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

Behnam Barzegar¹, Homayun Motameni²

¹Department of Computer Engineering, Nowshahr Branch, Islamic Azad University, Nowshahr, Iran
²Department 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 research 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, Flexible job 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
Solving Flexible Job-Shop Scheduling Problem …

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 processes 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.

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 J1, J2,...,Jn on set of M machines of M1,M2,...,Mm.

Each job has hj 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 Oj,h operations; h=1,....,hj where Oj,h presents h-th operation of j-th job, and hj presents number of required operations for j-th job. Machines set is presented by M = {M1, M2,..., Mm}. 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 Oj,h), set of jobs are assigned, which have capacity of performing that operation.

This set is presented asMj,h ⊂ M. Each machine would have specific process time for implementing operation. This specific process time for implementing each operation is presented with Pj,h.

In this study, we define Mj,h set with variable aij,h with value one and zero. If variable aij,h has value 1, it means that machine j has capacity for implementing operation Oj,h. For assignment, we use variable yij,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 Oj,h operation, machine j is selected.

Eventually, result solution from variable yij,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 tij and final time ftij 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 xij,k,l with value one or zero. If this variable has value 1, it means that operation Oj,k on machine j is implemented immediately after operation Oj,h. Also, seij,f,k presents start up time of job k after a job from family f on machine i.
Solving Flexible Job-Shop Scheduling Problem …  B. Barzegar, H. Motameni

\[ F_{f,j} = \begin{cases} 1 & \text{if } k \in f \\ 0 & \text{otherwise} \end{cases} \]

\[ a_{i,j,h} = \begin{cases} 1 & \text{if } O_{j,h} \text{can be performed on machine } i \\ 0 & \text{otherwise} \end{cases} \]

Variables of this model include:
\[ y_{i,j,h} = \begin{cases} 1 & \text{if machine } i \text{ selecte for operation } O_{j,h} \\ 0 & \text{otherwise} \end{cases} \]
\[ x_{i,j,h,k,l} = \begin{cases} 1 & \text{if } O_{j,h} \text{ presedes } O_{k,l} \text{ immediately on machine } i \\ 0 & \text{otherwise} \end{cases} \]

\[ C_{\text{max}}: \text{ Maximum time of constructing duration} \]
\[ m: \text{ A large number} \]
\[ t_{k,l}: \text{ Initial time for operation } O_{k,l} \]
\[ f_{k,l}: \text{ Final time for operation } O_{k,l} \]
\[ P_{i,k}: \text{ Process time for operation } O_{i,k} \text{ on machine } i \]
\[ S_{i,k}: \text{ Start up time for job } k \text{ on machine } i \text{ if previous job be job } j. \]

With having parameters \( S_{i,j,k}, P_{i,j,h}, a_{i,j,h}, f_{a}, m, n \) problem FJSP is modeling as follows:

(1) \( \min C_{\text{max}} \)

(2) \( t_{k,l} + S_{i,k,k} + P_{i,j,h} \leq f_{i,k} \text{ for } i=1,\ldots,m \ k=1,\ldots,n \ l=1,\ldots,h_k \)

(3) \( s_{i,k} = \sum_{j=1}^{m} \sum_{k=1}^{n} f_{j,k} \text{ for } i=1,\ldots,m \ j=1,\ldots,n \ k=1,\ldots,n \ f=1,\ldots,f_a \)

(4) \( f_{i,k} \leq t_{k,+l} \text{ for } k=1,\ldots,n \ l=1,\ldots,h_k \)

(5) \( f_{i,k} \leq C_{\text{max}} \text{ for } k=1,\ldots,n \)

(6) \( s_{i,k,l} \leq a_{i,k,l} \text{ for } i=1,\ldots,m \ k=1,\ldots,n \ l=1,\ldots,h_k \)

(7) \( t_{k,l} + P_{i,j,k} + S_{i,k,k} \leq t_{k,l} + (1-x_{i,j,h,k,l})M \text{ for } j=1,\ldots,n \ k=1,\ldots,n h=1,\ldots,h_j \ l=1,\ldots,h_i \)

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

(9) \( \sum_{i=1}^{m} s_{i,k} = 1 \text{ for } j=0,\ldots,n \ h=1,\ldots,h_j i=1,\ldots,m \)

(10) \( \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j,h,k} = y_{i,j,h} \text{ for } i=1,\ldots,m \ k=1,\ldots,n \ l=1,\ldots,h_k \)

(11) \( \sum_{i=1}^{m} \sum_{j=1}^{n} x_{i,j,h,k} = y_{i,j,k} \text{ for } i=1,\ldots,m \ j=1,\ldots,n \ h=1,\ldots,h_j \)

(12) \( x_{i,j,k,h,u} \leq x_{i,j,k,l} \text{ for } j=1,\ldots,n \ k=1,\ldots,n \ h=1,\ldots,h l=1,\ldots,h_i k=1,\ldots,m \)

(13) \( x_{i,j,k,h,u} \leq x_{i,j,k,l} \text{ for } j=1,\ldots,n \ k=1,\ldots,n \ h=1,\ldots,h l=1,\ldots,h_i k=1,\ldots,m \)

(14) \( x_{i,j,k,l}=0 \text{ for } i=1,\ldots,m \ k=1,\ldots,n \ l=1,\ldots,h_k \)

(15) \( s_{i,k,k}=0 \text{ for } i=1,\ldots,m \ k=1,\ldots,n \)

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

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 \( C_{\text{max}} \).

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.

Table 1 shows case of performing 3 jobs on 5 machines. Each job includes three following operations and Figure 1 shows another case of performing 3 jobs (with three sub-operations) on 3 machines and each run time is known as Gantt diagram.

<table>
<thead>
<tr>
<th></th>
<th>( M_1 )</th>
<th>( M_2 )</th>
<th>( M_3 )</th>
<th>( M_4 )</th>
<th>( M_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J_1 )</td>
<td>( O_{1,1} )</td>
<td>1</td>
<td>9</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>( O_{1,2} )</td>
<td>3</td>
<td>5</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>( O_{1,3} )</td>
<td>6</td>
<td>7</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>( J_2 )</td>
<td>( O_{2,1} )</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>( O_{2,2} )</td>
<td>2</td>
<td>8</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>( O_{2,3} )</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( J_3 )</td>
<td>( O_{3,1} )</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>( O_{3,2} )</td>
<td>5</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>( O_{3,3} )</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

![Figure 1. A Gantt-Chart representation of a solution for a 3 * 3 problem](image)

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 as \( V_i \). Eventually, movement of bird populations is obtained with two following equations:

\[
V_{id}(t+1) = W_n \times V_{id}(t) + c_1 r_1 \left(l_{best_id}(t) - X_{id}(t) \right) + c_2 \left(g_{best}(t) - X_{id}(t) \right) \tag{1}
\]

\[
X_{id}(t+1) = X_{id}(t) + V_{id}(t) \tag{2}
\]

Where \( d = 1, 2, \ldots, D \) and \( i = 1, 2, \ldots, N \) 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]).

$$W_n = W_{\text{max}} - \frac{(W_{\text{max}} - W_{\text{min}})\times n}{i\text{Ter}_{\text{max}}}(3)$$

$W_{\text{min}}$ and $W_{\text{max}}$ are respectively initial and final values of inertia weight. $(W)$, $i\text{Ter}_{\text{max}}$ 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 $(V_{\text{max}})$ 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 $(I_{\text{best}})$ and the best response in current generation$(g_{\text{best}})$. 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.

\[
X_i = (x_{i1}, \ldots, x_{id}, \ldots, x_{iD})(4)
\]

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)

\[
\text{Best}(t) = \max_{j \in \{1, \ldots, m\}} \text{fit}_j(t)(5)
\]

\[
\text{Worst}(t) = \min_{j \in \{1, \ldots, m\}} \text{fit}_j(t)(6)
\]

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 \( \text{fit} \) are denote mass and fitness of factor \( i-th \) in time \( t \), respectively.

\[
q_i(t) = \left( \frac{\text{fit}_i-Worst(t)}{\text{Best}(t)-Worst(t)} \right) (7)
\]

\[
M(t) = \frac{q_i(t)}{\sum_{j=1}^{n} q_j(t)}(8)
\]

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.

\[
F_{ij}^d(t) = \frac{G(t)-M(t)}{R_{ij}(t)+\varepsilon} (x_i^d(t) - x_j^d(t))(9)
\]

\( R \) is \text{ECLIDIAN} distance between factor \( i-th \) and factor \( j-th \) that is defined as relations (10). \( \varepsilon \) is also a small value for avoiding denominator from becoming zero.

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

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].

\[
F_i^d(t) = \sum_{j \in K_{best}, j \neq i} \text{rand} \times G(t) \frac{M_j(t)-M_i(t)}{R_{ij}(t)+\varepsilon} (x_i^d(t) - x_j^d(t))(11)
\]

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).

\[
d_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \Rightarrow
\]

\[
d_i^d(t) = \sum_{j \in K_{best}, j \neq i} \text{rand} \times G(t) \frac{M_j(t)}{R_{ij}(t)+\varepsilon} (x_i^d(t) - x_j^d(t))(12)
\]
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, \( \text{rand} \) is a random number with normal distribution in the interval \([0,1]\), and its random property is resultant of keeping search in random mood.

\[
V_i^d(t + 1) = \text{rand}_i \times V_i^d(t) + a_i^d(t) \tag{13}
\]

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).

\[
x_i^d(t + 1) = x_i^d(t) + V_i^d(t + 1) \tag{14}
\]

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 Figure 2, 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).
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).

```
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;
    
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]));
```

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);
```
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 lbest and 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];
    }
    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(Best_Loc[i] - loc_Arr))) +
        (c2 * r2 * (Math.Abs(side_X - loc_Arr))));
    if (v_Temp > v_max)
        v_Arr = v_Temp;
    else
        v_Arr = v_Temp;
}

Figure 9. implement related formula for PSO

Table 2. Experimental results

<table>
<thead>
<tr>
<th>Data</th>
<th>HJT</th>
<th>DP</th>
<th>MG</th>
<th>MX</th>
<th>GSFJ</th>
<th>CPU</th>
<th>Edata</th>
<th>vdata</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>611</td>
<td>609</td>
<td>609</td>
<td>609</td>
<td>35.5</td>
<td>574</td>
<td>574</td>
<td>574</td>
</tr>
<tr>
<td>102</td>
<td>655</td>
<td>655</td>
<td>655</td>
<td>655</td>
<td>3.1</td>
<td>535</td>
<td>532</td>
<td>530</td>
</tr>
<tr>
<td>103</td>
<td>573</td>
<td>554</td>
<td>550</td>
<td>563</td>
<td>557</td>
<td>574</td>
<td>574</td>
<td>574</td>
</tr>
<tr>
<td>104</td>
<td>578</td>
<td>568</td>
<td>568</td>
<td>568</td>
<td>569</td>
<td>481</td>
<td>479</td>
<td>478</td>
</tr>
<tr>
<td>105</td>
<td>503</td>
<td>503</td>
<td>503</td>
<td>503</td>
<td>4.1</td>
<td>460</td>
<td>458</td>
<td>457</td>
</tr>
<tr>
<td>106</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>833</td>
<td>45.9</td>
<td>801</td>
<td>800</td>
<td>799</td>
</tr>
<tr>
<td>107</td>
<td>765</td>
<td>765</td>
<td>765</td>
<td>765</td>
<td>37.3</td>
<td>752</td>
<td>750</td>
<td>750</td>
</tr>
<tr>
<td>108</td>
<td>845</td>
<td>845</td>
<td>845</td>
<td>845</td>
<td>52.2</td>
<td>767</td>
<td>767</td>
<td>765</td>
</tr>
<tr>
<td>109</td>
<td>878</td>
<td>878</td>
<td>878</td>
<td>878</td>
<td>58.4</td>
<td>859</td>
<td>854</td>
<td>853</td>
</tr>
<tr>
<td>110</td>
<td>866</td>
<td>866</td>
<td>866</td>
<td>866</td>
<td>54.6</td>
<td>806</td>
<td>805</td>
<td>804</td>
</tr>
<tr>
<td>111</td>
<td>1106</td>
<td>1103</td>
<td>1103</td>
<td>1104</td>
<td>1104</td>
<td>1073</td>
<td>1072</td>
<td>1071</td>
</tr>
<tr>
<td>112</td>
<td>960</td>
<td>960</td>
<td>960</td>
<td>960</td>
<td>80.2</td>
<td>937</td>
<td>936</td>
<td>936</td>
</tr>
<tr>
<td>113</td>
<td>1053</td>
<td>1053</td>
<td>1053</td>
<td>1053</td>
<td>89.7</td>
<td>1039</td>
<td>1038</td>
<td>1038</td>
</tr>
<tr>
<td>114</td>
<td>1151</td>
<td>1123</td>
<td>1123</td>
<td>1123</td>
<td>50.1</td>
<td>1071</td>
<td>1070</td>
<td>1070</td>
</tr>
<tr>
<td>115</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>1111</td>
<td>134.3</td>
<td>1993</td>
<td>1990</td>
<td>1990</td>
</tr>
<tr>
<td>116</td>
<td>924</td>
<td>915</td>
<td>892</td>
<td>892</td>
<td>44.2</td>
<td>717</td>
<td>717</td>
<td>717</td>
</tr>
<tr>
<td>117</td>
<td>757</td>
<td>707</td>
<td>707</td>
<td>707</td>
<td>93.0</td>
<td>646</td>
<td>646</td>
<td>646</td>
</tr>
<tr>
<td>118</td>
<td>864</td>
<td>843</td>
<td>842</td>
<td>850</td>
<td>844</td>
<td>77.8</td>
<td>674</td>
<td>669</td>
</tr>
<tr>
<td>119</td>
<td>850</td>
<td>796</td>
<td>796</td>
<td>797</td>
<td>100.2</td>
<td>725</td>
<td>703</td>
<td>700</td>
</tr>
<tr>
<td>120</td>
<td>919</td>
<td>864</td>
<td>857</td>
<td>865</td>
<td>186.6</td>
<td>756</td>
<td>756</td>
<td>756</td>
</tr>
<tr>
<td>121</td>
<td>1066</td>
<td>1046</td>
<td>1046</td>
<td>1041</td>
<td>133.4</td>
<td>861</td>
<td>846</td>
<td>835</td>
</tr>
<tr>
<td>122</td>
<td>919</td>
<td>890</td>
<td>882</td>
<td>918</td>
<td>910</td>
<td>99.1</td>
<td>790</td>
<td>772</td>
</tr>
<tr>
<td>123</td>
<td>980</td>
<td>953</td>
<td>950</td>
<td>972</td>
<td>101.1</td>
<td>884</td>
<td>853</td>
<td>842</td>
</tr>
<tr>
<td>124</td>
<td>952</td>
<td>918</td>
<td>909</td>
<td>918</td>
<td>152.7</td>
<td>825</td>
<td>820</td>
<td>808</td>
</tr>
<tr>
<td>125</td>
<td>970</td>
<td>955</td>
<td>941</td>
<td>970</td>
<td>124.2</td>
<td>822</td>
<td>802</td>
<td>791</td>
</tr>
<tr>
<td>126</td>
<td>1169</td>
<td>1138</td>
<td>1125</td>
<td>1156</td>
<td>1141</td>
<td>179.5</td>
<td>1086</td>
<td>1070</td>
</tr>
<tr>
<td>127</td>
<td>1230</td>
<td>1215</td>
<td>1186</td>
<td>1230</td>
<td>1213</td>
<td>135.4</td>
<td>1109</td>
<td>1100</td>
</tr>
<tr>
<td>128</td>
<td>1204</td>
<td>1169</td>
<td>1149</td>
<td>1169</td>
<td>1152</td>
<td>210.0</td>
<td>1097</td>
<td>1085</td>
</tr>
<tr>
<td>129</td>
<td>1210</td>
<td>1157</td>
<td>1118</td>
<td>1178</td>
<td>1163</td>
<td>212.6</td>
<td>1016</td>
<td>1004</td>
</tr>
<tr>
<td>130</td>
<td>1253</td>
<td>1225</td>
<td>1204</td>
<td>1225</td>
<td>1224</td>
<td>276.1</td>
<td>1105</td>
<td>1089</td>
</tr>
</tbody>
</table>

79
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 $F_{ij}$ 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 $edata$ 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 Mastroiilli 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 for $c_1$ 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


Solving Flexible Shop Scheduling Problem

B. Barzegar, H. Motameni