Optimality of the flexible job shop scheduling system based on Gravitational Search Algorithm

Behnam Barzegar[1]✉, Homayun Motameni[2]
(1)Department of Computer Engineering, Nowshahr Branch, Islamic Azad University, Nowshahr, Iran.
(2)Department of Computer Engineering, Sari Branch, Islamic Azad University, Sari, Iran.
barzegar@iauns.ac.ir; motameni@iausari.ac.ir

Received: 2011/09/14; Accepted: 2011/10/01 Pages: 1-12

Abstract

The Flexible Job Shop Scheduling Problem (FJSP) is one of the most general and difficult of all traditional scheduling problems. The Flexible Job Shop Problem (FJSP) is an extension of the classical job shop scheduling problem which allows an operation to be processed by any machine from a given set. The problem is to assign each operation to a machine and to order the operations on the machines, such that the maximal completion time (makespan) of all operations is minimized. The scheduling objective minimizes the maximal completion time of all the operations, which is denoted by Makespan. The goal of this research is to develop an efficient scheduling method based on Gravitational local search algorithm to address FJSP. we could reduce scheduling time and costs by transferring and delivering operations on existing machines, that is among NP-hard problems. Different methods and algorithms have been presented for solving this problem. Having a reasonable scheduled production system has significant influence on improving effectiveness and attaining to organization goals. In this paper, we design algorithm were proposed for flexible job shop scheduling problem (FJSP-GSA), that is based on Gravitational search algorithm (GSA). The experimental results showed that the proposed method has reasonable performance in comparison with other algorithms.

Keywords: *Gravitational search algorithm, Flexible job shop scheduling problem, Makespan, Mass, Gravitational force*

1. Introduction

Scheduled production system leads to avoiding stock accumulations, losses reduction, decreasing or even eliminating idol machines, and effort to better benefitting from machines for on time responding costumer orders and supplying requested materials in suitable time.

Classic job shop scheduling production systems contain N independent job on M machines. Each job includes one or more operations that must be implemented sequentially. Each operation needs specific process time. Flexible job shop scheduling production system is specific type of classic job shop scheduling production systems, in which one job could be implemented on set of machines.

Purpose of scheduling this problem is determining operation sequence for each machine, such that sequence order is kept and total time of operation (during implementing one job) be minimized.

In this paper, FJSP-GSA algorithm based on Gravitational local search is proposed for scheduling time optimization in FJSP.

Gravitational searching algorithm and proposed solution have been presented to made suitable time for implementing several operations on one job, which in fact is assigning proper machine to related operation.

2. Related work

Flexible job shop scheduling production system is one of the most important combined optimizing problems. The JSP is not only *NP*-hard, but it is one of the worst members in the class. An indication of this is given by the fact that one 10 * 10 problem formulated by Muth and Thompson [1] remained unsolved for over 20 years.

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 for the flexible job-shop scheduling production system (FJSP).

Bruker and Schlie et al [3] 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 [2] 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.

Jain and Meeran et al [7] provided a concise overview of JSPs over the last few decades and highlighted the main techniques. The JSP is the most difficult class of combinational optimization. Garey et al [8] demonstrated that JSPs are non-deterministic polynomial-time hard (NP-hard); hence we cannot find an exact solution in a reasonable computation time. The single objective JSP has attracted wide research attention. Most studies of single-objective JSPs result in a schedule to minimize the time required to complete all jobs, i.e., to minimize the make span. Many approximate methods have been developed to overcome the limitations of exact enumeration techniques. These approximate approaches include simulated annealing (SA) [9], tabu search [10-12] and genetic algorithms (GA) [13-15].

Fattahi, Saidi, Jolai [5] have considered hierarchical and integrated perceptions in relation to scheduling job shop production systems. They based on these perceptions and two SA and TA heuristics offered six combined algorithms and compared them. The concluded that combined algorithms from SA and TA along with hierarchical perception would provide better solutions than other algorithms. Te also offered in their article a new technique for introducing structure of solution in scheduling flexible job shop production problems.

Choi and Choi et al [4] have presented a local searching algorithm for scheduling job shop production problems. They regarded that there is possibility of a substitute operation for any operation. In this mode, for all operations a machine and process time are assigned and then, for some operations considered substitute machine and process time. Moreover, a run time has been considered for any operations, which is depended to the last operation.

In this study, we first considered problem with primary process and ignored substitute process, and regarded flexibility and obtained construction duration have been used as upper boundary. Then, local searching procedure is looking for better solution by using distribution rules. In this study, different distribution rules in local searching procedure have been considered.

Xia and Wu et al [6] have presented a hybrid optimizing perception for scheduling multi object *flexible job shop production system* problems. In their study, combination of two methods SA and *particle swarm optimization* have been used for optimizing flexible job shop production system problem. PSO algorithm is applied for assignment problem either for determining any operations use which machine. Value of object function is calculated b SA algorithm and implemented for each particle in PSO algorithm once.

Mastrolilli and Gambardella et al [17] 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 determining sequence of operations on each machine. Pezzella et al [18] and Gao et al [19] 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 [20] propose a parallel variable neighborhood search (VNS) algorithm for the FJSP based on independent VNS runs. Defersha and Chen et al [21] 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, its disjunctive graph model are presented. In section third, gravitational searching algorithm is explained and finally in section four, we explain proposed solution by using gravitational searching algorithm.

3. Flexible Job-shop Scheduling Problem

The FJSP can be an extension of the classical JSP; therefore, we can formulate the FJSP based on JSP. Consider a set of n jobs, noted $J=\{J1, J2 ,....Jn\}$, every job in the set J has a given number operations, and should be operated on a given machine from a

machine set named $M=\{M1,M2 ,....Mm\}$. So, there are n jobs and m machines. In the classical JSP problem, with n jobs and m machines, there are $n *m$ operations. However, in FJSP problems, the operation number can vary with the problem assumption. There are two kinds of FJSP, i.e., TFJSP and P-FJSP. For the T-FJSP, each job can be operated on every machine from the set M; for the P-FJSP, there is a problem constraint for the operating process, in table 1, we can see that one operation of a job must be processed by a set of machines in $M' \subseteq M$. In the sequencing stage for the FJSP, we must consider the candidate machine set size for every operation waiting for processed.

The detailed definition of the FJSP as follows:

- A set of J independent jobs.
- Each job Ji can be operated on a given set of machines Mi .

- The Oi, j represents the jth operation of Ji . The machines set waiting for processing the Oi, j noted by$Mk \subseteq M$.
- We use pi, j,k to represent the processing time of Oi, j operated on the kt machine.

There have two assumptions: a started operation cannot be interrupted; each machine only can process one operation at the same time.

The objective in our paper is to find the minimum time of the whole operations.

4. Disjunctive Graph

Disjunctive graphs are a well-known modeling concept for job-shop scheduling and related machine sequencing problems. In a disjunctive graph the nodes correspond to the operations to be scheduled and the weighted arcs represent precedence constraints between pairs of operations. A conjunctive arc (i, j) expresses the condition that operation i must precede operation j, while a pair of disjunctive arcs (i, j), (j, i) expresses the condition that i must precede j or vice versa. Disjunctive arcs result from the fact that two operations I and j on the same machine cannot overlap in time. Since there is a one-to-one correspondence between feasible semi-active schedules and feasible selections in a disjunctive graph, an optimal schedule minimizing makespan can be found be determining a feasible selection that minimizes the length of a longest path in the associated graph.

A distinctive property of disjunctive job-shop graphs is that each pair of disjunctive arcs contains two arcs with identical extremities but reverse orientation. We introduce a generalization of the classical disjunctive graph concept, allowing pairs of disjunctive arcs with different extremities. This generalization allows modeling a variety of sequencing problems in manufacturing and logistic systems. We consider in particular a version of the job-shop problem characterized by sequence-dependent set-up times and no buffers between machines.

As a main result, we show that the feasibility problem for generalized disjunctive graphs - in contrast to the feasibility problem in the job-shop case - is NP-complete. The feasibility problem addresses the question of whether a generalized disjunctive graph has a feasible selection, i.e. a complete, positive acyclic selection of disjunctive arcs. The proof is based on a polynomial-time reduction of the SAT-problem to the feasibility problem. We also discuss some implications of this complexity result on the design of solution methods.

As a second extension of the classical disjunctive graph model, we discuss the introduction of arbitrary arc weights. Non-positive arc weights allow to describe various types of conditions typical for scheduling problems, such as due dates, limitations on total duration of some operations, synchronization and no-wait constraints. We address the feasibility problem for disjunctive graphs with arbitrary weights. Summarizing some existing results, we show that the complexity of the feasibility problem depends only on the conjunctive part of the graph: if it is acyclic, there exists a feasible selection; if it has a cycle of positive weight, there is no feasible selection; if it has a cycle of non-positive weight, deciding on feasibility is NP complete.

5. Gravitational Search Algorithm (GSA)

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

System area is 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 which within it any answers of problem is 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 be not equal, and their amounts are determined base on suitability of each mass. *Mass state* is a point in space which is one of problem answers. After forming system, its rules are determined [23-24].

We suppose that only 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.

$$X_i = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^D\right) \tag{1}$$

Worst (t) and *best (t)* are for *minimization* problems and are calculated with relations (2) and (3). (For *maximization* problems is just enough to consider the inverse of these two relations).

$$\text{Best}(t) = \max_{j\in\{1,\ldots,m\}} fit_j(t) \tag{2}$$

$$\text{worst}(t) = \min_{j\in\{1,\ldots,m\}} fit_j(t) \tag{3}$$

We can account fitness of recent population with relation (4), and obtain mass of factor *i-th* in time *t* (i.e. with relation (5)), where *M* and *fit* are denote mass and fitness of factor *i-th* in time *t*, respectively.

$$q_i(t) = \frac{fit_i - worst\ (t)}{best\ (t) - worst\ (t)} \tag{4}$$

$$\mathrm{M}(t) = \frac{q_i}{\sum_{j=1}^{s}(t)} \tag{5}$$

In this system, force *F* is delivered on mass *i-th* from mass *j-th* in time *t* in the direction of dimension *d*, which value of this force is obtained base on relation (6), And in relation (6), *G(t)* is gravity constant in time *t* which is regulated in the beginning of operating algorithm, and is decreased by the time.

$$\mathrm{F}_{\mathrm{ij}}^{\mathrm{d}}(t) = \frac{G(t)\times M_j(t)}{R_{ij}(t)+\varepsilon}\,(X_j^d(t) - X_i^d(t)) \tag{6}$$

R is *ECLIDIAN* distance between factor *i-th* and factor *j-th* that is defined as relations (7)," " is also a small value for avoiding denominator from becoming zero.

$$\mathrm{ij} = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2 + \cdots + (n_2 - n_1)^2} \tag{7}$$

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 (8). In this relation, *rand* is a random number with normal distribution in the interval [0,1].

$$\mathrm{F}_{\mathrm{i}}^{\mathrm{d}}(t) = \sum_{j\epsilon kbest\ ,j\neq i} rand\ j \times G(t)\frac{M_j(t)\times M_i(t)}{R_{ij}(t)+\varepsilon}\,(X_j^d(t) - X_i^d(t)) \tag{8}$$

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 (9).

$$\mathrm{a}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t}) = \frac{\mathrm{F}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t})}{\mathrm{M}_{\mathrm{i}}(\mathrm{t})} \rightarrow \tag{9}$$

$$\mathrm{a}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t}) = \sum_{\mathrm{j\epsilon kbest\ ,j\neq i}} \mathrm{rand\ j} \times \mathrm{G(t)}\frac{\mathrm{M_j(t)} \times \mathrm{M_i(t)}}{\mathrm{R_{ij}(t)} + \varepsilon}\,(\mathrm{X}_{\mathrm{j}}^{\mathrm{d}}(\mathrm{t}) - \mathrm{X}_{\mathrm{i}}^{\mathrm{d}}(\mathrm{t}))$$

And speed of each mass is equal to sum of coefficient of mass recent speed and acceleration, and is explained as relation (10). 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.

$$\mathrm{V}_{\mathrm{i}}^{\mathrm{d}}(t+1) = rand_i \times V_i^d(t) + a_i^d(t) \tag{10}$$

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 (11).

$$\mathrm{X}_{\mathrm{i}}^{\mathrm{d}}(t+1) = \mathrm{X}_{\mathrm{i}}^{\mathrm{d}}(t)_i + V_i^d(t+1) \tag{11}$$

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 *8* to *11*. System parameters are updated in each stage (G, M).

Stop condition could be determined after passing specified time. In Figure 1, semi-code of this algorithm has been presented: (Rashedi and Nezamabadi-pour and Saryazdi et al,2009).

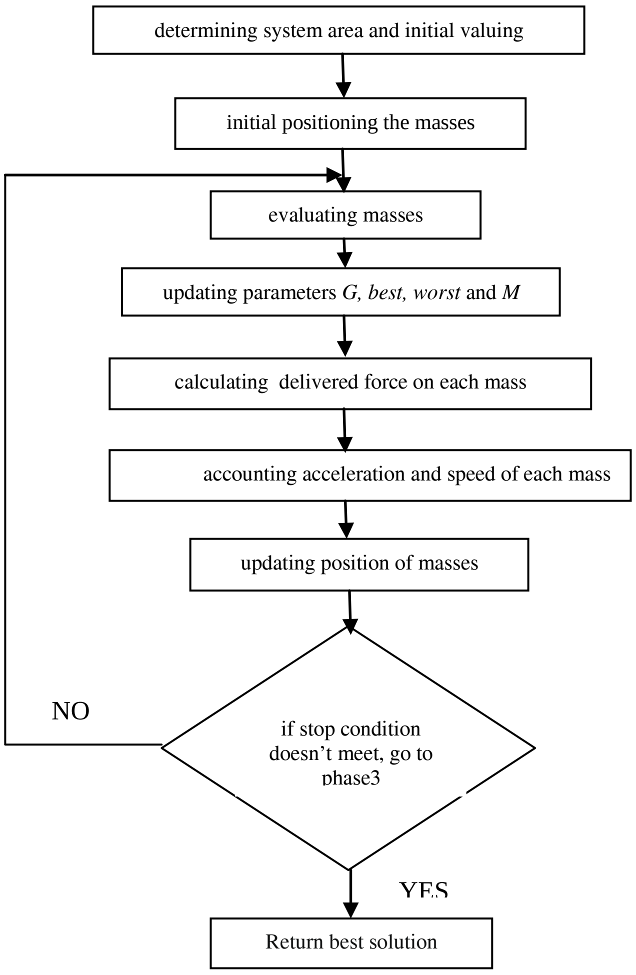
determining system area and initial valuing
initial positioning the masses
evaluating masses
updating parameters *G, best, worst* and *M*
calculating delivered force on each mass
accounting acceleration and speed of each mass
updating position of masses
NO
if stop condition doesn't meet, go to phase3
YES
Return best solution

Figure 1. semi-code of Gravitational Search Algorithm

6. Proposed Method Based on Gravitational Search Algorithm (FJSP-GSA)

Regarding to gravitational searching algorithm, each searching factor should be contain information for solving problems. This information says that for example each factor in any time might be aware that in each point of searching space, which operation is implementing on which machine. According to problem definition, we could consider a table like below table that in it, each operation is implementable on set of machines. But in any time, only one job is implemented on each machine, and then next operation should be implemented.

Table 1. Example of an job with 4 actions on 4 machines

	A1	A2	A3	A4
M1			■	
M2	■			
M3		■		
M4				■

You could see with some attention that above table is similar to N-minister problem table that in each column is placed just one minister (one job for each machine). The only difference is that in new table maybe several jobs are implemented on each machine.

To brief this table we use One-dimensional array and we assign to each factor in searching space.

It is obvious that each house of this array is assigned to one column of table, and value of that house states number of machines that related job would be implemented by them. For example, second house of below array indicates that second job (in second column) would be implemented on third machine.

In gravitational searching algorithm, each factor in searching space includes a one dimensional array which keeps summary of recent state of implemented operations on related machines. So, with having five masses, in fact five searching factors are applied for finding purpose state (minimum time for performing operation).

To indicate that bigger mass has better state, we should subtract total time of implementing an operation from a constant value (this value could be maximum required time for implementing a job which counts as a constraint). Result answer of this subtraction is q_i, conforming with formula (4). Now if base on formula (5), we divide fitness of one factor on sum of factors fitness, mass factor is attained.

Accounting delivered force, acceleration, speed and position of each mass are depended on dimension of each mass, and they are independent of each other.

Consider a two-dimensional space. If there are two masses in one column during applying calculations on dimension *X*, calculations should be stopped, since second mass doesn’t deliver force on first mass in direction of dimension *X*.

For example, in Figure 2, you see that two masses (A and B) are placed in one column, so they don’t deliver force in direction of dimension *X* on each other.

And similarly, two masses *C* and *D* are placed in one line, and so don’t deliver force on each other in direction of dimension *Y*. But pair masses (B,C), (B,D), (C,D), (A,D) are delivered force on each other in both directions of dimensions *X* and *Y*, and so calculations are applied on them completely.

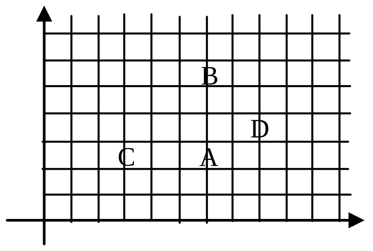

Figure 2. Two-dimensional space with 4 masses

Therefore, in first condition, we investigate un-parallelism of two masses in interested dimension. Then in order to account sum of delivered forces on related mass, we need to determine forces delivered from those masses which are placed in *Kbest* series.

Kbest array is filled with initial value of (-1). According to gravitational algorithm, at the first moment of operating algorithm, all masses deliver force on each other. After assessing first condition, we add number of masses to *Kbest* series (Figure 3).

```
for (byte j = 0; j < j_num; j++)
K_best[j] = -1;
```

```
for (byte i = 0; i <= mass_num; i++)
{
if (Loc_arr[0, i] >= n)
Loc_arr[0, i] = n;
if (Loc_arr[1, i] >= n)
Loc_arr[1, i] = n;
if (Loc_arr[0, k] != Loc_arr[0, i])
{
for (byte j = 0; j < mass_num; j++)
if (K_best[j] == -1)
{
K_best[j] = arr[Loc_arr[0, i], Loc_arr[1, i]];
break;
}
}
}
```

Figure 3. calculating K_best

It is obvious that according to gravitational algorithm, in next moment, we should add the condition of “being masses heavier” to the first condition, i.e. in addition to condition of un-parallelism of masses, those masses which are heavier than recent masses, should be added to *Kbest* series.

Now, we could write calculations as follow:

```
while ((K_best[l] >= 0) && (number <= mass_num))
{
R = Math.Sqrt((Math.Pow((Loc_arr[0, k_best_T] –
Loc_arr[0, k]), 2) + Math.Pow((Loc_arr[1, k_best_T] –
Loc_arr[1, k]), 2)));
F_arr[0, k] = F_arr[0, k]+((rand_obj.Next(100) /
100.0) * G * (Math.Abs((hiu_mass[k_best_T] -
hiu_mass[k])) /
(R + E))* Math.Abs(Loc_arr[0,k_best_T] -
Loc_arr[0,k]));
}
A_mass = F_arr[0, k] / hiu_mass[k];
V_arr[0, k] =((rand_obj.Next(100)/100.0)* V_arr[0, k])
+ A_mass;
x_temp= (Loc_arr[0,k] + Math.Round(V_arr[0,k]));
```

Figure 4. calculating for each masses

New positions of mass, has been specified. And it is obvious that researcher factor should have new state and finally new mass in new position of search space. But how these changes in state and mass should be created?

In proposed solution, we divide state array on *N*-dimensions of search space, i.e. for each dimension, we assign some houses to state array.

For example, we would have a state array with six houses and a three-dimensional search space, where we assign two houses for dimension *X* and two houses for dimension *Y* and two houses for dimension *Z* (Figure 5).

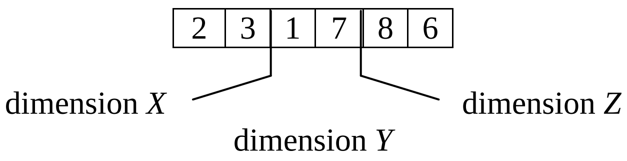

Figure 5. state array and dimensions

Attention that order of assigned dimensions to the houses are arbitrary, but with change in position of factor in search space, movement is determined in direction of related dimension, and only corresponding cells with that dimension may be change in state array, and other values remain constant. So factors could move in direction of their corresponding dimensions.

Way of changing values is important, and is explained as follow:

When a factor starts to move in one direction, we divide each corresponding value with related dimension on distance, then by calling neighborhood function, we specify that by replacing which value in state array total spent time would be decreased and corresponding mass found better state.

If in searching space is reminded just one mass, search is finished and with considering existing number of reminded mass (best mass) in array, list of machines is presented for processing reminded operations of one job in order to minimizing time production.

For instance, follow array shows that if first job is implemented by third machine, second job is implemented by fifth machine and so on, then we would have ideal time for producing or performing related job, Such that required time for performing one job on specified machines with above mentioned operations and ignoring other times (such as supplying materials, path stops, delivering times) would be as follows (Figure 6):

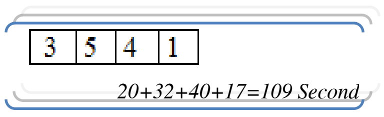

Figure 6. Result array

S No	Instance Name	Instance Size (*n* × *m*)	Obtained Values from the JSSP using Simple Genetic Algorithm	Goncalves *et al* [15]	Obtained Values from the Proposed Algorithm (FJSP-GSA)
1.	FT06	6 × 6	55	55	55
2.	LA01	10 × 5	686	666	633
3.	LA02	10 × 5	739	655	*680*
4.	LA03	10 × 5	670	597	484
5.	LA04	10 × 5	634	590	546
6.	LA05	10 × 5	593	593	544
7.	LA06	15 × 5	956	926	896
8.	LA07	15 × 5	994	890	860
9.	LA08	15 × 5	906	863	855
10.	LA09	15 × 5	956	951	*954*
11.	LA10	15 × 5	958	958	955
12.	LA11	20 × 5	1267	1222	1215
13.	LA12	20 × 5	1071	1039	1020
14.	LA13	20 × 5	1188	1150	1121
15.	LA14	20 × 5	1292	1292	1270
16.	LA15	20 × 5	1383	1207	1155
17.	LA16	10 × 10	1080	945	943
18.	LA17	10 × 10	906	784	745
19.	LA18	10 × 10	976	848	828
20.	LA19	10 × 10	1021	842	718
21.	LA20	10 × 10	1072	907	890

7. Experimental Results

The C#.Net 2008 language programming was used for testing the proposed algorithm and in this paper, we have used 21 instances that are taken from the OR-Library (Beasley, 1990) as benchmarks 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.

Goncalves et al [15] gives optimal Solution for most of the benchmark problems. But our proposed algorithm gives the optimal solution within a minimum considerable amount of time. We can't compare the computation times of Goncalves et al [15] with our proposed work, because the system configuration is not unique (Tamilarasi and kumar [22]).

8. Conclusion

This paper has presented a new Gravitational search algorithm for FJSP. Purpose of scheduling job shop production systems is determining sequence of operations on related machines, such that production time get optimized. Gravitational search algorithm is one of random-base algorithms for optimum finding in different problems, which has been established based on exploiting gravity rules in nature. Proposed solution which is offered for scheduling job shop production systems is based on this algorithm where optimal or near optimal solutions might be found. This solution correspond dimensions of searching space with state array houses, and while controlling the search guarantees improved search and proportioned states.

Finally, we believe that the methodology used in this paper can be extended to solve other scheduling problems.

9. References

[1] J.F. Muth and G.L. Thompson Industrial Scheduling. Prentice-Hall, Engle-wood Cliffs, N.J. (1963).
[2] P. Brandimarte. Routing and scheduling in flexible job shop by tabu search. Annals of Operation Research, 41, PP. 157-183. (1993).
[3] P. Bruker, & R. Schlie . Job shop scheduling with multi-purpose machines. Computing, 45, PP. 369-375. (1990).
[4] Choi IC & Choi DS. A local search algorithm for job shop scheduling problems with alternative operations and sequence–dependent setups. Computers & Industrial Engineering, 42, PP. 43-45. (2002).
[5] P. Fattahi, M.Saidi, F. Jolai. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing. (2002).
[6] W. Xia, Z. Wu .An effective hybrid optimization approach for multi-objective flexible jobshop scheduling problems. Computers & Industrial Engineering. 48, PP. 409–425. (2005).
[7] Jain and Meeran. Deterministic job-shop scheduling: Past, present and future. Europ. J. Operational Res. 113, PP. 390–434. (1999).
[8] MR Garey, DS Johnson and R. Sethi. The complexity of flowshop and jobshop scheduling. Maths. Operations Res. 1, PP. 117–129. (1976).
[9] HR. Lourenço. Local optimization and the job shop scheduling problem. Eur. J. Operational Res. 83, PP. 347–364. (1995).
[10] D. Sun, R. Batta and L. Lin .Effective job shop scheduling through active chain manipulation. Compu. & Operations Res. 22(2), pp.159–172.(1995).
[11] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. Managt. Sci. 42(6), pp. 797–813. (1996).
[12] F. Pezzella and E. Merelli. A tabu search method guided by shifting bottleneck for the job shop scheduling problem. Eur. J. Operational Res. 120(2), pp. 297–310. (2000).
[13] J. Bean. Genetic algorithms and random keys for sequencing and optimization. Operations Res. Soc. Am. J. Computing (ORSA). 6, pp.154–160. (1994).
[14] S. Kobayashi, I. Ono and M. Yamamura. An efficient genetic algorithm for job shop scheduling problems. In LJ. Eshelman (Ed.). Proce. 6th Intl. conf. on genetic algorithms. San Francisco, CA:Morgan Kaufman Publ. pp. 506–511. (1995).
[15] JF. Gonçalves, JJM. Mendes and Resende MGC. A hybrid genetic algorithm for the job shop scheduling problem. Europ. J. Operational Res. 167(1), pp. 77–95. (2005).
[16] L. Wang and DZ. Zheng. An effective hybrid optimization strategy for jobshop scheduling problems. Compu. & Operations Res. 28, pp. 585–596. (2001).
[17] M. Mastrolilli and LM. Gambardella. Effective neighborhood functions for the flexible job shop problem. J. Scheduling. 3(1), pp. 3–20. (2000).
[18] F. Pezzella, G. Morganti and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. Compu. & Operations Res. 35, pp. 3202–3212. (2008).
[19] J. Gao, L. Sun and M. Gen. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Compu. & Operations Res. 35, pp. 2892–2907. (2008).
[20] M. Yazdani, M. Amiri and M. Zandieh. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. Expert Sys. with Appli. Intl. J. 37(1), pp. 678–687. (2010).
[21] FM. Defersha and M. Chen. A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming. Proc. Intl. Conf. Compu. Sci. & Engg. 1, pp. 201–208. (2009).
[22] Tamilarasi and T. Anantha kumar. An enhanced genetic algorithm with simulated annealing for job-shop scheduling, International Journal of Engineering, Science and Technology Vol. 2, No. 1, 2010, pp. 144-151. (2010).
[23] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi. "GSA: A Gravitational Search Algorithm" , Information Sciences,179. pp. 2232–2248. (2009).
[24] B. Barzegar, A. Rahmani, K. Zamanifar, A. Divsalar. " Gravitational Emulation Local Search Algorithm for Advanced Reservation and Scheduling in Grid Computing Systems" , Fourth International Conference on Computer Sciences and Convergence Information Technology, (2009).