Journal of Advances in Computer Research
Quarterly ISSN: 2008-6148
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 4, No. 3, August2013), Pages: 69-81
www.jacr.iausari.ac.ir

Solving Flexible Job-Shop Scheduling Problem using Hybrid Algorithm Based on Gravitational Search Algorithm and Particle Swarm Optimization

Behnam Barzegar[1 *], Homayun Motameni[2]

1) Department of Computer Engineering, Nowshahr Branch, Islamic Azad University, Nowshahr, Iran
2)Department of Computer Engineering, Islamic Azad University, Sari Branch, Sari, Iran

barzegar@iauns.ac.ir; motameni@iausari.ac.ir

Received: 2013/07/04; Accepted: 2013/08/14

Abstract

Job Shop scheduling problem has significant importance in many researching fields such as production management and programming and also combined optimizing. Job Shop scheduling problem includes two sub-problems: machine assignment and sequence operation performing. In this paper combination of particle swarm optimization algorithm (PSO) and gravitational search algorithm (GSA) have been presented for solving Job Shop Scheduling problem with criteria of minimizing the maximal completion time of all the operations, which is denoted by Makespan. In this combined algorithm, first gravitational search algorithm finds best mass with minimum spent time for a job and then particle swarm Optimization algorithm is performed for optimal processing all jobs. Experimental results show that proposed algorithm for solving job shop scheduling problem, especially for solving larger problem presents better efficiency. Combined proposed algorithm has been named GSPSO.

Keywords: *Gravitational search algorithm, Particle Swarm Optimization algorithm, Flexiblejob shop scheduling*

1. Introduction

Scheduling has significant importance in manufacturing and industrial services and most often has been used in many cases such as production scheduling, transportation, provisions, communication and information process. Thus, scheduling problem has been under care of many researchers. In this paper, Job Shop Scheduling problem has investigated and its aim is performing all jobs in minimum run time. For suitable scheduling jobs, we should consider that one operation could be performed on set of machines and for doing so, JSP should solve two following sub-problem:

1) Assigning a suitable machine from set of machines to each operations.
2) Sequencing each operation on the selected machine.

Many of well-known scheduling problems are among NP-hard problems. It means that by increasing value of problem parameters, too much time is spent for offering an optimal solution, and sometimes solving problem would be impossible. Solving this kind of problems has encouraged many researchers to find an optimal solution for minimizing operation run time and also has decreased complexity of these kinds of

problems. Job Shop Scheduling problem is among NP-hard problems and we would solve this problem using our proposed combined algorithm called GSPSO.In this paper we have used combination of PSO and GSA algorithms for finding optimal solution in order to decrease operation run time.

In this method, first gravitational search algorithm finds best mass, i.e. best machine arrangement (minimum spent time) and then particle swarm algorithm processe it to minimize the maximal completion time of all the operations, which is denoted by Makespan.

2. Related work

The job-shop scheduling problem (JSP) has been studied for more than 50 years in both academic and industrial environments and also recently, many researchers have been done it for the flexible job-shop scheduling problem (FJSP).

Bruker and Schlie *et al* [2] who first considered this problem, offered a multilateral algorithm for solving flexible job shop problem with two jobs. In real world, for solving a problem with more than two jobs, two perceptions have been used: *hierarchical perception and integrated perception.*

In hierarchical perception, assigning any operation to the machines and determining operation sequences are performed individually. In other words, assignment and sequence determination are independent. But in integrated perception, sequence determination is based on this idea that in order to decreasing complexity, main problem should be decomposed into two problems called assignment and sequence determination. As this perception decomposes into two problems of assignment and sequence determination, is used more. Brandimarte *et al* [1] was the first one who used this perception for FJSP. He specified path determination with distribution rules and then focused on solving scheduling problem with TS algorithm. Xia and Wu *et al* [3] have presented a hybrid optimizing perception for scheduling multi object *flexible job shop* problems. In their study, combination of two methods SA and particle swarm optimization have been used for optimizing flexible job shop problem. PSO algorithm is applied for assignment problem either for determining any operations use whit machine. Value of object function is calculated by SA algorithm and implemented for each particle in PSO algorithm once.

Mastrolilli and Gambardella *et al* [4] proposed a tabu search procedure with effective neighborhood functions for the flexible job shop problem. Many authors have proposed a method of assigning operations to machines and then determined sequence of operations on each machine. Pezzella *et al* [5] and Gao *et al* [6] proposed the hybrid Genetic and variable neighborhood descent algorithm for this problem. There are only a few papers considering parallel algorithms for the FJSP.

Yazdani *et al* [7] propose a parallel variable neighborhood search (VNS) algorithm for the FJSP based on independent VNS runs. Defersha and Chen *et al* [8] describe a coarse-grain version of the parallel genetic algorithm for the considered. FJSP basing on island model of parallelization focusing on genetic operators used and scalability of the parallel algorithm. Both papers are focused on parallelization side of the programming methodology and they do not use any special properties of the FJSP.

The rest of the paper is as following: First, problem analyzing and in second section, particle swarm optimizing algorithm are presented. In third section, gravitational search algorithm is explained and finally in four section, we explain proposed combined

method. Also In five section, experimental results and in six section, conclusion is presented.

3. Flexible job-shop scheduling problem

In this section, mathematical model (combined of integer and linear programming) is presented for better understanding problem and also applying it for solving small problems optimally.

Flexible job shop scheduling problem contains N job on M machines. Each job includes some operations and for each operation there is an opportunity to use set of operational machines. Since flexible job shop scheduling problem has specific importance in production centers, it attains large attention from managers of production units.

Furthermore, specific mathematical characteristics of this problem that have offered effective strategies for solving this problem are interesting for researchers of this area of mathematics field.

The job-shop scheduling problems with multi-purpose machines (MPM job-shop problem) may be formulated as follows.

Simple form of flexible job shop scheduling production systems is classic job shop scheduling problem which schedules n job of J_1, $J2,\ldots,J_n$ on set of M machines of $M_1,M_2,\ldots,M_m$.

Each job has h_j operation that must be implemented serially. Subscript j indicates job, subscript h indicates operation and subscript i presents machines. The purpose of scheduling this problem is to determine the sequence of operations for each machine, so that a predefined object function like construction duration gets optimized.

Each job has one sequence of $O_{j,h}$ operations; $h=1,\ldots,h_j$ where $O_{j,h}$ presents *h-th* operation of *j-th* job, and h_j presents number of required operations for *j*-th job. Machines set is presented by $M = \{M_1, M_2\ldots, M_m\}$. Subscript i presents machine and subscript j presents job and subscript h is applied for operation.

To implement each h operation on j job (presented as $O_{j,h}$), set of jobs are assigned, which have capacity of performing that operation.

This set is presented as$M_{j,h} \subset M$. Each machine would have specific process time for implementing operation. This specific process time for implementing each operation is presented with $P_{i,j,h}$.

In this study, we define $M_{j,h}$ set with variable $a_{i,j,h}$ with value one and zero. If variable $a_{i,j,h}$ has value 1, it means that machine j has capacity for implementing operation $O_{j,h}$. For assignment, we use variable $y_{i,j,h}$ with value one or zero. This variable is determined by model. If this variable has value 1, it means that among operational machines for implement $O_{j,h}$ operation, machine j is selected.

Eventually, result solution from variable $y_{i,j,h}$ gives assignment problem solution (i.e. each operation among assignable machine is performed by which machine).

For solving sequence problem, we consider initial time $t_{k,l}$ and final time $ft_{k,l}$ for each operation. Value of those variables is determined by model. Moreover, an assumed job whose number of its operations is equal to number of machines is considered as initial job.

In this model, we use variable $x_{i,j,h,k,l}$ with value one or zero. If this variable has value 1, it means that operation $O_{k,l}$ on machine j is implemented immediately after operation $O_{j,h}$. Also, $se_{i,f,k}$ presents start up time of job k after a job from family f on machine i.

$$F_{f,j} = \begin{cases} 1 & if\ k \in f \\ 0 & Other\ wise \end{cases}$$

$$a_{i,j,h} = \begin{cases} 1 & if\ O_{j,h} can\ be\ performed\ on\ machine\ i \\ 0 & Other\ wise \end{cases}$$

Variables of this model include:

$$y_{i,j,h} = \begin{cases} 1 & if\ machcie\ i\ selecte\ for\ operation\ O_{j,h} \\ 0 & Other\ wise \end{cases}$$

$$x_{i,j,h,k,l} = \begin{cases} 1 & if\ \ O_{j,h} presedes\ O_{k,l}\ immediately\ on\ machine\ i \\ 0 & Other\ wise \end{cases}$$

C_{max} : Maximum time of constructing duration

m : A large number

$t_{k,l}$: Initial time for operation $O_{k,l}$

$ft_{k,l}$: Final time for operation $O_{k,l}$

$P_{i,k,l}$: Process time for operation $O_{k,l}$ on machine i

$S_{i,j,k}$: Start up time for job k on machine j if previous job be job j.

With having parameters $S_{i,f,k}$, $P_{i,j,h}$, $a_{i,j,h}$, fa ,m ,n problem FJSP is modeling as follows:

(1) minC_{max}

(2) $t_{k,l} + y_{i,k,l}.p_{i,k,l} \leq fi_{k,l}$ *for* $i=1,\ldots,m$ $k=1,\ldots,n$ $l=1,\ldots,h_k$

(3) $s_{i,j,k} = \sum F_{f,j}.se_{i,f,k}$ *for* $i=1,\ldots,m$ $j=1,\ldots,n$ $k=1,\ldots,n$ $f=1,\ldots,fa$

(4) $fi_{k,l} \leq t_{k,l+1}$ *for* $k=1,\ldots,n$ $l=1,\ldots,h_k-1$

(5) $fi_{k,l} \leq C_{max}$ *for* $k=1,\ldots,n$ $l=1,\ldots,h_k$

(6) $y_{i,k,l} \leq a_{i,k,l}$ *for* $i=1,..,m$ $k=1,\ldots,n$ $l=1,\ldots,h_k$

(7) $t_{j,h} + p_{i,j,h} + s_{i,j,k} \leq t_{k,j} + (1-x_{i,j,h,k,l})M$ *for* $j=0,\ldots,n$ $k=1,\ldots,n h=1,\ldots,h_j$ $l=1,\ldots,h_k i=1,\ldots,m$

(8) $f_{j,h} + s_{i,j,k} \leq t_{j,h+1} + (1-x_{i,k,l,j,h+1})M$ *for* $j=1,\ldots,n$ $k=0,\ldots,n$ $h=1,\ldots,h_j-1$ $l=1,\ldots,h_k i=1,\ldots,m$

(9) $\sum y_{i,j,h} = 1$ *for* $j=0,..,n$ $h=1,\ldots,h_j$ $i=1,\ldots,m$

(10) $\sum\sum x_{i,j,h,k,l} = y_{i,k,l}$ *for* $i=1,\ldots,m$ $k=1,\ldots,n$ $l=1,\ldots,h_k$

(11) $\sum\sum x_{i,j,h,k,l} = y_{i,k,l}$ *for* $i=1,\ldots,m$ $j=1,\ldots,n$ $h=1,\ldots,h_j$

(12) $x_{i,j,h,k,l} \leq y_{i,k,l}$ *for* $j=1,..,n$ $k=1,\ldots,n$ $h=1,\ldots,h$ $l=1,\ldots,h_k i=1,\ldots,m$

(13) $x_{i,j,h,k,l} \leq y_{i,k,l}$ *for* $j=1,..,n$ $k=1,\ldots,n$ $h=1,\ldots,h_j$ $l=1,\ldots,h_k i=1,\ldots,m$

(14) $x_{i,k,l,k,l} = 0$ *for* $i=1,..,m$ $k=1,\ldots,n$ $l=1,\ldots,h_k$

(15) $s_{i,k,k} = 0$ for i=1,..,m k=1,…,n

(16) $x_{i,j,h,k,l}, y_{i,j,h} \in \{0,1\}$

Constraint 1 is the object function of the problem which minimizes maximum completion time. Constraint 2 presents startup time and finishing time of each operation. Constraint 3 introduces run time for each job. Constraints 4 and 8 cause those pre-requirement limitations which are respected. Constraint 5 defines C_{max}.

Constraint 6 causes that required machines for each operation are selected among the assignable machines for that operation. Constraint 7 guarantees that if operation l from job k is performed after operation h from job j on machine i, its startup time is after finishing operation h from job j and also after process time of preparing machine i.

Constraint 9 causes that among all assignable machines for a specific operation, just one machine is selected. Constraints 10 and 11 state that just one operation after and before other operations are performed on machine *i*.

Constraints 12 and 13 state that each operation after and before other operations is performed just on its assignable machine. Constraint 14 guarantees that any operation is processed once.

Table1 shows case of performing 3 jobs on 5 machines. Each job includes three following operations and Figure1 shows another case of performing 3 jobs (with three sub-operations) on 3 machines and each run time is known as Gunt diagram.

*Table 1. Illustration of an example of 3 * 5 problem*

		M_1	M_2	M_3	M_4	M_5
J_1	$O_{1,1}$	1	9	3	7	5
	$O_{1,2}$	3	5	2	6	4
	$O_{1,3}$	6	7	1	4	3
J_2	$O_{2,1}$	1	4	5	3	8
	$O_{2,2}$	2	8	4	9	3
	$O_{2,3}$	9	5	1	2	4
J_3	$O_{3,1}$	1	8	9	3	2
	$O_{3,2}$	5	9	2	4	3
	$O_{3,3}$	4	5	7	1	6

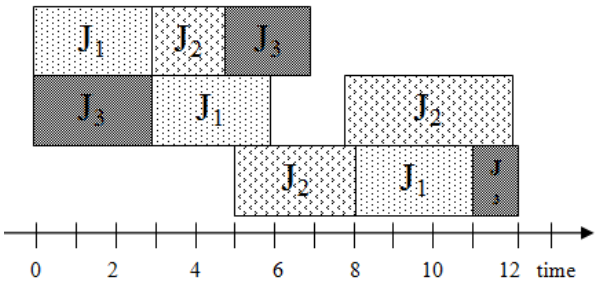

*Figure 1. A Gantt-Chart representation of a solution for a 3 * 3 problem*

4. Particle swarm optimization

In particle swarm optimization method, each proposed solution for respected problem that is called a particle, is considered as a point in search space.

There is one-to-one correspondence between space points and those vertices which are started from coordinate origin. Thus, we display position of particles as vertex. In each stage, obtained set of solution changes and we try to find better solutions for the respected problem.

If we consider searching space as a D-dimensional space, then we could display position of *i-th* particle as X_i and its speed asV_i. Eventually, movement of bird populations is obtained with two following equations:

$$V_{id}(t+1) = W_n \times V_{id}(t) + c_1 r_1\left(lbest_{id}(t) - X_{id}(t)\right) + r_2\left(gbest_{id}(t) - X_{id}(t)\right) \quad (1)$$

$$X_{id}(t+1) = X_{id}(t) + V_{id}(t) \quad (2)$$

Where $d = 1,2,\dots,D$ and $i = 1,2,\dots,N$ and N is equal to size of birds population, n is irritation number, W is inertia mass, c_1 and c_2 are respectively cognitive and social

coefficients which usually are selected in range of $[0,2]$ and r_1 and r_2 are random numbers which usually are selected in range of $[0,1]$.

Inertia mass parameters provide suitable distribution for speed in searching local and absolute optimal points. Selecting low values for these parameters provide local points, and choosing high values for them search a larger area, which in fact helps choosing absolute optimal points. In this paper, we first consider high value for this parameter (W) and gradually decrease this value while running program (Shi and Eberhart*et al* [14,15]).

$$W_n = W_{max} - \frac{(W_{max} - W_{min}) \times n}{iTer_{max}} \quad (3)$$

W_{min} and W_{max} are respectively initial and final values of inertia weight.

(W), $iTer_{max}$ is maximum number of stages and n is the number of current stage. As there is no process for controlling speed of current birds, it is necessary to consider a maximum value for it (V_{max}) which no bird should exceed of this value (Meraji and Afshar *et al* [16]).

Birds change their speed based on the best response that has obtained till now $(lbest)$ and the best response in current generation$(gbest)$. This speed is sum with bird position and so new position of bird is obtained. If current best position be better than best global position, this best current position is replaced with best global position. This process is repeated according to the size of bird populations.

5. Gravitational search algorithm

In GSA, optimization is done using gravitational rules and movement rules in an artificial discrete-time system.

System area is the same as problem definition area. According to gravitational rule, act and state of other masses are recognized through gravitational forces. So, this force could be used as a tool for transferring information. We can also use proposed solution for solving any optimization problem within which any answer of problem is within definable as a state in space and its degree of similarity with other answers of problem is mentioned as a distance. Value of masses in each problem is also mentioned in regards to purpose function. In first step, system space is determined. Area includes a multi-dimensional coordinated system in problem definition space. Each point in space is one of the answers of problem and search factors are also series of masses.

Each mass has three properties:

a) Mass state, b) Gravitational mass, c) Inertia mass.

Abovementioned masses are resulted from active gravitational mass and Inertia mass concepts in physics.

In physics, active gravitational mass is criteria of degree of gravitational force around a body, and Inertia mass is criteria of body resistance against movement. These two properties could not be equal, and their amounts are determined based on the suitability of each mass. Mass state is a point in space which is one of the problem answers. After forming system, its rules are determined.

We suppose that there are only gravity rule and movement rule. Their general forms are similar to nature rules and have defined as below:

Gravity Rule: Any mass in an artificial system attracts all other masses toward itself. The value of this force is proportional with gravitational mass of related mass and distance between two masses.

Movement Rule: Recent speed of each mass is equal to sum of the coefficient of last speed of that mass and its variable speed. Also, acceleration or variable speed is equal to delivered force on mass, divide on amount mass.

In following, we explain principals of this algorithm: Suppose that there is a system with S masses and within it, state of mass *i-th* is defined as relation (1), where x denotes position of mass *i-th* in dimension d and n denotes number of dimensions in the search space.

$$X_i = \left(x_i^1, \dots, x_i^d, \dots, x_i^D\right) \quad (4)$$

Worst (t) and Best (t) are for minimization problems and are calculated with relations (5) and (6).(For maximization problem is just enough to consider the inverse of these two relations)

$$\mathrm{Best}(t) = \max_{j\in\{1,\dots,m\}} fit_j(t) \quad (5)$$

$$\mathrm{Worst}(t) = \min_{j\in\{1,\dots,m\}} fit_j(t) \quad (6)$$

We can account fitness of recent population with relation (7), and obtain mass of factor *i-th* in time t (i.e. with relation (8)), where *M* and *fit* are denote mass and fitness of factor *i-th* in time t, respectively.

$$q_i(t) = \frac{fit_i - Worst(t)}{Best(t) - Worst(t)} \quad (7)$$

$$M(t) = \frac{q_i(t)}{\sum_{j=1}^{s}(t)} \quad (8)$$

In this system, force F is delivered on mass *i-th* from mass *j-th* in time t in the direction of dimension d, in which value of this force is obtained base on relation (9), And in relation (9), G(t) is gravity constant in time t which is regulated in the beginning of operating algorithm, and is decreased by the time.

$$F_{ij}^d(t) = \frac{G(t) - M_j(t)}{R_{i,j}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right) \quad (9)$$

R is *ECLIDIAN* distance between factor *i-th* and factor *j-th* that is defined as relations (10)," ε " is also a small value for avoiding denominator from becoming zero.

$$ij = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 + \cdots (n_2 - n_1)^2} \quad (10)$$

The force delivered on mass *i-th*in direction *d* at time *t* is equal to resultant of total force from *k* superior mass in population (*k* is better factor than recent factor).

Kbest denotes series of *k* superior masses in population. *K* value is not constant and is defined as a time-dependant value, such that all masses at the beginning influence on each other and deliver force, but by passing time, number of effective members in population is decreased linearly. And for accounting sum of delivered forces on mass *i-th* in dimension *d*, we could write (11). In this relation, *rand* is a random number with normal distribution in the interval [0,1].

$$F_i^d(t) = \sum_{j\epsilon K_{best}, j\neq i} \mathrm{rand}\, j \times G(t) \frac{M_j(t) - M_i(t)}{R_{ii}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right) \quad (11)$$

According to Newton's second movement rule; each mass takes acceleration in the direction of dimension *d*, which is proportional with delivered force on that mass, and has mentioned in relation (12).

$$d_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \Longrightarrow$$

$$d_i^d(t) = \sum_{j\epsilon K_{best}, j\neq i} \mathrm{rand}\, j \times G(t) \frac{M_j(t)}{R_{ii}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right) \quad (12)$$

And speed of each mass is equal to sum of coefficient of mass recent speed and acceleration, and is explained as relation (13). In this relation, rand is a random number with normal distribution in the interval [0,1], and its random property is resultant of keeping search in random mood.

$$V_i^d(t+1) = rand_i \times V_j^d(t) + a_i^d(t) \quad (13)$$

Now, mass should moves. It is obvious that more speed of the mass cause more movement in that dimension. New state of factor *i-th*is mentioned by relation (14).

$$x_i^d(t+1) = x_i^d(t) + V_j^d(t+1) \quad (14)$$

At the beginning of forming system, each mass (factor) is randomly positioned in one point of space that is an answer of problem. In each moment, masses are evaluated and then changing in the position of each mass is calculated after solving relations 11 to 14. System parameters are updated in each stage (G, M).

Stop condition could be determined after passing specified time. In Figure2, semi-code of this algorithm has been presented: (Rashedi and Nezamabadi-pour and Saryazdi *et al* [13]).

1) determining system area and initial valuing;

2) initial positioning the masses;

3) evaluating masses;

4) updating parameters G, best, worst and M ;

5) calculating delivered force on each mass;

6) accounting acceleration and speed of each mass;

7) updating position of masses;

8) if stop condition doesn't meet, go to phase3.

Figure 2.Semi-code of gravitational search algorithm

6. Proposed combined method

In our proposed algorithm, we first assign an array of sub-operation to each bird in PSO searching space and place some mass in GSA gravitational space, such that each mass in GSA searching space has an array which consists of number of selected machines. It means that by using GSA, we assign the best machine to one bird for performing each sub-operation and repeat this process for all birds by putting sub-operations in one circle. Finally, in return to PSO algorithm we obtain optimal arrangement for jobs.

This method first guarantees that each bird accepts sub-operation of several jobs, but one sub-operation couldn't be initialized for several birds. Position of each bird in space is determined randomly.

We fill k_best array with initial value of -1. We use following sample code for obtaining superior factors, which apply force on a mass from different dimensions (Figure 3).

```
while (k < mass_Num)
{
   for(j = 0;j < mass_Num-1 ; j++)
k_Best[j] = -1;
   for(i=0; i <= mass_Num-1; i++)
      {
!if (gls_Loc_Arr[k]) = gls_Loc_Arr[k])
{
if (k_Best[i] == -1)
{
                  k_Best[i]= gls_space[gls_Loc_Arr[k]];
 break;
}
}
    }
}
```

Figure 3.calculate k_best array

Based on applied force by each factor on respected mass, GSA should be calculated (sum of force and distance).

We could use the following pseudo-code for these calculations (Figure 4).

```
l = 0;
while (k_Best[l] != -1){
)if (k_Best[l] > 0
k_Best_Temp = k_Best[l] - 1;
else
k_Best_Temp = 0;
R=(Math.Sqrt((Math.Pow((gls_Loc_Arr[k_Best_Temp] –
gls_Loc_Arr[k]),2) +
Math.Pow((gls_Loc_Arr[k_Best_Temp] –gls_Loc_Arr[k]), 2)));
f_Arr[k] = f_Arr[k] + ((rand_obj.Next(100)/100.0) *G*
       (Math.Abs((gls_Hiu[k_Best_Temp] - gls_Hiu[k])) /
            * Math.Abs(gls_Loc_Arr[ k_Best_Temp] –(R+E))
          gls_Loc_Arr[k]));}
```

Figure 4.calculate of Force

To obtain fitness in GSA algorithm function, we calculate sum of the spent time on machines for each particle in searching space, which is known as heuristics.

Now, new parameters of acceleration speed and distance should be updated (Figure 5).

```
a_Mass = f_Arr[k] / gls_Hiu[k];
v_Arr[k] = ((rand_obj.Next(100)/ 100.0)*v_Arr[k]) +a_Mass;
new_loc = (gls_Loc_Arr[k] + v_Arr);
```

Figure 5.calculate of distance

The important point is that after finding optimal arrangement of machines for a bird, we assign those machines to the bird in PSO space. For this purpose, we could use following pseudo-code and an intermediate array called PSO_Machine_Swarm (Figure 6).

```
for (i = 0;i<mass_Num; i++)
{
  pso_Machin_Swarm[min_Mass_Index,i]=
                        mass_Arr[min_Mass_Index,i];
}
```

Figure 6.PSO_Machine_Swarm

In PSO algorithm, fitness of each bird should be used in appropriate with performed sub-operation on machines and so we obtain sum of spent time on machines. For this purpose, we consider the below code (Figure 7):

```
for (i=0;i< swarm_Num; i++)
{
for (int j=0;j< act_Num; j++)
{
hiu_Index=swarm_Arr[i, j];
   machin_No = pso_Machin_Swarm[i, j];
    act_No = j;
   hiu_Swarm[hiu_Index]=hiu_Swarm[hiu_Index]+
machin_Act_Time[machin_No, act_No];
   }
}
```

Figure 7.calculate of Fitness

In fitness function, we should determine Ibestand gbest. Following pseudo-code is presented for this purpose (Figure 8):

```
for (int i=0; i<swarm_Num; i++)
{
if (hiu_Swarm[i]< l_Best[i])
{l_Best[i]=hiu_Swarm[i];
lBest_Loc[i]=loc_Arr[i];
lBest_Loc[i]=loc_Arr[i];
}
if (hiu_Swarm[i] < g_Best)
  {  g_Best=hiu_Swarm[i];
    }
}
```

Figure 8.calculate of lbest and gbest

Now we should implement related formula for PSO (Figure 9).

It should be noted that applied machines for each sub-operation ought to be changed in appropriate with birds move in their own place (row of machine array), but birds don’t exit from optimal found state in GSA algorithm.

```
count++;
Wn = Wmax - (((Wmax - Wmin) * count) / iTermax);
r1=(rand_Obj.Next(10)/10.0);
r2=(rand_Obj.Next(10)/10.0);
loc = 0;
for (i=0; i < swarm_Num; i++)
{
v_Temp= ((Wn * v_Arr + (c1 * r1 *
            (Math.Abs(lBest_Loc[i] - loc_Arr))) +
(c2 * r2 * (Math.Abs(side_X - loc_Arr))));
if (v_Temp > v_max)
v_Arr = v_max;
else
v_Arr = v_Temp;
}
```

Figure 9.implement related formula for PSO

Table 2. Experimental results

	Edata						***rdata***						***vdata***					
Data	***HJT***	***DP***	***MG***	***MX***	***GSFJ***	***CPU***	***HJT***	***DP***	***MG***	***MX***	***GSFJ***	***CPU***	***HJT***	***DP***	***MG***	***MX***	***GSFJ***	***CPU***
101	611	609	609	609	609	35.6	574	574	571	574	574	48.8	573	572	570	572	573	54.1
102	655	655	655	655	655	3.1	535	532	530	535	533	64.7	531	529	529	529	529	44.3
103	573	554	550	563	557	25.2	481	479	478	478	478	82.4	482	479	477	479	482	50.7
104	578	568	568	568	569	75.5	509	504	502	509	507	49.8	504	503	502	502	502	67.6
105	503	503	503	503	503	4.1	460	458	457	457	458	43.4	464	460	457	460	460	81.9
106	833	833	833	833	833	45.9	801	800	799	799	800	94.2	802	800	799	802	802	12.7
107	765	765	762	765	765	37.3	752	750	750	750	750	86.5	751	750	749	750	750	36.3
108	845	845	845	845	845	52.2	767	767	765	769	767	54.3	766	766	765	766	766	68.7
109	878	878	878	878	878	59.4	859	854	853	853	855	104.5	854	853	853	853	853	78.2
110	866	866	866	866	866	54.6	806	805	804	804	805	98.7	805	805	804	807	805	12.8
111	1106	1103	1103	1103	1104	119.6	1073	1072	1071	1071	1071	113.3	1073	1071	1071	1071	1071	171.6
112	960	960	960	960	960	80.2	937	936	936	936	936	122.5	940	936	936	936	936	211.1
113	1053	1053	1053	1053	1053	89.7	1039	1038	1038	1038	1038	109.4	1040	1038	1038	1038	1040	127.8
114	1151	1123	1123	1123	1123	50.1	1071	1070	1070	1070	1070	61.1	1071	1070	1070	1070	1070	191.9
115	1111	1111	1111	1111	1111	134.3	1993	1990	1990	1990	1990	119.6	1091	1089	1089	1089	1089	257.4
116	924	915	892	892	895	44.2	717	717	717	717	717	14.9	717	717	717	717	717	3.5
117	757	707	707	707	707	93.0	646	646	646	646	646	28.8	646	646	646	646	650	3.9
118	864	843	842	850	844	77.8	674	669	666	674	672	44.1	663	663	663	663	663	3.6
119	850	796	796	796	797	100.2	725	703	700	715	710	33.3	617	617	617	617	617	4.5
120	919	864	857	865	862	186.6	756	756	756	756	756	82.3	756	756	756	756	756	2.8
121	1066	1046	1017	1046	1041	133.4	861	846	835	856	846	182.6	826	814	806	836	825	172.1
122	919	890	882	918	910	99.1	790	772	760	784	779	110.2	745	744	739	744	743	264.6
123	980	953	950	972	970	101.1	884	853	842	853	851	199.7	826	818	815	826	822	214.4
124	952	918	909	918	915	152.7	825	820	808	825	824	161.1	796	784	777	784	783	299.5
125	970	955	941	970	968	124.2	823	802	791	823	818	135.4	770	757	756	757	757	312.7
126	1169	1138	1125	1156	1141	179.5	1086	1070	1061	1081	1076	214.8	1058	1056	1054	1058	1057	260.0
127	1230	1215	1186	1230	1213	135.4	1109	1100	1091	1106	1110	260.0	1088	1087	1085	1087	1089	155.3
128	1204	1169	1149	1169	1152	210.0	1097	1085	1080	1097	1091	178.8	1073	1072	1070	1073	1073	212.1
129	1210	1157	1118	1178	1163	212.6	1016	1004	998	1008	1007	155.5	995	997	994	995	997	438.8
130	1253	1225	1204	1225	1224	276.1	1105	1089	1078	1089	1085	270.9	1071	1071	1069	1071	1071	355.8

7. Experimental results

The C#.Net 2008 language programming was used for testing the proposed algorithm and in this paper we have used three different sets of computational experiments in order to evaluate the efficiency to test our new proposed algorithm. We have used the Intel Pentium Core i5 Duo 2.4GHz Processor and 4GB RAM configuration system with Windows XP as the platform to run this algorithm and achieved the following results.

Consider first MPM job-shop instances with related machines generated in Hurink *et al* [9]. These instances are derived from the classical job shop instances, proposed in (Lawrence, [12]), by adding some machines to candidate sets of operations. More precisely, each set $F_{i,j}$ contains the machine used in the original problem, plus any of the other machines with a given probability. With different values of the probability, three sets of instances of different degree of flexibilities are generated. Each operation is associated on average with (m/2) machine in a *vdata* instance, with two machines in a *vdata* instance, and with less than two machines in a*edata* instance.

The new method denoted GSPSO is compared with the hierarchical methods of (Hurink *et al* [9]) hereafter denoted HJT and Dauzere-Peres and integrated approaches of Dauzere-Peres and Paulli [11] (denoted DP) and Mastrolilli and Gambardella [4] (denoted MG) and Mati and Xie [10] (denoted MX). Table 2 provides the results of this comparison. Column CPUs presents the computation times of the new method.

8. Conclusion

In this paper, a proposed algorithm called GSPSO has used for solving flexible Job Shop Scheduling problem, which is a combination of PSO and GSA algorithms. According to the obtained results, this algorithm has an acceptable ability in decreasing run time of all jobs and we could use it in different scheduling problems.

Comparing results of proposed algorithm with other algorithms we realize that this algorithm most of the times offers better results. In very low populations, proposed algorithm finds solution with some delay, but in higher population based on amount of calculations, quality of solutions would be ideal. Also, suitable values forc_1 and c_2 learning coefficients are in the range of$[0.5,1.75]$. First, we consider high value for inertia weight coefficient and gradually decrease this value while performing algorithm to its minimum value.

9. References

[1] Brandimarte ,Routing and scheduling in flexible job shop by tabu search. Annals of Operation Research. 41: 157-183, 1993.

[2] Bruker P, Schlie R, Job shop scheduling with multi-purpose machines. Compu. 45: 369-375, 1990.

[3] Xia W, Wu Z, An effective hybrid optimization approach for multi-objective flexible jobshop scheduling problems. Computers & Industrial Engineering. 48: 409–425,2005.

[4] Mastrolilli M, Gambardella LM, Effective neighborhood functions for the flexible job shop problem. Journal of Scheduling. 3(1): 3–20, 2000.

[5] Pezzella F, Morganti G, Ciaschetti G, A genetic algorithm for the flexible job-shop scheduling problem. Compu.& Operations Res. 35: 3202–3212, 2008.

[6] Gao J, Sun L, Gen M, A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Compu.& Operations Res. 35: 2892–2907, 2008.
[7] Yazdani M, Amiri M, Zandieh M, Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Sys.withAppli. Intl. J. 37(1): 678–687, 2010.
[8] Defersha FM, Chen M, A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming. Proc. Intl. Conf. Compu. Sci. &Engg. 1: 201–208, 2009.
[9] Hurink J, Jurisch B, Thole M, Tabu search for the job-shop scheduling problem with multi-purpose machines. OR Spektrum. 15:205-215.
[10] Mati Y, Xie X, A genetic-search-guided greedy algorithm for multi-resource shop scheduling with resource flexibility. IIE Transactions. 40:1228-1240, 2008.
[11] Dauzere-peres S, Paulli J, An integrated approach for modeling and solving the multiprocessor job-shop scheduling problem using tabu search. Annals of Oprations Research. 70:281-306, 1997.
[12] Lawrence S, Supplement to resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. Technical report, GSIA, Carnegie-Mellon University, Pittsburgh, PA, 1984.
[13] Rashedi, E., Nezamabadi-pour, H., Saryazdi, S, BGSA: binary gravitational search algorithm. Natural Computing, vol. 9, pp. 727-745, 2010.
[14] Shi Y, Eberhart R, Parameter selection in particle Swarm Optimization. In: Porto VW, SaravananN,Waagen D and Eiben AE (eds) Evolutionary Programming VII. 611-616, 1998.
[15] Shi Y, Eberhart R. Empirical study of Particle Swarm Optimization. Proceeding IEEE International Congres Evolutionary Computation, Washington, DC., USA.1945-50, 1999.
[16] Meraji S, Afshar M, Afshar A, Reservoir Operatin by Particle Swarm Optimization Algorithm.7th International Conference of Civil Enginnering. Icce7th. Tehran. Iran, 2005.