Comparison of two variants of particle swarm optimization algorithm for solving flexible job shop scheduling problem

S.Kamel/ENIT
National School of Engineering of Tunis
Tunisia
msouadpop@gmail.com

S.Boubaker/UOH
Hail University
Saudi Arabia
Sahbi.Boubaker@gmail.com

***Abstract*—The Scheduling is one of the most challenging problems faced in many different areas of everyday life. This problem can be formulated as a combinatorial optimization problem, and it has been solved with various methods using nature-inspired meta-heuristics and intelligent algorithms. We present in this paper a solution to the flexible ob shop scheduling problem using two variants of particle swarm optimization namely parametric version (PSO) and fully- adaptive one (TRIBES). TRIBES like PSO, is a computational method that mimics the behavior of flying birds and their means of information exchange. The candidate solutions in the swarm communicate and cooperate with each other, whereas individuals in an evolutionary algorithm compete for survival. A study comparing the performances of both solutions is described and the results are analyzed.**

Keywords—particle swarm optimization; TRIBES; scheduling; makespan ; Computational intelligence.

I. Introduction

Scheduling and sequencing are important factors to survive in the marketplace. They permit the allocation of jobs over time [1] when limited resources are available, where several constraints must be satisfied and a number of objectives should be optimized.

Most scheduling problems are complex combinatorial optimization problems. The flexible job shop scheduling problem (FJSP) is a branch of production scheduling that belongs to the NP-hard family [1]. It presents two difficulties. The first one is the assignment of each operation to a machine, and the second one is the scheduling of this set of operations in order to optimize a predefined criterion. The result of a scheduling algorithm must be a schedule that contains a start time and a resource assignment to each operation.

Many approaches have been developed to solve the FJSP. In general, these approaches use heuristic algorithms since that no exact algorithm can consistently solve FJSP instances with a great number of jobs. Most of these approaches have treated separately the assignment of operations to machines and the sequencing of operations on the resources or machines. In this paper, we introduce two algorithms based on particle swarm optimization (PSO). The first one requires the user to adjust some control parameters which is expensive and time consuming, especially as the optimum set of parameters is to be renewed for each problem. The second algorithm can solve any optimization problem without going through a preliminary step of tuning parameters. PSO is easily implemented while the code of TRIBES is a bit complicated.

Details of our two approaches based on PSO are presented in section III, IV and V. Section II introduces and formulates the flexible job shop scheduling problem .The experiments are given in section VI. Finally, brief conclusions and future perspectives are discussed in section VII.

II. Problem formulation

First, The problem consists in performing a set of n jobs J= $\{J_1, J_2, \ldots, J_n\}$ on a set of m machines $M=\{M_1, M_2, \ldots, M_m\}$ .Each machine can process only one operation at a time. Each job J_i has a linear sequence of ni operations $\{O_{i,1}, O_{i,2}, \ldots, O_{i,ni}\}$. There are precedence constraints among the operations of the same job. Nonetheless, there are no precedence constraints among the operations of different jobs. The realization of each operation $O_{i,j}$ (the operation j of job i) requires a machine M_k and a processing time $p_{i,j,k}$. Job preemption and job splitting are not allowed. The starting time of $O_{i,j}$ is $t_{i,j}$ and the ending time is $t_{fi,j}$. We can associate a table TFJSP of processing times to each flexible job shop scheduling problem such that TFJSP= $\{p_{i,j,k}$ integer$\neq 0$, $1 \le i \le n; 1 \le j \le n_i; 1 \le k \le m\}$. In literature the FJSP can be classified into two sub problems: the Total FJSP (T-FJSP) and the partial FJSP (PFJSP). In T-FJSP, each operation can be processed on any machine of M whereas, in P-FJSP, each operation can be processed on one machine of subset of M. In our paper, the two kinds of FJSP are considered and the objective is to find a schedule having a minimum makespan (or, the minimum time of the whole operations), denoted by $C_{max}=max_{1\le i\le n}(max_{1\le j\le ni}(t_{fi,j}))$. Moreover, we shall assume that the total number of operations to perform is greater than the number of machines, each job can start at time 0, breakdowns are not considered and machines are available at time 0.

Now that the problem has been suitably described, the next step is to select an appropriate solution strategy. The selection of a suitable solution strategy should, in most cases driven by the complexity of the problem at hand. Next sections focuse on the two techniques chosen to solve the FJSP to minimize the makespan.

III. PARTICLE SWARM OPTIMIZATION

The particle swarm optimization (PSO) is a population based search algorithm developed by Kennedy and Eberhart in 1995 ([3]) inspired by social behavior of bird flocking or fish schooling. Unlike Genetic Algorithms (GA), PSO has no evolution operators such as crossover and mutation. In PSO, the population is initialized randomly and the potential solutions, called particles fly through the search space with velocities which are dynamically adjusted according to their historical behaviors. In PSO, each particle is influenced by both the best solution that it has discovered so far and the best particle in its neighbors (local variant) or in the entire population (global variant). In PSO, each particle i modelised in an optimization problem which dimensionality is D, by a position vector $\vec{xi}=(x_{i1}, x_{i2},\ldots,x_{iD})$ and a velocity $\vec{vi}=(v_{i1}, v_{i2}, \ldots, v_{iD})$, is influenced by both the best solution that it has discovered so far $\vec{p_i}=(p_{i1}, p_{i2}, \ldots,p_{iD})$ and the best particle in its neighbors or in the entire population $\vec{g_i}=(g_{i1}, g_{i2}, \ldots, g_{iD})$.

Fig.1 shows the general flow chart of PSO. At each time step, the behavior of a given particle is a compromise between three possible choices:

- To follow its own way
- To go towards its best previous position
- To go towards the best neighbor

This compromise is formalized by equations (1) and (2) and illustrated by fig. 2.

At time t, the position and the velocity of each particle are adjusted according to equations (1) and (2):

$$v_{ij}(t+1)=w.v_{ij}(t)+c_1.r_1.(p_{ij}(t)-x_{ij}(t))+c_2.r_2.(p_{ij}(t)-x_{ij}(t)) \quad (1)$$

$$x_{ij}(t+1)=x_{ij}(t)+v_{ij}(t+1) \;;\; j \text{ in } \{1, ..., D\} \quad (2)$$

Where w is the inertia weight, (c_1, c_2) are two positive constants, called cognitive and social parameters and (r_1, r_2) are random numbers uniformly distributed in [0, 1].

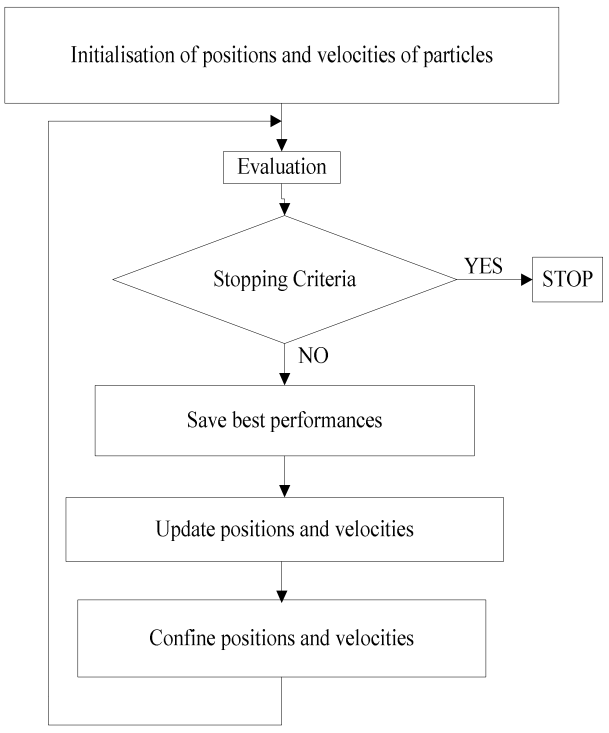
Initialisation of positions and velocities of particles
Evaluation
Stopping Criteria
YES
STOP
NO
Save best performances
Update positions and velocities
Confine positions and velocities

Fig. 1. PSO Flow Chart

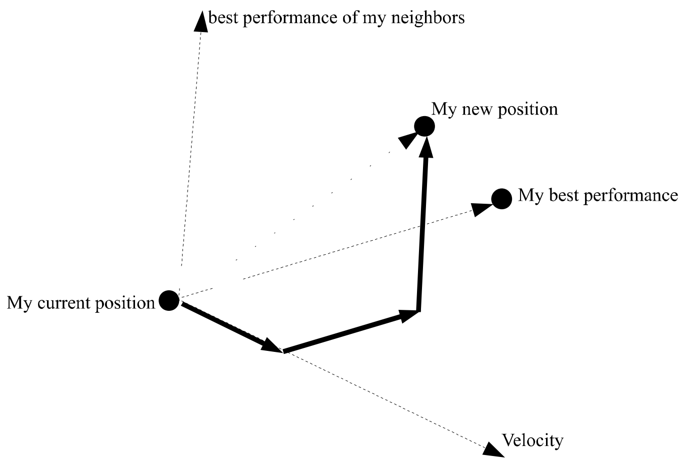

Fig. 2. An illustration of particle’s move

IV. PSO FOR FJSP

One of the key issues when designing the PSO algorithm lies on its solution representation which directly affects its feasibility and performance. The original PSO design is developed to solve continuous function. But FJSP is a combinatorial problem, the solution space is discrete. Therefore, the first issue is to find a suitable representation that let us avoid a correction process which will be very expensive in term of computation time. In our case, we implement a coding that takes into account all the constraints and the specifities of the problem. For the (n jobs, m machines, O operations) FJSP each particle is represented by a matrix having 4 rows and (2*O) columns. Fig. 3, Fig. 4 and Fig. 5 illustrate the solution representation of a particle corresponding to (3 jobs, 5 machines, 8 operations). The first and second halves of the first row of the particle (Fig. 3, Fig. 4) represent operations as a repetition of operations. For example (1,1,1) represents ($O_{1,1}$,$O_{1,2}$,$O_{1,3}$), (2,2,2) represents ($O_{2,1}$,$O_{2,2}$,$O_{2,3}$) and so on. The second row of the particle represents particle's position or the rational solution. Each dimension of the first half of the 2nd row maps one operation and each of the second half maps one machine. At this step, we use the Smallest Position Value (SPV) rule [3] to find the permutation of jobs. The smallest component of the particle's position in the 2nd row of Fig.4 is -2.1 which corresponds to job number 3, thus J_3 (or the first operation of J_3) is scheduled first. The second smallest component of the particle's position is -2 which corresponds to job number 1, therefore J_1 (or the first operation of J_1) is the second, etc. The 2nd row of Fig. 5 contains a random number in the interval [0,m] that indicates after being rounded to its nearest integer, the machine to which an operation is assigned during the course of TRIBES. The 3rd row of Fig. 4 indicates the sequence of jobs in the ordering and the 3rd row of Fig. 5 indicates the corresponding machines. Finally, the last row of Figure. indicates operations in the order and last row of Fig. 5 indicates starting times.

Repitition of jobs:$J_1J_1…J_2J_3J_3$	*Repitition of jobs:$J_1J_1…J_2J_3J_3$*
Position	*Position*
Sequence of Jobs	*Machines*
Sequence of operations	*Starting dates*

Fig. 3. Particle Representation

J_1	J_1	J_1	J_2	J_2	J_2	J_3	J_3
0.6	-2	6	2	6.6	3	2.5	-2.1
J_3	J_1	J_1	J_2	J_3	J_2	J_1	J_2
O_{31}	O_{11}	O_{12}	O_{21}	O_{32}	O_{22}	O_{13}	O_{23}

Fig. 4. The first half of the particle

J_1	J_1	J_1	J_2	J_2	J_2	J_3	J_3
4	0.5	2	0.6	4	0.9	3	2
M5	M1	M3	M1	M5	M1	M3	M3
t=0	T=0	t=1	t=1	t=2	t=2	t=3	t=4

Fig. 5. The Second half of the particle

V. TRIBES

Although metaheuristics are efficient, they suffer from a large number of algorithm-specific parameters which need to be optimized separately for each instance and problem type. To address this issue, a number of self-adaptive algorithms are developed. TRIBES, a metaphor for different sized groups of people moving in an unknown area or region, looking for a "good" place, is a parameter-free PSO [4]. TRIBES adapts the number of particles and the details of topology according to performance feedback of the swarm. The swarm in TRIBES is composed of sub-swarms called tribes. At the beginning, the swarm contains only one tribe containing one particle. Through generations the swarm in TRIBES evolves by structural and behavioral adaptations. Structural adaptation determines how particles are added or removed based on the behaviors of tribes and behavioral adaptation indicates how particles update their positions.

A. *Tribal relationships*

Each tribe represents a fully connected graph. Any particle in the swarm belongs to only one tribe. The set of informers of a particle forms its i-group. This i-group contains all the particles in the tribe of the considered particle and any particle of other tribes to which the particle at hand is connected. All links are symmetric (Fig. 6).

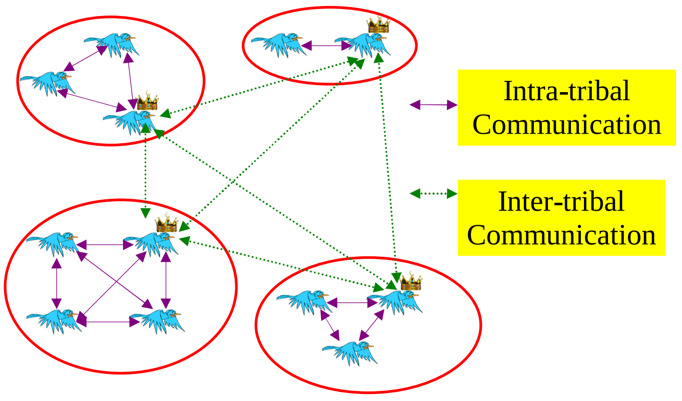

Fig. 6. Tribal relationships

B. *Structural adaptations*

As TRIBES algorithm evolves, if the performance of a particle was improved in the last iteration, the particle is labeled "good". Otherwise, it is labeled "bad". Relative to its tribe, the particle having the least good performance within a tribe is called the "worst". Similarly, we can determine the best particle of a tribe. The tribe also is labeled "good" if the number of its good particles exceeds the half of tribe size. Otherwise, the tribe is labeled "bad".

Good tribes do not need as many particles, therefore they will remove their weakest member and any external links to the particle to be deleted will be reallocated to the best performer in the tribe. Fig. 7 and Fig. 8 illustrates the deletion of a worst particle in a good tribe. If a good tribe contains only one particle, this good tribe will be removed only if its particle's best external informer has a better performance than its particle. In the case of removal of the whole tribe, all external links are reallocated to the best external informer of the particle to be deleted.

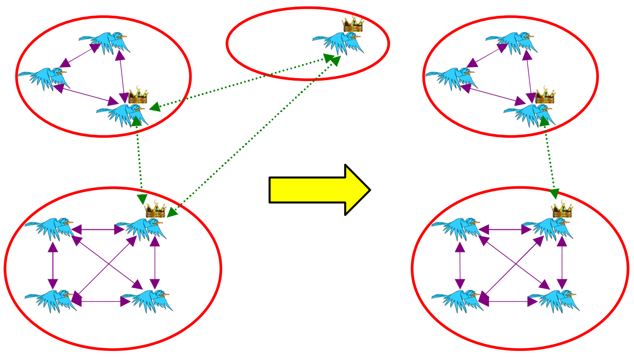

Fig. 7. Deletion from a mono-particle tribe

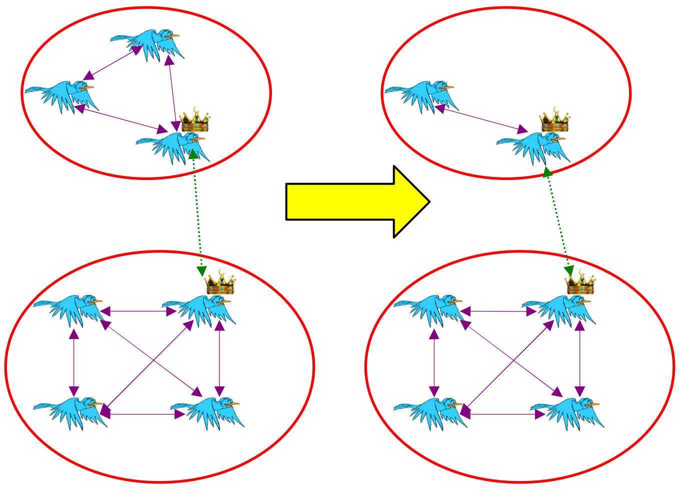

Fig. 8. Deletion from a multi-particle tribe

On the other hand, bad tribes need more information, that's why each bad tribe generates one particle at random and forms a link between its best particle and the generated one. At each structural adaptation step, the set of all new particles generated creates a new tribe.

Structural adaptation should not occur after each iteration. It takes place at the beginning of the algorithm and then periodically as it progresses. In fact, some time (iterations) are indispensable for information to propagate through the whole swarm. If after one structural adaptation the number of links in the swarm is NL, the next structural adaptation will occur after (NL/2) number of iterations. The algorithm that outlined Structural adaptation rules is available in [5].

C. *Behavioral adaptations*

In TRIBES, particles do not have explicit associated velocities. They update their positions according to their history. Each particle adopts a moving strategy based on its recent past. If a particle has just improved its position twice in succession, it is considered "excellent" and it will adopts the simple pivot method to adjust its position. Otherwise, positions of particles are updated using the noisy pivot method.

In the simple pivot method, two positions are used: the best position p of the given particle and the best position g of its best informer. Then two hyper-spheres of radius |p-g| are created around p and g. The new position is generated inside the intersection of the two hyper-spheres. In such way the newly generated point is most likely to be nearer to the best between p and g. In order to obtain such a behavior, two weights w_1 and w_2 proportional to the relative fitness of the considered particle and its best informer are generated and the new position is obtained by $w_1.h_p+w_2.h_g$, where h_p and h_g are two randomly generated points in the hyper-spheres surrounding p and g, respectively [5]. The noisy pivot method begins exactly like the simple pivot strategy. After having determined the new position it is again modified by adding a random noise.

Confinement in TRIBES is done as in classical PSO. In fact, if a component of the position of a particle tends to go out of the search space, it is then simply brought back to nearest acceptable values.

To summarize, the TRIBES flowchart is presented in the following figure (Fig. 9).

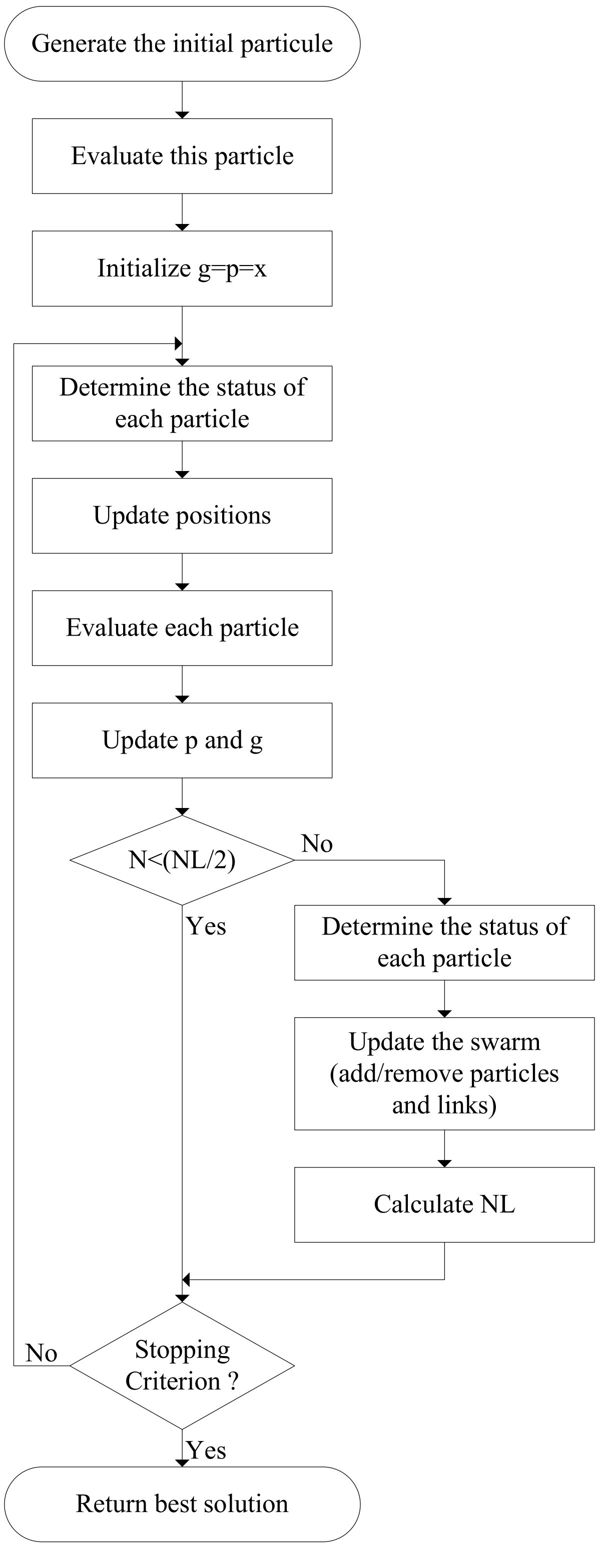

Fig. 9. Flow Chart of TRIBES

D. TRIBES for FSP

For combinatorial problems, we do not have a distance defined on them. Therfore, in moving strategies, we need to consider each dimension independently. For each dimension d, the new value of position component is computed as follows:

$$v_d=x(t)_d-x(t-1)_d \quad (3)$$

$$\delta p=p_{i,d}-x(t)_d \quad (4)$$

$$\delta g=g_{i,d}-x(t)_d \quad (5)$$

$$x(t+1)_d=x(t)_d+\chi(vd+rand\_gauss(\delta p,|\delta p|/2)) \quad (6)$$

$$x(t+1)_d=x(t)_d+\chi\ (rand\_gauss(\ \delta g,|\delta g|/2)) \quad (7)$$

Where $\chi=(1/(\varphi-1+(\varphi2-2*\varphi)1/2))$, $\varphi=2/0.97225$ and rand_gauss(m, s) function generates a number under the normal distribution with a mean m and a standard deviation s [4].

VI. RESULTS

This section described the computational tests which are used to evaluate the effectiveness and efficiency of PSO and TRIBES. Our algorithms have been implemented in C language and run on a PC Intel ® Core ™i3 CPU M370@2.40GHz 909 MHz, 1.85GB of RAM. Different instances of the present problem have been chosen to test PSO and TRIBES algorithms. These instances are taken from Kacem [7]. The non deterministic nature of our algorithms makes it necessary to carry out multiple runs on the same problem instance in order to obtain meaningful results. We run PSO and TRIBES ten times from different starting solutions. For PSO, we use the global variant that's why the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The parameters w was set to 0.729, c_1 was set to 1.49, c_2 was set to 1.49 and the swarm size was set to 50. Our solutions obtained either using PSO or TRIBES and those in literature are presented in Table I. This Table shows that experimental results are encouraging because the proposed algorithms are highly competitive to the state-of-the-art methods in the literature since PSO and TRIBES are able to find the optimal and best-known solutions on the instances studied.

TABLE I. EXPERIMENTAL RESULTS

Instance	**C_{max}**		
	Kacem[7]	***PSO***	***TRIBES***
Instance 1	5	5	5
Instance 2	11	11	11
Instance 3	11	11	11
Instance 4	7	7	7
Instance 5	11	11	11
Instance 6	17	17	17
Instance 7	16	16	16
Instance 8	6	6	6
Instance 9	14	14	14

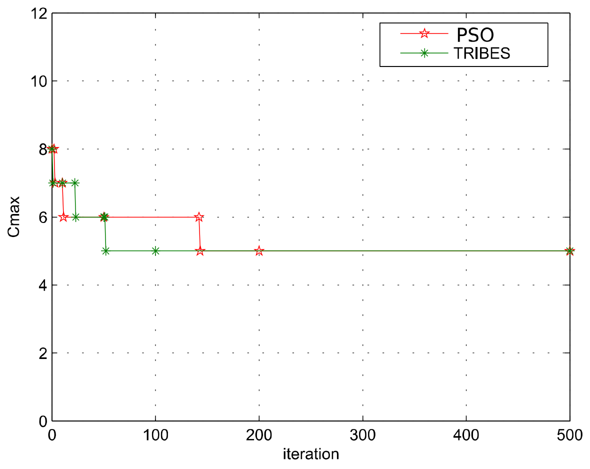

Fig. 10. Comparison of results of PSO & TRIBES (instance 1 of Kacem [7])

In order to compare our approaches based on PSO and TRIBES, we studied their convergences for minimizing Cmax criterion. Fig. 10 and Fig. 11 show that our approaches have led to the same results but with varying rates of convergence. Indeed, in Fig.10, convergence takes place at the 52th generation in the case of TRIBES at the 143th generation in the case of the PSO.

In Fig. 11, convergence takes place the 200th generation in the case of TRIBES, the 209th generation in the case of the PSO.

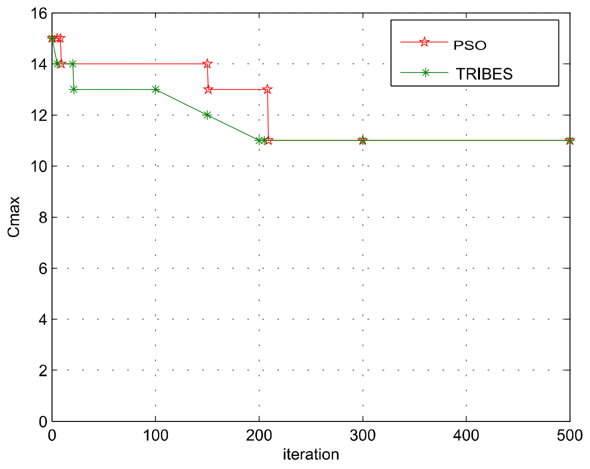

Fig. 11. Comparison of results of PSO & TRIBES (instance 3 of Kacem [7])

VII. Conclusion

The results obtained from the computational study highlight the effectiveness and efficiency of the proposed approaches. A more comprehensive study on a large number of instances should be made to test the efficiency of the proposed solution techniques. If you want a simple and easy implemented algorithm, you can choose particle swarm optimization algorithm. If you want to overcome the difficulty of parameter tuning present in all metaheuristics, you may choose TRIBES. Further investigation is needed to fully reveal the ability of TRIBES in tackling scheduling problems and solving other optimization problems. Future research should pay more attention to the hybridization of TRIBES and other metaheuristics in order to benefit from advantages of each algorithm.

References

[1] Carlier and P. Chrétienne. *Problèmes d'ordonnancement Modélisation/complexité/algorithmes*. Edition Masson, Paris, France, 1988.

[2] J. Kennedy and R.C. Eberhart. *Particle Swarm Optimization*. IEEE International Conferenceon Neural Networks (Perth, Australia), IEEE Service Center, Piscataway, NJ, IV, 1995, pp.1942-1948.

[3] S. mekni, B. Fayech Chaâr and M. Ksouri. *A novel particle swarm optimization for the multi-objective flexible job shop scheduling problem*. International Conference on Informatics and Control, Automation and Robotics (ICINCO 2008), 2008, pages 225-230.

[4] M. Clerc. *L'optimisation par essaims particulaires, versions paramétriques et adaptatives*, Edition Hermès, Lavoisier ; 2005.

[5] Y. Cooren. *Perfectionnement d'un algorithme adaptatif d'optimisation par essaim particulaire*. Applications en génie médical et en électronique, 2008, Thesis, Paris 12 University.

[6] S. Mekni, B. Fayech Chaâr and M. Ksouri. *TRIBES application to the flexible job shop scheduling problem.* IMS 2010 10th IFAC Workshop on Intelligent Manufacturing Systems,Lisbon, Portugal, July 1st -2nd 2010.

[7] I. Kacem, S. Hammadi and P. Borne. *Approach by localization and multiobjective evolutionary optimization for flexible job shop scheduling problems*. IEEE Trans Systems, Man and Cybernetics, Vol 32, 2002, pp 245-276.