

## A simulated annealing algorithm for JIT single machine scheduling with preemption and machine idle time

Vahid Majazi Dalfard

Islamic Azad University, Qazvin Branch, Faculty of Industrial and Mechanical Engineering, Qazvin, Iran

Vmd\_60@yahoo.co.nz

### Abstract

This article is related to concept of preemption in just-in-time single machine shop with allowable machine idle time. According to the operation research model of this problem, Lingo software can easily solve small problems, but it cannot solve the average and big problems. The purpose of this article is to provide a simulated annealing algorithm for solving this problem. The suggested simulated annealing algorithm gives the answers, which are quite close to the optimal solution in the tested problems. The runtime solution problem by using the suggested algorithm is much less than by using the Lingo software. The numerical examples show that the efficiency of SA algorithm is quite high.

**Keywords:** Just-in-time scheduling, single machine, preemption, machine idle time, simulated annealing algorithm

### Introduction

This paper considers a single machine scheduling problem with earliness and tardiness costs. Moreover, preemption and idle times are allowed. To determine the performance of production systems, the quality of the schedules for a single machine is considered. In addition, results and insights obtained for single machine problems frequently can be generalized for more complex scheduling environments, such as flow shops or job shops. Single machine problem is focused in this paper. Just-in-time scheduling problem (JITSP) is a well studied field of multi-criteria scheduling problems (Sourd, 2005). Because of the practical significance and relevance of the JITSP models, the scheduling community pays a considerable attention to these models. These production strategies, which have been implemented by many organizations, consider both early and tardy delivers as undesirable. Definitely, these models in which earliness is penalized are suitable for delivering perishable goods and the storage costs which cannot be ignored. An early job can cause inventory holding cost, such as opportunity cost of the money invested in inventory, storage and insurance costs and deterioration. Contrarily, a tardy job can cause customer dissatisfaction, contract penalties, loss of sale and loss of reputation (Liao & Cheng, 2007). So the criterion which involves both earliness and tardiness costs has received considerable attention for about two decades. For scheduling models with earliness and tardiness costs many papers have been issued. Baker and Scudder (1990) present a brilliant analysis for the initial work on early-tardy scheduling. Recent researches on these problems have been published by Hooegeveen (2005), Lauff and Werner (2004) and Gordon *et al.* (2002).

The non-preemptive single machine scheduling problem with earliness and tardiness costs has been regarded as an NP-hard problem which has been extensively studied in recent years (Lenstra *et al.*, 1977). Also several exact algorithms and heuristics were proposed (Davis & Kanet, 1993; Hooegeveen & Van De Velde, 1996; Sourd & Kedad-Sidhoum, 2003, 2008;

Hendel & Sourd, 2006; Bülbül *et al.*, 2007). Most results of just-in-time scheduling were presented by Jzefowska (2007). However, preemptive earliness tardiness scheduling problems seem to be disregarded because of the importance of preemptive problems in scheduling theory.

In non-preemptive problems, earliness-tardiness costs are function of the job completion time. When a job is completed after its determinate time the tardiness penalty occurs, conversely, if it is completed before that time the earliness penalty occurs. However, in a scheduling environment where preemption is allowed, the desired results may not be obtained by such functions. Definitely, in order to remove a job from a production line it should be completed as soon as possible. It is stated that the work-in process has to be minimized. If the only aspect that has effect on costs is completion time of the job then idle time within the execution of the job might not be penalized and the work-in-process will be large (Hendel *et al.*, 2009). Because of this problem we are going to compose a model in which the earliness and tardiness costs are consecutively correlated to the start and completion time of the job. The related model with non-preemption has been introduced by Hendel *et al.* (2009). For each job  $j_i$ , a due date  $d_i$  is given. This due date is equal to a target completion time. In order to penalize the preemption, the target start time which is equal to  $d_i - p_i$  is presented by Hendel *et al.* (2009).

Formally, we consider a set of jobs  $g = \{j_1, \dots, j_n\}$  which has to be scheduled on a single machine. Each job  $j_i \in g$  has a processing time  $p_i$ . We define tardiness costs as  $T_i = \max(0, C_i - d_i)$ , where  $C_i$  is the completion time of job  $j_i$ , and earliness costs as  $E_i = \max(0, -d_i - p_i - S_i)$  where  $S_i$  is the starting time of job  $j_i$ . We want to minimize  $\sum_{i=1}^n (\alpha_i E_i + \beta_i T_i)$ . If all earliness penalties are zero, there is an optimal schedule without job interruption, so that we have an instance of  $1 || \sum W_i T_i$  that is strongly NP-hard (Lenstra *et al.*, 1977). The aim of this paper is to

investigate JIT single machine scheduling problems for which preemption is allowed. However, the resulting problem is equivalent to  $1|pmtn, r_i | \sum_{i=1}^n (\alpha_i E_i + \beta_i T_i)$  that is NP-hard (Lenstra *et al.*, 1977).

For many production settings, assuming that the allowed machine idle time is also suitable. Indeed, idle time must be avoided in case the machine has limited capacity or high operating costs. This assumption is also validated when starting a new production run engaging high setup costs or times. But, in preemptive problems presence of machine idleness is very useful (Hendel *et al.*, 2009). Because, optimal solutions usually include some idle periods during which no activity is processed. Due to the complexity of these problems, some meta-heuristic algorithms were applied. Local search method has been known for obtaining relevant results for just-in-time scheduling problems (Sourd & Kedad-Sidhoum, 2008). A neighborhood search solution for the preemptive problem has been proposed (Hendel *et al.*, 2009). They considered the problem as general case for the objective of finding good sequences via neighborhood search. In this paper a simulated annealing (SA) proposed for finding good solutions in reasonable computational times.

### Problem definition

In this section, we formally define the considered problem and formulate a new non-linear mathematical model based on just-in-time scheduling problem where preemption and idle time are allowed. The proposed model deals with the minimization earliness-tardiness scheduling problem with preemption. The problem can be stated as follows. A set of  $n$  independent jobs  $\{j_1, j_2, \dots, j_n\}$  has to be scheduled on a single machine which can handle at most one job at a point of time. The machine is assumed to be continuously available and breakdown is not occurred. There is a due date  $d_i$  for each job  $j_i$  which is to be processed on a single machine. Each of the  $n$  jobs is available for processing at time zero and has a determinate processing time  $p_i$ , earliness weight  $\alpha_i$ , and tardiness weight  $\beta_i$ . We regard as a schedule where, for some job  $j_i$  there is some idle time between  $S_i$  and  $C_i$  and then we can build a schedule with a lower or equal scheduling cost such that there is no idle time between  $S_i$  and  $C_i$  (Hendel *et al.*, 2009). In this non-linear programming, we assume that all the model parameters are integer.

Notations:

*Indices:*  $i$  Index for jobs;  $k$  Index for periods;  $H$  Number of periods

*Parameters*

$n$ : Number of jobs;  $d_i$  Due date of job  $j_i$ ;  $p_i$  Processing time of job  $j_i$ ;  $\alpha_i$  Earliness penalty of job  $j_i$ ;  $\beta_i$  Tardiness penalty of job  $j_i$ ;  $M$  A large positive number

### Variables

$Z$  Weighted earliness and tardiness;  $s_i$  Start time of job

$j_i$ ;  $s'_i$ : A helper variable for calculating start time of job

$j_i$ ;  $c_i$  Completion time of job  $j_i$ ;  $E_i$  Earliness of job  $j_i$ ,

$E_i = -\max(0, d_i - p_i - s_i)$ ;  $T_i$  Tardiness of job  $j_i$ ,

$T_i = -\max(0, c_i - d_i)$

$x_{ik} = \begin{cases} 1 & \text{If job } j_i \text{ is selected for time } k \\ 0 & \text{Otherwise} \end{cases}$

$\delta_{ik} = \begin{cases} 1 & \text{if } x_{ik} = 1 \\ M & \text{if } x_{ik} = 0 \end{cases}$

### The mathematical model

The objective function and constraints can be formulated as follows:

$$\min Z = \sum_{i=1}^n (\alpha_i E_i + \beta_i T_i)$$

where:

$$\sum_{k=1}^H x_{ik} + p_i \quad i=1,2,\dots,n \quad (1)$$

$$\sum_{i=1}^n x_{ik} \leq 1 \quad k=1,2,\dots,H \quad (2)$$

$$C_i = \max_{k=1}^H (k * x_{ik}) \quad i=1,2,\dots,n \quad (3)$$

$$S'_i = \min_{k=2}^H ((k-1) * \delta_{ik}) \quad i=1,2,\dots,n \quad (4)$$

$$S''_i = (k * \delta_{ik}) - 1 \quad i=1,2,\dots,n, k=1 \quad (5)$$

$$S_i = \min(S'_i, S''_i) \quad i=1,2,\dots,n \quad (6)$$

$$T_i = \max(0, C_i - d_i) \quad i=1,2,\dots,n \quad (7)$$

$$E_i = \max(0, d_i - p_i - S_i) \quad i=1,2,\dots,n \quad (8)$$

$$\delta_{ik} = M(1 - x_{ik}) + 1 \quad i=1,2,\dots,n, k=1,2,\dots,H \quad (9)$$

The objective function illustrates the total penalized earliness and tardiness simultaneously. Constraint (1) guarantees that each job  $j_i$  is separated into  $p_i$  operations, which have unit execution times. These operations are to be assigned to  $H$  distinct time slots  $[t-1, t)$  where  $1 \leq t \leq H$  and  $H$  is the length of the schedule. Constraint (2) ensures that in each time slots

$[t-1, t)$  only one part of one job can be performed and this restriction is able to consider the machine idle time. Constraints (3) regards the completion time of each job. Constraints (4), (5), (6) and (9) are related to calculating the start time of each job. Constraint (7) explains tardiness cost for each job. Constraint (8) illustrates earliness cost for each job. Furthermore, in this problem, a job can be preempted and later continued from the point at which interruption has occurred. Thus, interruption of a job can be happened several times and hereupon a job has several start times calculated by Constraints (4), (5) and (9). Constraint (6) determines the actual start time (minimum of start times) of each job.

**Complexity of JITSP**

*Validation of solution method*

To validate our proposed model we illustrate some numerical examples. As an example, if we solve a problem with five jobs. Using Lingo considering parameters listed in Table 1, we observe that the problem is solved in 23 h and 24 min and 40 s with 435,955, 862 iterations and the optimal solution is 32, due to several variables and constraints. Thus, this section is devoted to using simulated annealing (SA). Sourd and Kedad-Sidhoum (2008) presented that local search methods are known to obtain good results for just-in-time scheduling problems. We propose a simulated annealing for the preemptive problem and for more illustration some examples are solved. By considering the experimental results, it is observed that the solutions of these two methods (exact & meta-heuristic) are converged for small size problems, i.e. the best solutions provided by both of the methods are close to each other. When we increase the size of the problem, SA gives better solutions in reasonable time which surpasses Lingo.

*Complete enumerative*

As we know, in single machine scheduling problem, if all conditions and initial assumptions of the basic model are going on, solution space only depends on the number of jobs and the solution has  $n!$  different states (Baker, 1974). Allowance of idle time and preemption in JIT environment is suitable (Hendel *et al.*, 2009). Hence, if preemption is allowed then depending on unit of job interruption, solution space will change. For example, if we consider unit of the job interruption equal to one, then the number of states of the solution changes to  $(\sum_{i=1}^n p_i)!$ .

Solution space can be increased to  $(\sum_{i=1}^n p_i + idletime)!$  states, if we also consider the machine

Table 1. The values of the parameters.

Job $j_i$	$p_i$	$d_i$	$\alpha_i$	$\beta_i$
$j_1$	1	26	1	1
$j_2$	2	26	1	1
$j_3$	9	26	1	1
$j_4$	5	26	1	1
$j_5$	9	26	1	1

idle time. To prevent extensive increase in solution space, a dominant set has been proposed. In section 6, it will be observed that the optimal solutions exist in this set. The solution space of this dominant set obtained applying the proposed length of the process

$$(\max \left\{ \sum_{i=1}^n p_i, d_{\max} \right\} + n) \cdot \text{Thus, the}$$

proposed number of idle time periods has been obtained. Number of idle time periods

$$= (\max \left\{ \sum_{i=1}^n p_i, d_{\max} \right\} + n) - \sum_{i=1}^n p_i$$

The solution space will have  $(\max \left\{ \sum_{i=1}^n p_i, d_{\max} \right\} + n)!$

states, when

The process length is considered to be

$$(\max \left\{ \sum_{i=1}^n p_i, d_{\max} \right\} + n) \text{ and}$$

job interruption is considered as one unit of time. The term  $(\max \left\{ \sum_{i=1}^n p_i, d_{\max} \right\})$  guarantees that the solution

space always stays feasible because the length of the process is always greater than or equal to sum of the processing times.

*Trade off between the best solution & computational time*

The aim of algorithms which used for solving these problems is obtaining the best possible solutions in a reasonable time. Because of special structure and high complexity, the length of the process in JITSP is a factor that plays an important role in the quality of solution and computational time. Here, the purpose is finding the most efficient length of process experimentally. In reasonable computational time, achieving global solutions in small size problems and a local optimum in large size problems depends on the length of the process. We can achieve reasonable results about the length of the process using Lingo in small size problems and SA in large size problems. By solving many examples with different sizes, maximum length of the process was obtained experimentally by a tradeoff between the quality of solution and its computational time. These problems have

been solved many times with different number of the length of the process. While solving these examples using Lingo, the length of the process starts from  $(\sum_{i=1}^n p_i)$  and increases gradually.

For example, we solved two instances with four jobs that the results show the accuracy of our idea.

Table 2. Problem parameters of two instances subject to trade off between best solution & computational time.

Instance n	Job $j_i$	$p_i$	$d_i$	$\alpha_i$	$\beta_i$	$\sum_{i=1}^n p_i$	$d_{\max}$	H
Instance 1	$j_1$	2	8	4	1	17	20	24
	$j_2$	5	10	2	3			
	$j_3$	7	10	2	3			
	$j_4$	3	20	3	2			
Instance 2	$j_1$	2	3	1	2	16	22	26
	$j_2$	8	6	3	1			
	$j_3$	5	22	2	3			
	$j_4$	1	15	1	2			

Fig. 1. Computational time-length diagram of instance 1.

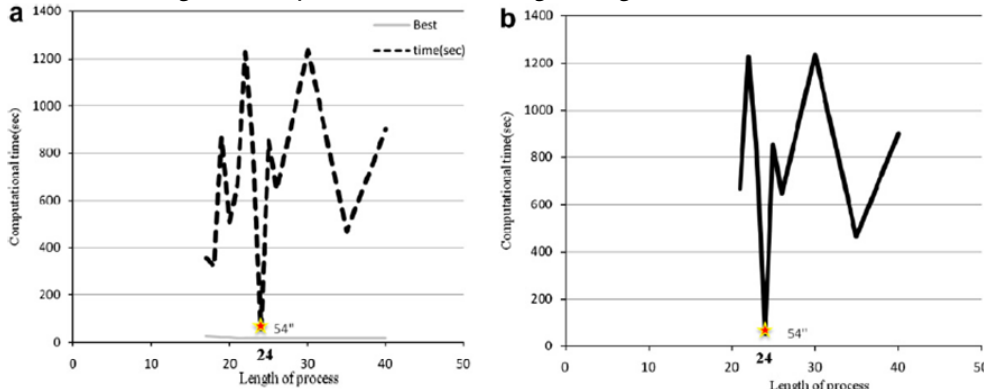
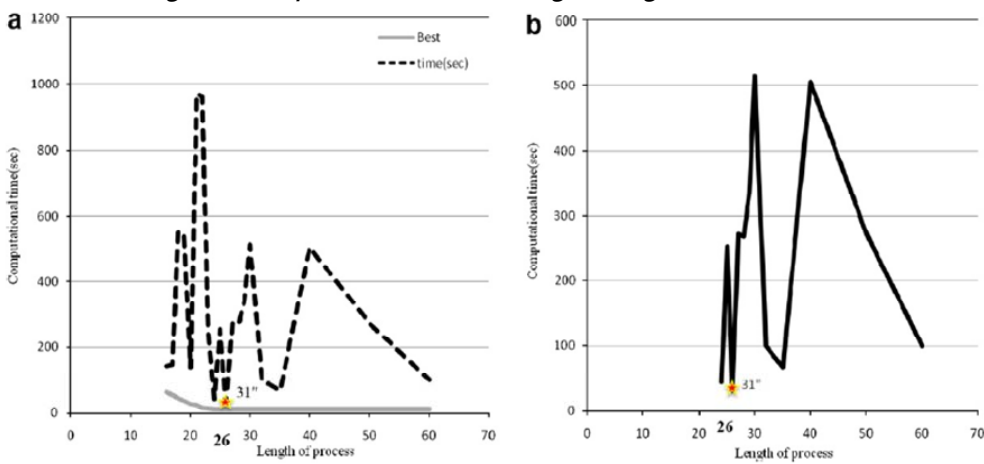


Fig. 2. Computational time-length diagram of instance 2.



Problem parameters of two instances subject to trade off are shown in Table 2. Based on the results two diagrams have been drawn that the first one is related to computational time-length of process on all runs (Fig. 1a & 2a) and the second one is related to computational time-length of process of the optimal solution (Fig. 1b & 2b). Fig. 1 and 2 shows that the proposed number of periods in Lingo or the length of the sequence in SA has the best computational time in optimal solution. The proposed length of the process in Instance 1 is 24 with the optimal solution 18 obtained in 54 s. Also the proposed length of the process in Instance 2 is 26 with the optimal solution 12 obtained in 31 s.

**A Simulated annealing algorithm solution approach**

*General simulated annealing algorithm:*

The general algorithm SA, which was introduced at the first time in 1953 by Metropolis, is as follows:  
 Select an initial Solution  $a=a^0 \in S_1$ ,  
 Select an initial temperature  $T=T_0 > 0$ ,  
 Set temperature change counter  $t=0$ ,  
 Repeat freezing process,  
 Set repetition counter  $i=0$ ,  
 Repeat-equilibrium process  
 Generate solution b: a neighbour or a,  
 Calculate  $\Delta f=f(a) -f(b)$ ,  
 If  $(\Delta f > 0)$  then  $a:=b$ ,

Else if random  $(0, 1) < \exp(-\frac{\Delta f}{t})$  then  $a:=b$ ,

$i=i+1$   
 Until  $i=N(t)$   
 $t:t+1$ ,  
 $T=T(t)$   
 Until stopping criterion true

The main steps to implement a SA are: 1-Determining the way of giving the first answer as starting the search point; 2-Determining the initial temperature; 3-Determining the rate of temperature decrease; 4-Determining the way of neighborhood creation; 5-Determining the number of reviewed neighborhoods in each temperature; 6-Scale stop.

The general procedure of SA is shown in Fig. 3.

*The proposed SA algorithm for JITSP*

For solving a single machine scheduling problem with allowable preemption and machine idle time, a simulated annealing algorithm is proposed. The main components of SA for

implementation are as follows.

*Creating the initial answer:* In this programme 1000 random answers by the length  $h$  have been created and the answer with the best objective function was selected as the start point.

*Initial temperature:* Temperature is one of the parameters that play an important role in acceptance or rejection the variation of the objective functions. Selection initial value of temperature should be such a way that in the early stages, many undesirable answers could be selected. This is because of the possibility of the development and changes the answer (Kirkpatrick, 1983).

Actually the initial temperature tells us how much we let the answer be deteriorated. It is more accurate to say how much each deterioration has probability. The accept probability of each deterioration answer is

$$e^{-\frac{\text{Index of deterioration the answer}}{\text{Temp}}}$$

In order to be our index, almost independent of the problem size we should place it equal to

$$\frac{\text{Scale of deterioration the objective function}}{\text{objective function}}$$

*Determining the rate of temperature decrease*

In order to reduce the probability of accepting the unfavorable answers, we should reduce the temperature. The way of changing this parameter is on the base of the

changes of the temperature functions  $T_k = \alpha T_{k-1}$ ,  $\alpha < 1$  in this paper  $\alpha = 0.95$  is selected (Kirkpatrick, 1983).

**Determining the way of creating neighbourhood:** In this paper we change two time units of H. if the functions get better, we can accept this relocation and if it deteriorate, we should find the size of deterioration of objective function. We can find the index of deterioration of the objective function via the following equation (Kirkpatrick, 1983):

$$\text{Index of deterioration the objective function} = \frac{\text{Scale of deterioration the objective function}}{\text{The objective function}}$$

and we produce a random number between 0 and 1 by uniform distribution.

if  $\frac{-\text{Index of deterioration the answer}}{e \cdot \text{Temp}} > \text{rand}(0,1)$ , Then we

accept deterioration of the answer, if not, another neighborhood will be chosen.

**Determining the number of neighbourhoods which are reviewed in each temperature**

In order to reach the better answers, more iteration is necessary. These iterations should be determined so that the runtime will be minimized. At the same time the answer must be favourite. In this paper the number of the iteration are constant and equal to 1000.

**Scale stop:** The runtime of the calculation is dependent to the scale stop. It is important how this scale is efficient in determining the favorite answer. The algorithm ends at that time when the answers in each temperature don't be changed by increasing the temperature. This state is called freezing state. This status is assumed as the scale stop. In this paper the final temperature is assumed 0.002. For solving these problems, a set of test problems are needed for comparing the results of different methods. For producing our instances, proper values should be created. Moreover, each of these parameters has special distribution function that will be explained in the following. To present the efficiency of the proposed approach, problems

with different sizes are considered. The small size problems are associated with 3 and 4 jobs, medium size with 10 and 15 jobs, and large size with 50 and 60 jobs. There is no specified release dates. The processing times are generated from the discrete uniform distribution [1, 9] and earliness-tardiness penalties are drawn from the discrete uniform distribution [1, 4]. The due dates of each job is drawn from the uniform distribution  $[d_{\min} - \lambda, d_{\min} + \lambda]$  where

$$d_{\min} = P(1 - TEF) \quad P = \sum_{i=1}^n P_i \quad \text{and} \quad \lambda = P(RDD/2)$$

The two parameters TEF and RDD are the tardiness/earliness function and relative range of due dates, respectively. The values of TEF do not affect the quality of the solution, but the values of RDD do. RDD gets the values of 0.2, 0.5 and 0.8. Also TEF gets the values of 0.2, 0.35 and 0.5.

Fig. 3. The general procedure of SA.

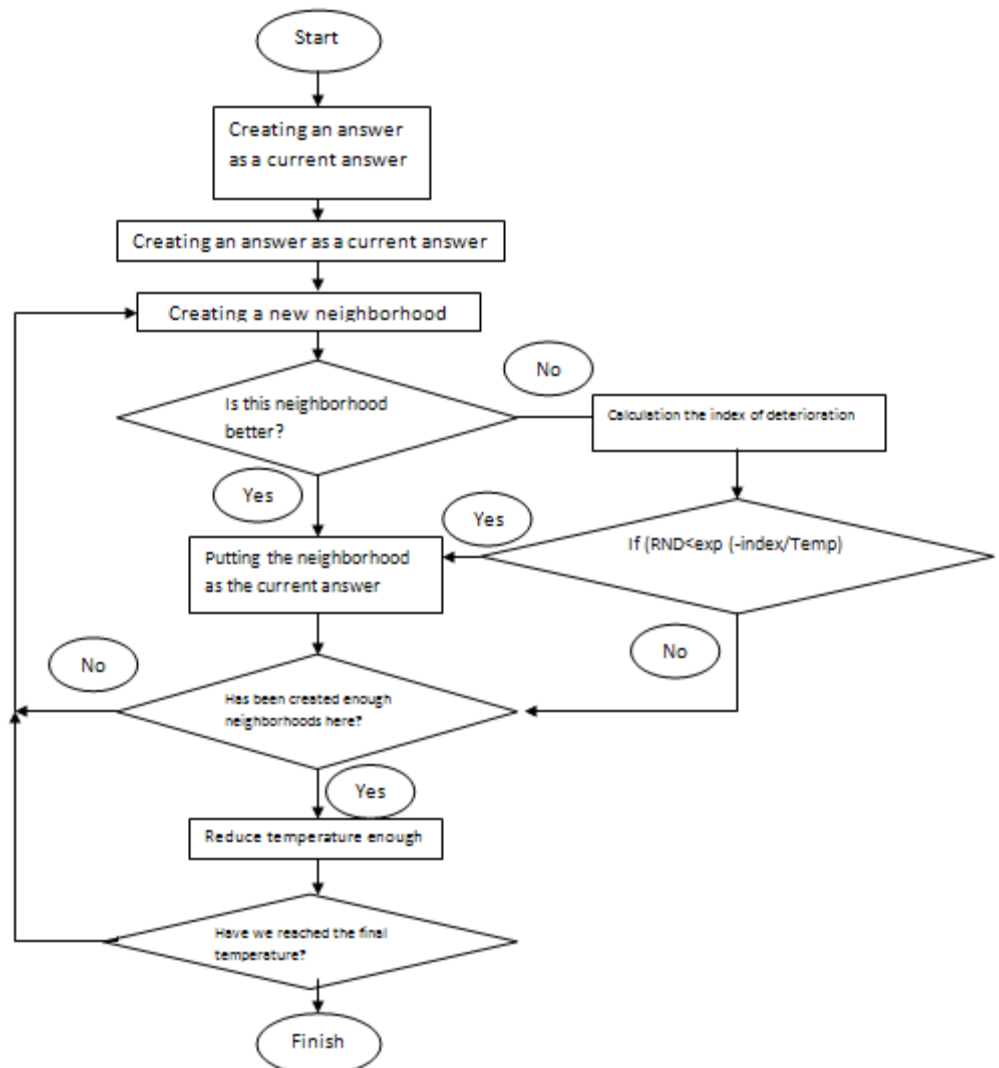




Table 3. Comparison of results of the model solved by Lingo 8 with the proposed SA (matlab 7).

Problem name	No. of job	Global solver (Lingo)			Proposed SA	
		Best solution	Optimal solution	Computational time	Best solution	Computational time
Vmd01	3	14	14	0:00:27	14	0:00:48
Vmd02	3	29	29	0:01:16	29	0:00:48
Vmd03	3	9	9	0:00:09	9	0:00:47
Vmd04	3	16	16	0:06:12	16	0:00:46
Vmd05	3	24	24	0:03:19	24	0:00:46
Vmd06	3	8	8	0:00:6	8	0:00:46
Vmd07	3	34	34	0:02:11	34	0:00:47
Vmd08	3	11	11	0:00:58	11	0:00:49
Vmd09	3	17	17	0:00:49	17	0:00:45
Vmd10	4	19	19	0:15:16	19	0:00:42
Vmd11	4	3	3	0:00:58	3	0:00:49
Vmd12	4	34	34	0:07:38	34	0:00:41
Vmd13	4	28	28	0:01:09	28	0:00:40
Vmd14	4	72	72	0:34:21	72	0:00:42
Vmd15	4	44	44	0:09:19	44	0:00:41
Vmd16	4	14	14	0:06:46	14	0:00:49
Vmd17	4	19	19	0:11:25	19	0:00:48
Vmd18	4	12	12	0:2:16	12	0:00:49
Vmd19	10	315	-	17:34:28	91	0:00:46
Vmd20	10	-	-	10:00:00	112	0:00:49
Vmd21	10	-	-	10:00:00	59	0:00:48
Vmd22	10	-	-	10:00:00	345	0:00:48
Vmd23	10	-	-	10:00:00	86	0:00:49
Vmd24	10	-	-	10:00:00	325	0:00:50
Vmd25	10	-	-	10:00:00	216	0:00:50
Vmd26	10	-	-	10:00:00	937	0:00:47
Vmd27	10	-	-	10:00:00	198	0:00:48
Vmd28	15	-	-	10:00:00	92	0:00:50
Vmd29	15	-	-	10:00:00	65	0:00:52
Vmd30	15	-	-	10:00:00	192	0:00:52
Vmd31	15	-	-	10:00:00	111	0:00:51
Vmd32	15	-	-	10:00:00	39	0:00:51
Vmd33	15	-	-	10:00:00	419	0:00:48
Vmd34	15	-	-	10:00:00	861	0:00:45
Vmd35	15	-	-	10:00:00	229	0:00:49
Vmd36	15	-	-	10:00:00	76	0:00:49
Vmd37	50	-	-	20:00:00	12295	0:00:41
Vmd38	50	-	-	20:00:00	12867	0:00:48
Vmd39	50	-	-	20:00:00	18952	0:00:46
Vmd40	50	-	-	20:00:00	14652	0:00:48
Vmd41	50	-	-	20:00:00	14212	0:00:49
Vmd42	50	-	-	20:00:00	23815	0:00:49
Vmd43	50	-	-	20:00:00	20909	0:00:47
Vmd44	50	-	-	20:00:00	19046	0:00:45
Vmd45	50	-	-	20:00:00	16820	0:00:56
Vmd46	60	-	-	22:00:00	29145	0:00:48
Vmd47	60	-	-	22:00:00	27629	0:00:53
Vmd48	60	-	-	22:00:00	23158	0:00:46
Vmd49	60	-	-	22:00:00	24861	0:00:52
Vmd50	60	-	-	22:00:00	23067	0:00:49
Vmd51	60	-	-	22:00:00	28006	0:00:45
Vmd52	60	-	-	22:00:00	29128	0:00:47
Vmd53	60	-	-	22:00:00	25219	0:00:51
Vmd54	60	-	-	22:00:00	23841	0:00:49

### Parameter setting

In this section, the results of the computational experiments are used to evaluate the performance of the proposed algorithm for just-in-time single machine scheduling problem. There are nine instances for each problem size. At this point, some information about parameter analysis would be useful. Initially, several experiments were conducted on test problems in order to determine the tendency for the values of parameters. Six test problems were used for this purpose. Test problems were made of variable number of jobs that vary from 3 to 60 jobs (Table 3). In each step, only one of the parameters was tested. Each test was repeated four times. We considered the following values for the several parameters required by the proposed SA:

Initial temperature (IT): four levels (1, 0.90, 0.80 & 0.75).  
Temperature decrease rate (TDR): four levels (0.02, 0.04, 0.05 & 0.08).  
Final temperature (FT): three levels (0.001, 0.002 & 0.005).  
Number of iteration in each temperature (NIT): one level (1000).

Test results showed that these values were suitable for the problem. Later, additional tests were conducted in order to determine the best values. After completing the tests, Taguchi analysis is applied for the different values of parameters. The best values of the computational experiments for just-in-time single machine scheduling problem with sum of the penalized earliness and tardiness objective were obtained for IT=0.9, TDR=0.05, FT = 0.002 and NIT = 1000. These values were set as the default value of the parameters.

### Conclusions and future work

Recently, the single machine scheduling problem has been becoming attentively studied field, as using in lots of industrial areas. Many researchers have been done and several solution approaches have been proposed in literature. We considered the expansion of the classical single machine earliness-tardiness scheduling problem where preemption is allowed and idle time is also considered. For this problem we presented the new mathematical model. The proposed model finds the sequence configuration with the aim of minimizing the scheduling costs. An efficient algorithm based on SA was planned to solve the mathematical model. In order to verify the performance of the proposed SA, 54 test problems were solved. Computational results showed the superiority of the proposed methodology in the sequencing of jobs as compared with exact methods. In addition, the usability of the algorithm in having appropriate computational time has been proved. The proposed SA can be applied for multiple objectives case considering other criteria like maximum tardiness, maximum earliness, mean tardiness and mean earliness.

### References

1. Baker KR (1974) Introduction to sequencing and scheduling. John Wiley & Sons, Inc., Duke University.
2. Baker KR and Scudder G (1990) Sequencing with earliness and tardiness penalties: A review. *Opera. Res.* 38, 22-36.
3. Bülbül K, Kaminsky P and Yano C (2007) Preemption in single machine earliness/tardiness scheduling. *J. Scheduling.* 10, 271-292.
4. Davis JS and Kanet JJ (1993) Single-machine scheduling with early and tardy completion costs. *Naval Res. Logistics.* 40, 85-101.
5. Gordon V, Proth JM and Chu C (2002) A survey of the state of the art of common due date assignment and scheduling research. *Euro. J. Opera. Res.* 139, 1-25.
6. Hendel F, Runge N and Sourd F (2009) The one-machine just-in-time scheduling problem with preemption. *Discrete Optimization.* 6, 10-22.
7. Hendel Y and Sourd F (2006) Efficient neighbourhood search for the one-machine earliness-tardiness scheduling problem. *Euro. J. Opera. Res.* 173, 108-119.
8. Hoogeveen JA (2005) Multicriteria scheduling. *Euro. J. Opera. Res.* 167, 592-623.
9. Hoogeveen JA and Van De Velde SL (1996) A branch-and-bound algorithm for single machine earliness-tardiness scheduling with idle time. *INFORMS J. Computing.* 8, 402-412.
10. Józefowska J (2007) Just-in-time scheduling. In: Models and algorithms for computer and manufacturing systems. Springer (ISBN: 978-387-71717-3).
11. Kirkpatrick S, Gelatt CD and Vecchi MP (1983) Optimization by simulated annealing. *Sci. New Series* 220(4598), 671-680.
12. Lauff V and Werner F (2004) Scheduling with common due date, ear lines and tardiness penalties for multimachine problems: A survey. *Math. Computer Model.* 40, 637-655.
13. Lenstra JK, Rinnooy Kan AH and Brucker GP (1977) Complexity of machine scheduling problems. *Annl. Discrete Maths.* 1, 343-362.
14. Liao CJ and Cheng CC (2007) A variable neighborhood search for minimizing single machine weighted earliness and tardiness with common due date. *Computers Industrial Engg.* 52, 404-413.
15. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH and Teller E (1953). Equations of state calculations by fast computing machines. *J. Chem. Phys.* 21(6), 1087-1092.
16. Sourd F (2005) Punctuality and idleness in just-in-time scheduling. *Euro. J. Opera. Res.* 167, 739-751.
17. Sourd F and Kedad-Sidhoum S (2003) The one machine problem with earliness and tardiness penalties. *J. Scheduling.* 6, 533-549.
18. Sourd F and Kedad-Sidhoum S (2008) A faster branch-and-bound algorithm for the earliness-tardiness scheduling problem. *J. Scheduling.* 11, 49-58.