10.24425/acs.2019.129378

*Archives of Control Sciences*
*Volume* 29(LXV), 2019
*No.* 2, *pages 205–226*

# Cellular particle swarm optimization with a simple adaptive local search strategy for the permutation flow shop scheduling problem

JUAN C. SECK-TUOH-MORA, JOSELITO MEDINA-MARIN, ERICK S. MARTINEZ-GOMEZ, EVA S. HERNANDEZ-GRESS, NORBERTO HERNANDEZ-ROMERO and VALERIA VOLPI-LEON

Permutation flow shop scheduling problem deals with the production planning of a number of jobs processed by a set of machines in the same order. Several metaheuristics have been proposed for minimizing the makespan of this problem. Taking as basis the previous Alternate Two-Phase PSO (ATPPSO) method and the neighborhood concepts of the Cellular PSO algorithm proposed for continuous problems, this paper proposes the improvement of ATPPSO with a simple adaptive local search strategy (called CAPSO-SALS) to enhance its performance. CAPSO-SALS keeps the simplicity of ATPPSO and boosts the local search based on a neighborhood for every solution. Neighbors are produced by interchanges or insertions of jobs which are selected by a linear roulette scheme depending of the makespan of the best personal positions. The performance of CAPSO-SALS is evaluated using the 12 different sets of Taillard's benchmark problems and then is contrasted with the original and another previous enhancement of the ATPPSO algorithm. Finally, CAPSO-SALS is compared as well with other ten classic and state-of-art metaheuristics, obtaining satisfactory results.

**Key words:** flow shop, particle swarm optimization, local search strategy, hybrid search method, cellular automata, scheduling

## 1. Introduction

Flow Shop Scheduling Problem (FSSP) has represented a relevant research area for over sixty years due to its simplicity and combinatorial nature [14], which makes the problem a common place to prove new algorithms for discrete optimization. The FSSP can be easily understood and implemented in a computer program to be analyzed. However, since its discrete nature, there is a combinatorial explosion of possible solutions for a linear increment in the number of jobs

J. Seck-Tuoh-Mora (corresponding author), J. Medina-Marin, E.S. Martinez-Gomez, E.S. Hernandez-Gress, N. Hernandez-Romero, and V. Volpi-Leon are with Engineering Department, Autonomous University of Hidalgo State, Carr. Pachuca-Tulancingo, Col. Carboneras, Mineral de la Reforma, Hidalgo 42184, Mexico. E-mails: {jseck, jmedina}@uaeh.edu.mx, ericksmage@gmail.com, {nhromero, volpi}@uaeh.edu.mx

This work was partially supported by National Council for Science and Technology (CONACYT) with project number CB-2014-237323.

Received 23.05.2018.

to be scheduled. Thus, FSSP is relevant for theoretical reasons as a well-known case of an NP-complete problem. Besides, a large number of automated manufacturing systems follow the behavior of a flow shop. Therefore, research in this area and variations of the FSSP are important to control and optimize current manufacturing systems [2].

The FSSP consists of finding the optimal order of $n$ jobs to be processed by $m$ machines. Every job has the same sequence of operations passing from machine 1 up to machine $m$. All jobs are independent and available at time 0, each machine can only process on job at the same time, and preemption, failure and set-up times are not considered. The most common objective in a FSSP is to minimize the completion time of the last job in the last machine, better known as makespan $C_max$ [21, 26]. Many methods have ben proposed to solve the FSSP. Exact methods are limited to be applied for FSSPs with a few number of jobs and machines because of the complexity in the number of possible solutions and high calculation time. In recent years, many metaheuristic methods have been published in order to solve larger FSSP instances with hundreds of jobs and dozens of machines. Besides, there is a trending on proposing and implementing new algorithms able to compute high quality solutions for large FSSP instances in a reasonable time.

One of the most popular and applied metaheuristcs is the particle swarm optimization (PSO) algorithm. PSO is an optimization method that simplifies the local and global principles observed in the behavior of different groups of animals (like birds of fish) to solve complex problems (food search or escape from predators). The PSO algorithm is implemented using very simple rules that update velocity and position of solutions (or particles) based on uncomplicated linear equations [8]. PSO is very popular nowadays because many researchers have worked with it in order to improve its exploration and exploitation capabilities to solve complex and particular optimization problems [11, 16] and [1].

One adaptation from the original PSO is the one where the concept of cellular automata (CA) neighborhood is taken to enhance the local search of particles. This modification is known as Cellular Particle Swarm Optimization (CPSO) [28]. CPSO has been applied in different practical problems as the optimization of a milling process [10] and the layout of truss structures [12]. In the theoretical and the previous practical cases, CPSO has demonstrated stronger robustness and obtained better optimal values than original PSO. Other result obtained with an hybrid PSO and CA algorithm is presented in [15] for the optimal design of adaptive infinite impulse response filters. In that work, the suggested algorithm is different from CPSO in the sense of using a differential evolution method to produce the neighborhood of every particle instead of a random search. The most common application of PSO is in continuos problems, but discrete versions have been proposed as well to optimize combinatorial problems. In the latter case, the equations of PSO are modified to work with discrete elements. However, up to our knowledge, a hybrid process of PSO and CA has not yet been investigated for the optimization of FSSPs.

Several papers have been delivered to solve the FSSP based on PSO algorithm [5,17,19,24,25,31,33] obtaining good results, but these algorithms have two main drawbacks: these are still suffering from the problem of premature convergence and easily trapped into local optimum, or are complemented with complicated and complex local search methods. In order to refrain the decrement of swarm diversity conserving the simplicity of the method, a two point crossover operator and a repulsive process is introduced in the Alternate Two Phases PSO (ATPPSO) algorithm, which can make the solutions fly toward some promising areas [34]. Nevertheless, ATPPSO has a drawback: the lack of local search ability.

Based on the above consideration, the motivation of the current study is the application of a novel hybrid optimization technique based on the combination of ATPPSO, CA and a simple local search (CAPSO-SALS) in the makespan minimization for FSSPs. In particular, the solutions in the ATPPSO take a fixed number of neighbors generated by interchange or insertion of jobs selected by a linear operator, in order to improve their position. This hybridization improves the performance of the original ATPPSO in the makespan minimization.

The original ATPPSO has been improved by a local search mechanism in [34], known as I-ATPPSO. In I-ATPPSO, several operators have been added to the original ATPPSO to enhance its performance, specially the local search is realized by a mutator operator based on the variable neighborhood search [9]. However, this mutator has a quadratic complexity with regard of the number of jobs to be scheduled. In this sense, the simple local search in CAPSO-SALS has a linear complexity, which provides and advantage over the I-ATPPSO algorithm.

The paper is organized as follows: Section 2 describes the preliminaries of the FSSP. Section 3 explains the basics of PSO, CPSO and the original ATPPSO applied to FSSPs. Section 4 exposes the simple adaptive local search method used to improve the original ATPPSO in this paper. Section 5 presents the CAPSO-SALS algorithm proposed in this paper to minimize the makespan in FSSPs. Section 6 shows the sensitivity analysis, comparison of CAPSO-SALS with the original ATPPSO and I-ATPPSO and computational results comparing CPSO-SALS with other ten classic and state-of-art metaheuristic methods, obtaining sa- tisfactory results. The last section gives the concluding remarks of the paper.

## 2. Flow shop scheduling problem

The flow shop scheduling problem consists of the assignment of a set of $n$ jobs $W = \{J_1, \ldots, J_n\}$, each job $J_i$ is composed by a set of operations $J_i = \{O_{i,1}, \ldots, O_{i,m}\}$ processed by a set of machines $U = \{M_1 \ldots M_m\}$. Every operation $O_{i,j}$ has associated a value $T_{i,j}$ that represents the processing time of operation $O_{i,j}$; for $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$.

Each machine can process only one job at a time. It is assumed that the machine sequence is identical for all jobs, and the job sequence is the same for all machines.Therefore, a job schedule is represented as a permutation $\pi = \{\pi(1), \ldots, \pi(n)\}$ of $W$. The completion time of the operation $O_{i,j}$ is indicated by $C_{i,j}$. The model of the flow shop scheduling problem can be defined in the following way:

$$\begin{aligned} C_{\pi(1),1} &= T_{\pi(1),1}\,, \\ C_{\pi(1),j} &= C_{\pi(1),j-1} + T_{\pi(1),j}\,, \\ C_{\pi(i),1} &= C_{\pi(i-1),1} + T_{\pi(i),1}\,, \\ C_{\pi(i),j} &= \max\{C_{\pi(i-1),j},\ C_{\pi(i),j-1}\} + T_{\pi(i),j}\,, \end{aligned} \tag{1}$$

where $i \in \{2, \ldots, n\}$ and $j \in \{2, \ldots, m\}$. The makespan is defined as $C_{\max} = C_{\pi(n),m}$ or time on machine $m$ for all the jobs. The objective in this paper is the minimization of $C_{\max}$. For simplicity, the makespan of $\pi$ will be described by $C(\pi)$.

## 3. Particle swarm optimization

PSO is a continuous evolutionary algorithm initialized with a population of random solutions or particles, and searches optimal values by updating the population in generations. Each particle is conformed by a candidate solution $x$, its fitness, a velocity $v$ and a memory $x_b$ of the best candidate solution encountered by the particle with its recorded fitness. Each particle seeks in the search space with a velocity dynamically adjusted according to the own memory and the information of its colleagues. Thus, particles evolve towards better search areas. The velocity of a particle is calculated by:

$$v^{t+1} = wv^t + \varphi_1\left(x_b^t - x^t\right) + \varphi_2\left(x_g^t - x^t\right) \tag{2}$$

and the solution is updated by:

$$x^{t+1} = x^t + v^{t+1}, \tag{3}$$

where $w$ is the inertia weight, $\varphi_1$ and $\varphi_2$ are uniform distributed random numbers that determine the weight between the attraction to particle $x_b$ which is the best particle position, and $x_g$ which is the overall best particle.

Cellular particle swarm optimization (CPSO) is a variant of PSO proposed by Shi, Liu, Gao and Zhang [28], particles in CPSO improve their positions using a scheme similar to cellular automata. The concept of cellular automata was first proposed by Von Neumann and Ulam [3]. The operation of a cellular automaton raises from homogeneously interconnected cells evolving synchronously at discrete time steps obeying one common transition function [20]. Thus, a large array of simple elements can generate complex behaviors interacting locally with its neighbors. Particles in the swarm (or smart-cells) communicate with cells outside the swarm in order to improve their fitness value, the best cell in the neighborhood will be selected as the new smart-cell. That neighborhood concept will be used in the algorithm proposed in this paper for the FSSP.

### 3.1. Alternate two phases particle swarm optimization

The hybrid alternate two phases particle swarm optimization (ATPPSO) is a variant of the classical PSO where discrete versions of Eqs. 2 and 3 are proposed to handle permutations [34]. In ATPPSO, velocity and state of every particle are updates as follows:

$$v^{t+1} = v^t \otimes \pi_g + \otimes \pi_b\,, \tag{4}$$

$$\pi^{t+1} = \pi^t \otimes v^{t+1}, \tag{5}$$

where $\otimes$ is the attractive crossover operation depending on the best global position $\pi_g$ and the personal best position $\pi_b$ of a solution $\pi$.

The attractive crossover operator generates a new solution by combining two other sequences. To obtain a permutation of jobs as a new solution, a pair of crossing point is randomly selected in the first particle. The jobs inside crossing point are copied into the solution. Then the remaining places are filled up by taking in order each valid (unrepeated) job from the second particle (Fig. 1). However, because the velocities of particles approach to zero when velocity and current position have the same permutation, the attractive crossover is easily trapped in local optima.

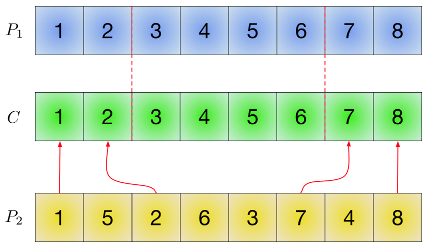


Figure 1: Example of the attractive crossover for permutations of 8 jobs, the resultant sequence is showed in the middle.

In order to overcome this problem, a repulsive process $\odot$ is defined (Eq. (6)), in which each particle is repelled away from its personal worst position and attracted to its personal best position simultaneously, to escape from local optima.

$$v^{t+1} = v^t \odot \pi_w + \otimes \pi_b \,. \tag{6}$$

Similar to the attractive crossover operator, first, a pair of crossing random points is selected along the first parent and the jobs inside their crossing points are copied into the solution. Then, the remaining places are filled up by taking in the reverse order each unrepeated job from the second particle (Fig. 2).

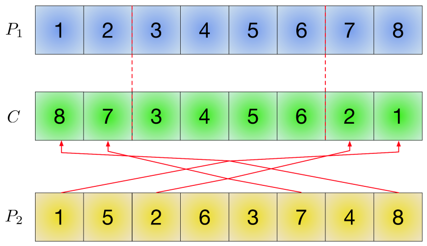


Figure 2: Example of the repulsive crossover for permutations of 8 jobs, the resultant sequence is illustrated in the middle.

## 4. Simple adaptive local search

Taking the ideas behind CPSO and ATPPSO, this paper proposes a hybrid algorithm called CAPSO-SALS, where the ATPPSO is complemented by a simple adaptive local search. The ATPPSO algorithm has been also improved with a local search in [34], where a tabu list and a mutation operator is implemented to increase the diversity of the swarm and improve the exploitation capacity. We believe that ATPPSO can be improved in an easier way obtaining similar results.

The ATPPSO algorithm utilizes three populations, the set of particles (hereafter called smart-cells), their best positions and their worst positions. Following the idea of CPSO [28], the ATPPSO procedure will be enhanced alternating the particle swarm process with a simple adaptive local search (SALS) based on a linear criterion to improve the population of best positions independently. This process will enhance the ATPPSO behavior when new velocities are calculated following Eq. (4) and (6).

Following the influence of CPSO, the local search will be implemented by neighborhoods for the best position of every smart-cell. If the best position of a smart-cell has a not-so-good makespan with regard of the others, it is possible that it may be in a local minimum, so its neighborhood will be generated with high probability of changing the first jobs to escape from local minima (Fig. 3 (A)). On the other hand, if the best position of a smart-cell has a good makespan compared to the rest of best positions, that position will have a greater probability to be modified in the last jobs in order to produce a neighborhood of new solutions and exploit the current position in a finer way (Fig. 3 (B)).

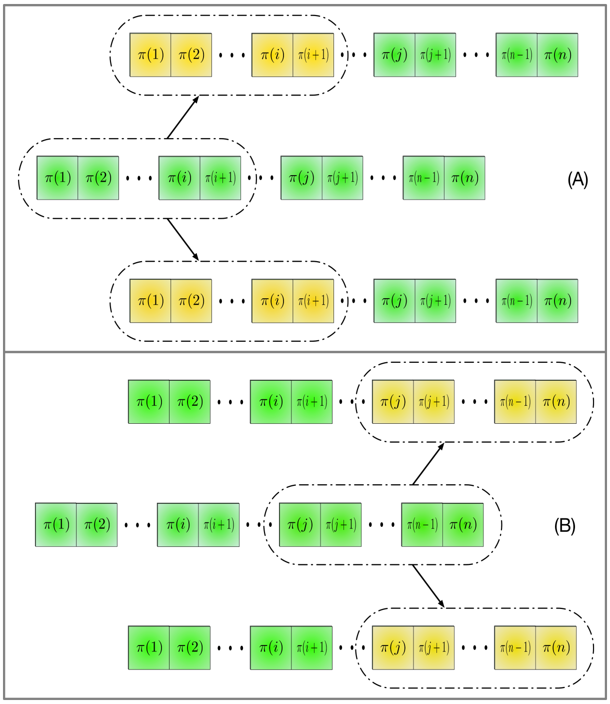


Figure 3: Neighbors generated by the local search. (A) If the best position of a smart-cell has a not-so-good makespan, the first jobs have more probability to be selected to produce new solutions. (B) If the best position of a smart-cell has a good makespan, the last jobs would be chosen to produce new solutions.

There are several local search methods based on job interchanges and insertions, such as a variable neighborhood search, exhaustive search, simulated annealing, age of solutions, selecting jobs not already scheduled, using a pool of new solutions or a roulette wheel scheme [7, 18, 22, 35]. Thus, interchanges and insertions are the most common and successful operators used in the optimization of FSSPs, therefore the local search proposed in this paper will be based on these operators (Fig. 4).

Based on the makespan of the best position of a smart-cell, an initial job is selected following a linear roulette wheel scheme. Two random positions are generated taking two different values from 1 to $n$; the job in the first position is interchanged or inserted in the second position to obtain a new neighbor. This process is described in detail below.

For any smart-cell $\pi \in \Pi$, let $\tau$ be the best position reached by $\pi$ during the process, let $\tau_g$ be the best position for all smart-cells and let $\tau_w$ be the worst of

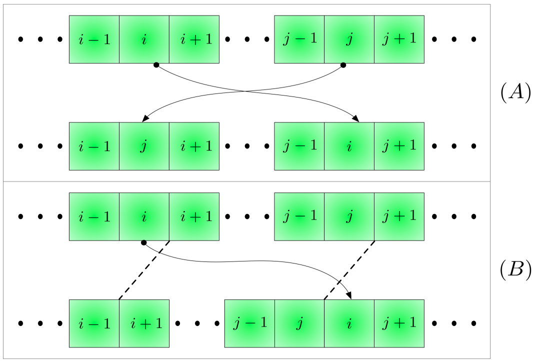


Figure 4: Interchange (A) and insertion (B) operators used in the local search neighborhood.

the current best positions in the optimization process. The normalized fitness of every $\tau$ is defined as:

$$P(\tau) = \frac{C(\tau_w) - C(\tau)}{C(\tau_w) - C(\tau_g)} . \tag{7}$$

In this case, $P(\tau_g) = 1$, $P(\tau_w) = 0$ and $0 \leqslant P(\tau) \leqslant 1$ for every other best position of a smart-cell.

There will be a greater probability of changing the last job places in $\tau_g$, a greater probability of changing the first job places in $\tau_w$, and all the job places have the same probability to be changed in the local search process when $P(\tau) = 1/2$. In the last case, if we have to optimize the scheduling of $n$ jobs, we are going to select one initial job place from 1 to $n-1$ and from this initial spot, select at random another subsequent job place and make an interchange or insertion. Therefore, the probability of selecting each job place is $1/(n-1)$.

We can define a factor $-1 \leqslant \epsilon \leqslant 0$ to regulate the probability of choosing an initial job place in $\tau$ to produce new solutions in its neighborhood. To control de probability of selecting an initial job place according to the normalized fitness $P(\tau)$, we define $\Omega = -1 + 2P(\tau)$ as a ponderation of $\epsilon$. In this case:

$$\begin{aligned} P(\tau) &= 0 & &\rightarrow \Omega = -1, \\ P(\tau) &= 1/2 & &\rightarrow \Omega = 0, \\ P(\tau) &= 1 & &\rightarrow \Omega = 1. \end{aligned} \tag{8}$$

The probability of choosing the job at the first place of $\tau$ is defined as:

$$\alpha = \frac{1 + \Omega\epsilon}{n-1} . \tag{9}$$

Notice that $\alpha$ is bigger when $P(\tau)$ is low to encourage the selection of the first job places in $\tau$, and it is lower for high values of $P(\tau)$ to aim the choice of the last job places in $\tau$. The remaining probability to select from job in place 2 until place $n - 1$ is given by:

$$\beta = 1 - \alpha(n - 1). \tag{10}$$

Equation (10) indicates that every job place has at least $\alpha$ probability to be chosen and the residual probability $\beta$ will be divided proportionately from places 2 until $n - 1$. Observe that $\beta$ will be positive for best solutions $\tau$ with good makespan values, and it will be negative for best positions with not-so-good makespan values. With this, the step size for the rest of job places is defined as:

$$\gamma = \beta / Fib(n - 2), \tag{11}$$

where $Fib(n - 2)$ is the Fibonacci number to calculate the corresponding proportional probability to be accumulated from place 2 to $n - 1$. The probability of every job place $1 \leqslant i \leqslant n - 1$ to be selected as the initial one to search locally new solutions is calculated as:

$$\delta(i) = \alpha + \gamma * (i - 1). \tag{12}$$

Figure 5 illustrates the behavior of $\delta$ for every job place depending of the value $\Omega$ associated with the makespan of $\tau$. Finally, the cumulative probability of choosing the job place $i$ as the initial one is given by:

$$\Delta(i) = \sum_{k=1}^{i} \delta(k). \tag{13}$$

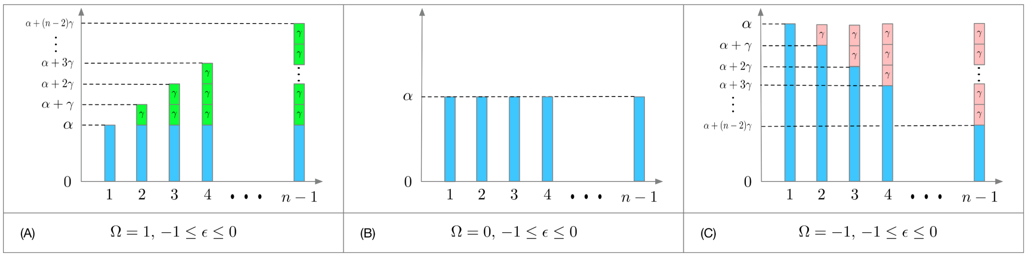


Figure 5: Probability to select every job place in the local search. (A) If $\tau$ has a good makespan, the last job places have more probability to be selected to produce new solutions. (C) If $P(\tau) = 1/2$, all jobs could be chosen with equal probability. (C) If $\tau$ has a not-so-good makespan, the first job places would be chosen to obtain better solutions.

A random number $r \in (0, 1)$ is generated to decide the initial job place $i$ to be selected in $\tau$ such that $r < \Delta(i)$. Once $i$ has ben chosen, another job place $j > i$ is calculated at random. With these job places $i$ and $j$, an interchange or an insertion is made to obtain the new neighbor of $\tau$ according to the current iteration number $mod\ m_s$, where $m_s$ is fixed at the beginning of the optimization process.

## 5. Cellular Alternate PSO with Simple Adaptive Local Search (CAPSO-SALS)

In this section we explain how ATPPSO is complemented by neighborhoods for every best solution of each smart-cell, where neighbors are generated using the adaptive linear local search described above.

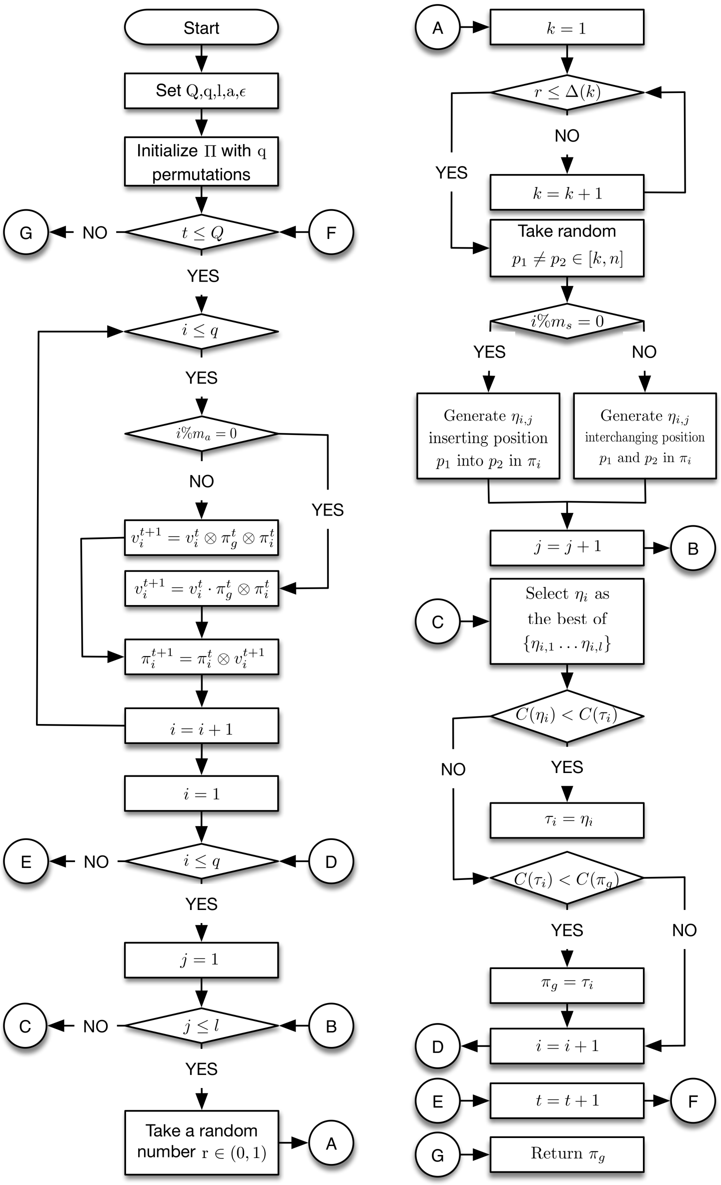

Start
Set Q,q,l,a,ϵ
Initialize Π with q permutations
$t \le Q$
NO
YES
G
F
$i \le q$
$i\%m_a = 0$
$v_i^{t+1} = v_i^t \otimes \pi_g^t \otimes \pi_i^t$
$v_i^{t+1} = v_i^t \cdot \pi_g^t \otimes \pi_i^t$
$\pi_i^{t+1} = \pi_i^t \otimes v_i^{t+1}$
$i = i + 1$
$i = 1$
E
D
$j = 1$
$j \le l$
C
B
Take a random number $r \in (0,1)$
A
$k = 1$
$r \le \Delta(k)$
$k = k + 1$
Take random $p_1 \neq p_2 \in [k,n]$
$i\%m_s = 0$
Generate $\eta_{i,j}$ inserting position $p_1$ into $p_2$ in $\pi_i$
Generate $\eta_{i,j}$ interchanging position $p_1$ and $p_2$ in $\pi_i$
$j = j + 1$
Select $\eta_i$ as the best of $\{\eta_{i,1} \ldots \eta_{i,l}\}$
$C(\eta_i) < C(\tau_i)$
$\tau_i = \eta_i$
$C(\tau_i) < C(\pi_g)$
$\pi_g = \tau_i$
$t = t + 1$
Return $\pi_g$

Figure 6: Flowchart for CAPSO-SALS algorithm

The parameters of the proposed algorithms are: $Q$ number of iterations, $q$ number of particles, $l$ number of neighbors per best solution of each smart-cell, $m_a$ to select the attractive or repulsive part of ATPPSO, $\epsilon$ to regulate the probability of choosing an initial job place, and $m_s$ to determine if interchanges or insertions are made to obtain the new neighbors of every best position (Algorithm 1).

**Algorithm 1** CAPSO-SALS algorithm

```
1:  //Initialization
2:  Set the control parameters: $Q$, $q$, $m_a$, $l$, $\epsilon$, $m_s$
3:  initialize $\Pi$ with $q$ random permutations
4:  for $t = 1$ to $Q$ do
5:      //ATPPSO part
6:      for $i = 1$ to $q$ do
7:          if $t \% m_a = 0$ then
8:              $v_i^{t+1} = v_i^t \otimes \pi_g^t \otimes \pi_i^t$
9:          else
10:             $v_i^{t+1} = v_i^t \odot \pi_g^t \otimes \pi_i^t$
11:         end if
12:         $\pi_i^{t+1} = \pi_i^t \otimes v_i^{t+1}$
13:     end for
14:     //Cellular Automata part
15:     for $i = 1$ to $q$ do
16:         //Create the neighborhood of $\pi_i$
17:         for $j = 1$ to $l$ do
18:             Generate a random number $r \in (0, 1)$
19:             $k = 1$
20:             while $r < \Delta(k)$ according to Eq. 12 do
21:                 $k = k + 1$
22:             end while
23:             Generate two random positions $p_1 < p_2 \in [k, n]$
24:             if $t \% m_s = 0$ then
25:                 Generate a new solution $\eta_{i,j}$ inserting job from position $p_1$ to position $p_2$ in $\pi_i$
26:             else
27:                 Generate a new solution $\eta_{i,j}$ interchanging jobs from positions $p_1$ to $p_2$ in $\pi_i$
28:             end if
29:         end for
30:         //Evolution rule of CPSO
31:         Select $\eta_i$ as the neighbor in $\{\eta_{i,1} \dots \eta_{i,l}\}$ with minimum makespan
32:         if $C(\eta_i) < C(\tau_i)$ then
33:             $\tau_i = \eta_i$
34:         end if
35:         //Update best solution if needed
36:         if $C(\tau_i) < C(\pi_g)$ then
37:             $\pi_g = \tau_i$
38:         end if
39:     end for
40: end for
41: Return $\pi_g$
```

### 5.1. Computational complexity analysis

The time complexity of the proposed method is briefly analyzed. CAPSO-SALS consists of four main steps: initialization, attractive two-phase PSO, cellular local search with adaptive local search, and halting judgment. Note that the CAPSO-SALS has $q$ best positions, and each one generates $l$ new particles by adaptive linear local search. Let $Q$ be the maximum iteration number. Initialization step contains a loop ($q$ times) to generate permutations of $n$ jobs, so its time complexity is $O(qn)$. For attractive two-phase PSO, there is a loop ($q$ times) performing three permutation crossovers, so its time complexity is $O(q3n)$. For adaptive linear local search, there is a triple loop ($q$, $l$ and $n$ times), after the last loop, an interchange or an insertion is performed ($O(n)$). Therefore, its time complexity is $O(ql2n)$. For halting step, its time complexity is $O(1)$. Therefore, the time complexity of the proposed method is $O(Qqln)$.

The parameter $l$ could be fixed in advance. So, when the swarm size or number of jobs is large, the contribution of $l$ can be neglected. Then the computational complexity of CAPSO-SALS is $O(Qqn)$, so our algorithm has the same computational complexity as the original ATPPSO. This complexity is lower than the one reported by new algorithms using improved local search methods whose complexity is $O(Qqn^2)$; for instance, the immunity-based hybrid genetic algorithm [2] or the particle swarm optimization with expanding neighborhood topology [19].

From experimental results, we could see that CAPSO-SALS outperforms ATPPSO and the most effective methods reported in [2] and [19] on most sets of problems, so we could say it is efficient. Moreover, since the rapid development of computer, the quality of solutions is more important when problems are solved in convenient time.

## 6. Simulation results and comparison

### 6.1. Sensitivity analysis

A sensitivity analysis for CAPSO-SALS has been performed by varying different factors such as number of smart-cells $q$, number of iterations $Q$, value $mod_a$ to apply the repulsive crossover of ATTPSO, number of neighbors $l$ for every smart-cell, value $\epsilon$ to regulate de probability of choosing an initial job for the adaptive linear local search and value $mod_s$ to select an interchange or an insertion for the adaptive local search.

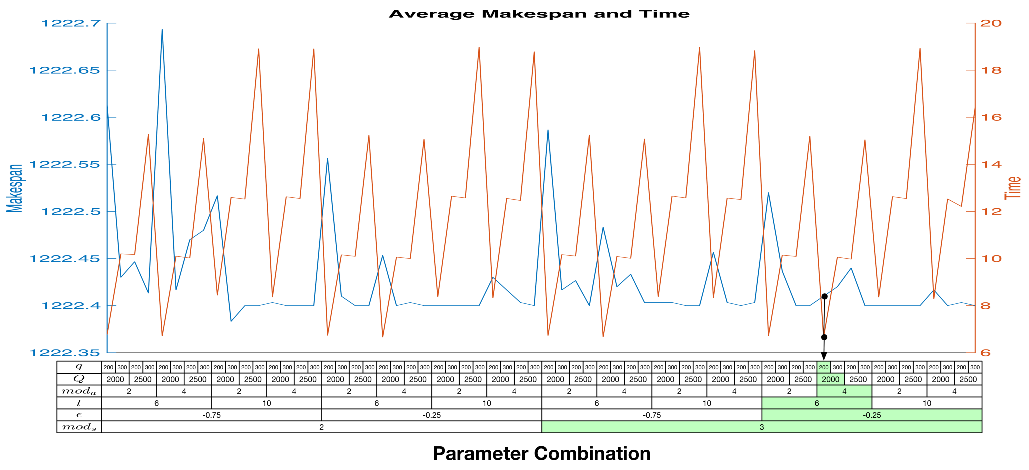
In order to calibrate the algorithm, all possible combinations of the previous factors are tested with different levels as defined in Table 1. The total number of combinations is $2^6 = 64$.

Table 1: Levels of parameters for the sensitivity analysis

| Factors | $q$ | $Q$ | $m_a$ | $l$ | $\epsilon$ | $m_s$ |
|---|---|---|---|---|---|---|
| Levels | 200 | 2000 | 2 | 6 | $-0.75$ | 2 |
| | 300 | 2500 | 4 | 10 | $-0.25$ | 3 |

Our optimization program has been executed on a Mac OS environment using Intel Core Xeon, 3.5 GHz, 32G RAM memory, and the codes were developed using MATLAB 7.14 and optimized by mex command. Our algorithm is analyzed with a random set of 10 FSSP instances, each one with 20 jobs and 5 machines produced using processing times uniformly distributed between 1 and 99 [29]. For each parameter combination we observe the value for the average makespan obtained over 30 runs.

Figure 7 plots at once the average makespan and the corresponding execution time (in secs.) obtained in the sensitivity analysis. From these solutions, one notices that a bigger number of neighbors increases the execution time, when a larger number of smart-cells and iterations is considered.



| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 |
| $Q$ | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | |
| $mod_a$ | 2 | | | | 4 | | | | 2 | | | | 4 | | | | 2 | | | | 4 | | | | 2 | | | | 4 | | | |
| $l$ | 6 | | | | | | | | 10 | | | | | | | | 6 | | | | | | | | 10 | | | | | | | |
| $\epsilon$ | -0.75 | | | | | | | | | | | | | | | | -0.25 | | | | | | | | | | | | | | | |
| $mod_s$ | 2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $q$ | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 | 200 | 300 |
| $Q$ | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | | 2000 | | 2500 | |
| $mod_a$ | 2 | | | | 4 | | | | 2 | | | | 4 | | | | 2 | | | | 4 | | | | 2 | | | | 4 | | | |
| $l$ | 6 | | | | | | | | 10 | | | | | | | | 6 | | | | | | | | 10 | | | | | | | |
| $\epsilon$ | -0.75 | | | | | | | | | | | | | | | | -0.25 | | | | | | | | | | | | | | | |
| $mod_s$ | 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 7: Sensitivity analysis for the CAPSO-SALS algorithm

From these solutions, the selected combination of parameters, which produces a minimum makespan and execution time, is the following one: number of smart-cells $q = 200$, number of iterations $Q = 2000$, $mod_a = 4$ for the repulsive crossover, number of neighbors $l = 6$, probability of choosing an initial job for the adaptive local search $\epsilon = -0.25$, and $mod_s = 3$ for executing an insertion in the local search.

## 6.2. Performance comparison of the ATPPSO and CAPSO-SALS

For evaluating the proposed CAPSO-SALS contrasted with the original ATPPSO method, we use the best and average makespan for selected instances of the last 4 FSSP sets from the Taillard set problems defined in the OR Library http://people.brunel.ac.uk/mastjjb/jeb/orlib/flowshopinfo.html and commonly employed to compare new optimization methods. In these instances, there are 12 sets and these are: $20 \times 5$ (i.e. 20 jobs and 5 machines), $20 \times 10$, $20 \times 20$, $50 \times 5$, $50 \times 10$, $50 \times 20$, $100 \times 5$, $100 \times 10$, $100 \times 20$, $200 \times 10$, $200 \times 20$ and $500 \times 20$. There are 10 problems inside every size set.

An improvement of the ATPPSO called I-ATPPSO has been also presented in [34]. This algorithm implements more complex control operators than CAPSO-SALS. For instance, I-ATPPSO employs a tabu list, an activity threshold measure and a variable neighborhood search realized by a mutation operator which is greedy in CPU time consumption ($O(n^2)$).

Table 2 shows the statistics of the best and the average makespan values of the proposed CAPSO-SALS algorithm, the ATPPSO and the I-ATPPSO methods. We take the last 12 results for the specific Taillard instances reported in [34] because these ones represent the most difficult examples to optimize. These problems goes from 100 to 500 jobs and 10 to 20 machines. For a fair comparison, in these instances the number of iterations in the CAPSO-SALS algorithm has been defined in 6000 for every run, and 20 runs for each problem as it has been calculated in [34]. The best results are presented in bold.

Table 2: Performance of ATPPSO, I-ATPPSO and CAPSO-SALS

| Problem | Size | Optimal | ATPPSO | iATPPSO | CAPSO-SALS |
|---|---|---|---|---|---|
| ta081 | $100 \times 20$ | 6202 | 6471/6527.3 | 6375/6445.5 | **6369/6413.1** |
| ta085 | $100 \times 20$ | 6314 | 6514/6577.5 | **6448**/6520.9 | 6461/**6493.3** |
| ta090 | $100 \times 20$ | 6434 | 6641/6690.6 | 6577/6604.7 | **6558/6588.2** |
| ta091 | $200 \times 10$ | 10862 | 10950/11000.2 | **10885**/10920.4 | 10892/**10901.6** |
| ta095 | $200 \times 10$ | 10524 | 10537/10620.5 | 10537/**10562.8** | **10533**/10568 |
| ta0100 | $200 \times 10$ | 10675 | 10747/10846.2 | **10685**/10744.5 | 10694/**10738.9** |
| ta0101 | $200 \times 20$ | 11195 | 11571/11644.2 | 11488/11549.4 | **11463/11543.3** |
| ta0105 | $200 \times 20$ | 11259 | 11636/11734.2 | **11518/11574.2** | 11583/11621 |
| ta0110 | $200 \times 20$ | 11288 | 11730/11783 | 11655/11707.5 | **11643/11701.4** |
| ta0111 | $500 \times 20$ | 26059 | 26961/27011.9 | 26670/26783.4 | **26621/26742.2** |
| ta0115 | $500 \times 20$ | 26334 | 26984/27102.9 | **26739/26851.9** | 26740/26866 |
| ta0120 | $500 \times 20$ | 26457 | 27144/27246.5 | **26900/27017.7** | 26956/27077.2 |

It can be observed in Table 2 that CAPSO-SALS outperforms the original ATPPSO in best and average values for all the instances, proving that CAPSO-SALS represents an improvement over the ATPPSO method. Besides, CAPSO-

SALS obtains 6 minimum makespan and 8 better averages than I-ATPPSO. For the three problems with 500 jobs and 20 machines, I-ATPPSO shows a better performance than CAPSO-SALS. This can be explained by the large number of jobs, provoking that the initial probability of Eq. (9) be too small to define a significative difference. Further investigation may use a selective sensitivity analysis to obtain a better performance of CAPSO-SALS for this particular case.

Table 2 confirms our idea concerning the utility to hybrid cellular automata concepts with an adaptive local search to improve the ATPPSO method. However, the neighborhood concept from cellular automata has a linear processing cost. This is explained by the number of additional operations induced by the fixed neighbors for every smart-cell.

This better performance is mainly due to the effectiveness of the proposed adaptive local search, which produces changes in the scheduling by interchanges and insertions. Last jobs have more probability to be moved if a solution is closer to the best global solution in order to improve the exploitation of good search regions. On the other hand, worst solutions that may be trapped in a local minimum have a greater probability to change the position of the jobs in the first positions. Thus, the algorithm can explore regions in the search space with new information.

It can be can concluded that CAPSO-SALS is better in performance than ATPPSO and has a similar efficiency that I-ATPPSO with a lower complexity due to the simplicity of its local search strategy.

### 6.3. Comparative results of the proposed hybrid algorithm against other metaheuristics

For evaluating the proposed CAPSO-SALS we use the average error ratio $E$. In the literature, $E$ is often used to evaluate the performance of algorithms applied to deal with FSSPs. The error ratio is calculated as follows:

$$E = \frac{Cmax - Best}{Best}\ \%, \tag{14}$$

where $Cmax$ is the best makespan obtained for the algorithm and $Best$ is the known minimum makespan for the problem. We compare the proposed hybrid CAPSO-SALS to 10 between the most frequently referred and recent state-of-the-art metaheuristics in the related sche- duling literature for the FSSP.

In Table 3, a comparison with those algorithms is performed, presenting the average quality of the error ratio defined in Eq. 14 for the ten instances of every Taillard set problem. Those values are obtained by the solutions of the proposed algorithm (CAPSO-SALS) and the other 10 algorithms from the literature. Those are: and artificial chromosomes with genetic algorithm (ACGA) of Chang et al. [4], an ant colony optimization algorithm (ACS) [32], a discrete differential evolution (DDE) algorithm [23], a combinatorial particle swarm optimization (ComPSO) of Jarboui et al. [13], the genetic algorithm of Reeves (GAReev) [27], a hybridization of genetic algorithm with variable neighborhood search (GA-VNS) of Zobolas et al. [35], a modified genetic algorithm (MGGA) of Tang et al. [30], a particle swarm optimization with expanding neighborhood topology (PSONET) of Marinakis et al. [19], a self-guided genetic algorithm (SGGA) of Chen et al. [6], and an immunity-based hybrid genetic algorithm (VacGA) of Bessedik et al. [2]. With the aim to compare them, we have used the data from Marinakis et al. [19] and Bessedik et al. [2]. The results, averaged by instance size are shown in Table 3.

Table 3: Average *E* over the optimum solution or lowest known upper bound for Taillard's instances obtained by the methods compared.

| Problems | CAPSO-SALS | ACGA | ACS | DDE | ComPSO | GA Reev |
|---|---|---|---|---|---|---|
| $20 \times 5$ | 0 | 1.08 | 1.19 | 0.46 | 1.05 | 0.53 |
| $20 \times 10$ | 0 | 1.62 | 1.7 | 0.93 | 2.42 | 1.79 |
| $20 \times 20$ | 0 | 1.34 | 1.6 | 0.79 | 1.99 | 1.40 |
| $50 \times 5$ | 0 | 0.57 | 0.43 | 0.17 | 0.9 | 0.19 |
| $50 \times 10$ | 0.74 | 2.79 | 0.89 | 2.26 | 4.85 | 2.11 |
| $50 \times 20$ | 3.1 | 3.75 | 2.71 | 3.11 | 6.4 | 3.60 |
| $100 \times 5$ | 0.02 | 0.44 | 0.22 | 0.08 | 0.74 | 0.16 |
| $100 \times 10$ | 0.54 | 1.71 | 1.22 | 0.94 | 2.94 | 0.80 |
| $100 \times 20$ | 2.63 | 3.47 | 2.22 | 3.24 | 7.11 | 3.32 |
| $200 \times 10$ | 0.62 | 0.94 | 0.64 | 0.55 | 2.17 | – |
| $200 \times 20$ | 3.34 | 2.61 | 1.3 | 2.61 | 6.89 | – |
| $500 \times 20$ | 2.58 | – | 1.68 | – | – | – |
| Average | 1.13 | 1.84 | 1.28 | 1.37 | 3.40 | 1.54 |

| Problems | GA-VNS | MGGA | PSONET | SGGA | VacGA |
|---|---|---|---|---|---|
| $20 \times 5$ | 0 | 0.81 | 0 | 1.1 | 0.08 |
| $20 \times 10$ | 0 | 1.4 | 0.07 | 1.9 | 0.93 |
| $20 \times 20$ | 0 | 1.06 | 0.08 | 1.6 | 0.07 |
| $50 \times 5$ | 0 | 0.44 | 0.02 | 0.52 | 0.08 |
| $50 \times 10$ | 0.77 | 2.56 | 2.11 | 2.74 | 2.34 |
| $50 \times 20$ | 0.96 | 3.82 | 3.83 | 3.94 | 3.69 |
| $100 \times 5$ | 0 | 0.41 | 0.09 | 0.38 | 0.14 |
| $100 \times 10$ | 0.08 | 1.5 | 1.26 | 1.6 | 1.44 |
| $100 \times 20$ | 1.31 | 3.15 | 4.37 | 3.51 | 3.43 |
| $200 \times 10$ | 0.11 | 0.92 | 1.02 | 0.8 | – |
| $200 \times 20$ | 1.17 | 3.95 | 4.27 | 2.32 | – |
| $500 \times 20$ | 0.63 | – | 2.73 | – | – |
| Average | 0.4 | 1.82 | 1.65 | 1.85 | 1.35 |

From Table 3, there are a number of important conclusions about CAPSO-SALS. The algorithm finds the optimum in the first set for all instances. This happens in the same way only with the GA-VNS algorithm. Both algorithms have as local search phase a neighborhood generation; in the case of CAPSO-SALS based on neighbors generated by interchanges and insertions, and for GA-VNS applying a more complex VNS algorithm. We can conclude that the combination of a population based algorithm (like PSO in the proposed algorithm and a genetic algorithm in GA-VNS) with a very strong local search technique increases both the exploration and exploitation abilities of the algorithm. Comparing our proposed hybrid algorithm with others in Table 3, the results showed that CAPSO-SALS has yielded a lower average in 9 out of the other 10 algorithms used for the comparisons. This difference is up to 300% in the case of another particle swarm optimization variant (ComPSO). These results shows as well that CAPSO-SALS provides better results than PSONET and VacGA, which were recently proposed in literature.

Algorithms which deal directly with discrete values, like genetic algorithms or ant colony systems, may have in general an advantage when are compared with discrete versions of PSO algorithm. However, the proposed CAPSO-SALS performs better than the six out of seven versions of those approaches (only GA-VNS performs better). The most important comparison is versus two PSO algorithms and the DDE algorithm. This happens because for these three strategies, as in the proposed algorithm, a discretization from the basic equations that characterize these methods should be implemented. The results of the proposed algorithm in almost every average results of all sets are better than the other three implementations. The average results of DDE for the sets of 200 jobs are better from the proposed algorithm (0.55 vs 0.62 and 2.61 vs 3.34), but the average value is better in CAPSO-SALS (1.13 vs 1.37) and the DDE algorithm has no results for the difficult set of 500 jobs. Compared to ComPSO and PSONET, the proposed algorithm presents better average results for all the sets, only PSONET have obtained the same performance for the first set. This fact shows that the proposed CAPSO-SALS is very efficient for the solution of this kind of problems.

For the largest set of 500 jobs, there are only four algorithms reporting results and CAPSO-SALS is in third place according to the corresponding $E$. However, it outperforms the other recent PSO approach (PSONET). This extensive experimentation shows that the proposed CAPSO-SALS algorithm is an easy and innovative improvement for PSO based methods solving FSSPs, which has competitive performance as far as simple approaches are concerned.

In Table 4, the average CPU times in seconds of the proposed algorithm and of other two recent algorithms for each of the 12 sets are presented. It can be observed that the proposed algorithm is slightly faster than the other two algorithms for the instances of 20 machines, and in the average it needs 103.45 s to find the best in an instance, while the other two algorithms need between 93.38 s and 105.04 s.

Table 4: Average CPU times (in seconds) of CAPSO-SALS and other two recent algorithms (PSONET and VacGA) from literature in Taillard benchmark instances for the FSSP.

| Problems | CAPSO-SALS | PSONET | VacGA |
|---|---|---|---|
| $20 \times 5$ | 6.55 | 3.45 | 36.4 |
| $20 \times 10$ | 8.95 | 15.25 | 35.6 |
| $20 \times 20$ | 14.35 | 24.52 | 36.2 |
| $50 \times 5$ | 15.22 | 6.15 | 56.3 |
| $50 \times 10$ | 21.13 | 23.55 | 96.7 |
| $50 \times 20$ | 34.29 | 44.25 | 103.2 |
| $100 \times 5$ | 30.79 | 22.85 | 150.4 |
| $100 \times 10$ | 42.73 | 60.35 | 210.2 |
| $100 \times 20$ | 67.63 | 131.15 | 220.4 |
| $200 \times 10$ | 88.39 | 124.18 | – |
| $200 \times 20$ | 136.68 | 255.42 | – |
| $500 \times 20$ | 360.83 | 409.54 | – |
| Average | 103.45 | 93.38 | 105.04 |

## 7. Conclusion

In this paper, we have proposed the improvement of the alternate two phases PSO called CAPSO-SALS. This algorithm is a hybridization of the ATPPSO algorithm with concepts of cellular automata and a simple adaptive local search for the flow shop scheduling problem under the makespan minimization criterion. The algorithm uses a flexible local neighborhood topology where the size of the neighborhood is fixed since the beginning of the optimization process. The main challenge was to present a simple algorithm combining the advantages of the exploration abilities of the ATPPSO algorithm with the exploitation abilities of a local neighborhood structure by means of an adaptive local search.

The parameters of the CAPSO-SALS has been calibrated by an extensive design of experiments. The algorithm has been tested in 120 benchmark instances that are usually used in the literature. CAPSO-SALS has given better results in general when it was compared with the original ATPPSO algorithm and similar solutions to I-ATPPSO algorithm that has a more complex local search process. CAPSO-SALS produced very good results as well against other algorithms recently reported in literature. The results are very promising and show that the CAPSO-SALS is competitive with other successful methods.

Furthermore, the CAPSO-SALS could be modified to take into account more realistic aspects of the problem such as sequence-dependent setup times (SDST flow shop), unrelated parallel machines at each stage (general hybrid flow shop), or the existence of due dates. Other perspective is the application of the CAPSO-

SALS algorithm for the solution of other NP-hard combinatorial optimization problems.

The performance of the CAPSO-SALS could be further improved with the NEH heuristic, or applying a selective sensitivity analysis for particular cases of FSSPs. We will study the effect of these options on the CAPSO-SALS in the future.

## References

[1] A. Banks, J. Vincent, and C. Anyakoha: A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, **7**(1), (2008) 109–124.

[2] M. Bessedik, F.B.S. Tayeb, H. Cheurfi, and A. Blizak: An immunity-based hybrid genetic algorithms for permutation flowshop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, **85**(9-12), (2016), 2459–2469.

[3] A.W. Burks: *Essays on cellular automata*. University of Illinois Press, 1970.

[4] P.C. Chang, S.H. Chen, C.Y. Fan, and V. Mani: Generating artificial chromosomes with probability control in genetic algorithm for machine scheduling problems. *Annals of Operations Research*, **180**(1), (2010), 197–211.

[5] C.L. Chen, S.Y. Huang, Y.R. Tzeng, and C.L. Chen: A revised discrete particle swarm optimization algorithm for permutation flow-shop scheduling problem. *Soft Computing*, **18**(11), (2014), 2271–2282.

[6] S.H. Chen, P.C. Chang, T. Cheng, and Q. Zhang: A self-guided genetic algorithm for permutation flowshop scheduling problems. *Computers & Operations Research*, **39**(7), (2012), 1450–1457.

[7] X. Dong, M. Nowak, P. Chen, and Y. Lin: Self-adaptive perturbation and multi-neighborhood search for iterated local search on the permutation flow shop problem. *Computers & Industrial Engineering*, **87**, 176–185 (2015)

[8] K.L. Du and M. Swamy: *Particle swarm optimization*. In: Search and Optimization by Metaheuristics, pp. 153–173. Springer, 2016.

[9] K. Fleszar and K.S. Hindi: An effective vns for the capacitated p-median problem. *European Journal of Operational Research*, **191**(3), (2008), 612–622.

[10] L. Gao, J. Huang, and X. Li: An effective cellular particle swarm optimization for parameters optimization of a multi-pass milling process. *Applied Soft Computing*, **12**(11), (2012), 3490–3499. https://doi.org/10.1016/j.asoc.2012.06.007, http://www.sciencedirect.com/science/article/pii/S1568494612002785.

[11] E. García-Gonzalo and J. Fernández-Martínez: A brief historical review of particle swarm optimization (pso). *Journal of Bioinformatics and Intelligent Control*, **1**(1), (2012), 3–16.

[12] S. Gholizadeh: Layout optimization of truss structures by hybridizing cellular automata and particle swarm optimization. *Computers & Structures*, **125** (2013), 86–99. http://dx.doi.org/10.1016/j.compstruc.2013.04.024. http://www.sciencedirect.com/science/article/pii/S0045794913001557

[13] B. Jarboui, S. Ibrahim, P. Siarry, and A. Rebai: A combinatorial particle swarm optimisation for solving permutation flowshop problems. *Computers & Industrial Engineering*, **54**(3), (2008), 526–538.

[14] S.M. Johnson: Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics* (*NRL*), **1**(1), (1954), 61–68.

[15] P. Lagos-Eulogio, J.C. Seck-Tuoh-Mora, N. Hernandez-Romero, and *J. Medina-Marin*: A new design method for adaptive iir system identification using hybrid cpso and de. *Nonlinear Dynamics*, **88**(4), (2017), 2371–2389.

[16] S. Lalwani, R. Kumar, and N. Gupta: A review on particle swarm optimization variants and their applications to multiple sequence alignment. *Journal of Applied Mathematics and Bioinformatics*, **3**(2), (2013), 87.

[17] C.J. Liao, C.T. Tseng, and P. Luarn: A discrete version of particle swarm optimization for flowshop scheduling problems. *Computers & Operations Research*, **34**(10), (2007), 3099–3111.

[18] R. Liu, C. Ma, W. Ma, and Y. Li: A multipopulation pso based memetic algorithm for permutation flow shop scheduling. *The Scientific World Journal*, **2013** (2013).

[19] Y. Marinakis and M. Marinaki: Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. *Soft Computing*, **17**(7), (2013), 1159–1173.

[20] H.V. McIntosh: *One Dimensional Cellular Automata*. Luniver Press, 2009.

[21] T. Morton and D.W. Pentico: *Heuristic scheduling systems: with applications to production systems and project management*, vol. 3. John Wiley & Sons, 1993.

[22] Q.K. Pan and R. Ruiz: Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, **222**(1), (2012), 31–43.

[23] Q.K. Pan, M.F. Tasgetiren, and Y.C. Liang: A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, **55**(4), (2008), 795–816.

[24] Q.K. Pan, M.F. Tasgetiren, and Y.C. Liang: A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research*, **35**(9), (2008), 2807–2839.

[25] Q.K. Pan, L. Wang, M.F. Tasgetiren, and B.H. Zhao: A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion. *The International Journal of Advanced Manufacturing Technology*, **38**(3–4), (2008), 337–347.

[26] M.L. Pinedo: *Scheduling: theory, algorithms, and systems*. Springer, 2016.

[27] C.R. Reeves: A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, **22**(1), (1995), 5–13.

[28] Y. Shi, H. Liu, L. Gao, and G. Zhang: Cellular particle swarm optimization. *Information Sciences*, **181**(20), (2011), 4460–4493, Special Issue on Interpretable Fuzzy Systems, http://dx.doi.org/10.1016/j.ins.2010.05.025. http://www.sciencedirect.com/science/article/pii/S0020025510002288.

[29] E. Taillard: Benchmarks for basic scheduling problems. *European Journal of Operational Research*, **64**(2), (1993), 278–285.

[30] L. Tang and J. Liu: A modified genetic algorithm for the flow shop sequencing problem to minimize mean flow time. *Journal of Intelligent Manufacturing*, **13**(1), (2002), 61–67.

[31] M.F. Tasgetiren, Y.C. Liang, M. Sevkli, and G. Gencyilmaz: A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, **177**(3), (2007), 1930–1947.

[32] K.C. Ying and C.J. Liao: An ant colony system for permutation flow-shop sequencing. *Computers & Operations Research*, **31**(5), (2004), 791–801.

[33] M. Yu, Y. Zhang, K. Chen, and D. Zhang: Integration of process planning and scheduling using a hybrid ga/pso algorithm. *The International Journal of Advanced Manufacturing Technology*, **78**(1–4), (2015), 583–592.

[34] C. Zhang, J. Ning, and D. Ouyang: A hybrid alternate two phases particle swarm optimization algorithm for flow shop scheduling problem. *Computers & Industrial Engineering*, **58**(1), (2010), 1–11.

[35] G. Zobolas, C.D. Tarantilis, and G. Ioannou: Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, **36**(4), (2009), 1249–1267.