Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 9 (4): 662-670, 2009
ISSN 1812-5654

A Simulated Annealing Algorithm for Flexible Job-Shop Scheduling Problem

[1]M. Yazdani, [1]M. Gholami, [2]M. Zandieh and [3]M. Mousakhani
[1]Faculty of Industrial and Mechanical Engineering, Islamic Azad University, Qazvin, Iran
[2]Department of Industrial Management, Faculty of Management and Accounting,
Shahid Beheshti University, Tehran, Iran
[3]Faculty of Management and Accounting, Islamic Azad University, Qazvin, Iran

Abstract: This study addresses the flexible job-shop scheduling problem to minimize makespan. In fact, the FJSP mainly presents two difficulties. The first one is to assign each operation to a machine out of a set of capable machines and the second one is to sequence the assigned operations on all machines. Hence, to solve this NP-hard problem, a simulated annealing algorithm is proposed. The meta-heuristic algorithm explores the solution space using a stochastic local search while trying to avoid local optima through accepting probabilistic moves to the worse solutions. The neighborhood search structures of assignment and sequencing are used for generating neighboring solutions to search the solution space. To evaluate the performance of the algorithm, twenty benchmark problems adopted from the literature are used. Consequently, the computational results validate the quality of present approach.

Key words: Scheduling, flexible job-shop, simulated annealing, computational optimization, flexible manufacturing systems

INTRODUCTION

Scheduling problems exist almost everywhere in real-world situations (Gen and Cheng, 1997), especially in the industrial engineering world. Most scheduling problems are complex combinatorial optimization problems and they are mainly very difficult to solve. The job-shop scheduling problem (JSP), known as an NP-hard problem, is one of the hardest combinatorial optimization problems in the area. In JSP, a set of n jobs must be processed on m machines where each job i consists of n_i operations that should perform on the machines while satisfying precedence constraints. Each operation of a job should be processed only on one predetermined machine. The machines are incessantly available and they can process one operation at a time without interruption. JSP aims to find the appropriate sequencing of operations on the machines to optimize the performance indicator. Herein, a typical performance indicator for JSP is the makespan, which is defined as the time needed to complete all the jobs.

The flexible job-shop scheduling problem (FJSP) is a generalization of the classical JSP, where operations are allowed to be processed on any among a set of available machines. Inasmuch as FJSP introduces a further decision level known as job routes besides the sequencing of operations on machines, it is more difficult than the classical JSP.

Since these types of problems belong to NP-hard class, no exact method has so far been introduced to be able to tackle these problems within a reasonable amount of time. Hence, a variety of heuristic procedures such as dispatching rules, local search and metaheuristics procedures such as tabu search, simulated annealing and genetic algorithm have been applied to solve these problems and find optimal or near optimal schedule in a reasonable time, instead of looking for an exact solution. These heuristic and meta-heuristic algorithms can be classified into two main categories: hierarchical approaches and integrated approaches. In hierarchical approaches, the assignment of operations to machines and the sequencing of operations on the resources or machines are treated separately, whereas in integrated approaches, the assignment and sequencing problems are not separated. Indeed, the hierarchical approaches are based on the idea of decomposing the original problem in order to reduce its complexity. Despite the fact that the integrated approaches are much more difficult to solve, they generally achieve better results.

Brandimarte (1993) applied hierarchical approach for FJSP based on decomposition. He solved the routing sub-problem using some existing dispatching rules; then using

Corresponding Author: M. Zandieh, Department of Industrial Management, Faculty of Management and Accounting, Shahid Beheshti University, Tehran, Iran Tel/Fax: (009821) 88091567

a tabu-search algorithm, he concentrated on the sequencing sub-problem. Dauzere-Peres and Paulli (1997) defined a new neighborhood structure for the problem where there was no distinction between reassigning and resequencing an operation. A tabu search procedure is proposed based on integrated approach. Mastrolilli and Gambardella (2000) improved Dauzere-Peres and Paulli (1997) tabu search techniques and presented two neighborhood functions for this problem. Kacem *et al.* (2002) studied a Genetic Algorithm (GA) based on an integrated approach for FJSP and used a chromosome representation that combined both routing and sequencing information. They propounded a localization approach to solve the resource assignment problem and an evolutionary approach controlled by the assignment model for FJSP. Xia and Wu (2005) worked on a hierarchical approach to solve multi-objective FJSP. The approach made use of particle swarm optimization to assign operations on machines and simulated annealing algorithm to schedule operation on each machine (local search). Fattahi *et al.* (2007) proposed a mathematical model and two meta-heuristics approaches, simulated annealing and tabu search algorithms, for solving FJSP. Moreover, considering two developed meta-heuristics and concerning the integrated and hierarchical approaches, six different searching algorithms are presented. Pezzella *et al.* (2008) propounded a genetic algorithm for the problem based on integrated approach. This GA employed a mix of different rules for generating the initial population, selection of individuals and reproduction operators.

Gao *et al.* (2008) studied the FJSP with three objectives: min makespan, min maximal machine workload and min total workload. They developed a hybrid Genetic Algorithm (hGA) based on an integrated approach for this problem. Under the framework of the GA, Variable Neighborhood Descent (VND) is applied to each newly generated offspring to improve its quality before injecting it into the population. Tay and Ho (2008) solved the multiobjective flexible job-shop problems using dispatching rules discovered through genetic programming. They also applied an integrated approach to devise their algorithm. Experimental results showed that composite dispatching rules generated by the genetic programming framework outperforms the single dispatching rules and the composite dispatching rules selected from literature with respect to minimum makespan, mean tardiness and mean flow time objectives. Shi-Jin *et al.* (2008) developed a filtered-beam-search-based heuristic algorithm (HFBS) to find sub-optimal schedules within a reasonable computational time for the FJSP with multiple objectives of minimizing makespan, the total workload of machines and the workload of the most loaded machine.

Given the literature of FJSP, the heuristics and meta-heuristics have corroborated their merits to solve the problem. Simulated Annealing (SA) is one of the renowned meta-heuristics which have been applied successfully to miscellaneous optimization problems. SAs have proven their efficiency and effectiveness in a wide variety of optimization problems. This paper proposes SA algorithm based on an integrated approach to solve FJSP.

PROBLEM DEFINITION

The problem is to organize the execution of n jobs on m machines. There is a set $J = (J_1, J_2, ..., J_n)$ of jobs in this problem. The machine set is noted as Ma, $Ma = (M_1, M_2, ..., M_m)$. Each job J_i consists of a sequence of n_i operations $(O_{i,1}, O_{i,2}, ..., O_{i,n(i)})$ to be executed one after the other according to the given sequence. Each operation $O_{i,j}$, i.e., the operation j of job J_i, can be processed on any among a subset of compatible machines called $Ma(O_{i,j}) \in Ma$. If $Ma(O_{i,j}) = Ma$ for all operations, we have a total flexibility, otherwise flexibility would be partial. The following conditions and constraints in FJSP are as follows:

- Machines are independent from each other.
- Jobs are independent from each other; there are no precedence constraints among the operations of different jobs.
- All jobs and machines are available at time 0.
- Setup times of machines and move times between operations are negligible.
- The processing time of each operation is machine-dependent.
- At a given time, a machine can only execute one operation.
- Pre-emption is not allowed, i.e., each operation must be completed without interruption once it starts.
- The processing time of operation $O_{i,j}$ on machine M_r is $P_{i,j,M_r} > 0$.

The problem is assigning each operation to an appropriate machine (routing problem) and to sequence the operations on the machines (sequencing problem) in order to minimize the makespan, i.e., the time needed to complete all the jobs, which is defined as:

$$C_{max} = \max(C_i)$$

where, C_i is the completion time of job J_i. Data related to a given problem with partial flexibility is shown in Table 1,

in which rows correspond to operations and columns represent machines. In this example, symbol ∞ entry in the table means that a machine cannot execute the corresponding operation, i.e., it does not belong to the subset of compatible machines for that operation.

SIMULATED ANNEALING

Simulated annealing is a popular local search algorithm (meta-heuristic) for solving combinatorial optimization problems (Kirkpatrick *et al*., 1983; Cerny, 1985). SA searches the set of all possible solutions, reducing the chance of getting stuck in local optima by allowing moves to inferior solutions under the control of a randomized scheme. The name of the approach points to a direct analogy with the way that liquids freeze and crystallize or that metals cool and anneal.

The SA algorithm starts with an initial solution and iteratively moves towards other existing solutions. The algorithm generates a neighboring solution S' in the neighborhood of the candidate solution S. Then, the change of objective function value, $\Delta_{s,s'} = f(s')-f(s)$, is calculated. In case $\Delta_{s,s'}$ value is smaller than zero, to move to the neighboring solution is acceptable; otherwise, if $\Delta_{s,s'}$ value is greater than zero, to reduce the probability to get trapped in local optima, the SA may accept to move to an inferior neighboring solution depending on a randomized scheme. More precisely, the move is still accepted if $R<\exp(-\Delta_{s,s'}/T)$, where, T is a control parameter, called temperature and R is a uniform random number between interval (0,1). SA algorithm generally starts from a high temperature and then the temperature is gradually lowered. At each temperature, a search is carried out for a certain number of iterations. When the termination condition is satisfied, the algorithm will stop. The general SA algorithm is shown in Fig. 1.

SA FOR FJSP

Here, we meticulously elucidate the proposed SA algorithm to optimize the FJSP. Following the explanation of initialization of the algorithm, the main algorithm is illuminated.

Initialization step: In this step, it is necessary to define structural elements of SA such as Encoding scheme, Initial solution, Neighborhood search structures, Initial temperature, the law of decrease of temperature, Number of neighborhood search in each temperature and termination condition.

Encoding scheme: In this article, we use task sequencing list provided by Kacem *et al*. (2002) for represent our SA algorithm solutions. This string consists of representing the schedule in a list of NT operations: $(NT = \sum_{i=1}^{n} n_i)$. The length of the string is equal to the total number of operations. Each operation in this string is introduced by 3 indicators (i, j, k), where i signifies the job which an operation belongs to, j characterizes the progressive number of that operation within job i and k denotes the machine assigned to that operation. Consider the problem in Table 1.

Table 1: Processing times

	Machines			
Operations	M_1	M_2	M_3	M_4
$O_{1,1}$	4	7	6	5
$O_{1,2}$	2	6	∞	5
$O_{2,1}$	4	5	7	∞
$O_{2,2}$	5	∞	6	3
$O_{2,3}$	∞	5	4	7
$O_{3,1}$	5	3	∞	6
$O_{3,2}$	∞	∞	4	∞
$O_{4,1}$	2	4	∞	5
$O_{4,2}$	∞	4	2	∞
$O_{4,3}$	5	4	6	3

Initialization: Select an initial solution (s_0), an initial temperature (T_0), Number of neighborhood search in each temperature (nt) and termination condition.

Set $T \longleftarrow T_0$ and $s \longleftarrow s_0$;
repeat
  repeat
    Randomly select $s' \in N(s)$;
    Calculate $\Delta_{s,s'} = f(s') - f(s)$;
    if $\Delta_{s,s'} \le 0$ then $s \longleftarrow s'$;
    else generate random R uniformly in the range (0,1);
    if $R < \exp(-\Delta_{s,s'}/T)$ then $s \longleftarrow s'$;
  Until iteration_countre = nt
    Decrease of the temperature T;
Until termination condition is met.

Fig. 1: General simulated annealing algorithm step

S:	(3, 1, 2)	(2, 1, 1)	(3, 2, 3)	(4, 1, 1)	(2, 2, 4)	(4, 2, 3)	(1, 1, 4)	(1, 2, 1)	(4, 3, 2)	(2, 3, 3)

Fig. 2: Representation of solution s

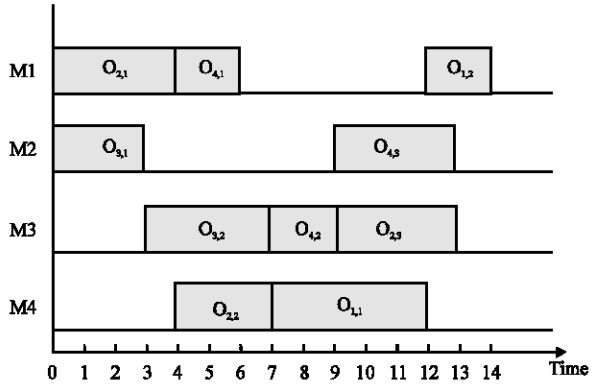

Fig. 3: Gantt chart of the solution

The solution s = ($O_{3,1}$, M_2), ($O_{2,1}$, M_1), ($O_{3,2}$, M_3), ($O_{4,1}$, M_1), ($O_{2,2}$, M_4), ($O_{4,2}$, M_3), ($O_{1,1}$, M_4), ($O_{1,2}$, M_1), ($O_{4,3}$, M_2), ($O_{2,3}$, M_3) is one of the feasible solutions from solution space S_p for the mentioned problem in Table 1. The representation of this solution in the task sequencing list format is provided in Fig. 2.

For example, 7th cell including (1, 1, 4) signifies that the operation $O_{1,1}$ should be processed on machine 4 in second order and after operation $O_{2,2}$. Gantt chart of this solution is shown in Fig. 3 where, C_{max} = 14.

Initial solution: At first an initial population is generated. For generating initial assignments, two approaches presented by Pezzella *et al.* (2008) are used: Assignment rule 1 which searches for the global minimum in the processing times table and Assignment rule 2 which foresees to permute randomly the jobs and the machines before to apply the approach by localization of Kacem *et al.* (2002). The implementation of mix of these two rules generates the initial set of assignments. 40% of initial assignments can be generated by rule 1 and 60% by rule 2. Once the initial assignments are generated, it is necessary to determine how to sequence the operations on the machines. Obviously, the sequencing is feasible if it respects the precedence constraints among operations of the same job. The sequencing of the initial assignments is obtained by randomly selecting a job (random), being known as a dispatching rule. Then, the best solution is chosen from initial population for initial solution of algorithm. To optimize each problem, the number of initial population should be determined based on the size and complexity of the problem. In the computational result, we show how the number of initial population for the problems can be set.

NEIGHBORHOOD SEARCH STRUCTURE

The main function of Neighborhood Search Structure (NSS) is to produce a neighboring solution from current solution via making a slight change in it. A variety of NSSs have been applied to scheduling problems. These NSSs must work in a way that they prevent any infeasible solution. In SA search algorithm, the choice of NSSs can greatly influence algorithm performance. In this section, two types of NSSs employed in local search of the presented algorithm are explained. Each NSS is defined by a function.

Neighborhood Search Structure for Sequencing (NSS-Se): To change the sequence of operations in the candidate solution string, NSS-Se is utilized so that the assignment of operations to machines remains unchanged. In this NSS, With regard to the precedence constraints among operations of the same job, changes are implemented through the substitution of positions of operations which are placed in adjacent cells in candidate solution string. Each time NSS-Se is executed, the number of repetitions of the function related to this NSS equals k_{max} which tallies with the number of changes or moves in a candidate solution, aiming at generating a neighboring solution. The steps of the execution of NSS-Se are spelled out below:

First, the function iteration counter k begins with $1(k \leftarrow 1)$. Function of this NSS comprises two major steps: (a) one cell from among the existing cells in candidate solution string is selected (b) If the move is feasible in respect of precedence constraints, the operation of selected cell is substituted for the operation of its preceding cell. Otherwise, the operation of selected cell is substituted for the operation of its succeeding cell. In this case, the first repetition of function is accomplished and the next one begins, i.e., k←k+1. This process continues until the repetitions of the function of this NSS are completed ($k = k_{max}$) to acquire the neighboring solution. NSS-Se can be utilized along with various measures of k_{max} in the presented algorithm.

Neighborhood search structure for assignment (NSS-As): This NSS is used to change the assignment of operations to machines in candidate solution string. In this type of NSS, sequencing of operations on the machines does not change. Each time NSS-As is executed, the number of repetitions of the function related to the NSS equals A_{max} which tallies with the number of changes

in a candidate solution, aiming at generating a neighboring solution. Also, an appropriate strategy has been employed to select operations which we want to make a change in their assignments. To execute this strategy, at the start of the presented algorithm, in the initialization step, those operations which are operated on more than a machine are identified and located in a set (namely Z). An operation belonging to set Z is utilized whenever function in connection with this NSS is iterated. The operation is selected from set Z, because this NSS has no impact on operations which fail to operate on more than one machine. The steps of the execution of NSS-As are presented in the following:

First, the function iteration counter A starts from 1 (A←1). Then one operation is randomly selected for set Z. In the next step, the place of this operation, i.e., the cell containing the operation in the candidate solution string, must be specified. Subsequently, changing the assignment of the selected operation starts. From among the available machines to process the selected operation $O_{i,j}$ ($Ma_{i,j}$), a new machine is chosen at random and an operation will be assigned to it. Accordingly, the first repetition of the function will be accomplished. Afterward, another repetition begins; in other words, one unit is added to the iteration counter (A←A+1) To achieve a neighboring solution, this process continues until repetitions of function concerning the NSS-As are completed ($A = A_{max}$). NSS-As is capable of using dissimilar measures of A_{max} in the presented algorithm.

Cooling Schedule: The basic aim of utilizing cooling schedule is to control the behavior of SAs. In general, the cooling schedule is specified by several parameters and methods, namely the initial temperature, Number of neighborhood search in each temperature, the law of decrease of the temperature and the termination condition of algorithm. As the procedure of SA proceeds, the temperature is gradually lowered under the law of decrease of temperature. In general, there are three laws for lowering the temperature in the literature of SA: linear, exponential and hyperbolic. In this study, we use a linear function to reduce the temperature; the function is shown in Eq. 1:

$$T_i = T_0 - i.\frac{T_0 - T_f}{N} \quad (1)$$

where, T_0 denotes initial temperature, T_f denotes final temperature, i represents a stage in the algorithm, T_i represents the temperature of stage i and N is the maximum number of iterations of algorithm.

An appropriate initial temperature should be high enough to create equal opportunity for all states of the search space to be visited and at the same time, it should not be rather too high to perform quite a lot of unnecessary searches in high temperature. The initial temperature is acquire via Eq. 2:

$$T_0 = [\overline{\Delta_f^+}] \quad (2)$$

where, Δ_f^+ is any positive differences in fitness value when a neighboring solution is achieved from the candidate solution and $\overline{\Delta_f^+}$ is the average value of these differences. Besides, we do not consider the negative or neutral differences in fitness value to compute the average value. In Eq. 2, the symbol [] denotes that the initial temperature, T_0, is set to the integer part of the average value of differences. The initial temperature of the algorithm should be determined empirically for each particular problem on hand.

Given the linear function applied to reduce the temperature, N iterations of algorithm, i.e. maximum number of iterations of algorithm, should perform to terminate it. This approach will result meeting the final temperature T_f. The values of parameters N and number of neighborhood search in each temperature, i.e. nt, for each problem, mainly depend on the limitation of time and the extent in which we want to search the solution space.

Proposed algorithm: The proposed SA algorithm steps are shown in Fig. 4. The algorithm comprises two internal and external loops. The external loop controls the termination condition; besides, the internal loop, as a local search, searches the solution space extensively. After the initialization of the algorithm is done, the initial solution s_0 is set to the input solution of local search step ($s_{in} \leftarrow s_0$) In each iteration of the local search step, the neighboring solution s' is produced from the neighborhood of current solution s by neighborhood search structure. The parameter *l*, depicted in Fig. 4, determines the sequence in which neighborhood search structures are applied. After the neighboring solution s' is produced, the difference in the fitness value (the change in value of makespan) is computed. In case the difference is negative, the algorithm moves from current solution s to the neighboring solution s'. When the difference in the fitness value is zero, we use a stochastic method to accept the solution; that is, a uniform random number, i.e., R~U(0, 1), is generated. If R is smaller than 0.5, the move will be toward neighboring solution s'. Besides, in case of the positive difference, the move will be toward neighboring solution s', only if $R<\exp(-\Delta_{s,s'}/T)$. If none of the conditions above is met, the

Initialization

Select encoding scheme, initial solution (s_0), initial temperature (T_0), number of neighborhood search in each temperature (nt), low of decrease temperature, set of neighborhood search structures and termination condition.

Main part of algorithm

Set $T \longleftarrow T_0$, $s^* \longleftarrow s_0$ and $s_{in} \longleftarrow s_0$;
for $i = 1 : N$
  Set $l \longleftarrow 1$ and $s \longleftarrow s_{in}$;
  for $j = 1 : nt$ (Local search loop)
    if $l = 1$ then generate point $s' \in N_1(s) \longleftarrow Se(s)$; (NSS-$Se$ with $k_{max} = 3$)
    elseif $l = 2$ then generate point $s' \in N_2(s) \longleftarrow As(s)$; (NSS-$As$ with $A_{max} = 1$)
    elseif $l = 3$ then generate point $s' \in N_3(s) \longleftarrow Se(s)$; (NSS-$Se$ with $k_{max} = 4$)
    elseif $l = 4$ then generate point $s' \in N_4(s) \longleftarrow As(s)$; (NSS-$As$ with $A_{max} = 2$)
    elseif $l = 5$ then generate point $s' \in N_5(s) \longleftarrow Se(s)$; (NSS-$Se$ with $k_{max} = 5$)
    elseif $l = 6$ then generate point $s' \in N_6(s) \longleftarrow As(s)$; (NSS-$As$ with $A_{max} = 3$)
    elseif $l = 7$ then generate point $s' \in N_7(s) \longleftarrow Se(s)$; (NSS-$Se$ with $k_{max} = 6$)
    endif
    if $\Delta_{s,s'} < 0$ then $s \longleftarrow s'$ and $l \longleftarrow l$;
    elseif $\Delta_{s,s'} = 0$ then generate random R uniformity in the range (0,1);
      if $R < 0.5$ then $s \longleftarrow s'$ and $l \longleftarrow l$;
      endif
    elseif $\Delta_{s,s'} > 0$ then generate random R uniformity in the range (0,1);
      if $R < \exp(-\Delta_{s,s'}/T)$ then $s \longleftarrow s'$ and $l \longleftarrow l+1$;
      endif
    else $l \longleftarrow l+1$;
    endif
    if $l = 8$ then $l \longleftarrow 1$;
    endif
  endfor
  The best solution obtained from each temperature is selected and and it place in s_i^*;
  Set $s_{in} \longleftarrow s_i^*$;
  if $f(s_i^*) < f(s^*)$ then $s^* \longleftarrow s_i^*$;
  endif
Update T : Decrease of temperature according to linear low;
endfor

Fig. 4: The proposed SA algorithm steps for FJSP

move toward neighboring solution s' will not occur. Moreover; if either the first condition or the second condition is met, the neighborhood search structure will be kept; otherwise, the next neighborhood search structure will be used $(l \leftarrow l+1)$. After all the iterations related to local search loop in a particular temperature is done, the best solution of the local search step, s_i^*, is selected and it will be determined as the input solution for the local search step in the next temperature $(s_{in} \leftarrow s_i^*)$. Furthermore, if s_i^* is better than the best solution having found in the searches so far, i.e., s*, it will replace the best solution $(s^* \leftarrow s_i^*)$.

At the end of any iterations of the algorithm, the temperature reduces according to the linear function. The algorithm will proceed until the stopping condition is met. In the cooling schedule section, it is described that this condition is equivalent to perform N iterations of the algorithm; as a result, the algorithm will meet final temperature.

In the proposed algorithm, both the problem of assignment and the problem of sequencing are tackled simultaneously and symmetrically, which corroborates the integrated approach applied in present algorithm.

COMPUTATIONAL RESULTS

The results describes the computational tests which are used to evaluate the effectiveness and efficiency of the presented SA algorithm. We implement the algorithm in MATLAB 7.1 and run on a PC with 2.0 GHz Intel Core 2 Duo and 1 GB of RAM memory. To valuate the performance of the proposed algorithm, we use 20 problems generated by Fattahi *et al*. (2007) (Fdata).

These testing problems are divided to two categories: small size problems (SFJS1:10) and medium and large size problems (MFJS1:10). Using a mathematical model, Fattahi *et al.* (2007) found the optimal values of the objective functions for the problems of small size. They also offered Upper and lower bounds for the problems of medium and large size. Besides, they proposed six algorithms based on the integrated and hierarchical approaches to solve FJSP. In integrated approach, two algorithms: ISA algorithm and ITS algorithm were presented and in hierarchical approaches four algorithms: HSA/SA, HSA/TS, HTS/TS and HTS/SA algorithms were presented. Among these six algorithms, the HTS/SA algorithm obtains better results for the Fdata problems. We also solve the Fdata problems; then compare present results with recent results obtained by these problems. Table 2 shows the SA parameters which we set for each problem. The final temperature T_f is set to 0.1.

The non-deterministic nature of the presented algorithm makes it necessary to carry out multiple runs on the same problem instance in order to obtain meaningful results. We select the best solution (best makespan) for each problem after ten runs of SA algorithm from different initial solutions.

Table 3 compares best results over ten runs of SA, to the best results of HTS/SA algorithm proposed by Fattahi *et al.* (2007) on Fdata. The first and second columns symbolize the name and size of the problem, respectively. Size of these problems are determined by indexes (n, h, m) in which index n denotes number of jobs, h denotes the maximum number of operations for all jobs and m denotes the machine number. In the third column, the lower bound values (optimal values) for the problems of small size and the lower bound and the upper bound values for the problems of medium and large size are drawn.

The fourth column refers to the best makespan resulted from ten runs of SA algorithm, C_{max} and the average makespan of ten runs of this algorithm, $AV(C_{max})$, that are shown in the fifth column. The sixth and seventh columns represent, respectively, the best makespan and

Table 2: Parameters setting for the SA algorithm

Problem	No. of initial population	Initial temperature (T_0)	No. of neighborhood search in each temperature (nt)	Maximum No. of iterations of algorithm (N)
SFJS1	20	35	30	20
SFJS2	20	32	30	20
SFJS3	20	76	50	50
SFJS4	30	84	50	50
SFJS5	30	38	50	50
SFJS6	30	75	50	50
SFJS7	30	102	50	50
SFJS8	30	62	100	50
SFJS9	50	55	200	50
SFJS10	50	132	200	100
MFJS1	100	120	200	200
MFJS2	100	112	200	300
MFJS3	100	125	200	400
MFJS4	200	136	200	500
MFJS5	200	130	200	500
MFJS6	300	145	300	1000
MFJS7	300	157	300	2000
MFJS8	500	152	300	3000
MFJS9	500	170	500	3000
MFJS10	500	150	500	3000

Table 3: Comparison between the SA algorithm and the HSA/TS algorithm

			Results of proposed SA		Results of HTS/SA		
Problem	Size (n, h, m)	(LB, UB)	C^*_{max}	AV (C_{max})	C^*_{max}	AV (C_{max})	dev (%)
SFJS1	2.2.2	66	66	66.0	66	66..0	0
SFJS2	2.2.2	107	107	107.0	107	107.0	0
SFJS3	3.2.2	221	221	221.0	221	221.0	0
SFJS4	3.2.2	355	355	355.0	355	355.0	0
SFJS5	3.2.2	119	119	119.0	119	119.0	0
SFJS6	3.3.3	320	320	320.0	320	336.0	0
SFJS7	3.3.5	397	397	397.0	397	397.0	0
SFJS8	3.3.4	253	253	253.0	256	269.6	+1.17
SFJS9	3.3.3	210	210	210.0	210	218.0	0.00
SFJS10	4.3.5	516	516	516.0	516	516.0	0.00
MFJS1	5.3.6	(396, 470)	468	468.0	469	499.0	+0.21
MFJS2	5.3.7	(396, 484)	448	454.6	468	494.2	+4.27
MFJS3	6.3.7	(396, 564)	468	472.0	538	542.0	+13.01
MFJS4	7.3.7	(496, 684)	561	567.0	618	629.4	+9.22
MFJS5	7.3.7	(414, 696)	514	526.5	625	630.4	+17.76
MFJS6	8.3.7	(469, 786)	634	649.2	730	749.8	+13.15
MFJS7	8.4.7	(619, 1433)	899	902.0	947	977.0	+5.06
MFJS8	9.4.8	(619, 1914)	897	912.4	922	973.8	+2.21
MFJS9	11.4.8	(764, 2908)	1101	1104.0	1105	1147.8	+0.36
MFJS10	12.4.8	(944, 4960)	1258	1272.0	1384	1428.2	+9.10
Average improvement							+3.776

Table 4: Mean relative errors based on Fdata problems

	Proposed SA	ISA	ITS	HTS/TS	HTS/SA	HAS/SA	HSA/TS
MRE	14.47	29.27	33.84	20.61	19.9	24.29	26.48

the average makespan obtained by ten runs of HTS/SA algorithm. Also, we made the use of a measure namely the relative deviation to compare results of our algorithm with HTS/SA algorithm. The relative deviation is defined as:

$$dev = [(C^*_{max}(comp) - C^*_{max}(SA))/C^*_{max}(comp)] \times 100\% \quad (3)$$

where C^*_{max} (SA) the best makespan is obtained by present algorithm and C^*_{max} (comp) is the best makespan of the algorithm we compare to. Measures concerning relative deviation measure are presented in eighth column.

The computational results reveal that the proposed algorithm in this study obtains the optimal values for all problems of small size, i.e. SFJS1:10. Besides, for the problems of medium and large size, i.e., MFJS1:10, the proposed algorithm, significantly, performs better than the HTS/SA algorithm. As it indicates in Table 3, the SA outperforms HTS/SA when the relative deviation measure is applied.

Another measure which is used to evaluate the solutions is the Relative Error (RE). The relative error is acquired as Eq. 4:

$$RE = [(C^*_{max} - LB)/LB] \times 100\% \quad (4)$$

where, the best makespan value, C^*_{max}, is obtained by the reported algorithm and LB is the best-known lower bound.

In Table 4, considering Fdata problems, the Mean Relative Error (MRE) of the best solution obtained by our SA and six different algorithms proposed by Fattahi *et al.* (2007) are presented. Given MRE, the results showed that our algorithm outperforms other algorithms.

In order to study the performance of the proposed SA more accurately, with respect to RE criterion, the means plot and LSD (Least Significant Difference) intervals at 95% confidence level are presented in Fig. 5 for the proposed SA and six other algorithms. It can be inferred from the Fig. 5 that the proposed SA is statistically outperform other six algorithms.

The schedule obtained from our SA algorithm for problem MFJS10 of Fdata with the makespan of 1258 is presented in the following. The starting and completion times of the operations assigned to each machine are as follows:

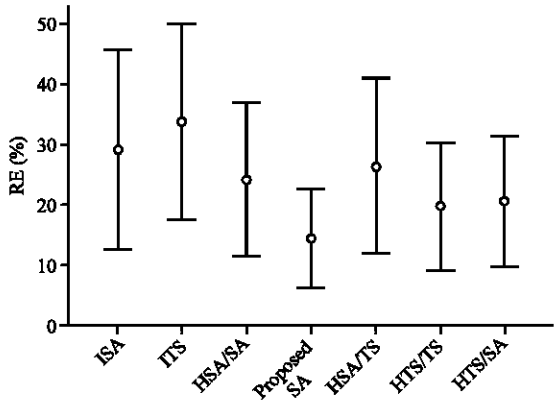

Fig. 5: Means plot and LSD intervals for the algorithms

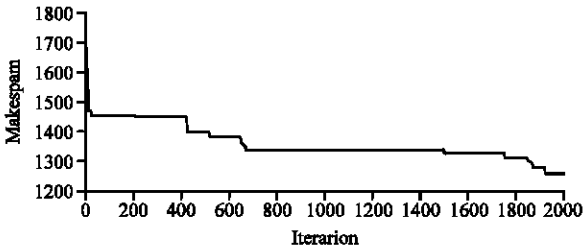

Fig. 6: Convergence of the algorithm to minimize makespan for MFJS10 problem

M_1: ($O_{5,1}$: 0-128) ($O_{3,1}$: 128-215) ($O_{2,1}$: 215-429) ($O_{1,1}$: 429-676)

M_2: ($O_{4,1}$: 0-65) ($O_{12,1}$: 65-422) ($O_{2,2}$: 429-495) ($O_{7,1}$: 495-652) ($O_{11,1}$: 652-802) ($O_{1,2}$: 802-932)

M_3: ($O_{10,1}$: 0-257) ($O_{9,1}$: 257-407) ($O_{12,2}$: 422-646) ($O_{7,2}$: 652-776) ($O_{11,2}$: 802-982)

M_4: ($O_{8,1}$: 0-245) ($O_{6,1}$: 245-399) ($O_{6,2}$: 399-549) ($O_{4,3}$: 549-694) ($O_{9,2}$: 694-914) ($O_{1,3}$: 932-1082)

M_5: ($O_{5,2}$: 128-175) ($O_{5,3}$: 175-285) ($O_{4,2}$: 285-458) ($O_{3,3}$: 458-558) ($O_{6,3}$: 558-678) ($O_{10,3}$: 693-838) ($O_{9,3}$: 914-954) ($O_{2,4}$: 788-908)

M_6: ($O_{8,2}$: 245-469) ($O_{10,2}$: 469-693) ($O_{2,3}$: 693-788) ($O_{4,4}$: 788-958) ($O_{9,4}$: 958-1108) ($O_{11,4}$: 1108-1258)

M_7: ($O_{3,2}$: 215-360) ($O_{8,3}$: 469-647) ($O_{12,3}$: 647-825) ($O_{7,3}$: 825-1003) ($O_{11,3}$: 1003-1053) ($O_{10,4}$: 1053-1203) ($O_{6,4}$: 1203-1253)

M_8: ($O_{5,4}$: 285-450) ($O_{3,4}$: 558-723) ($O_{8,4}$: 723-873) ($O_{12,4}$: 873-1103) ($O_{1,4}$: 1103-1248)

The convergence of the proposed SA algorithm to minimize makespan for the MFJS10 test problem is shown in Fig. 6.

The results obtained from the computational study have shown that the proposed algorithm is a viable and effective approach for FJSP.

CONCLUSION AND FUTURE STUDY

In this study, an effective simulated annealing algorithm for flexible job- shop scheduling problem is developed. The objective function is considered as the minimization of the makespan time. Task sequencing list is used to represent the solutions and a combination of strategies is utilized for generating the initial solution. To determine the initial temperature, a proper approach is devised. Besides, a linear method is used to reduce the temperature. Two neighborhood search structures in relation to assignment and sequencing problems are employed in process of presented algorithm. Twenty benchmark problems of FJSP are used to evaluate the performance of the presented algorithm.

The computational results reveal the competent performance of the proposed algorithm to optimize the FJSP. There are potentially unlimited opportunities to research in FJSP. For instance, a population based simulated annealing algorithm to optimize the FJSP can be developed. In addition, using VNS algorithm to search the solution space instead of conventional local search used in SA can be devised for the next studies. Finally, other performance indicators and process constraints can be considered for a future study.

REFERENCES

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by taboo search. Ann. Operat. Res., 41: 157-183.

Cerny, V., 1985. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. J. Optimiz. Theor. Appl., 45: 41-51.

Dauzere-Peres, S. and J. Paulli, 1997. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. Ann. Operat. Res., 70: 281-306.

Fattahi, P., M. Saidi Mehrabad and F. Jolai, 2007. Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J. Intel. Manuf., 18: 331-342.

Gao, J., L. Sun and M. Gen, 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput. Operat. Res., 35: 2892-2907.

Gen, M. and R. Cheng, 1997. Genetic Algorithms and Engineering Design. 1 Edn., John Wiley and Sons, New York, ISBN: 0-471-12741-8.

Kacem, I., S. Hammadi and P. Borne, 2002. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. IEEE T. Syst. Man Cybernetics C., 32: 1-13.

Kirkpatrick, S., Jr. C.D. Gelatt and M.P. Vecchi, 1983. Optimization by simulated annealing. Science, 220: 671-680.

Mastrolilli, M. and L.M. Gambardella, 2000. Effective neighborhood functions for the flexible job shop problem. J. Sched., 3: 3-20.

Pezzella, F., G. Morganti and G. Ciaschetti, 2008. A genetic algorithm for the flexible job-shop scheduling problem. Comput. Operat. Res., 35: 3202-3212.

Shi-Jin, W., Z. Bing-Hai and X. Li-Feng, 2008. A filtered-beam-search-based heuristic algorithm for flexible job-shop scheduling problem. Int. J. Prod. Res., 46: 3027-3058.

Tay, J.C. and N.B. Ho, 2008. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput. Ind. Eng., 54: 453-473.

Xia, W. and Z. Wu, 2005. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problem. Comput. Ind. Eng., 48: 409-425.