Chapter 5

Multi-agent based Particle Swarm Optimization for JSSP

5.1 Introduction

Modelling of social learning which has attracted interest in recent years is drawn from a swarm metaphor. An intelligent swarm is defined as a population of interacting individuals that optimizes a function or goal by collectively adapting to the environment. Swarm intelligence (SI) has been defined in (Bonabeau et al, 1999) as the emergent collective intelligence of a group of simple agents. SI systems are typically made up of a population of simple agents interacting locally with each other and with their environment. Local interactions between the agents lead to the emergence of complex global behaviour. Although the systems can exhibit remarkable coordination of activities between individuals, this coordination does not stem from a center of control or a directed intelligence, rather it is self-organizing. The swarm intelligence rests on five basic principles (Kennedy & Eberhart, 1995):
• **Proximity principle:** The population should be able to carry out simple space and time computations.

• **Quality principle:** The population should be able to respond to quality factors in the environment.

• **Diverse response principle:** The population should not commit its activities along excessively narrow channels.

• **Stability principle:** The population should not change its mode of behavior every time the environment changes.

• **Adapability principle:** The population must be able to change behavior mode when it is worth the computational price.

Two popular approaches of swarm intelligence (SI) are those inspired by studies of social insects such as ant colonies, and those inspired by studies of the flocking behavior of birds and fish such as particle swarm optimization. Particle swarm optimization (PSO) is an population-based stochastic heuristic technique that simulates the flight of a flock of birds or fish. The PSO was proposed by (Kennedy & Eberhart, 1995). A PSO algorithm mimics the behaviour of flying birds and their means of information exchange to solve optimization problems. Information is socially shared among individuals to direct the search towards the best position in the search space. There are many similar points between PSO and GA. Both algorithms start with a group of a randomly generated population; both have fitness values to evaluate the population. Both update the population and search for the optimum with random techniques. Both systems do not guarantee finding optimal solutions. However, the PSO technique does not use genetic operators like crossover and mutation. The information sharing mechanism in the PSO is significantly different from
the GA. In the GA, chromosomes share information with each other. So, the whole population moves like one group towards an optimal area. In the PSO, only global or local best particle gives out the information to others. The PSO is popular in research due to its following attractive features:

- **Ease of implementation:** Implementation of the PSO is done using basic mathematical operations.

- **Flexibility:** Often no major adjustments have to be made when adapting the PSO to a new problem.

- **Robustness:** The solutions of the PSO are almost independent of the initialization of the swarm. Additionally, very few parameters need to be tuned to obtain quality solutions.

- **Possibility to combine discrete and continuous variables:** Some authors addressed this possibility as a special feature of the PSO (Sensarma et al., 2002). There are potential dangers associated with the relaxation process necessary for handling the discrete variables (Abido, 2002). Simple round-off calculations may lead to significant errors.

- Easily tune and balance between local and global exploration.

- **Parallelism:** The PSO is inherently well suited for parallel computing. The swarm population can be divided between many processors to reduce computation time.

Originally, PSO was designed for optimization problems with continuous search space (Eberhart and Kennedy, 1995). Recently, PSO has been applied to solve discrete combinatorial optimization and scheduling problems such as Travelling Salesman Problem (Pang et al., 2004), Flow-
Ge et al., (2007) proposed hybrid evolutionary algorithm (HEA) that combines the PSO and the SA techniques to solve JSSP. The hybridization of the PSO with the SA makes it more effective than the standard PSO algorithm.

In this chapter the standard PSO algorithm has been explained and a hybrid model of the PSO with the local search procedure has been proposed for the JSSP and tested. The proposed model is called multi-agent based particle swarm optimization.

5.2 Standard PSO Algorithm

PSO algorithm is similar to the evolutionary algorithm in that the system is initialized with a population of random solutions called a swarm. An individual which is a potential solution is called a particle. The particle flies in a D-dimensional search space with a velocity that is dynamically adjusted according to the flying experience of the individual and its colleagues. Suppose that the position of the $i^{th}$ particle of the swarm is represented by a D-dimensional vector, $X_i = (x_{i1}, x_{i2}, \ldots, x_{id})$. The velocity of this particle is represented by another D-dimensional vector, $V_i = (v_{i1}, v_{i2}, \ldots, v_{id})$. All the particles in the swarm are measured by a fitness function. The particle keeps its best position attained so far in the vector $Pb_i = (pb_{i1}, pb_{i2}, \ldots, pb_{id})$. The best fitness value of the particle $i$ is called the personal best value and is denoted as $Pbest_i$. The swarm keeps the global best-so-far attained position by the whole swarm in the vector $Gb = (gb_1, gb_2, \ldots, gb_d)$. The global best-so-far fitness value in the swarm is denoted as $Gbest$. This kind of PSO is called global PSO. Sometimes a local best fitness value $Lbest_i$ in the neighbourhood of the particle $i$ is used instead of
the \textit{Gbest}. In this case the PSO is called the local PSO. In the local PSO, the neighbourhood is defined for an individual particle as a subset of particles it is able to communicate with. A number of neighbourhood configurations have been discussed in (Kennedy, 1999) and (Kennedy \& Mendes, 2002). The neighbourhood structure could be a ring, wheel, or fully connected graph. Particles in the same neighbourhood communicate with each other by exchanging information for moving towards better positions. The flow of information through the swarm depends on the neighbourhood structure. Figure 5.1 presents three neighbourhood structures for PSO. In a highly connected neighbourhood structure, the information about the best particle in the swarm is quickly transmitted through the whole swarm. This means faster convergence, which implies a higher risk to converge to local minima. The PSO with fully-connected topology is equivalent to the global PSO. In this chapter, the global PSO has been used.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{neighbourhood_topologies.png}
\caption{Neighbourhood topologies}
\end{figure}

In the global PSO, particles update their velocities and positions by means of the following equations:
\[ v'_j = w \times v_{ij}^{t-1} + c_1 \times r_1 \times (pbest_j - x_{ij}^{t-1}) + c_2 \times r_2 \times (gbest - x_{ij}^{t-1}) \]  \hspace{1cm} (5.1)

\[ x'_{ij} = x_{ij}^{t-1} + v'_j \]  \hspace{1cm} (5.2)

where, \( t \) is the iteration index, \( j = 1, 2, ..., d \) is the dimension index, \( w \) is the inertia weight, \( c_1 \) and \( c_2 \) are the acceleration constants, and \( r_1 \) and \( r_2 \) are uniform random numbers between 0 and 1.

A standard PSO algorithm can be presented by the following pseudo-code:

\begin{verbatim}
Standard PSO
for all particle i do
    Initialize position \( X_i \) randomly in the search space
end for
while termination criteria are not satisfied do
    for all particle i do
        Set \( Pbest_i \) as the best position found by the particle so far
    end for
    Set \( Gbest \) as the best position found by the swarm so far
    for all particle i do
        Update velocity according to Equation 5.1
        Update position according to Equation 5.2
    end for
end while
\end{verbatim}

\section{5.3 PSO parameters}

El-Gallad et al. (2002) have studied the various input parameters required for working the particle swarm optimizer. Some of these parameters are described below:
• **Population of the swarm:** This factor depends upon the problem being optimized. Smaller swarms may be more successful for some problems while larger ones may be useful for others. However, if the swarm size is too small it may result in convergence upon a local optimum while on the other hand very large swarms may increase computational time. Hence, balance need to be adjusted between the complexity of the algorithm and the risk of getting trapped in local optima by selecting a proper swarm size.

• **Number of iterations:** The uncertainty in the velocity update equation, which introduced by the stochastic factors, results in a global exploration of the search space. This makes it possible for the algorithm to arrive at the global optimum mostly likely only if the algorithm runs for a sufficiently large number of iterations. The use of the word sufficient is in itself indicative of the problem specific nature of this parameter. Indeed the permissible error margin, which strongly dictates the computational time, varies with the problem at hand.

• **Velocity of particles:** This factor determines the fineness with which the hyperspace under consideration is searched. If the value of this parameter is too high, then the particles may fly past the optimal solution and may even oscillate about a certain position. On the other hand if this value is too low, then the particles could get stuck at a local optimum. In order to circumvent this issue, an adaptive velocity technique may be applied. According to this approach, in the event that the solution found is oscillatory, the value of velocity is allowed to gradually decrease in a random fashion thereby helping the particles to get out of the oscillation and at the same time allowing
the swarm to explore new areas. The velocity of the particle in the Equation 5.1 is limited in the range \([v_{\text{max}}, v_{\text{min}}]\), where \(v_{\text{max}}\) and \(v_{\text{min}}\) are the maximum and minimum velocities allowed for the particle. The updating the location of the particle in the Equation 5.2 is to lead moving the particle toward compound vector of inertia component, cognition component and social component. By doing so, the opportunity for the particle to reach the optimal solution is increased. Figure 5.2 graphically shows the components of the compound vector of the Equation 5.1.

![Figure 5.2 Compound velocity vector of the PSO](image)

From the Figure 5.2 and the Equation 5.1, it is noticeable that a particle tries to maximize its fitness through three types of behaviours; the habitual behaviour, the cognitive behaviour and the social behaviour. The three behaviours are explained as follows:

- **The habitual behaviour** is represented by the first term of the Equation 5.1; \(w \times v_{i}^{t-1}\), where the particle trusts its own status at present location and provides a basic momentum.
The cognitive behaviour is represented by the second term of the Equation 5.1; \([c_1 \times r_1 \times (p_{bh} - x_{i}^{t-1})]\). This behaviour captures the previous local knowledge of the agent based on the best position \(P_{bh}\) of the particle obtained so far. Hence, the coefficient \(c_1\) is called the cognitive constant.

The social behaviour is represented by the third term of the Equation 5.1; \([c_2 \times r_2 \times (g_{b} - x_{i}^{t-1})]\). This behaviour computes the common global knowledge about the best position \(G_{b}\) obtained in the whole swarm. Therefore, the coefficient \(c_2\) is called the social constant.

The parameters setting of the PSO have a major influence on the behaviour of the algorithm. When only social values are used to update the particle velocities (\(c_1 = 0 \& c_2 \neq 0\)), the PSO algorithm converges rapidly to the best initial optimum found from the swarm and no exploration in own search regions is allowed to improve the solutions. In contrast, when \(c_1 \neq 0 \& c_2 = 0\), each particle in the swarm will explore around its best-found personal solution requiring a very large number of iterations to agree into a global solution, which is not necessarily the global optimum. Inertia weight \(w\) was introduced in (Shi and Eberhart, 1998) to balance the global exploitation and local exploration abilities of the particles. Exploration ability is related to the algorithm's tendency to explore new regions of the search space, while exploitation is the tendency to search a smaller region more thoroughly. A large value of \(w\) gives the particle high velocity which prevents particle from becoming trapped in local optima. A small value of \(w\) encourages the particle to exploit the same search space area. The inertia weight is usually set to be a large
value initially in order to achieve better global exploration, and gradually it is reduced for obtaining more refined solutions.

5.4 PSO variants

Originally, PSO algorithm is found by (Eberhart and Kennedy, 1995). Many developments have been made to the PSO to improve its performance. Some of the PSO variants are:

5.4.1 PSO with Constriction Coefficient

The constriction coefficient was introduced as an outcome of a theoretical analysis of swarm dynamics by Clerc (1999) and by Clerc and Kennedy (2002). Velocity update of the particle is constricted as followings:

\[ v'_j = \chi \times \left[ v_{j}^{t-1} + c_1 \times r_1 \times (p_{bj} - x_j^{t-1}) + c_2 \times r_2 \times (g_{bj} - x_j^{t-1}) \right] \tag{5.3} \]

where

\[ \chi = \frac{2}{2 - \varphi - \sqrt{\varphi^2 - 4 \times \varphi}} \tag{5.4} \]

\( \chi \) is a constriction factor, \( \varphi \) is acceleration rate \( (\varphi = c_1 + c_2) \), and \( r_1 \) and \( r_2 \) are the uniform random numbers in the range between 0 and 1. Convergence is guaranteed under the condition that \( \varphi > 4 \). The constriction factor improves the convergence of the particle over time by damping the oscillations once the particle is focused on the best point in an optimal region. The main disadvantage of this method is that the particles may follow wider cycles and may not converge when the individual best performance is far from the global best performance.
5.4.2 Fully Informed Particle Swarm

The Fully Informed Particle Swarm (FIPS) was first described by Mendes et al. (2004). The FIPS is based on the assumption that individuals are influenced by their whole neighborhood in a specific way. In the traditional particle swarm, a particle with k neighbors selects one (the best particle) to be a source of influence, and ignores the others. A Fully Informed Particle therefore has knowledge about the best solutions found so far by any neighbor and not just the best one. The velocity update in FIPS is as follows:

\[ v_i^t = \chi \times \left[ v_i^{t-1} + \frac{1}{N_i} \times \sum_{k=1}^{N_i} \phi \times r \times (p_{b_k}^{t-1} - x_i^{t-1}) \right] \] (5.5)

where \( \chi \) is a constriction factor, \( N_i \) is the number of neighbours of the particle \( i \), \( \phi \) is acceleration rate, and \( r \) the uniform random numbers in the range between 0 and 1, \( p_{b_k} \) is the personal best-so-far position of the particle \( k \) from the neighbourhood of the particle \( i \). The performance of FIPS is generally more dependent on the neighborhood topology.

5.4.3 Binary PSO

A binary PSO (BPSO) was first developed in (Kennedy and Eberhart, 1997) as a variant of PSO to solve some discrete binary optimisation problems. In the binary PSO, each particle is encoded as a binary string in the solution space. In other words, a particle flies in a search space restricted to zero and one. The concept of velocity loses its physical meaning and assumes the value of a probability instead. The speed of the particle must be constrained to the interval \([0, 1]\). A logistic sigmoid
transformation function can be used to limit the speed of the particle. The sigmoid function is given as follows:

\[ Sigmoid \quad (v^i_y) = \frac{1}{1 + e^{-v^i_y}} \] (5.6)

The update equation of the BPSO is made in three steps. First, the equation 5.1 is used to update the velocity of the particle. Second, the sigmoid function in the equation 5.6 is used to limit the velocity in the interval \([0, 1]\). Third, the new position of the particle is obtained as follows:

\[
\text{if } (\text{Rand} < Sigmoid(v^i_y)) \text{ then } x^i_y = 1 \text{ else } x^i_y = 0 \] (5.7)

where Rand is a random number drawn from a uniform distribution in the range \([0,1]\).

The concept of BPSO is different from the typical PSO, since the relevant variables are derived from the changes of probabilities. It is hard to identify the relation between the current status and the previous status of the particle. The selection of parameters, such as the inertia weight and the acceleration constants, is also problematic. The binary version of the Particle swarm optimization is suitable for problems that have discrete binary search spaces.

### 5.5 Proposed Multi-Agent based Particle Swarm Optimization for JSSP

The multi-agent based particle swarm optimization (MAPSO) has been proposed as a swarm of particles. The swarm in the MAPSO is represented as MAS, where each particle in the swarm is considered as an agent. Given
a JSSP of size $n \times m$, where $n$ is the number of jobs and $m$ is the number of machines. A schedule is described as a sequence of all operations of all the jobs. The sequence is represented as $S = [s_1, s_2, ... s_{nxm}]$, where each operation in the sequence is denoted with its job number; $s_k = i$ indicates an operation of the job $J_i$. This representation is equivalent to the OBR which has been described in the Section 4.5.3.

PSO cannot directly employ the particle position $X_i$ as a JSSP solution. For JSSP, a particle position vector $X_i$ is mapped into the scheduling Sequence $S_i$ and vice versa. The Operation-Particle Position Sequence (OPPS) from (Liu, 2007) is employed to represent the relation between the particle position vector and the operation sequence. A position $X_i = (x_{i1}, x_{i2}, ... x_{id})$ of a particle $P_i$ is mapped to a feasible solution of JSSP that is the operation sequence of all jobs $S_i = (s_1, s_2, ... s_{m \times n})$. Let $MapSeq()$ be the one-to-one function applied to each particle’s position to obtain the corresponding sequence as shown in Figure 5.3. The dimension of the particle is equal to number of operations in the sequence (i.e. $d = m \times n$). The sequence of all the operations is obtained when the components of particle position vector is sequenced in increasing order (i.e. $x_{ij} \leq x_{ij+1}$: $j = 1, 2, ... d-1$).

![Figure 5.3 Mapping between JSSP Space and PSO Space](image-url)
For example, suppose a JSSP with 3 jobs and 3 machines. Let an initial particle $i$ be set to a uniformly randomly generated position $X_i$ as given in the Table 5.1.

**Table 5.1** Initial particle before ordering

<table>
<thead>
<tr>
<th>Operations</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>3</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_{1,1}$</td>
<td>$x_{i1}$</td>
<td>$x_{i2}$</td>
<td>$x_{i3}$</td>
<td>$x_{i4}$</td>
<td>$x_{i5}$</td>
<td>$x_{i6}$</td>
<td>$x_{i7}$</td>
<td>$x_{i8}$</td>
<td>$x_{i9}$</td>
</tr>
<tr>
<td>$X_i$</td>
<td>2.1</td>
<td>3.8</td>
<td>1.5</td>
<td>2.7</td>
<td>2.2</td>
<td>1.1</td>
<td>1.4</td>
<td>4.3</td>
<td>3.5</td>
</tr>
</tbody>
</table>

The sequence is derived from the Table 5.2 after ordering the operations according to the increasing order of the components of the given position. The Table 5.2 shows the resulting sequence [2, 3, 1, 1, 2, 2, 3, 1, 3] which represents a feasible solution to the JSSP.

**Table 5.2** Initial particle after ordering

<table>
<thead>
<tr>
<th>Sequence</th>
<th>2</th>
<th>3</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O_{2,3}$</td>
<td>$x_{i6}$</td>
<td>$x_{i7}$</td>
<td>$x_{i3}$</td>
<td>$x_{i1}$</td>
<td>$x_{i5}$</td>
<td>$x_{i4}$</td>
<td>$x_{i9}$</td>
<td>$x_{i2}$</td>
<td>$x_{i8}$</td>
</tr>
<tr>
<td>$X_i$</td>
<td>1.1</td>
<td>1.4</td>
<td>1.5</td>
<td>2.1</td>
<td>2.2</td>
<td>2.7</td>
<td>3.5</td>
<td>3.8</td>
<td>4.3</td>
</tr>
</tbody>
</table>

It is widely observed that the PSO algorithm have good ability for global search, while the ability for fine search is relatively weak. That is to say, PSO algorithm can easily find the area of the solution space in which good solutions are located, while the procedure to find the good solutions is not equally easy for PSO algorithm. Thus, it is important for PSO algorithm to enhance local search ability. The self-learning method is applied in the proposed model in each of the iterations to all agents of the system to enhance the solution quality. The self-learning method corresponds to the proactive characteristic of the MAS. The self-learning method is applied by the local search procedure which is presented in
Section 4.8.5. The local search procedure alters the sequence of operations in the schedule to improve the schedule. For a JSSP of size n×m, consider the agent \( i \) which contains the sequence \( S_i = (s_1, s_2 \ldots s_{m\times n}) \) at the iteration \( t \). The proposed local search procedure when applied to the sequence \( S_i' \) yields an improved sequence \( S_i'' \). The local search procedure ensures that the makespan of the sequence \( S_i'' \) is less than or equal to the makespan of the sequence \( S_i' \). The procedure incorporates the self-learning capability of the agent. The complexity of the local search procedure is \( O(L^2) \), where \( L \) is the length of the sequence \( S_i' \). However, by improving the makespan of the sequence \( S_i' \) using the self-learning capability of each agent, the personal best position \( P_{best} \) may be altered which in turn may change the global best position \( G_{best} \). Due to improvements in \( P_{best} \) and \( G_{best} \) caused by the local search applied to particles, the convergence to a better solution is more guaranteed than the pure PSO algorithm. The flowchart of MAPSO is given in the Figure 5.4. The pseudo-code of the MAPSO algorithm is given as follow:

**MAPSO**

**Step1.** Swarm initialization:

// \( t \) is the iteration number.
// \( k \) is the number of particles.
\( t \leftarrow 1 \)
for \( 1 \leq i \leq k \) do
begin
    Initialize \( X_i \) \( 0 \) \( 2 \)
    Initialize \( V_i \) \(-2 \) \( 2 \)
    \( S_i = \text{MapSeq}(X_i) \)
    \( S_i.C_{\text{max}} = \text{Makespan}(S_i) \)
end for

**Step2.** Swarm Evaluation:

for \( 1 \leq i \leq k \) do
Find Pbest\textsubscript{i} for the particle i;
Find Gbest of the current swarm;

Step3. Swarm updating:
for 1\leq i \leq k do
begin
Update \textit{V}\textsubscript{i} using Equation 5.1;
Update \textit{X}\textsubscript{i} using Equation 5.2;
end for

Step4. Local update:
for 1\leq i \leq k do
begin
\textit{S}\textsubscript{i} = MapSeq(\textit{X}\textsubscript{i})
\textit{S}\textsubscript{i}.\textit{C_{max}} = Makespan(\textit{S}\textsubscript{i})
\hat{\textit{S}}\textsubscript{i} = LS\_Procedure (\textit{S}\textsubscript{i}, \textit{S}\textsubscript{i}.\textit{C_{max}})
end for

Step5. Finding the minimum solution in the swarm:
\textit{minC_{max}} = \min\{ \textit{S}\textsubscript{i}.\textit{C_{max}} : 1 \leq i \leq k \}

Step6. Termination:
If ((t = Max\_iterations) \textbf{or} (\textit{minC_{max}} = C'\_{max})) then end
\textit{t} \leftarrow \textit{t+1}
Loop to Step 2

In the first step of the MAPSO algorithm, a swarm of particles is created and initialized. The position vector of the particle is uniformly randomly initiated in the range [0, 2] using the function \textit{Initialize}. The velocity vector of the particle is uniformly randomly initiated in the range [-2, 2] using the function \textit{Initialize}. The position vectors of the particles mapped to the JSSP sequences (solutions) using the function \textit{MapSeq} where one particle is mapped to one JSSP sequence. The makespan of the sequence is calculated using the function \textit{Makespan(S)} where S in the JSSP sequence. In the second step of the MAPSO algorithm, each particle
Figure 5.4 Flow chart of the MAPSO

calculates the best so far personal position vector Pbest; i.e. the best makespan that the particle has attained so far. Then, the swarm calculates the best so far global position vector Gbest; i.e. the best makespan that the swarm has attained so far. In the third step of the MAPSO algorithm, the particles of the swarm update their position and velocity vectors by the equations 5.1 and 5.2 respectively. In the forth step of the MAPSO algorithm, each particle of the swarm applies LS_Procedure() on its JSSP sequence to improve its makespan. In the fifth step of the MAPSO algorithm, the minimum solution in the swarm is calculated. The termination criteria are examined in the sixth step and the algorithm stops
when the maximum iteration number is reached or the optimal makespan is attained, otherwise, the algorithm repeats from the Step 2. Both of $\text{Makespan}(S)$ and $\text{LS}_\text{Procedure}(S, C_{\text{max}})$ have been presented in Section 4.8.5.

5.6 Implementation and Results

The proposed MAPSO algorithm is implemented using the agent-based simulation toolkit REPAST. The computing system for the experiments was a computer system which has the configurations: Intel C2D 1.8 GHz processor, DDR2 2.5 GB RAM, and OS 64-bit Windows Vista.

5.6.1 Parameter Tuning

Empirical experiments have been done to tune the parameters of the MAPSO on the well-known FT10 benchmark instance of the JSSP. Cognitive and social constants $c_1$ and $c_2$ have been chosen in the range $[0.5,2]$. The inertia weight $w$ is initialized to 1 and is decremented in each iteration during the run of the algorithm by the factor of $0.98$ (i.e. $w_{t+1} = w_t \times 0.98$). The minimum velocity is set to -2, and the maximum velocity is set to 2. Table 5.3 shows the average minimum makespans (Avg) and the average number of iterations (Avlitr) for some of the parameter sets for $c_1$ and $c_2$. MAPSO model has been executed 50 times for each of the parameter sets with the maximum number of iterations being 500.
Table 5.3 Parameters tuning on FT10: $c_1$ and $c_2$

<table>
<thead>
<tr>
<th>parameter sets</th>
<th>Avg</th>
<th>AvItr</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1 = 0.5, c_2 = 0.5$</td>
<td>930</td>
<td>74.82</td>
</tr>
<tr>
<td>$c_1 = 1, c_2 = 1$</td>
<td>930</td>
<td>70.33</td>
</tr>
<tr>
<td>$c_1 = 1.5, c_2 = 1.5$</td>
<td>930</td>
<td>77.96</td>
</tr>
<tr>
<td>$c_1 = 1, c_2 = 2$</td>
<td>930</td>
<td>103.66</td>
</tr>
<tr>
<td>$c_1 = 2, c_2 = 1$</td>
<td>930</td>
<td>77.43</td>
</tr>
<tr>
<td>$c_1 = 2, c_2 = 2$</td>
<td>930</td>
<td>67.13</td>
</tr>
</tbody>
</table>

In Table 5.3, the average minimum makespan is 930 for all the parameter sets. This is due to the efficiency of the MAPSO in solving the FT10 instance where the MAPSO algorithm reaches the best known solution for FT10 (i.e. 930) in every run. However, the average number of iterations (AvItr) differs for each parameter set due to the stochastic behavior of the algorithm. The parameter set ($c_1 = 2, c_2 = 2$) is chosen from the Table 5.3 to be used henceforth, since the MAPSO algorithm achieves the least average number of iterations with this set.

5.6.2 Experimental Results

Eleven JSSP benchmark instances (FT06, FT10, FT20, LA01, LA06, LA11, LA16, LA 21, LA26, LA31, LA36) from OR Library have been considered for the experiment. The MAPSO has been executed 50 times (runs) on each of the JSSP benchmark instances. The parameters settings for the MAPSO based on the experiments in Section 5.6.1 are given in the Table 5.4. It has been noticed from the experiments that there is no further improvement in the minimum makespan after 500 iterations. Therefore, the maximum number of iterations is set as 500.
### Table 5.4 Parameters settings for MAPSO

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum iterations</td>
<td>500</td>
</tr>
<tr>
<td>Swarm size</td>
<td>100</td>
</tr>
<tr>
<td>Cognitive acceleration constant ($c_1$)</td>
<td>2</td>
</tr>
<tr>
<td>Social acceleration constant ($c_2$)</td>
<td>2</td>
</tr>
<tr>
<td>Inertia weight ($w_0$)</td>
<td>$0.98 \times w_{t-1}$</td>
</tr>
</tbody>
</table>

Table 5.5 shows the results of the MAPSO in terms of Best, Itr, Avg, and AvItr. In Table 5.5, Ins is the JSSP benchmark instances, Bk is the best known makespan from the literature, Best is the best minimum makespan obtained from the 50 runs of the algorithm, Itr is the iteration number in which the algorithm attains the best minimum makespan, Avg is the average minimum makespan obtained from all the 50 runs of the algorithm, and AvItr is the average iteration number to attain the best known makespan from all the 50 runs of the algorithm.

### Table 5.5 Experimental results of MAPSO

<table>
<thead>
<tr>
<th>Ins</th>
<th>Bk</th>
<th>MAPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best</td>
</tr>
<tr>
<td>FT06</td>
<td>55</td>
<td>55</td>
</tr>
<tr>
<td>FT10</td>
<td>930</td>
<td>930</td>
</tr>
<tr>
<td>FT20</td>
<td>1165</td>
<td>1165</td>
</tr>
<tr>
<td>LA01</td>
<td>666</td>
<td>666</td>
</tr>
<tr>
<td>LA06</td>
<td>926</td>
<td>926</td>
</tr>
<tr>
<td>LA11</td>
<td>1222</td>
<td>1222</td>
</tr>
<tr>
<td>LA16</td>
<td>945</td>
<td>945</td>
</tr>
<tr>
<td>LA21</td>
<td>1046</td>
<td>1046</td>
</tr>
<tr>
<td>LA26</td>
<td>1218</td>
<td>1218</td>
</tr>
<tr>
<td>LA31</td>
<td>1784</td>
<td>1784</td>
</tr>
<tr>
<td>LA36</td>
<td>1268</td>
<td>1268</td>
</tr>
</tbody>
</table>
The MAPSO have been compared with the PSO (Liu, 2007) and the HEA (Ge et al., 2007) in terms of the best minimum makespan obtained from the 50 runs of the algorithm (i.e. Best). In addition, The MAPSO have been compared with the HEA in terms of the iteration number in which the algorithm attains the best minimum makespan (i.e. Itr). The HEA is a hybrid evolutionary algorithm based on the PSO and the simulated annealing approaches to solve the JSSP. Table 5.6 shows the results of the PSO, the HEA, and the MAPSO.

**Table 5.6 Experimental results of PSO, HEA, and MAPSO.**

<table>
<thead>
<tr>
<th>Ins</th>
<th>Bk</th>
<th>PSO</th>
<th>HEA</th>
<th>MAPSO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Best</td>
<td>Best</td>
<td>Itr</td>
</tr>
<tr>
<td>FT06</td>
<td>55</td>
<td>55</td>
<td>55</td>
<td>4</td>
</tr>
<tr>
<td>FT10</td>
<td>930</td>
<td>945</td>
<td>930</td>
<td>19688</td>
</tr>
<tr>
<td>FT20</td>
<td>1165</td>
<td>1169</td>
<td>1169</td>
<td>25802</td>
</tr>
<tr>
<td>LA01</td>
<td>666</td>
<td>666</td>
<td>666</td>
<td>27</td>
</tr>
<tr>
<td>LA06</td>
<td>926</td>
<td>926</td>
<td>926</td>
<td>19</td>
</tr>
<tr>
<td>LA11</td>
<td>1222</td>
<td>1222</td>
<td>1222</td>
<td>10</td>
</tr>
<tr>
<td>LA16</td>
<td>945</td>
<td>949</td>
<td>945</td>
<td>3606</td>
</tr>
<tr>
<td>LA21</td>
<td>1046</td>
<td>1140</td>
<td>1046</td>
<td>90076</td>
</tr>
<tr>
<td>LA26</td>
<td>1218</td>
<td>1281</td>
<td>1218</td>
<td>238099</td>
</tr>
<tr>
<td>LA31</td>
<td>1784</td>
<td>1812</td>
<td>1784</td>
<td>3217</td>
</tr>
<tr>
<td>LA36</td>
<td>1268</td>
<td>1321</td>
<td>1287</td>
<td>98670</td>
</tr>
</tbody>
</table>

Following all the observations from Table 5.6:

- For the instance **FT06**, the PSO, the HEA, and the MAPSO attain the best known makespan (i.e. 55). However, the HEA attains the best known makespan in 4 iterations while the MAPSO attains the best known makespan in 1 iteration.
- For the instance **FT10**, the MAPSO and the HEA attain the best known makespan (i.e. 930) while the PSO fails to attain the best known makespan. However, the HEA attains the best known makespan in 19688 iterations while the MAPSO attains the best known makespan in 59 iterations.

- For the instance **FT20**, the MAPSO attains the best known makespan (i.e. 1165), while the PSO and the HEA fail to attain the best known makespan. The HEA attains the best known makespan in 25802 iterations while the MAPSO attains the best known makespan in 102 iterations.

- For the instance **LA01**, the PSO, the HEA, and the MAPSO attain the best known makespan (i.e. 666). However, the HEA attains the best known makespan in 27 iterations while the MAPSO attains the best known makespan in 1 iteration.

- For the instance **LA06**, the PSO, the HEA, and the MAPSO attain the best known makespan (i.e. 926). However, the HEA attains the best known makespan in 19 iterations while the MAPSO attains the best known makespan in 1 iteration.

- For the instance **LA11**, the PSO, the HEA, and the MAPSO attain the best known makespan (i.e. 1222). However, the HEA attains the best known makespan in 10 iterations while the MAPSO attains the best known makespan in 1 iteration.

- For the instance **LA16**, the PSO fails to attain the best known makespan, while the HEA and the MAPSO attain the best known makespan (i.e. 945). However, the HEA attains the best known makespan in 111 iterations while the MAPSO attains the best known makespan in 1 iteration.
makespan in 3606 iterations while the MAPSO attains the best known makespan in 52 iterations.

- For the instance **LA21**, the PSO fails to attain the best known makespan while the HEA and the MAPSO attain the best known makespan (i.e. 1046). However, the HEA attains the best known makespan in 90076 iterations while the MAPSO attains the best known makespan in 217 iterations.

- For the instance **LA26**, the PSO fails to attain the best known makespan while the HEA and the MAPSO attain the best known makespan (i.e. 1218). However, the HEA attains the best known makespan in 238099 iterations while the MAPSO attains the best known makespan in 37 iterations.

- For the instance **LA31**, the PSO fails to attain the best known makespan while the HEA and the MAPSO attain the best known makespan (i.e. 1784). However, the HEA attains the best known makespan in 3217 iterations while the MAPSO attains the best known makespan in 3 iterations.

- For the instance **LA36**, the MAPSO attains the best known makespan (i.e. 1268), while the PSO and the HEA fail to attain the best known makespan. The HEA attains the minimum makespan (i.e. 1287) in 98670 iterations while the MAPSO attains the best known makespan in 180 iterations.

In summary, The MAPSO algorithm attains the best known makespans for the eleven JSSP benchmark instances. PSO algorithm attains the best known makespans on four JSSP benchmark instances: FT06, LA01, LA06, and LA11 and fails on other JSSP benchmark
instances. Therefore, the proposed MAPSO model overcomes the PSO on the benchmark instances FT10, FT20, LA16, LA21, LA26, LA31, and LA36. The HEA attains the best known makespans on nine JSSP benchmark instances and fails on The HEA fails to attain the best known makespans on FT20 and LA36. Therefore, the proposed MAPSO model overcomes the HEA on the benchmark instances FT20 and LA36. Also, the MAPSO attains the best known makespans on the eleven JSSP benchmark instances in far less iterations than the HEA.

Figure 5.5 displays the implementation of MAPSO in REPAST for the instance LA26, where the minimum makespan was 1218 after 56 iterations. This minimum makespan is equal to the best known makespan for this instance. The output window of REPAST shows the current best makespan, average makespan, and the current generation number during the run of the MAPSO model.

### 5.7 CONCLUSIONS

A multi-agent based particle swarm model has been developed to solve traditional JSSP. The MAPSO model is integrated with a proposed local search method to implement the self-learning aspect of agents in order to enhance the quality of the solutions. The MAPSO algorithm have been tested on eleven standard JSSP instances from the OR-Library. The MAPSO have been compared with the PSO and the HEA. The experiments indicate that the MAPSO algorithm outperforms both the PSO and the HEA in terms of attaining the best known makespan and the number of iterations needed to obtain the best known makespan for the experimented
JSSP benchmark instances. The MAPSO algorithm has been published as a part of the paper (Balid and Minz, 2008b).

Figure 5.5 Implementation of MAPSO in the REPAST