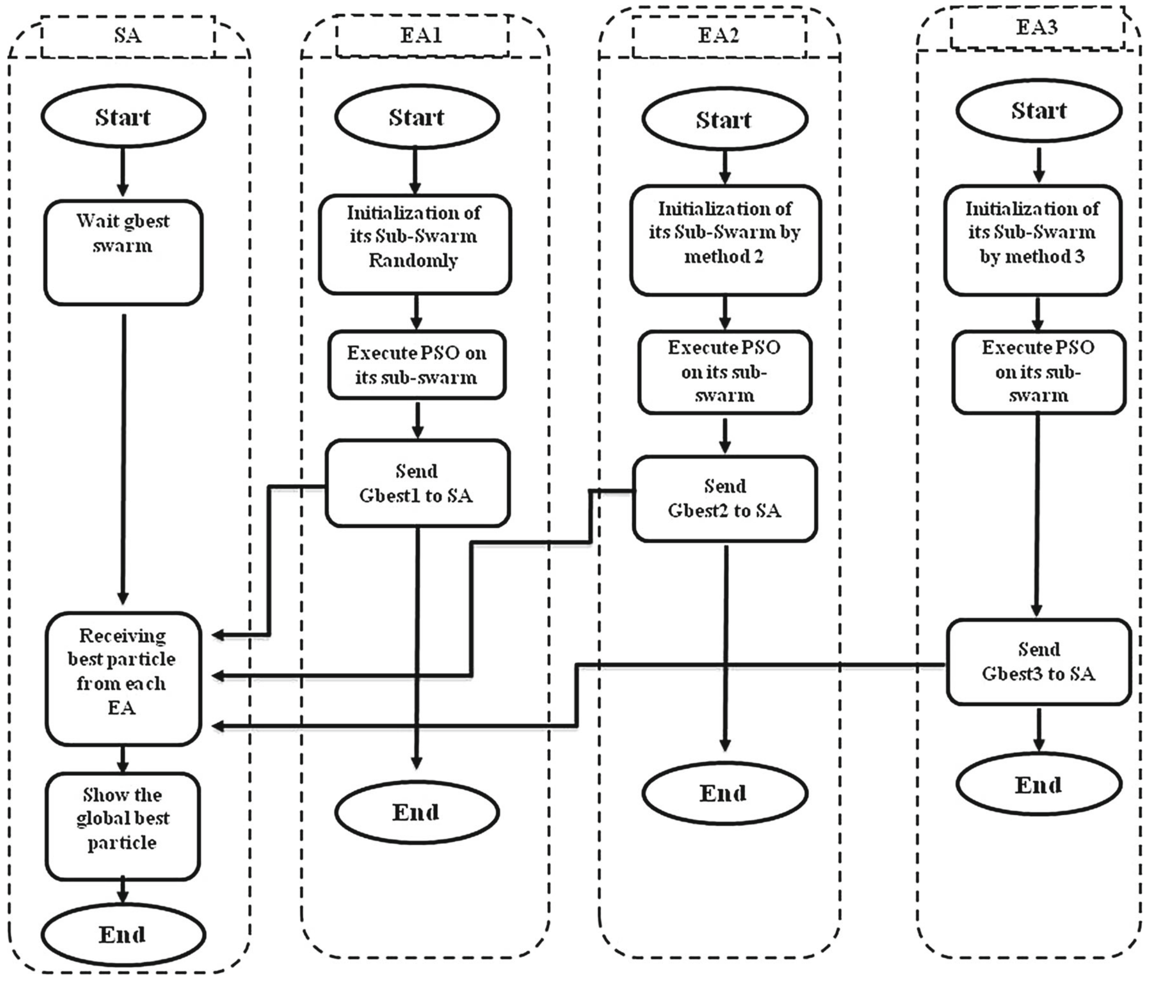
SA
EA1
EA2
EA3
Start
Wait gbest swarm
Receiving best particle from each EA
Show the global best particle
End
Initialization of its Sub-Swarm Randomly
Execute PSO on its sub-swarm
Send Gbest1 to SA
Initialization of its Sub-Swarm by method 2
Send Gbest2 to SA
Initialization of its Sub-Swarm by method 3
Send Gbest3 to SA

Fig. 2 Architecture of the proposed multi agent MAPSO1

The BA sends the initial population to the synchronizer agent SA and then waits until it receives the best particle swarm. Finally, The BA executes the PSO algorithm and shows the best particle that has the minimum makespan.

Synchronization agent (SA)

The acquaintances of SA are composed of all agents existing in the system. This SA agent divides the population received from the BA in sub-populations and sends them to the Execute Agent (EA). This agent is responsible also for triggering the migration phase (coordinating the migration between the EAs described hereafter) and the collection phase (receiving the best particle from EA).After receiving all best particles, the SA sends the global best swarm among the best particles to the BA.

Execute agent (EA)

This agent runs, in the first stage, the PSO algorithm with a subpopulation received from the synchronizer SA agent. In the second stage, they awaits the start of the migration phase, then at the beginning of this phase, they transmits a sub-swarm of a certain migrant best particle (defined in advance) to its neighbors according to a the migration policy. The EA receiver agent replaces a certain initial particles in their swarm by the received particles and they runs the PSO algorithm with a new sub-population. Finally, at the collection phase, they transmit the best particle found to the SA. The number of Execute Agent can be configured according to the number of machines of the production unit. In our architecture, we define two Execute EA agents.

Migration policy

Communication between sub-populations of EAs is carried out by exchanging migrants. Each EA executes the PSO algorithm on its sub-population, and then sends a message to the SA informing it of the end of its execution. The SA is a synchronization agent, which coordinates migration between sub-populations of EA agents. After receiving a message from all the EAs, the SA broadcasts a back to all of the

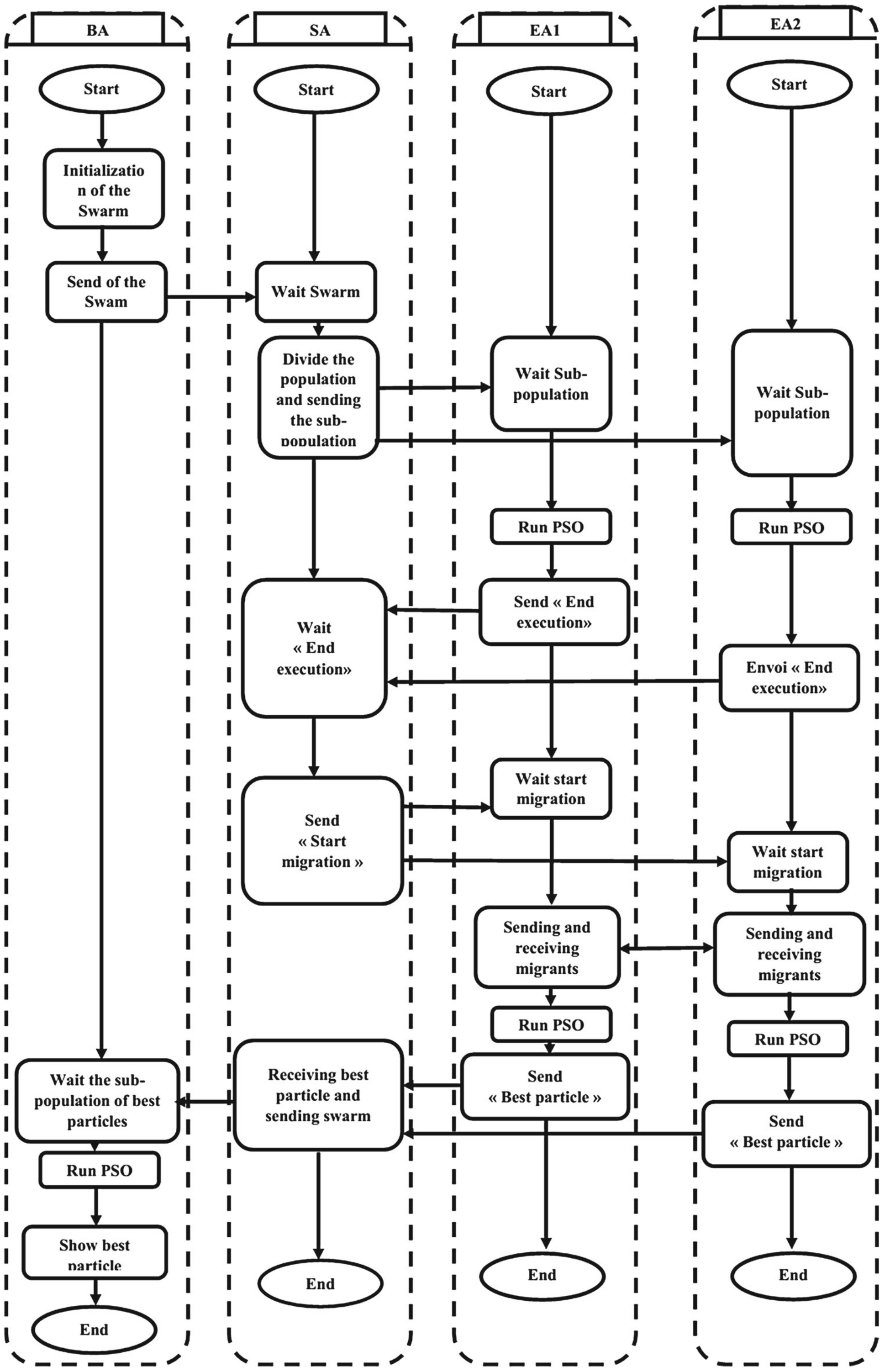

Fig. 3 Architecture of the proposed multi agent MAPSO2

EA agents notifying them for the start of the migration phase where each EA exchanges some of its best particles with its neighbors. Particles with low fitness value in the sub-population are replaced with the best particles of neighbors.

Experimental results

To test the effectiveness and performance of the PSO algorithm, first we carried out experiments with three different

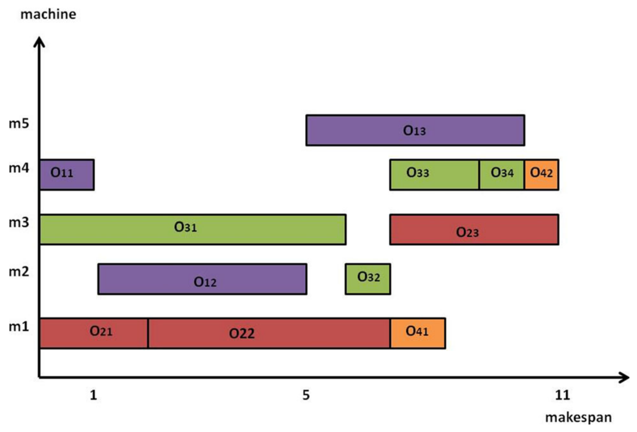

Fig. 4 The Gantt chart of instance 1 (4 jobs * 5 machines)

Table 1 Compared results on problem 4 * 5

Algorithm	Makespan
Our PSO	11
AL+GA	16
PSO+SA	11
GA	13
GA+HC	12

benchmark FJSP instances: two FJSP instances with total flexibility, and one with partial flexibility. The characteristics of these instances are available in (Jia et al. 2007; Motaghedi-larijani et al. 2010). Each instance can be characterized by number of jobs (n), number of machines (m), and the related operations O_{ij} associated to each job i. Each instance is run for 20 times and all the results are presented in next section. More experiments are then performed on large instances from Brandimarte (Brandimarte 1993).

The Flexible Jop Scheduling Multi Agent PSO (FJS MAPSO) was implemented in Netbeans with Java language. For the development of the multi-agent system we choose JADE (Java Agent Development framework) which is a free and an open source agent development platform. This middleware facilitates the creation of agents and their communication using message passing. The implementation has been performed on a machine based on Intel processor "Core2Duo" clocked at 2.0 GHz and having 3070 MO of memory.

PSO results using Kacem instances (Kacem et al. 2002)

Test on 4 × 5 problem

First, a small scale instance taken from (Motaghedi-larijani et al. 2010) is used.

The PSO parameters are as follow: ***swarmsize=100, Max_Iteration=200.***

The schedule's Gantt chart representation corresponding to the best obtained solution is shown in Fig. 4.

Comparing the results of our algorithm with some other well known reference algorithms is shown in Table 1. This table clearly shows the outperformance of our algorithm for small scale instances.

Where:

'AL+GA' refers to (Kacem et al. 2002), 'PSO+SA' refers to (Xia and Wu 2005), 'PSO+TS' refers to (Zhang et al. 2009) and 'GA+HC' refers to (Motaghedi-larijani et al. 2010).

Test on 10 × 10 problem

To evaluate the efficiency of the proposed PSO algorithm for a middle scale instance, a 10×10 problem taken from (Jia et al. 2007) is tested. The PSO parameters are as follow: ***Swarm_Size=500, Max_Iteration=500***.

The schedule's Gantt chart representation corresponding to the obtained solution is shown in Fig. 5. The comparison of our proposed algorithm with other reference algorithms is shown in Table 2.

Where

'Temporal decomposition', 'Classic' GA' and 'Approach by Localization' refers to (Kacem et al. 2002), 'PSO+SA' refers to (Xia and Wu 2005), 'PSO+TS' refers to (Zhang et al. 2009) and 'GA+HC' refers to (Motaghedi-larijani et al. 2010).

Test on 8 × 8 problem with partial flexibility

Finally, a middle scale instance problem 8×8 with partial flexibility taken from (Motaghedi-larijani et al. 2010) is experimented.

The PSO parameters are as follow: ***Swarm_Size* =500, *Max_Iteration* =500**.

The schedule's Gantt chart representation corresponding to the obtained solution is shown in Fig. 6.

Comparing in this case the obtained solution with the reference algorithms is shown in Table 3.

Where:

'Classic GA' and 'AL+CGA' refers to (Kacem et al. 2002), 'PSO+SA' refers to (Xia and Wu 2005), 'PSO+TS' refers to (Zhang et al. 2009).

Finally, all these obtained results clearly prove that our developed PSO is effective and efficient to solve flexible job-shop scheduling problem without any hybridization. However it is known that PSO method lacks from premature convergence and for this reason most of the previous reference papers are using the PSO by hybridization with other meth-

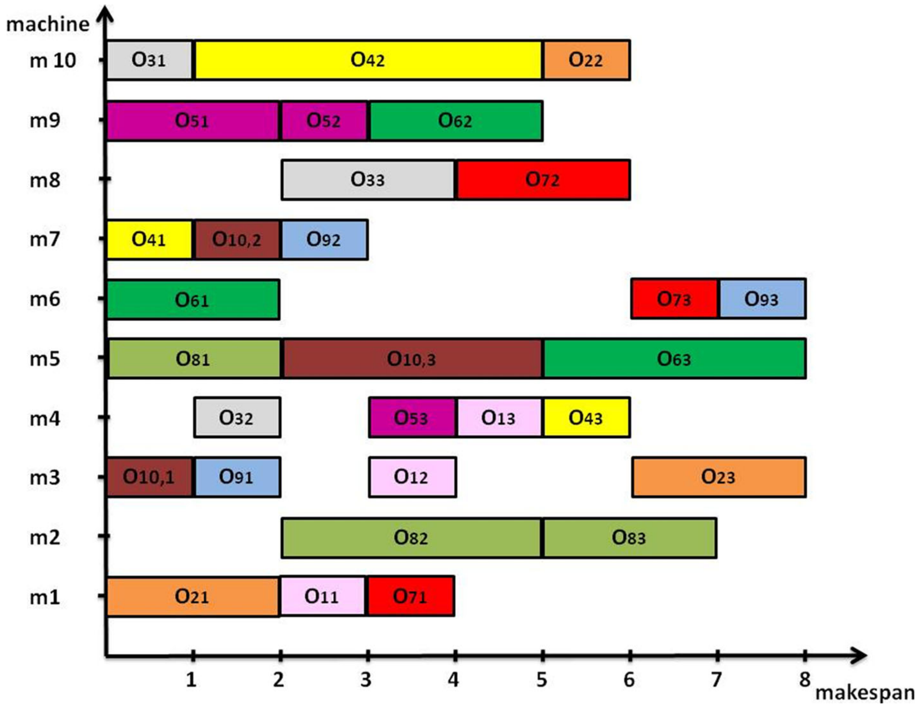

Fig. 5 The Gantt chart of instance 2 (10 jobs * 10 machines)

Table 2 Comparison of results on problem 10 * 10

Algorithm	Makespan
Our PSO	8
Temporal decomposition	16
Classic GA	7
Approach by localization	8
PSO+SA	7
PSO+TS	8
GA+HC	8

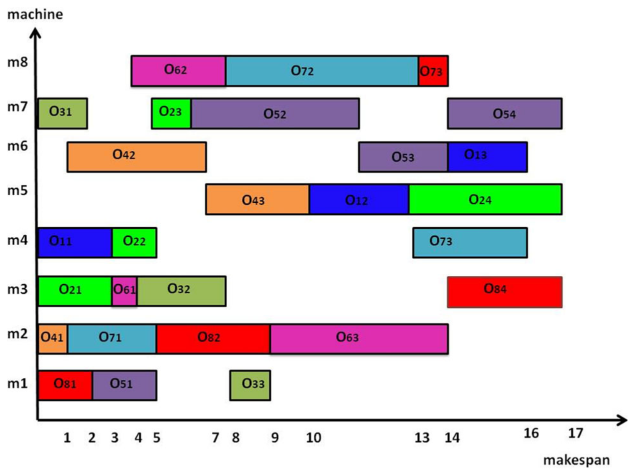

Fig. 6 The Gantt chart of instance 2 (8 jobs * 8 machines)

ods, we have a proof here that an appropriate implementation of the PSO can lead to good results without any hybridization.

PSO results on brandimarte instances (Brandimarte 1993)

To better confirm and strongly validate our PSO approach, additional experiments are performed using a decent number of problem instances from the widely utilised Brandimarte benchmark (Brandimarte 1993). The obtained results are shown in Table 4 and compared with other methods from the litterature.

Where:

'TS' refers to (Brandimarte 1993), 'PSO' refers to (Girish and Jawahar 2009), 'PSO + TS'refers to (Li et al. 2010) and 'MATSLO' refers to (Henchiri and Enngirou 2013)

Mk1, Mk2,.., Mk10 are the first ten instances proposed by (Brandimarte 1993). For these instances, the number of jobs varies from 10 to 20, the number of machines from 6 to 15 and the number of operations from 58 to 232.

As it is reported in Table 5, our PSO can achieve better results for Mk2, Mk5, Mk6, Mk8 and Mk9 instances. However, for the remaining Mk1, Mk3, Mk4, Mk7 and MK10 instances the obtained performance of our algorithm is comparable to the best achievable result from any of the other reference methods. This is due the premature convergence of the PSO. In addition, and in comparison with the cited references, our approach has a lower computational complexity and is much more easer to implement. this in line with our objective to not only get the best makespan, but to make also the approach as simpler as possible for future implementation on embedded systems capable of making real time decisions according to the state of resources and any unplanned or unforeseen events.

Table 3 Comparison of results on problem 8 * 8

Algorithm	Makespan
Our PSO	17
Classic GA	16
AL+CGA	16
PSO+SA	15
PSO+TS	15

Table 4 Comparison of results on Bandimarte instances

Instance	TS	PSO	PSO+TS	MATSLO	Our PSO
MK1	42	40	40	40	41
MK2	32	27	27	32	**26**
MK3	211	204	204	207	207
Mk4	81	62	63	67	65
Mk5	186	178	173	188	**171**
Mk6	86	78	65	85	**61**
Mk7	157	147	145	154	173
Mk8	523	523	523	523	**523**
Mk9	369	341	331	437	**307**
Mk10	296	252	223	380	312

Better results compared to other methods are in bold

Table 5 Comparison between PSO, MAPSO1 and MAPSO2

Instance	Makespane	PSO CPU time (s)	MAPSO1 CPU time (s)	MAPSO2 CPU time (ms)
4 * 5	11	0.353	0.332	17.849
10 * 10	8	7.316	7.307	37.816
8 * 8	17	5.698	5.589	26.984

Experimental results for the distributed approaches

To illustrate the performance of the proposed Multi-agent system *MAPSO1* and *MAPSO2* models, test experiments are carried out on problem instances from (Jia et al. 2007; Motaghedi-larijani et al. 2010). The result performance parameters are concentrated on evaluating the makespan with the CPU processing time of each approach using the same configuration. The obtained results are shown in Table 5. The proposed PSO, *MAPSO1* and *MAPSO1* models are run 20 times for each problem and for all these cases the same makespan solution is obtained for all the PSO, *MAPSO1* and *MAPSO2* approaches, but with a specific CPU processing time for each case.

As it is shown from Table 5, the MAPSO1 is faster than the classical PSO for all the problem instances studied. This is due to the parallel distribution of the PSO processing to the different Execute Agents such that each sub-population is evolving separately.

pgFor the proposed *MAPSO2*, the overall processing time is longer than the Centralized PSO as this is including the wait times required for all the Execute Agents (EA) to synchronize their activities. However, the total job is divided among all the EA agents and in this case each agent will be individually performing lower processing. For example, the Synchronization Agent SA divides the swarm received from the BA into two sub-populations and sends them to the different EA agents for processing. The size of sub-population noted sub-swarm of each agent is equals to the half of the swarm size and consequently lower processing load is required to be performed by each agent element. This is a very important feature in case we targeting a PSO implementation on multiple embedded systems having each a limited memory and processing power resources. In addition, the advantage of MAPSO2 model is that all agents are integrated on the optimization phase, with also the contribution of the migration policy that is a recently used method to guide the search into new areas of search space in order to maximize the chances for the system to converge more easily towards the best particle solution.

Conclusions

This paper presents a particle swarm optimization algorithm to solve the FJSP problem. The experimental results indicate that the algorithm is very effective and can play a definite part in directing real production. However, the industry and manufacturing systems are characterized by disruptions, unplanned events and unforeseen incidents that can happen at any time, such as machine breakdown, maintenance, break connection, etc. Our aim in this paper is not only to implement a fast solution, but an efficient algorithm that can be easily reconfigured for an embedded system suitable for an unpredictable environment. For that aim, we distribute the PSO into a multi agent system (MAS) to decentralize decisions and to make such that each entity will participate in the resolution of the whole problem. Two MAS approaches that distribute the PSO in a multi agent system are proposed. The two MAS architectures and the centralized PSO are tested and compared using reference instances. The objective is to choose the best MAPSO architecture of that can be easily reconfigured for an embedded system capable of making real time decisions according to the state of resources and to unplanned events. An interesting direction for future researches is to develop an embedded MAPSO in which each entity will participate in the resolution of the problem. As the problem resolution will be distributed on multiple embedded systems, it is important to control the energy consumption while guaranteeing the quality of the solution.

References

Asadzadeh, L., & Zamanifar, K. (2010). An agent-based parallel approach for the job shop scheduling problem with genetic algorithms. *Mathematical and Computer Modelling*, *52*, 1957–1965.

Azzouz, A., Ennigrou, M., Jlifi, B., & Ghédira, K. (2012). Combining tabu search and genetic algorithm in a multi-agent system for solving flexible job shop problem. In *Eleventh Mexican international conference on artificial intelligence*, pp. 83–88.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research.*, *41*, 157–183.

Binh, H., & Cing, T. (2008). Solving multiple-objective flexible job shop problems by evolution and local search. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *38*(5), 674–685.

Bastos-Filho, C., Carvalho, D., Figueiredo, E., & Miranda, P. (2009). Dynamic clan particle swarm optimization. In *Ninth international conference on intelligent systems design and applications*, pp. 249–254.

Clerc, M., & Kennedy, J. (2002). The particle swarm explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, *6*(1), 58–73.

Clerc, M. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New optimization techniques in engineering* (pp. 219–239). Berlin: Springer.

Chen, Y., Li, Z., & Wang, Z. (2004). Multi-agent-based genetic algorithm for JSSP. In *Proceedings of the third international conference on machine learning and cybernetics 2004*, pp. 267–270.

Ennigrou, M., & Ghédira, K. (2004). Approche multi-agents basée sur la recherche tabou pour le job shop flexible. In *14ème congrès francophone de reconnaissance des formes et intelligence artificielle*, Toulouse, France.

Ennigrou, M., & Ghédira, K. (2008). New local diversification techniques for the flexible job shop problem with a multi-agent approach. *Journal of Autonomous Agents and Multi- Agent Systems*, *17*(2), 270–287.

Emin, M. (2012). Coordinating metaheuristic agents with swarm intelligence. *Journal of Intelligent Manufacturing*, *23*(4), 991–999. doi:10.1007/s10845-010-0435-y.

Fernández, M., & Raupp, F. (2014). A Newton-based heuristic algorithm for multi-objective flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, pp. 1–8. doi:10.1007/s10845-014-0872-0.

Girish, B., & Jawahar, N. (2009). A particle swarm optimization algorithm for flexible job shop scheduling problem. In *5th annual IEEE conference on automation science and engineering*, Bangalore, India, pp. 298–303.

Gao, K. Z., Suganthan, P. N., Pan, Q. K., Chua, T. J., Cai, T. X., & Chong, C. S. (2014). Discrete harmony search algorithm for flexible job shop scheduling problem with multiple objectives. *Journal of Intelligent Manufacturing*, pp. 1–12.

Gen, M., & Lin, L. (2014). Multi-objective evolutionary algorithm for manufacturing scheduling problems: state-of-the-art survey. *Journal of Intelligent Manufacturing*, *25*(5), 849–866. doi:10.1007/s5-013-0804-4.

Henchiri, A., & Enngirou, M. (2013). Particle swarm optimization combined with tabu search in a multi-agent model for flexible job shop problem. *Computer Science*, *7929*, 385–394.

Jia, Z., Chen, Z., & Tang, J. (2007). An improved particle swarm optimization for multi-objective flexible job-shop scheduling problem. In: *Proceedings of 2007 IEEE international conference on grey systems and intelligent services*, Nanjing, China, pp. 1587–1592.

Jun-jie, B., Yi-guang, G., Ning-sheng, W., & Dun-bing, T. (2009). An improved PSO algorithm for flexible job shop scheduling with lot-splitting. In *Intelligent systems and applications, 2009. ISA 2009. International workshop on. IEEE*, 2009. pp. 1–5.

Jalilvand-Nejad, A., & Fattahi, P. (2013). A mathematical model and genetic algorithm to cyclic flexible job shop scheduling problem. *Journal of Intelligent Manufacturing*, *1–14*, 1–14. doi:10.1007/s10845-013-0841-z.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, pp. 1942–1948.

Kennedy, J. (1999). Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In *Evolutionary computation, vol. 3*.

Kacem, I., Hammadi, S., & Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions Systems, Man, and Cybernetics, Part C: Applications and Reviews*, *32*(1), 1–13.

Kennedy, J., & Mendes, R. (2003). Neighborhood topologies in fully-informed and best-of-neighborhood particle swarms. In *IEEE workshop on soft computing in industrial applications*, pp. 45–50.

Liu, H., Abraham, A., & Grosan, C. (2007). A novel variable neighborhood particle swarm optimization for multi-objective flexible job-shop scheduling problems. *Digital Information Management*, *1*, 138–145.

Li, J., Pan, Q., Xie, S., Jia, B., & Wang, Y. (2010). A hybrid particle swarm optimization and tabu search algorithm for flexible job-shop scheduling problem. *International Journal of Computer Theory and Engineering*, *2*(2), 1793–8201.

Lai, P., & Wu, H. (2013). Using heuristic algorithms to solve the scheduling problems with job-dependent and machine-dependent learning effects. *Journal of Intelligent Manufacturing*. doi:10.1007/s10845-010-0435-y.

Motaghedi-larijani, A., Sabri-laghaie, K., & Heydari, M. (2010). Solving flexible job shop scheduling with multi objective approach. *International Journal of Industrial Engineering & Production Research, 21*(4), 197–209.

Meeran, S., & Morshed, M. S. (2012). A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study. *Journal of Intelligent Manufacturing*, *23*, 1063–1078. doi:10.1007/s10845-011-0520-x.

Nouiri, M., Jemai, A., Bekrar, A., Niar, S., & Ammari, A. C. (2013). An effective particle swarm optimization to solve flexible job shop scheduling problem. In *Presented at the 5th IESM conference*. Morocco: Rabat.

Pezzellaa, F., Morgantia, G., & Ciaschettib, G. (2008). Genetic algorithm for the flexible job-shop scheduling. *Computers & Operations Research*, *35*, 3202–3212.

Trelea, I. (2003). The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, *85*, 317–325.

Tanga, J., Zhanga, G., Lina, B., & Zhang, B. (2011). A hybrid algorithm for flexible job-shop scheduling problem. *Procedia Engineering*, *15*, 3678–3683.

Venter, G., & Sobieszczanski-Sobieski, J. (2006). Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. *Journal of Aerospace Computing, Information, and Communication*, *3*(3), 123–137.

Wang, Y., & Xiang, Q. (2008). Particle swarms with dynamic ring topology. *Evolutionary Computation*, *3*, 419–423.

Wei, X., & Dongmei, F. (2012). Multi-agent system for flexible job-shop scheduling problem based on human immune system. In *Proceedings of the 31st Chinese control conference*, Hefei, China, pp. 2476–2480.

Xia, W., & Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, *48*, 409–425.

Zambrano rey, G., Bekrar, A., Prabhu, V., & Trentesaux, D. (2014). Coupling a genetic algorithm with the distributed arrival-time control for the JIT dynamic scheduling of flexible job-shops. *International Journal of Production Research*, *52*(12), 3688–3709.

Zhang, G., Shao, X., Li, P., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering*, *56*, 1309–1318.

Zhang, G., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, *38*, 3563–3573.