Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2012, Article ID 651310, 20 pages
doi:10.1155/2012/651310

*Research Article*

# Solving Flexible Job-Shop Scheduling Problem Using Gravitational Search Algorithm and Colored Petri Net

**Behnam Barzegar,[1] Homayun Motameni,[2] and Hossein Bozorgi[3]**

[1] *Department of Computer Engineering, Islamic Azad University, Nowshahr Branch, Nowshahr 4817713655, Iran*
[2] *Department of Computer Engineering, Islamic Azad University, Sari Branch, Sari, Iran*
[3] *Department of Computer Engineering, Mazandaran University of Science and Technology, Iran*

Correspondence should be addressed to Behnam Barzegar, behnam.barzegar@yahoo.com

Received 31 December 2011; Revised 6 April 2012; Accepted 14 April 2012

Academic Editor: Nazim I. Mahmudov

Copyright © 2012 Behnam Barzegar et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

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 customer orders and supplying requested materials in suitable time. In flexible job-shop scheduling production systems, we could reduce time and costs by transferring and delivering operations on existing machines, that is, among NP-hard problems. The scheduling objective minimizes the maximal completion time of all the operations, which is denoted by Makespan. 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, new algorithm were proposed for flexible job-shop scheduling problem systems (FJSSP-GSPN) that is based on gravitational search algorithm (GSA). In the proposed method, the flexible job-shop scheduling problem systems was modeled by color Petri net and CPN tool and then a scheduled job was programmed by GSA algorithm. The experimental results showed that the proposed method has reasonable performance in comparison with other algorithms.

## 1. Introduction

Classic job-shop scheduling problem 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 problem system is specific

type of classic job-shop scheduling production systems, in which one job could be implemented on a 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, FJSSP-GSPN algorithm based on gravitational local search and time Petri net is proposed for scheduling time optimization in FJSSP. The proposed algorithm comprised two stages: in the first stage, the system was modeled by timed Petri net and the simulation of results was done with Petri net method. In the second stage, a new algorithm based on gravitational local search algorithm was proposed which is called FJSSP-GSPN.

In the simulation stage, in order to determine performance of the system, one job has been simulated by CPN tool with several suboperations. Also, in the second stage, gravitational searching algorithm and proposed solution have been presented to make suitable time for implementing several operations on one job, which in fact is assigning proper the machine to the related operation.

The rest of the paper is as follows: In Section 3, problem analyzing and in Section 4, its disjunctive graph model are presented. In Section 5, colored Petri Net and in Section 6 simulating phase with CPN tool is described. In Section 7, gravitational searching algorithm is explained and finally in section eight, we explain the proposed solution by using the gravitational searching algorithm.

## 2. Related Work

Flexible job-shop scheduling problem system is one of the most important combined optimization problems, and is kind of NP-hard problem.

The job-shop scheduling problem (JSSP) 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 problem system (FJSSP).

Brucker and Schlie [1] who first considered Job-shop scheduling with multipurpose Multipurpose computing machines, 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 decrease complexity, the main problem should be decomposed into two problems called assignment and sequence determination. Frequent usage of this perception is due to decomposition it into two assignment problem and sequence determination problem. Brandimarte [2] was the first one who used this perception for FJSSP. He specified path determination with distribution rules and then focused on solving scheduling problem with tabu search algorithm.

Jain and Meeran [3] 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. [4] demonstrated that JSPs are nondeterministic 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, that is, to

minimize the makespan. Many approximate methods have been developed to overcome the limitations of exact enumeration techniques.

These approximate approaches include simulated annealing (SA) (Lourenço [5]), tabu search (Sun et al. [6]; Nowicki and Smutnicki [7]; Pezzella and Merelli [8]), and genetic algorithms (GA) (Bean [9]; Kobayashi et al. [10]; Gonçalves et al. [11]; Wang and Zheng [12]).

Fattahi et al. [13] have considered hierarchical and integrated perceptions in relation to scheduling job-shop production systems. They based on these perceptions 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.

I. C. Choi and D. S. Choi [14] have presented a local searching algorithm for scheduling job-shop production problems. They regarded that there is a possibility of a substitute operation for any operation. In this mode, a machine and a process time are assigned for all operations, and then for some other operations, alternative machines and process time are dedicated. Moreover, a run time has been considered for any operations, which is depended of the last operation.

Xia and Wu [15] have presented a hybrid optimizing perception for scheduling multiobject *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. Dedicated PSO algorithm for either assignment problem or determining operations uses designated machine. Value of object function is calculated by SA algorithm and implemented for each particle in PSO algorithm once.

Mastrolilli and Gambardella [16] 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. [17] and Gao et al. [18] 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. [19] propose a parallel variable neighborhood search (VNS) algorithm for the FJSP based on independent VNS runs. Defersha and Chen [20] 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 the parallelization side of the programming methodology and they do not use any special properties of the FJSP.

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

## 3. Flexible Job-Shop Scheduling Problem Systems' Analysis

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

Flexible job-shop scheduling production system contains $N$ job on $M$ machines. Each job includes some operations and for each operation there is an opportunity to use a set of operational machines. As flexible job-shop scheduling systems have considerable importance in production centers, they have attracted attention of production unit managers.

Furthermore, specific mathematical characteristics of this problem that have offered effective strategies for solving this problem are interested for researchers of this area of mathematics field. Simple form of flexible job-shop scheduling production systems is classic job-shop scheduling production system which schedules $n$ job of $J_1, J_2, \ldots, J_n$ 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 determining the sequence of operations for each machine, such 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 $O_{j,h}$), a set of jobs are assigned, which have the capacity of performing that operation. This set is presented as $M_{j,h} \subset M$. Each machine would have a 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 value of this variable be 1, then it means that machine $j$ is selected among operational machines for implementing $O_{i,j}$ operation.

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

For solving the 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 a model. Moreover, an assumed the job which the number of its operations is equal to number of machines is considered as the 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 the startup time of job $k$ after a job from family $f$ on machine $i$.

$$F_{f,j} = \begin{cases} 1 & \text{if } k \in f \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

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

Variables of this model include:

$$y_{i,j,h} = \begin{cases} 1 & \text{if machine } i \text{ seclect for operation } O_{j,h} \\ 0 & \text{otherwise} \end{cases}$$
$$x_{i,j,h,k,l} = \begin{cases} 1 & \text{if } O_{j,h} \text{ proceceds } O_{k,l} \text{ immediately on machine } i \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

$C_{\max}$: 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 $i$, and $S_{i,j,k}$: start up time for job $k$ on machine $j$ if the previous job is job $j$.

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

(1) $\min C_{\max}$;

(2) $t_{k,l} + y_{i,k,l} \cdot p_{i,k,l} \le fi_{k,l}$ for $i = 1,\ldots,m,\ k = 1,\ldots,n,\ l = 1,\ldots,h_k$;

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

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

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

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

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

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

(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) $x_{i,j,h,k,l} \le y_{i,k,l}$ for $j = 1,\ldots,n,\ k = 1,\ldots,n,\ h = 1,\ldots,h_j,\ l = 1,\ldots,h_k,\ i = 1,\ldots,m$;

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

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

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

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

Constraint 1 is the object function of a 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 that prerequirement limitations are respected. Constraint 5 defines $C_{\max}$. Constraint 6 causes that required machines for each operation is selected among 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 the process time of preparing machine $i$. Constraint 9 causes that among all assignable machines for a specific operation just one machine selected. Constraints 10 and 11 imply that only one

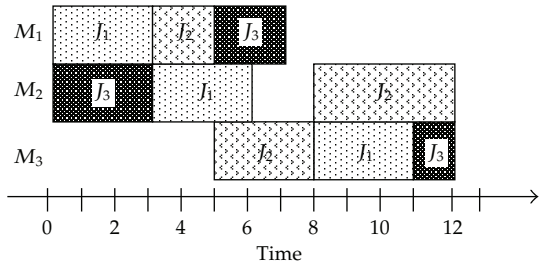



**Figure 1:** A Gantt chart representation of a solution for a $3 * 3$ problem.

**Table 1:** Illustration of an example of $3 * 3$ problem.

| job | Operations routing (processing time) | | |
|---|---|---|---|
| 1 | 1 (3) | 1 | 1 (3) |
| 2 | 1 (2) | 2 | 1 (2) |
| 3 | 2 (3) | 3 | 2 (3) |

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

## 4. The Disjunctive Graph Model

The JSSP can be described as a disjunctive graph $G = (V; C \cup D)$, where (1) $V$ is a set of nodes representing operations of the jobs together with two special nodes, a *source* (0) and a *sink,* representing the beginning and end of the schedule, respectively.

(2) $C$ is a set of conjunctive arcs representing technological sequences of the operations. (3) $D$ is a set of disjunctive arcs representing pairs of operations that must be performed on the same machines. The processing time for each operation is the weighted value attached to the corresponding nodes.

Figure 2 shows this in a graph representation for the problem given in Table 1. The Gantt-Chart is a convenient way of visually representing a solution of the FJSSP. An example of a solution for the $3 * 3$ problem in Table 1 is given in Figure 1.

Job-shop scheduling can also be viewed as defining the ordering between all operations that must be processed on the same machine, that is, to fix precedences between these operations. In the disjunctive graph model, this is done by turning all undirected (disjunctive) arcs into directed ones. A *selection* is a set of directed arcs selected from disjunctive arcs. By definition, a selection is *complete* if all the disjunctions are selected. It is *consistent* if the resulting directed graph is acyclic.

## 5. Colored Petri Nets

The Petri Nets theory was born from the thesis defended by Carl Adam Petri in the Faculty of Mathematics and Physics of the Technical University of Darmstadt (Germany) in 1962, entitled "Communication with autonomata." At the end of the 60s and beginning of the 70s,

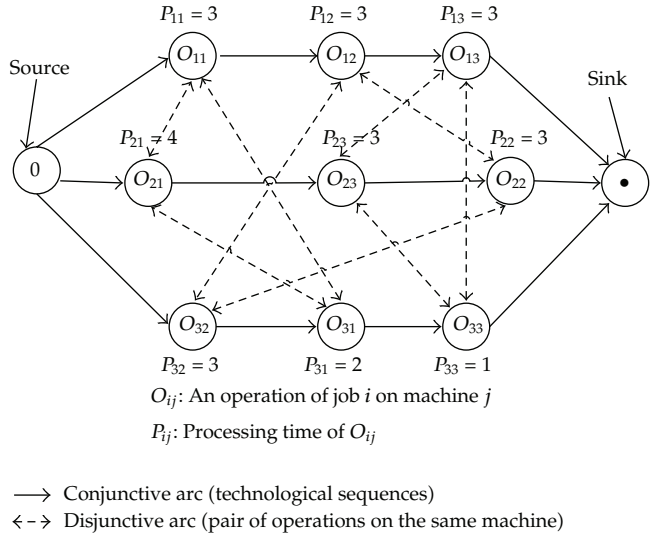



**Figure 2:** A disjunctive graph of a $3 * 3$ problem.

researchers from MIT in the USA developed the foundations of the concept of the Petri nets as we know today.

According to Murata [21], Petri nets are a type of bipartite, directed, and weighted graph, which can capture the dynamics of a discrete-event system. The Petri nets provide a compact representation of a system because they do not represent explicitly all the space of states from the modeled system.

An ordinary Petri net is a 4-tuple $PN = (P, T, Pre\ post)$, formed by a finite set of places $P$ of dimension $n$, a finite set of transitions $T$ of dimension $m$, an input condition $Pre: P \times T \rightarrow N$, and an output condition $Post: P \times T \rightarrow N$. To each place an integer nonnegative number denominated token is associated.

Models with time restrictions can be developed via Petri nets, as shown for example, in [22]. Manufacturing, transportation, and telecommunication systems are some of the examples of application of that methodology.

A limitation of the ordinary Petri nets, also called *place/transition Petri nets*, is the fact that they demand a large quantity of places and transitions to represent complex systems (as most real systems are). As the net expands, the general view of the modeled system starts to get compromised, and the analysis of the modeled system becomes difficult to do.

Real systems often present similar processes which occur in parallel or concurrently, and they differ from each other only by their inputs and outputs. In the colored Petri nets, the quantity of places, transitions, and arcs is, generally, sensibly reduced via the addition of data to the structure of the net.

According to Jensen [23] a more compact representation of a Petri net is obtained via the association of a data set (denominated token colors) to each token. The concept of color is analogous to the concept of type, common among the programming languages.

Colored Petri net (CPN) is a tool by which validation of discrete-event systems are studied and modeled. CPNs are used to analyze and obtain significant and useful information from the structure and dynamic performance of the modeled system. Colored Petri nets mainly focus on synchronization, concurrency, and asynchronous events. The

graphic features of CPNs specify the applicability and visualization of the modeled system. Furthermore, synchronous and asynchronous events present their prioritized relations and structural adaptive effects. The main difference between CPNs and Petri nets (PN) is that in CPNs the elements are separable but in PNs they are not. Colored indicates the elements specific feature. The relation between CPNs and ordinary PNs is analogous to high-level programming languages to an assembly code (Low-level programming language). Theoretically, CPNs have precise computational power but practically since high-level programming languages have better structural specifications, they have greater modeling power.

CPN's drawback is its nonadaptivity therefore it is not possible to access the previous information available in CPNs. If there is more than one transition activated then each transition can be considered as the next shot. This colored Petri net's characteristic indicates that since several events occur concurrently and event incidences are not similar, then when events occur they do not change by time and this phenomenon is in contrast with the real and dynamic world. Simulation would be similar to execution of the main program. Our Purpose is to use the simulated model for analyzing the performance of the systems, as a result here the system problems and the weak points would be identified. However, classic CPN tools can do nothing to improve and solve problems and also it would not be possible to predict the next optimized situation.

According to Jensen [23], a colored Petri net is a 9 tuple:

$$\mathrm{CPN} = (\Gamma, P, T, A, N, C, G, E, I), \tag{5.1}$$

where $\Gamma$ is a finite, nonempty set of types, denominated *colors set*; $P$ is a finite set of places of dimension $n$; $T$ is a finite set of transitions of dimension $m$; $A$ is a finite set of arcs so that $P \cap T = P \cap A = T \cap A = \emptyset$; $N$ is a node function, defined from $A$ by $P \times T \cup T \times P$; $C$ is a color function, defined from $P$ on $\Gamma$; $G$ is a guard function, defined from $T$; $E$ is a function of expression of arcs, defined from $A$; $I$ is an initiation function, defined from $P$.

The *colors set* determines the types, operations and functions which can be associated to the expressions utilized on the net (arc functions, guards, colors, etc.) The sets *P, T, A* and *N* have analogous significance to the vertexes and precedence functions sets defined for the ordinary Petri nets. Color functions map every place on the net, including them in a color set. Guard functions map all the transitions on the net, moderating the stream of tokens according to Boolean expressions. Arc functions map each arc on the net, associating them to a compatible expression with the possible color sets. Finally, initialization functions map the places on the net associating them to the existent multisets.

There are four main components in Petri net.

(1) (•) Token: specify existence in system.

(2) (⬭) Place: temporary place for maintenance of Tokens.

(3) (→) Arc: show Token directions.

(4) (▯) Transition: specify main operation in system. A system can be modeled just by using these four simple elements.

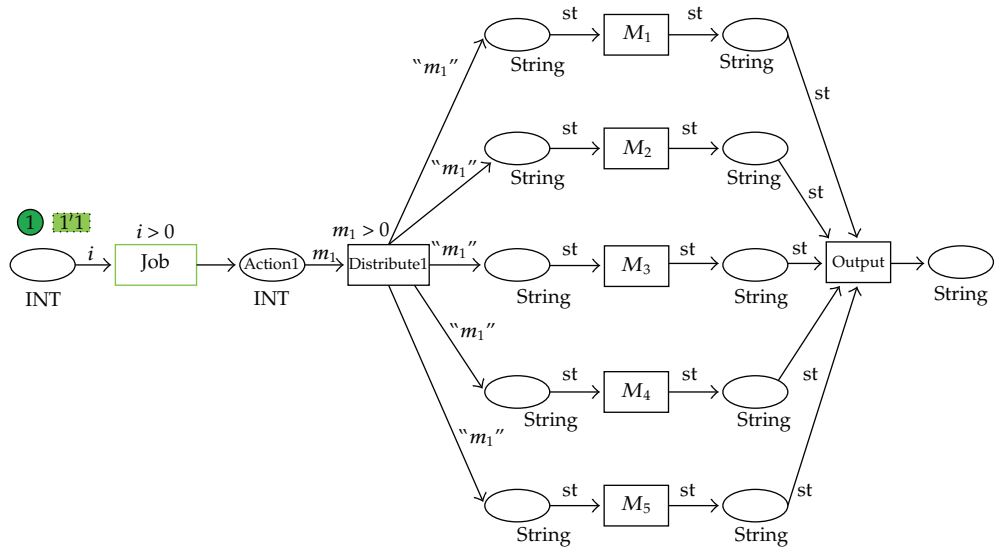



**Figure 3:** Example model for one job with one operation on five machines.

### *5.1. CPN Tool*

CPN Tools is an industrial-strength computer tool for constructing and analysing CPN models. Using CPN Tools, it is possible to investigate the behaviour of the modelled system using simulation, to verify properties by means of state space methods and model checking, and to conduct simulation-based performance analysis. User interaction with CPN Tools is based on direct manipulation of the graphical representation of the CPN model using interaction techniques, such as tool palettes and marking menus. The functionality of the tool can be extended with user-defined Standard ML functions [24].

## 6. The First Stage (Simulating Problem with CPN Tool)

Regarding problem analyzing and offered comments, performance of scheduling operation system in a flexible production job-shop is simulated with CPN tool. This simulation which could be contains one or more operations for one job, is implemented on one machine. Also, this simulation is extension able on more operations or even more jobs. First, suppose (Figure 3) that you have one job with one operation and a job-shop with several machines (up to 5 machines).

Next to each place, there is an inscription which determines the set of token colors (data values) that the tokens on the place are allowed to have. The set of possible token colors is specified by means of a type (as known from programming languages), and it is called the color set of the place. By convention the color set is written below the place. The places have the color set INT and STRING, color sets are defined using the CPN ML keyword colset.

The color sets are defined as:

```
colset STRING = string;
colset INT  =  int;
```

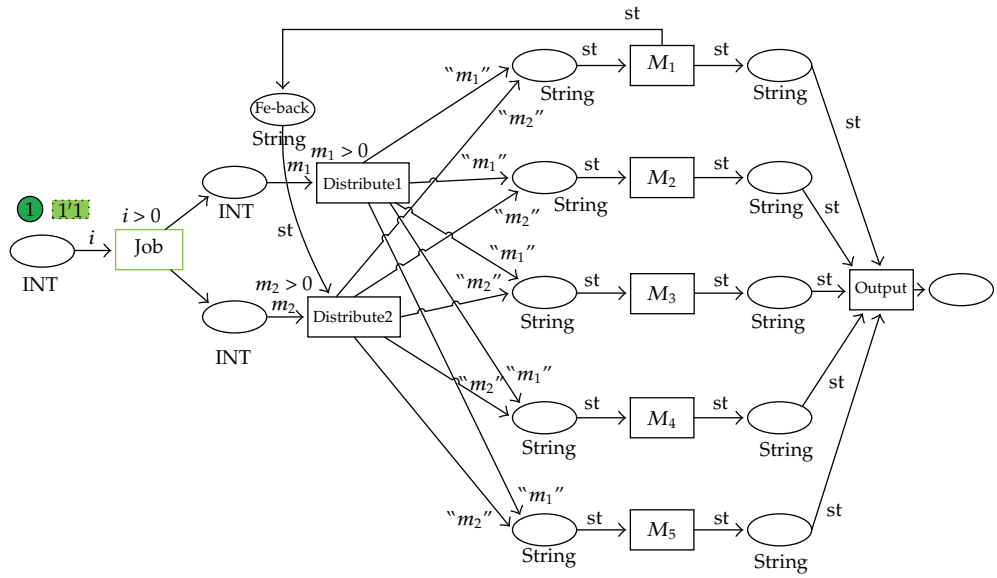



**Figure 4:** Example model for two jobs with one operation on five machines.

The arc expressions are written in the CPN ML programming language and are built from typed variables, constants, operators, and functions. When all variables in an expression are bound to values (of the correct type) the expression can be evaluated. An arc expression evaluates to a multiset of token colors. As an example, consider the two arc expressions *i* and *st* on the three arcs connected to the transition. They contain the variables *i* and *st* declared as follows:

var $i$ : INT;

var $st$ : STRING.

According to problem analysis, this job could be implemented on each machine, but the ideal machine is the one that performs this job in less time, so each machine sends operational time for the related job to the comprising section. The comprising section recognizes minimum time among input values and displays it with the name of the related machine.

Now suppose that we have one job with two operations, and according to problem definition, this operation should be implemented serially (Figure 4). Thus, after that first operation is implemented, second operation would be allowed by sending a return signal.

As you can see, this simulation for one job that has up to 5 operations has been implemented on five machines (Figure 5), but this simulation could be modeled on *j* job with *N* operation and *M* machines, too.

## 7. Gravitational Search Algorithm

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

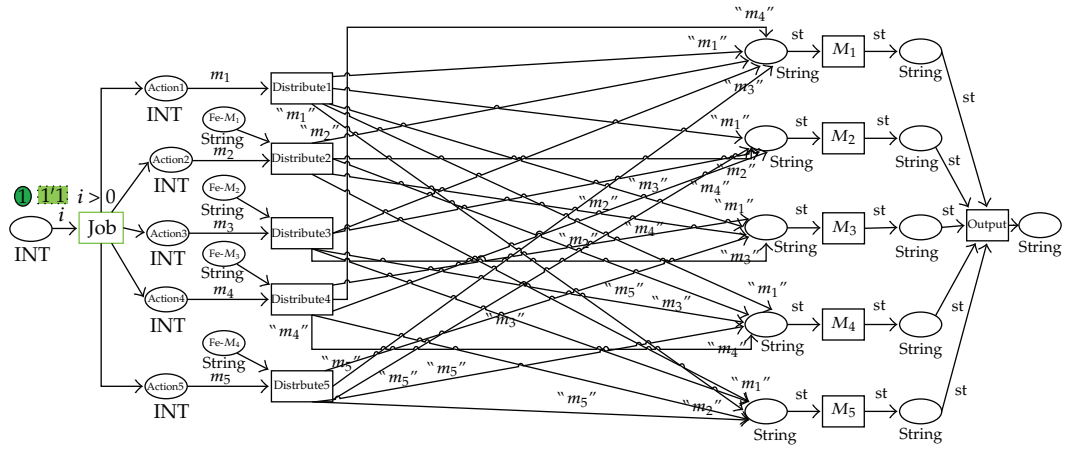



**Figure 5:** Example model for five jobs with one operation on five machines.

System area is the same as problem definition area. According to the 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 the 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 the first step, system space is determined. Area includes a multidimensional coordinated system in problem definition space.

Each point in space is one of the answers of a problem and search factors are also the series of masses.

Each mass has three properties:

(a) mass state, (b) gravitational mass, (c) inertia mass.

Above mentioned masses are resulted from *active gravitational mass* and *inertia mass* concepts in physics.

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

We suppose that there is only the gravity rule and movement rule. Their general forms are similar to nature rules and have been defined as follows.

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

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

In the 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 (7.1), where $x$ denotes position of mass $i$th in dimension $d$, and $n$ denotes number of dimensions in the search space.

$$X_i = \left(x_i^1, \ldots, x_i^d, \ldots, x_i^D\right) \tag{7.1}$$

Worst($t$) and Best($t$) are for the *minimization* problems and are calculated as follows. (for *maximization* problems it is just enough to consider the inverse of these two relations).

$$\begin{aligned} \text{Best}(t) &= \max_{j\in\{1,\ldots,m\}} \text{fit}_j(t) \\ \text{worst}(t) &= \min_{j\in\{1,\ldots,m\}} \text{fit}_j(t). \end{aligned} \tag{7.2}$$

We can account fitness of recent population with relation (7.3), and obtain mass of factor $i$th in time $t$ (i.e., with relation (7.4)), where $M$ and fit are denoted mass and fitness of factor $i$th in time $t$, respectively.

$$q_i(t) = \frac{\text{fit}_i - \text{worst}(t)}{\text{Best}(t) - \text{worst}(t)}, \tag{7.3}$$

$$M(t) = \frac{q_i(t)}{\sum_{j=1}^{s}(t)}. \tag{7.4}$$

In this system, force $F$ is delivered on mass $i$th from mass $j$th in time $t$ in the direction of dimension $d$, the value of this force is obtained based on the following; $G(t)$ is gravity constant in time $t$ which is regulated in the beginning of the operating algorithm, and it is decreased by the time.

$$F_{ij}^d(t) = \frac{G(t) \times M_j(t)}{R_{ij}(t) + \varepsilon}\left(x_j^d(t) - x_i^d(t)\right). \tag{7.5}$$

$R$ is the *ECLIDIAN* distance between factor $i$th and factor $j$th that are defined as, "$\varepsilon$" is also a small value for avoiding denominator from becoming zero.

$$ij = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2 + \cdots + (n_2 - n_1)^2}. \tag{7.6}$$

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). $K_{\text{best}}$ denote 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 the

**Table 2:** Semi-code of gravitational algorithm.

| |
|---|
| (1) Determining system area and initial valuing |
| (2) Initial positioning OF the masses |
| (3) Evaluating THE 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 does not meet, go to phase 3 |

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

$$F_i^d(t) = \sum_{j \in K_{\text{best}}, j \neq i} \text{rand} j \times G(t) \frac{M_j(t) \times M_i(t)}{R_{ii}(t) + \varepsilon} \left( x_j^d(t) - x_i^d(t) \right). \tag{7.7}$$

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.

$$\begin{aligned} a_i^d(t) = \frac{F_i^d(t)}{M_i(t)} \Longrightarrow \\ a_i^d(t) = \sum_{j \in K_{\text{best}}, j \neq i} \text{rand} j \times G(t) \frac{M_j(t)}{R_{ii}(t) + \varepsilon} \left( x_j^d(t) - x_i^d(t) \right). \end{aligned} \tag{7.8}$$

Speed of each mass is equal to sum of coefficient of the mass' recent speed and acceleration, and is explained as follows. 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.

$$V_i^d(t+1) = \text{rand}_i \times V_i^d(t) + a_i^d(t). \tag{7.9}$$

Now, mass should move. It is obvious that more speed of the mass, causes more movement in that dimension. New state of factor $i$th is mentioned by relation (7.10).

$$x_i^d(t+1) = x_i^d(t) + V_i^d(t+1). \tag{7.10}$$

At the beginning of the 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 Table 2, semicode of this algorithm has been presented.

| 2 | 3 | 1 | 4 |
|---|---|---|---|

**Figure 6:** Example of state array.

**Table 3:** Example of a job with 4 actions on 4 machines.

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

## 8. Proposed Method Based on Gravitational Search Algorithm (FJSSP-GSPN)

In regards to gravitational searching algorithm, each searching factor should 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 (see Table 3), 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.

You could see with some attention that Table 3 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 the 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 the searching space one sample of it (Figure 6).

It is obvious that each house of this array is assigned to one column of the table, and value of that house states the number of machines that the related job would be implemented by. For example, second house of array indicates that the 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 the 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 (7.3). Now if based on formula (7.4), 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 the application calculations on dimension $X$, calculations should be stopped, since second mass does not deliver force on first mass in direction of dimension $X$.

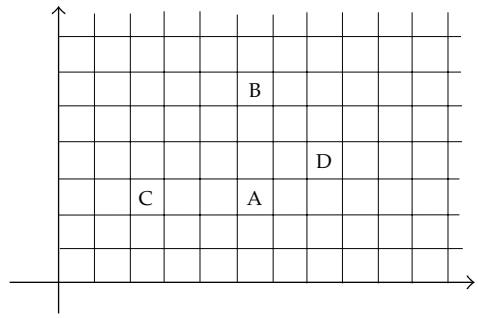



**Figure 7:** Two-dimensional space with 4 masses.

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

**Algorithm 1:** First condition: unparallelism of two masses.

For example, in Figure 7, you see that two masses (A and B) are placed in one column, so they do not deliver force in direction of dimension $X$ on each other.

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

Therefore, in first condition, we investigate unparallelism 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 $K_{\text{best}}$ series (Algorithm 2).

$K_{\text{best}}$ 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 $K_{\text{best}}$ series as it has shown in Algorithm 1.

It is obvious that according to gravitational algorithm, in the next moment, we should add the condition of "being heavier masses" to the first condition, that is, in addition to

```
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]));
```

**Algorithm 2:** Calculating for each mass.

```
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])) /
        (R+E))*Math.Abs(gls_Loc_Arr[ k_Best_Temp] –
            gls_Loc_Arr[k]));
}
```

**Algorithm 3:** Calculating $R$ and force.

condition of unparallelism of masses, those masses which are heavier than recent masses, should be added to $K_{\text{best}}$ series.

Now, we could write calculations based on Algorithm 2.

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

We could use following pseudocode for these calculations (Algorithm 3).

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

In proposed solution, we divide state array on $N$-dimensions of search space, that is, 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 8).

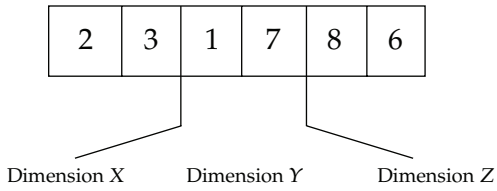



**Figure 8:** State array and dimensions.

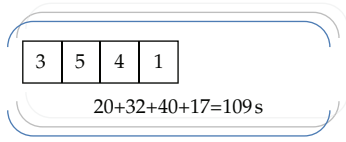



**Figure 9:** Result array.

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 the related dimension, and only corresponding cells with that dimension may be changed 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 just one mass is remained, search would be finished, and by considering number of remained mass in array (best mass), list of machines is presented for processing remained operation of respected job, so that production time would be minimized.

For instance, array (Figure 9) shows that if the first job is implemented by the third machine, the second job is implemented by the 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).

## 9. Experimental Results

To illustrate the effectiveness and performance of the proposed algorithm in this paper, we consider 43 instances from two classes of standard JSP test problems: instances FT06, FT10, and FT20 designed by Fisher and Thompson and instances LA01–LA40 designed by Lawrence. All the test problems are taken from web ftp://mscmga.ms.ic.ac.uk/pub/jobshop1.txt [30].

All the runs were carried out on a Intel Pentium Core i5 Duo 2.4 GHz Processor and 4 GB RAM configuration system. The algorithm was coded in C# language under the operation system Windows XP. Numerical results are compared with those reported in some existing literature works using other approaches [25–30], including some heuristic and metaheuristic algorithms. Take the benchmark problem FT10 for instance.

**Table 4:** Experimental results.

| Problem | Size $(n, m)$ | $C^*$ | Nowicki [25] | TSSB [26] | Sabuncuoglu [27] | HIA [28] | F&F [29] | HPSO [30] | Obtained Values from the proposed algorithm (GSPN) |
|---|---|---|---|---|---|---|---|---|---|
| FT06 | 6,6 | 55 | 55 | 55 | — | 55 | 55 | 55 | 55 |
| FT10 | 10,10 | 930 | 930 | 930 | 1016 | 930 | 930 | 930 | 930 |
| FT20 | 20,5 | 1160 | 1165 | 1165 | — | 1165 | 1165 | 1165 | 1160 |
| LA01 | 10,5 | 666 | 666 | 666 | 666 | 666 | 666 | 666 | 666 |
| LA02 | 10,5 | 653 | 655 | 655 | 704 | 655 | 655 | 655 | 653 |
| LA03 | 10,5 | 590 | 597 | 597 | 650 | 597 | 597 | 597 | 590 |
| LA04 | 10,5 | 590 | 590 | 590 | 620 | 590 | 590 | 590 | 590 |
| LA05 | 10,5 | 593 | 593 | 593 | 593 | 593 | 593 | 593 | *594* |
| LA06 | 15,5 | 919 | 926 | 926 | 926 | 926 | 926 | 926 | 919 |
| LA07 | 15,5 | 890 | 890 | 890 | 890 | 890 | 890 | 890 | 890 |
| LA08 | 15,5 | 862 | 863 | 863 | 863 | 863 | 863 | 863 | 862 |
| LA09 | 15,5 | 951 | 951 | 951 | 951 | 951 | 951 | 951 | 951 |
| LA10 | 15,5 | 958 | 958 | 958 | 958 | 958 | 958 | 958 | 958 |
| LA11 | 20,5 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 | 1222 |
| LA12 | 20,5 | 1030 | 1039 | 1039 | 1039 | 1039 | 1039 | 1039 | 1030 |
| LA13 | 20,5 | 1145 | 1150 | 1150 | 1150 | 1150 | 1150 | 1150 | 1145 |
| LA14 | 20,5 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 | 1292 |
| LA15 | 20,5 | 1207 | 1207 | 1207 | 1207 | 1207 | 1207 | 1207 | 1207 |
| LA16 | 10,10 | 936 | 945 | 945 | 988 | 945 | 945 | 946 | 936 |
| LA17 | 10,10 | 784 | 784 | 784 | 827 | 784 | 784 | 784 | ***790*** |
| LA18 | 10,10 | 845 | 848 | 848 | 881 | 848 | 848 | 848 | 845 |
| LA19 | 10,10 | 842 | 842 | 842 | 882 | 842 | 842 | 842 | 842 |
| LA20 | 10,10 | 897 | 902 | 902 | 948 | 902 | 907 | 902 | 897 |
| LA21 | 15,10 | 1023 | 1046 | 1047 | 1154 | 1046 | 1052 | 1057 | 1023 |
| LA22 | 15,10 | 927 | 927 | 927 | 985 | 927 | 927 | 927 | 927 |
| LA23 | 15,10 | 1032 | 1032 | 1032 | 1051 | 1032 | 1032 | 1032 | ***1043*** |
| LA24 | 15,10 | 935 | 935 | 935 | 992 | 938 | 941 | 938 | ***938*** |
| LA25 | 15,10 | 964 | 977 | 977 | 1073 | 979 | 982 | 979 | 964 |
| LA26 | 20,10 | 1203 | 1218 | 1218 | 1269 | 1218 | 1218 | 1218 | 1203 |
| LA27 | 20,10 | 1228 | 1235 | 1236 | 1316 | 1235 | 1242 | 1236 | 1228 |
| LA28 | 20,10 | 1203 | 1216 | 1216 | 1373 | 1216 | 1225 | 1225 | 1203 |
| LA29 | 20,10 | 1140 | 1152 | 1160 | 1152 | 1168 | 1176 | 1168 | 1140 |
| LA30 | 20,10 | 1355 | 1355 | 1355 | 1435 | 1355 | 1355 | 1355 | 1355 |
| LA31 | 30,10 | 1784 | 1784 | 1784 | 1784 | 1784 | 1784 | 1784 | 1784 |
| LA32 | 30,10 | 1850 | 1850 | 1850 | 1850 | 1850 | 1850 | 1850 | 1850 |
| LA33 | 30,10 | 1719 | 1719 | 1719 | 1719 | 1719 | 1719 | 1719 | ***1725*** |
| LA34 | 30,10 | 1721 | 1721 | 1721 | 1780 | 1721 | 1721 | 1721 | 1721 |
| LA35 | 30,10 | 1888 | 1888 | 1888 | 1888 | 1888 | 1888 | 1888 | 1888 |
| LA36 | 15,15 | 1268 | 1268 | 1268 | 1401 | 1268 | 1281 | 1279 | ***1275*** |
| LA37 | 15,15 | 1391 | 1397 | 1407 | 1503 | 1411 | 1418 | 1423 | 1391 |
| LA38 | 15,15 | 1196 | 1196 | 1196 | 1297 | 1201 | 1213 | 1196 | ***1200*** |
| LA39 | 15,15 | 1230 | 1233 | 1233 | 1369 | 1240 | 1250 | 1247 | 1230 |
| LA40 | 15,15 | 1212 | 1222 | 1229 | 1347 | 1233 | 1228 | 1236 | 1212 |

Table 4 summarizes the results of the experiments on the 43 instances. The contents of the table include the name of each problem, the scale of the problem (size $n \times m$), the value of the best-known solution for each problem ($C^*$), the value of the best solution found by using the proposed algorithm (GSPN). From the table, it can be seen that the proposed algorithm is able to find the best known solution for 37 instances, and the deviation of the minimum found makespan from the best known solution is also very small. The proposed algorithm can yields good solution with respect to almost all other algorithms, The superior results indicate the successful incorporation of PSO and SA, which facilitates the escape from local minimum points and increases the possibility of finding a better solution. Therefore, it can conclude that the proposed GSPN solves the JSP efficiently.

## 10. Conclusion

In this paper, gravitational search algorithm for solving the flexible job-shop scheduling problem is presented. Due to their special structure, our Gravitational search-based algorithm works faster and more efficiently than other known algorithms. Our primary objective is to show that the proposed exploiting gravity leads to an efficient heuristic for the FJSSP. We presented computational results derived by testing the algorithms which have been developed in this paper on a number of benchmark problems. The gravitational search algorithms yield excellent results for almost all problems. Finally, we believe that the methodology used in this paper can be extended to solve other scheduling problems.

## Acknowledgments

The authors gratefully acknowledge the comments of the referee which improved the presentation of the paper.

## References

[1] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[2] P. Brandimarte, "Routing and scheduling in a flexible job shop by tabu search," *Annals of Operations Research*, vol. 41, no. 3, pp. 157–183, 1993.

[3] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: past, present and future," *European Journal of Operational Research*, vol. 113, no. 2, pp. 390–434, 1999.

[4] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of Operations Research*, vol. 1, no. 2, pp. 117–129, 1976.

[5] H. R. Lourenço, "Local optimization and the jobshop scheduling problem ," *European Journal of Operational Research*, vol. 83, no. 2, pp. 347–364, 1995.

[6] D. Sun, R. Batta, and L. Lin, "Effective job shop scheduling through active chain manipulation," *Computers and Operations Research*, vol. 22, no. 2, pp. 159–172, 1995.

[7] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996.

[8] F. Pezzella and E. Merelli, "A tabu search method guided by shifting bottleneck for the job shop scheduling problem," *European Journal of Operational Research*, vol. 120, no. 2, pp. 297–310, 2000.

[9] J. Bean, "Genetic algorithms and random keys for sequencing and optimization," *Journal on Computing*, vol. 6, pp. 154–160, 1994.

[10] S. Kobayashi, I. Ono, and M. Yamamura, "An efficient genetic algorithm for job shop scheduling

problems," in *Proceedings of the 6th International Conference on Genetic Algorithms*, L. J. Eshelman, Ed., pp. 506–511, Morgan Kaufman, San Francisco, Calif, USA, 1995.

[11] J. F. Gonçalves, J. J. M. Mendes, and M. G. C. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European Journal of Operational Research*, vol. 167, no. 1, pp. 77–95, 2005.

[12] L. Wang and D.-Z. Zheng, "An effective hybrid optimization strategy for job-shop scheduling problems," *Computers & Operations Research*, vol. 28, no. 6, pp. 585–596, 2001.

[13] P. Fattahi, M. Saidi Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *Journal of Intelligent Manufacturing*, vol. 18, no. 3, pp. 331–342, 2007.

[14] I. C. Choi and D. S. Choi, "A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups," *Computers and Industrial Engineering*, vol. 42, no. 1, pp. 43–58, 2002.

[15] W. Xia and Z. Wu, "An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems," *Computers and Industrial Engineering*, vol. 48, no. 2, pp. 409–425, 2005.

[16] M. Mastrolilli and L. M. Gambardella, "Effective neighbourhood functions for the flexible job shop problem," *Journal of Scheduling*, vol. 3, no. 1, pp. 3–20, 2000.

[17] F. Pezzella, G. Morganti, and G. Ciaschetti, "A genetic algorithm for the flexible job-shop scheduling problem," *Computers and Operations Research*, vol. 35, no. 10, pp. 3202–3212, 2008.

[18] J. Gao, L. Sun, and M. Gen, "A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems," *Computers & Operations Research*, vol. 35, no. 9, pp. 2892–2907, 2008.

[19] M. Yazdani, M. Amiri, and M. Zandieh, "Flexible job-shop scheduling with parallel variable neighborhood search algorithm," *Expert Systems with Applications*, vol. 37, no. 1, pp. 678–687, 2010.

[20] F. M. Defersha and M. Chen, "A coarse-grain parallel genetic algorithm for flexible job-shop scheduling with lot streaming," in *Proceedings of the International Conference on Computational Science and Engineering (CSE '09)*, pp. 201–208, August 2009.

[21] T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[22] B. Berthomieu and M. Diaz, "Modeling and verification of time dependent systems using time Petri nets," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 259–273, 1991.

[23] K. Jensen, *Coloured Petri Nets Vol. 1: Basic Concepts*, EATCS Monographs on Theoretical Computer Science, Springer, Berlin, Germany, 1992.

[24] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri nets and CPN tools for modelling and validation of concurrent systems," *International Journal of Software Tools Technology Transfer*, vol. 9, no. 3, pp. 213–254, 2007.

[25] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996.

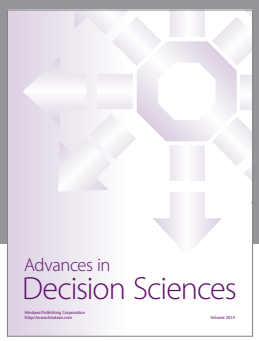
[26] F. Pezzella and E. Merelli, "A tabu search method guided by shifting bottleneck for the job shop scheduling problem," *European Journal of Operational Research*, vol. 120, no. 2, pp. 297–310, 2000.

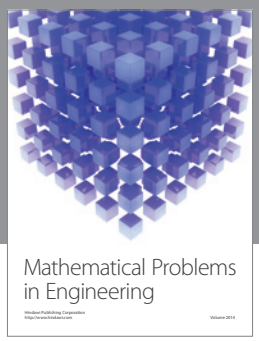
[27] I. Sabuncuoglu and M. Bayiz, "Job shop scheduling with beam search," *European Journal of Operational Research*, vol. 118, no. 2, pp. 390–412, 1999.

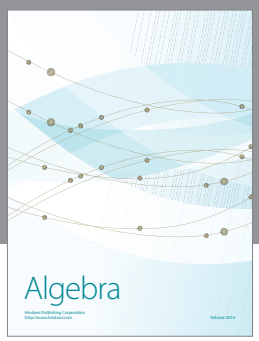
[28] H. W. Ge, L. Sun, Y. C. Liang, and F. Qian, "An effective PSO and AIS-based hybrid intelligent algorithm for job-shop scheduling," *IEEE Transactions on Systems, Man, and Cybernetics Part A*, vol. 38, no. 2, pp. 358–368, 2008.

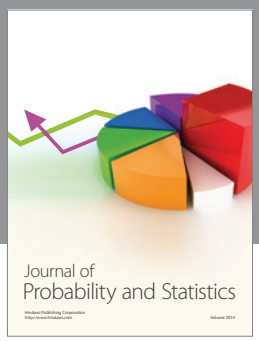
[29] C. Rego and R. Duarte, "A filter-and-fan approach to the job shop scheduling problem," *European Journal of Operational Research*, vol. 194, no. 3, pp. 650–662, 2009.

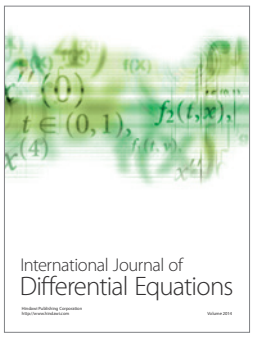
[30] S. Cun-li, L. Xiao-bing, W. Wei, and B. Xin, "A hybrid particle swarm optimization algorithm for job-shop scheduling problem," *International Journal of Advancements in Computing Technology*, vol. 3, no. 4, 2011.

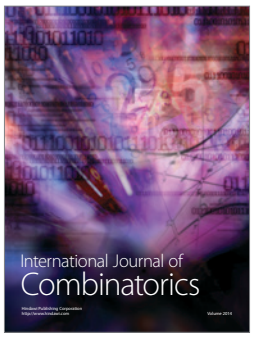


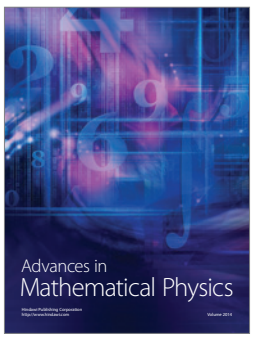


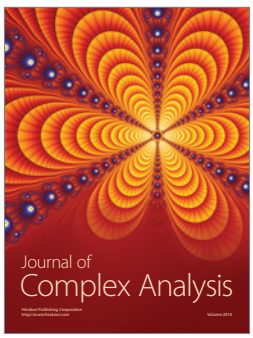


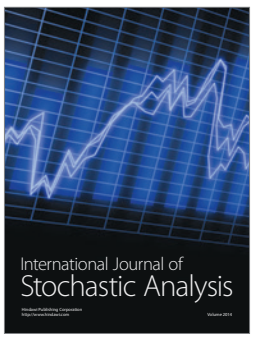


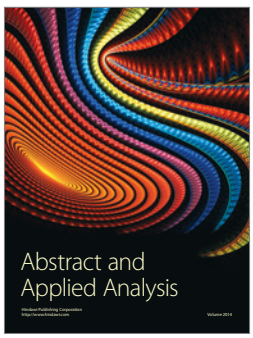


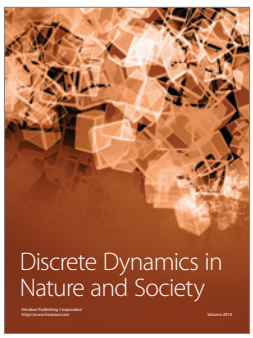


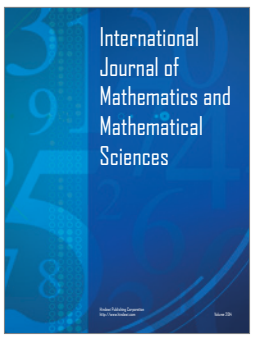


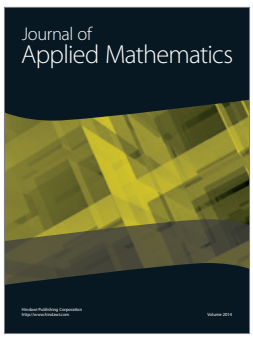












Hindawi
Submit your manuscripts at
http://www.hindawi.com