INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 24 (2017) 1119–1137
DOI: 10.1111/itor.12195

Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem

Marco Antonio Cruz-Chávez[a], Martín G. Martínez-Rangel[b] and Martín H. Cruz-Rosales[c]

[a] *CIICAP, Autonomous University of Morelos State, Cuernavaca, Morelos, México*
[b] *FCAeI, Autonomous University of Morelos State, Cuernavaca, Morelos, México*
[c] *FC, Autonomous University of Morelos State, Cuernavaca, Morelos, México*
E-mail: mcruz@uaem.mx [Cruz-Chávez]; mmtzr@uaem.mx [Martínez-Rangel]; mcr@uaem.mx [Cruz-Rosales]

Received 27 November 2014; received in revised form 8 May 2015; accepted 12 June 2015

Abstract

This paper presents a simulated annealing algorithm accelerated by a partial scheduling mechanism and a cooling schedule mechanism that is a function of the standard deviation. This facilitates a rapid approach to good solutions in the flexible job shop scheduling problem (FJSSP). The results demonstrate that for benchmark instances of several sizes, simulated annealing that implements the proposed mechanism converges more quickly to good solutions than simulated annealing that does not implement the proposed mechanism.

Keywords: partial scheduling; controlled simulated annealing; standard deviation; complexity

1. Introduction

In everyday life, various problems frequently occur in the area of combinatorial optimization (Dash and Kajiji, 2014; Niroomand and Vizvar, 2015). The most prominent scheduling problems are considered NP-hard and complex, and require many computational resources (Alba et al., 2013). The flexible job shop scheduling problem (FJSSP) is one of the most well-known scheduling problems, and is also one of the most difficult NP-complete problems to solve (Kacem et al., 2002a). FJSSP can be grouped according to the characteristics of the problem. One group processes the problems with total flexibility, in which there is availability of all machines for each operation. One machine, from all machines involved in the process, is selected to perform each operation. Another group processes the problems with partial flexibility. In this group, not all machines are available for all operations. This latter type of problems is more difficult to handle in the FJSSP (Kacem et al., 2002a; Ho and Tay, 2004; Ida and Oka, 2011; Nouiri et al., 2013; Zhang and Manier, 2013; Bozejko et al., 2014). Problems in the partial flexibility group have application in the manufacturing industry. In addition, the literature shows that they can be successfully used for various scheduling problems (Shaw, 1988; Gu et al., 1997; Wang et al., 2003; Babayan and He, 2004). These problems serve as a reference for other problem resolution techniques in the field of resource assignment, for example, the vehicle routing problem and the resource assignment problem in classrooms (Liu et al., 2010; Wu et al., 2013; Rahimi et al., 2015; Zheng et al., 2015). Traditional approaches for resolution of the FJSSP are as varied as the different formulations of the problem. They include fast, simple heuristics, taboo search (Liouane et al., 2007), evolutionary approaches (Gu et al., 2006; Zhang et al., 2007; Ma et al., 2014a; Ma et al., 2014b; Yuan and Xu, 2015; Zheng et al., 2015), Monte-Carlo Tree Search (Wu et al., 2013), simulated annealing (SA) (Shivasankaran et al., 2014), and modern hybrid metaheuristics that consolidate the advantages of different approaches (Zhou et al., 2014). The FJSSP is an extension of classic JSSP, and incorporates all the difficulties and complexities of this problem (Mastrolilli and Gambardella, 2000; Kacem et al., 2002a, 2002b; Ho and Tay, 2004; Chen and Chen, 2008; Amiri et al., 2010; Mati et al., 2011; Knopp et al., 2014). The difference between these two problems is that JSSP limits the performance of each operation to a single machine, while FJSSP allows multiple machines to perform each operation (Mastrolilli and Gambardella, 2000; Ho and Tay, 2004).

According to the complexity theory (Papadimitriou and Steigliths, 1998), there are not exact methods that can solve an FJSSP in polynomial time. For this reason, metaheuristics are used, which are bound in polynomial time, to search for the global optimum. These metaheuristics usually give good solutions very close to the global optimum (Applegate and Cook, 1991). The FJSSP model is used in manufacturing environments that are related to control and/or production planning. If better solutions were found to this model, it would enable more efficient use of limited resources, such as manufacturing machinery, because more optimal scheduling of resources would be possible.

Many of the metaheuristics used in FJSSP are characterized using an iterated local search, requiring the use of a neighborhood function. For this reason, the development of more efficient and effective mechanisms to accelerate iterated local search is important for improving the efficiency of these metaheuristics. Many metaheuristics that use iterated local search have been proposed to find good solutions to the FJSSP. A well-known example is the SA algorithm (Kirkpatrick et al., 1983; Cerney, 1985). In literature, various approaches that use heuristics with local search have been proposed, which enable FJSSP to find good solutions. In Mastrolilli and Gambardella (2000), some neighborhood functions that can generate very efficient solutions are proposed, which can be used by iterated local search algorithms. One of these approaches is presented in Hansmann and Hoeck (1997), in which the local search is based on the Laarhoven neighborhood (Van Laarhoven et al., 1992). It accepts only solutions below the lower bound in its local search in order to escape local optima, thereby increasing the probability of finding the global optimum. In Amiri et al. (2010), a variable neighborhood search is used to solve the FJSSP. Linear coding, known as sequenced task list, is used for representing the solutions in this search. To generate the solution space with this method, two neighborhood structures related to the sequencing problem and three neighborhood structures related to the assignment problem are used. The two sets of structures are combined (both for allocation and for sequencing) to select a solution candidate. In Mati et al. (2011), the resource assignment problem is presented as a complex dynamic programming problem. They consider some resources as flexible and take into account block restrictions for genetic algorithms.

All these approaches of iterated local search require the evaluation of the solution quality at every step of the heuristic algorithm. To achieve this, the most commonly used scheduling algorithm is the one presented in Zalzala and Fleming (1997), Kacem et al. (2002a), Liouane et al. (2007), Zhang et al. (2007), Gao et al. (2014). In this algorithm, scheduling is performed as a function of time for each of the operations involved in the process and part of the problem instance. This scheduling is carried out in order to obtain the end time for the last operation that runs in the system; this time is the makespan.

This paper presents an SA algorithm that applies partial scheduling to accelerate the neighborhood search in FJSSP. The partial scheduling algorithm reported good results for JSSP in Cruz-Chávez et al. (2007), with a 50% faster average to obtain the makespan evaluation as compared to the classical scheduling algorithm (Nakano and Yamada, 1991; Yamada and Nakano, 1992; Zalzala and Fleming, 1997; Kacem et al., 2002a; Liouane et al., 2007; Zhang et al., 2007). The partial scheduling proposal in this paper for FJSSP has two major variants to the proposal made in Cruz-Chávez et al. (2007). It has a selective mechanism for choosing a particular machine or a set of machines that can perform the same operation. It also has a control mechanism that allows for a balance of loads on the machines used during the process of assigning a sequence based on the machines. The proposed mechanism is applied to SA for FJSSP. A cooling sequence is also applied, which uses standard deviation to improve the algorithm convergence (Martínez-Rangel et al., 2007).

This work consists of the following sections. Section 1 is the introduction. Section 2 describes the FJSSP and the model that it represents. Section 3 describes the mechanism of partial scheduling in FJSSP. Section 4 describes the SA algorithm that is tuned using standard deviation. To evaluate solutions in the neighborhood search, the partial scheduling mechanism is applied to FJSSP. An analysis of the complexity of the SA algorithm for the FJSSP is included. Section 5 presents the computational results. Finally, Section 6 explains the conclusions.

2. The FJSSP

This paper addresses the partial FJSSP (Brandimarte, 1993; Mastrolilli and Gambardella, 2000; Kacem et al., 2002a, 2002b; Ho and Tay, 2004; Liouane et al., 2007), in which each machine can process only some of the operations involved in the process. These problems have a great similarity to the problems that arise in industry, due to the existence of multiple machines that may exist in a job shop, not all of which can process all the jobs. Table 1 presents a partial FJSSP; this figure shows that a machine can only run some operations of some jobs. For example, the set of operations that M_2 can run is $M_2 = \{O_{2,1}, O_{1,2}, O_{3,2}, O_{1,3}, O_{2,3}, O_{3,3}\}$. It is also important to note that the processing time for an operation may be different on each machine. For example, operation one of job two, $O_{1,2}$, when run on the machine M_1, has a processing time of $t(O_{1,2}, M_1) = 4$. If the operation is executed on machine M_2, then $t(O_{1,2}, M_2) = 6$ and if the operation is executed on machine M_3, then $t(O_{1,2}, M_3) = 5$.

The FJSSP generally consists of a set of N jobs, a set of M machines, and a set O of operations, where each job consists of a subset of O in sequence. Each operation i that belongs to a job j has duration$\mu(O_{i,j})$. In a schedule of jobs, a start time for each operation $st(O_i)$ is defined such that:

- A machine cannot process more than one operation at a time.

Table 1
Instance of Partial-FJSSP

		M_1	M_2	M_3
J_1	$O_{1,1}$	6	.	5
	$O_{2,1}$	2	4	.
	$O_{3,1}$	6	.	.
J_2	$O_{1,2}$	4	6	5
	$O_{2,2}$	6	.	6
	$O_{3,2}$	.	5	3
J_3	$O_{1,3}$	2	3	.
	$O_{2,3}$	4	2	6
	$O_{3,3}$	5	4	.

- There is an order of precedence in the operations of a job.
- An operation that has already started cannot be interrupted.

The time in which all operations are executed and completed in FJSSP is known as makespan. In this work, the objective for the FJSSP is to find a schedule that minimizes the makespan.

2.1. The disjunctive graph model

The model presented below of FJSSP is an extension of the JSSP model presented in Roy and Sussman (1964). The general problem FJSSP is defined by the graph $G = (V, A, E, FL, P, O, \mu)$, where

$$V = O \cup \{I, F\}$$

$$A = \left\{\left\{\left\{I, O_{1,j}\right\}, \left\{O_{i,j}, O_{i+1,j}\right\}, \left\{O_{m,j}, F\right\}\right\} \mid \forall i, j : O_{i,j} \in O \wedge \left(O_{i,j} \prec O_{i+1,j}\right)\right\}$$

$$E = \left\{\left\{O_{i,j}, O_{i',j'}\right\} \mid \forall i, j, i, j, j \neq j , k : O_{i.j}, O_{i',j'} \in O \wedge \left(M_k(O_{i,j}) \prec M_k(O_{i',j'}) \vee M_k(O_{i',j'}) \prec M_k(O_{i,j})\right)\right\}$$

$$FL = \left\{\left\{O_{i,j}\right\} \mid \forall i, j : \left\{O_{i,j}\right\} \subseteq O \wedge \left\{M_k(O_{i,j})\right\} \subseteq M\right\}$$

$$P = \{\{p_{i,j,k} + st_{i,j,k}\} | \forall i, j, k : O_{i,j} \in O \wedge M_k(O_{i,j}) \in M \wedge (p_{i,j,k} + st_{i,j,k}) > 0\}$$

$$\mu : OM \rightarrow IN.$$

Vertices in set V represent operations. There are two vertices (fictitious operations) that have no processing time, these are the initial operation I and final operation F.

Each conjunctive arc in the set of arcs A unites a pair of operations belonging to the same job. The first operation of each job is directly connected to the operation I, $(I, O_{1,j})A$, while the last operation m of each of the jobs is connected with the operation F, $(O_{m,j}, F)A$. The precedence constraint between a pair of operations of the same job j is represented by a conjunctive arc $[O_{i,j}, O_{i+1,j}]A$. Here $A = \{\{I, O_{1,j}\}, \{O_{i,j}, O_{i+1,j}\}, \{O_{m,j}, F\}\}$.

Each disjunctive arc that is in the set of edges E, connects a pair of operations that can be performed on the same machine or on different machines. In the set E, resource capacity constraints are represented for disjunctive arcs where $M_k(O_{i,j})$ precedes $M_k(O_{i',j'})$ or $M_k(O_{i',j'})$ precedes $M_k(O_{i,j})$, between pairs of operations $(O_{i,j}, O_{i',j'})$ O. The possible direction of each edge depends on which operation of the pair $(O_{i,j}, O_{i',j'})$ is first performed on machine M_k.

The set of operations FL is the set that corresponds to the property that flexible systems have; each operation $O_{i,j}$ of the subset of operations $\{O_{i,j}\}$ and set O can choose more than one machine from the subset of machines $\{M_k(O_{i,j})\}$ of set M to be processed. When machine M_k is selected, it is the only one that executes the operation $O_{i,j}$.

The set P is the set of processing times p, plus the start time st of each operation i of job j executed on machine M_k.

The function μ indicates that all operations can be executed on more than one machine $\{M_k(O_{i',j'})\}$. Collectively with the set $\{O_{i,j}\} \subset O$, where $O_{i,j}$ can be executed on only one machine, the processing time p_{ijk} is defined as the set of positive integers IN. There is set $OM = \{\{Mk(O_{i',j'})\}, \{O_{i,j}\}\}$, where $i' \neq i$ and $j' \neq j$. The processing time $\mu(O_{i,j})$ is for each operation of OM and for the fictitious operations $\mu(I) = \mu(F) = 0$.

3. Partial scheduling mechanism

The partial scheduling mechanism requires an initial solution S_o for FJSSP. This method is used in Cruz-Chávez et al. (2007) for JSSP; in this paper it is modified for use in FJSSP. The mechanism of partial FJSSP scheduling uses a neighborhood structure of adjacent operation pairs, which randomly selects a pair of operations $O_{i',j'} \prec O_{i,j}$ of S_o to be executed on the machine M_a, between the operation $O_{i,j}$ and operation $O_{i',j'}$ that immediately precedes it. There should be no slack time between them. It randomly selects machine M_b, then operation $O_{i,j}$ is allocated to its earliest start time before the operations executed in M_b. This must be done without violating precedence constraints. For example, if the operations in M_b belong to the same job as $O_{i,j}$, the precedence constraints must be respected. This occurs because in the FJSSP, a machine is allowed to execute more than one operation of the same job. Operations in solution S_o with a completion time of less than or equal to the start time of $O_{i,j}$ keep the same scheduling. Therefore, the partial scheduling starts with the operation that was permuted. The procedure is explained in detail below with an example of an instance of FJSSP presented in Table 1. An instance of three jobs and three machines is presented in Table 1. The machines can process each of the operations in their respective times. In the case of the operation $O_{3,1}$, there is only one machine that can process it, $t(O_{3,1}, M_3) = 6$. Otherwise the operation corresponds to $O_{2,3}$, which has three different machines that can process it, each with its respective processing time.

Table 2 presents data from a first solution S_o for the instance of FJSSP presented in Table 1. For each of the operations, a consecutive number for each is established, from $O_1 = O_{1,1}$ to $O_9 = O_{3,3}$. This information is reflected in Table 2, column 2. In column 1, a turn T is assigned to each of the operations as they are executed in the system. If two or more operations have the same start time (*Ini*), the turn is assigned randomly. Column 3 (*Job*) indicates the job required to execute the operation O. Column 4 (M_k) shows the machine executing the operation O. Column 5 (*pt*) presents

Table 2
Data for the first solution (S_o)

	Process control									
T	*O*	*Job*	*Mk*	*pt*	*In*	*Ao*	*Pr*	*Ep*	*Ini*	*E*
1	4	2	1	4	0	0	0	0	0	4
2	1	1	3	5	0	0	0	0	0	5
3	7	3	1	2	1	4	0	0	4	6
4	5	2	3	6	1	1	4	4	5	11
5	8	3	2	2	0	5	7	6	6	8
6	2	1	1	2	1	7	1	5	6	8
7	9	3	2	4	0	0	8	8	8	12
8	3	1	1	6	0	0	2	8	8	14
9	6	2	3	3	0	0	5	11	11	14

the processing time of operation O on the machine. Column 6 (*In*) is a flag; if $In = 1$, then operation O can be permuted with another. If $In = 0$, it indicates that operation O cannot be permuted with any other operation. Column 7 (A_o) shows that the operation in that column is adjacent to the operation O with which it can be permuted. Column 8 (*Pr*) presents the case of an operations pair in the same job; it indicates the operation that precedes the operation in column 2. Column 9 (*Ep*) shows the completion time of the operation in *Pr*. Column 10 (*Ini*) indicates the start time of operation O in turn T. Finally, column 11 (*E*) states the end time of the operation O in turn T, where $E = Ini + pt$.

To permute a pair of operations O_i, $O_{j,}$ executed on the same machine, it is essential that the first operation O_j have a precedent operation O_i (without slack time between O_i, O_j). For example, O_i must precede O_j, otherwise O_j cannot be permuted ($In = 0$). O_j also cannot be permuted if the preceding operation O_i belongs to the same job. According to the solution S_o, presented in Table 2, O_4, which corresponds to the first operation of job J_2 (see Table 1), was assigned $T = 1$, machine M_1, and has a duration $pt = 4$ time units. If this is the operation selected as O_j, it cannot be permuted ($In = 0$). This is because O_j has no preceding operation O_i (*Pr*) because it is the first operation that is executed on M_1. Therefore, $Pr = 0$ and the completion time of the preceding operation O_i is $Ep = 0$. The start time of O_4 is $Ini = 1$ and the completion time is $E = 4$. Table 2 shows that the operations that can be permuted are the operations O_7, O_5, and O_2 (with value $In = 1$).

Figure 1 shows the Gantt chart of the solution S_o. It is noted that operations O_7, O_5, and O_2, each have an adjacent operation on the same machine that precedes them without slack time ($O_4 \prec O_7,\ O_1 \prec O_5,\ O_7 \prec O_2$) and with operations that correspond to different jobs (see Table 2, column 3). Figure 1 shows that there are three operations (O_3, O_9, and O_6) that have no slack time with the operation that precedes them in the assigned machine. These cannot be considered candidates to be exchanged because of the fact that each of these operations with their respective preceding operation belongs to the same job (so permutation would violate precedence constraints). Figure 1 shows that the makespan of the solution S_o is 14 time units, since the operation O_6 in M_3 and operation O_3 in M_1 are the last to finish (at the same time) at 14 units time. Figure 1 shows each operation, its turn, operation, and job (*T, O, J*).

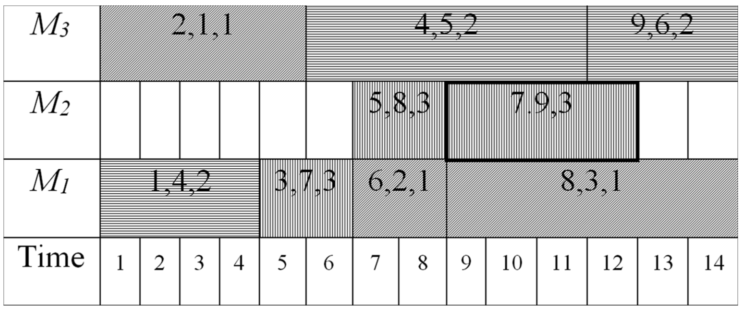


Fig. 1. Gantt chart of scheduling solution S_o.

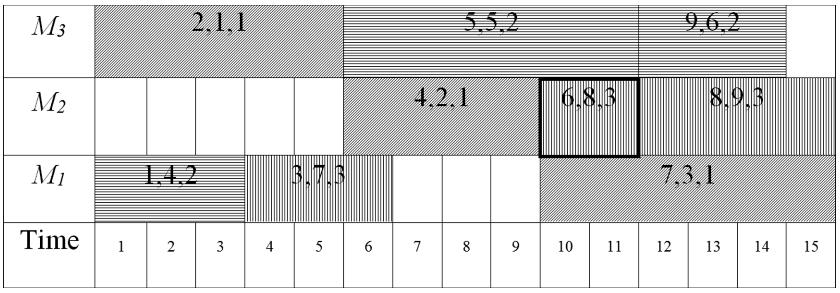


Fig. 2. New scheduling before the permutation of operations 2 and 7 in M_1. Solution $S_{1.}$

3.1. Scheduling

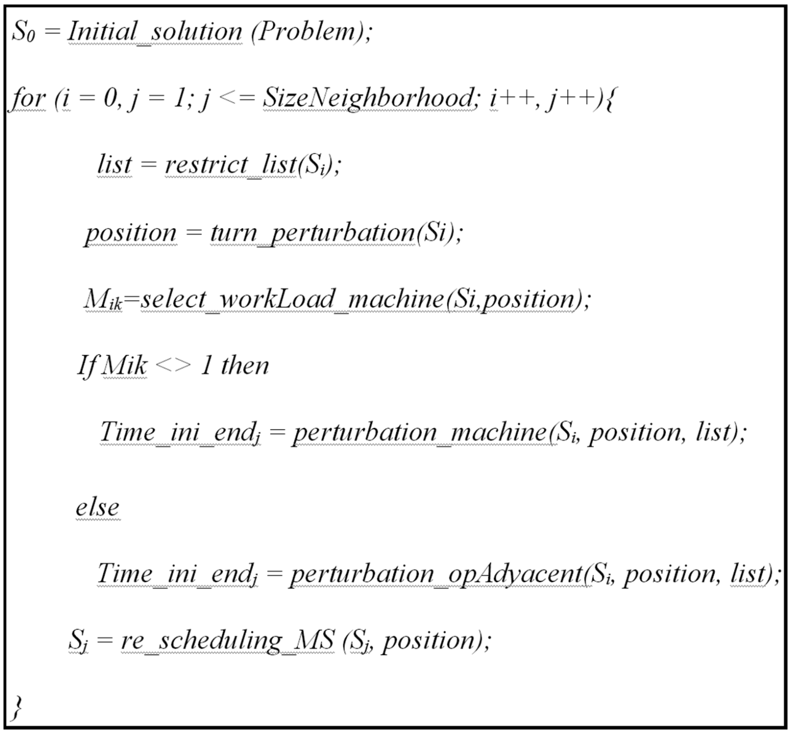
A JSSP solution, when applied to a neighborhood structure that permutes pairs of adjacent operations that have no slack time between them, generates feasible solutions (Cruz-Chávez et al., 2006). The partial scheduling method applied to JSSP in Cruz-Chávez et al. (2007) applies this neighborhood structure. Once the permutation of a pair of operations is executed, the proposed algorithm generates a partial scheduling of the problem sequence from the operations affected during the permutation. In the partial scheduling method applied to JSSP (Cruz-Chávez et al., 2007) for a permutation, it randomly selects a turn T from the list of all the operations that can be exchanged (if $In = 1$), such that the operation in turn T can be permuted by the operation that the precedes it in the same machine. For the case of FJSSP, with the same procedure, it can be seen on review of Table 2 that the operations that can be permuted are O_7, O_5, and O_2. Likewise, O_2 is randomly selected to be permuted with three possibilities. Instead of being permuted with O_7 (see Fig. 1) that precedes it in the same machine, a machine able to process O_2 is randomly chosen. This machine does not have to be the one currently processing O_2. The operation O_2 is assigned the earliest possible start time, taking care that the new assignment does not violate the precedence constraint between pairs of operations of the same job. For example, operation O_2 can be processed in M_1 and M_2 in 2 and 4 time units, respectively (see Table 1). In S_o, represented in Fig. 1, the operation O_2 is executed in M_1 with a start time of 7 time units. The partial scheduling method randomly chooses the machine M_2 and verifies that O_2 is allocated in its earliest start time possible, respecting precedence constraints. In this case, the result is presented in Fig. 2, where the earliest possible time

Table 3
Data for the new solution S_1 (result of the permutation of operations 2 and 7)

	Process control									
T	*O*	*Job*	*Mk*	*d*	*In*	*Ao*	*Pr*	*Ep*	*Ini*	*E*
1	4	2	1	4	0	0	0	0	1	4
2	1	1	3	5	0	0	0	0	1	5
3	7	3	1	2	1	4	0	0	5	6
4	2	1	2	4	0	0	1	5	6	9
5	5	2	3	6	1	1	4	4	6	11
6	8	3	2	2	1	2	7	6	10	11
7	3	1	1	6	0	0	2	9	10	15
8	9	3	2	4	0	0	8	11	12	15
9	6	2	3	3	0	0	5	11	12	14

without violating precedence constraints with operation O_1 that is executed in the same job, was 6 time units. The result is a feasible sequence. At first, operation O_2 was in turn $T = 6$ (see Table 2), but after the reassignment, it moved to turn $T = 4$ (see Table 3). In cases where an operation to be permuted can only be executed in one machine, then it is permuted with the preceding operation in the same machine (like the partial scheduling procedure for JSSP). The scheduling of turn $T = 1$ to $T = 3$ suffered no change in the turn of each operation (see Tables 2 and 3), so these operations do not require a new scheduling. Consequently, the scheduling of operations is only carried out in turns 4 through 9 using the scheduling algorithm presented in Nakano and Yamada (1991). The result of the new solution S_1 is presented in Fig. 2 and the data for the new S_1 solution are presented in Table 3. For this example, there is a makespan of 15 time units.

Figure 3 presents the partial scheduling algorithm (Partial-S) that applies the proposed procedure. The algorithm starts by choosing an FJSSP problem instance, then finds an initial solution S_o and obtains the solution's makespan (*scheduling*). From S_o, a list of candidate operations (*restrict_list*) is selected that can be permutated ($In = 1$). Candidate operations are those with an adjacent operation and without slack time on the assigned machine. The pairs of adjacent operations cannot be part of the same job. From the list of candidate operations, one is selected and its position (*position*) is obtained in order to be permutated with an operation from a different machine that can also process it (*select_workLoad_machine*). The selected machine is different from the assigned one (*perturbation_machine*). In the event that the selected operation has a single machine that can process it, then the operation is permutated with the adjacent preceding operation on the same machine (*perturbation_opAdjacent*) as performed in partial scheduling for JSSP, always ensuring that the permutation does not violate the precedence constraint (adjacent operations that belong to the same job cannot be permutated on the same machine). The start time and end time of the operation used for the permutation (*Time_ini_endj*) is obtained. Then a partial scheduling is made from the new schedule of operations that are in the selected position (*position*) by the function (*re_scheduling_MS*) to obtain the complete scheduling of the new solution S_i and its makespan. The partial scheduling of the new schedule is made by the *re_scheduling_MS* function that uses the classic scheduling algorithm presented in Nakano and Yamada (1991), Yamada and Nakano (1992), Zalzala and Fleming (1997).



```
S0 = Initial_solution (Problem);
for (i = 0, j = 1; j <= SizeNeighborhood; i++, j++){
    list = restrict_list(Si);
    position = turn_perturbation(Si);
    Mik=select_workLoad_machine(Si,position);
    If Mik <> 1 then
        Time_ini_endj = perturbation_machine(Si, position, list);
    else
        Time_ini_endj = perturbation_opAdyacent(Si, position, list);
    Sj = re_scheduling_MS (Sj, position);
}
```

Fig. 3. Local search algorithm with partial scheduling (Partial-S).

4. Accelerated SA algorithm

SA is a stochastic local search technique to approximate the minimum value of the cost function $f : S \rightarrow R$ on a finite set of solutions S. It is an iterative method that searches in the solution space with an iterated local search using a neighborhood function $N(s)$. By generating a new solution neighbor s' of s, the candidate solution s' is accepted as a new solution if $f(s') \leq f(s)$. When $f(s') > f(s)$, then the new solution is evaluated with an acceptance probability of $P(s)$ based on the Boltzmann function, which involves the control parameter T, and the difference in the quality values solution $\Delta s = f(s') - f(s)$. Initially, T has very high values and as the algorithm progresses, T decreases, which influences the acceptance probability of the solution s'.

In FJSSP, s' is a neighbor of s, which is a schedule of the problem. The cost function $f(s)$ is defined in this work by the makespan (MS). The neighborhood $N(s)$ of s is defined as the set of feasible solutions that can be generated from s in a single step, that is, a permutation of a pair of operations (O_i, O_j) assigned to a machine M_k.

SA requires initializing the control parameters. The required parameters include parameters T_o, T_f, and the coefficient of control γ, which controls the decrease rate of T. The length of the Markov chain required for an optimization problem is defined by the size of the problem neighborhood, which indicates the number of iterations performed in the Metropolis algorithm before generating a decrease in T.

The tuning of SA needs uniform distribution $P(s)$ of the Boltzmann function and the distribution $P(n)$ of n random numbers generated to determine the acceptance or rejection of a solution based on $P(s)$. The problem used for achieving equality in terms of uniform distribution of probabilities is $P(s) = e^{-(f(s')-f(s))/T}$, which has a Poisson distribution, while $P(n)$ has a normal distribution. The principles of probability accept that $P(s)$ has a greater dispersion. This is true for the proper behavior of the phenomenon of SA; the beginning of SA obtains a very high probability of acceptance when T is very high, and as T decreases and approaches zero, the probability of acceptance $P(s)$ is minimal. The cooling sequence in this proposal defines the initial value of the parameter as twice the standard deviation of a sample of randomly generated solutions. When the solutions are compared in the Boltzmann function, they must be in a range (upper bound and lower bound) determined by the average of the solutions generated, more or less two times the standard deviation. The probability principle can accept 75% of all solutions generated in the process of SA, which is equivalent to 95% in processes that follow a normal distribution (Kendall and Stuart, 1958). According to the above, the tuning of the control parameter of SA is based on the standard deviation of a set of 65,000 solutions for each of the selected test instances of FJSSP (Martínez-Rangel et al., 2007). The only differences accepted are those in the cost function $|\Delta s| = -(f(s') - f(s))$ within the range defined as two times the standard deviation. The distribution of $P(s)$ can be uniform in the SA process, the value of T_o is taken as two times the same standard deviation ($T_o = 2 \times \sigma$).

The values of each of the parameters involved in the SA process, taking into account the standard deviation, are defined as follows:

- For each of the benchmarks used in the tests, randomly generate a set of enough solutions Ω to be able to identify the standard deviation of the quality of the solutions that comprise it.
- T_o value is equal to two times the standard deviation ($2 \times \sigma$) found for the set Ω.
- The decrease in temperature is defined by $T \leftarrow \gamma \times T$, where $\gamma = 0.998$, the value obtained by an empirical sensibility analysis.
- The length of the Markov chain is twice the neighborhood size of the problem. $Ve = 2 \times (m \times (n - 1))$, according to the neighborhood structure used, which performs permutations in adjacent operation pairs without slack time (Cruz-Chávez et al., 2006), where n is the number of jobs and m is the number of machines in the problem, which determines the number of iterations in Metropolis.
- A solution s' of the problem may be evaluated for acceptance or rejection by the Boltzmann probability distribution function, whenever Δs is not more than twice the standard deviation found in set Ω.
- The partial scheduling algorithm seen in Section 3 is used to evaluate the solution s'. This speeds up the local search in SA.

The above points are used to accelerate the process of SA. Figure 4 shows the SA algorithm, which takes into account the above points, facilitating a more rapid convergence to good solutions for the FJSSP benchmarks used in this work.

```
Input Data: S, $\sigma$, $T_f$, MS=f(S) ; T = 2 *$\sigma$, SizeNeighborhood = m*(n-1)*2;
while (T >0){
   NNeighbor = 0;
   while (NNeighbor < SizeNeighborhood ){
      pick S «= N ( S ) ;
      If (f(S')<=f(S))
      {
         S=S' ;
         if (MS > f(S')) MS = f(S');
            NNeighbor++;
      } //end if
      else
      {
         $\Delta S$ =f(S')-f(S)
         if ($\Delta S$ <= 2 *$\sigma$)
         {
               n←(0<probability<1)
               if (n<exp (-$\Delta S$ /T)))
               {
                  S = S« //accept new solution
                  NNeighbor++;
               }
               else
                  S = S; //reject new solution
         }//end if
         else    S = S; //reject new solution
      } //end else
   } //end while Markov
   T = $\gamma$ * T
}// end while $T_f$ >0
```

Fig. 4. Accelerated simulated annealing algorithm.

4.1. Asymptotic complexity of the accelerated SA algorithm

According to Aarts and Van Laarhoven (1985), the computational complexity of the SA algorithm is presented in the following expression (1), where τ is the time to generate and evaluate a possible solution, L is the length of the Markov chain, and R is the solution space of the problem being evaluated.

$$O\,(\tau L \ln |R|)\,. \tag{1}$$

In the case of FJSSP, the maximum number of steps required to generate and evaluate a solution in the worst cases with a partial scheduling algorithm has the complexity $O(nm)$. L is (nm), which equals the size of the neighborhood. The size of the solution space in the worst cases of an FJSSP is bound, as is the solution space of a classical JSSP $(n!)^m$. More solution space results from the flexibility of FJSSP when, in the worst cases, operations of jobs can be executed by any machine

Table 4
Standard deviation obtained for $\Omega = 65{,}000$

Benchmarks	Job/Mach/Op	Std. dev.
mk01	10/6/55	11.9
mk02	10/6/58	10.54
mk04	15/8/90	16.58
mk07	20/5/100	36.79
mk05	15/4/106	22.43
mk03	15/8/150	47.30
mk06	10/15/150	23.87
mk08	20/10/225	26.34
mk09	20/10/240	32.49
mk10	20/15/240	28.45

$n((nm)!)$. Then R has a complexity of $R \in (O((n!)^m) + O(m((nm)!)))$. Therefore the asymptotic complexity of the SA algorithm for FJSSP involves making substitutions forτ, L, and R in (1). This is presented in (2) for the JSSP portion, and in (3) for the flexible portion.

$$O\left((nm)^2 \ln\left((n!)^m\right)\right) \tag{2}$$

$$O\left((nm)^2 \ln\left(m\left((nm)!\right)\right)\right). \tag{3}$$

In (2), if $\ln(n!)$ is pushed toward the limit where n approaches infinity, then the limit is infinity, $\ln(n!) \to \alpha$ and $\ln(n) \to \alpha$, $\ln(n!) \approx \ln(n)$, and yields expression (4).

$$O\left(n^2 m^3 \ln(n)\right). \tag{4}$$

In (3), $\ln(m(nm)!)$ is pushed toward the limit where nm approaches infinity, then the limit is $\ln(nm^2)$ and yields expression (5).

$$O\left((nm)^2 \ln\left(nm^2\right)\right). \tag{5}$$

The computational complexity of the accelerated SA algorithm for FJSSP is presented in (6).

$$O\left((nm)^2 \left(m \ln(n) + \ln\left(nm^2\right)\right)\right). \tag{6}$$

5. Computational results

The proposed algorithm was implemented in ANSI C using the Visual C on a PC with 2 GHz and 1 GB of RAM, running Windows XP. To test the efficiency of the proposed mechanism, a set of benchmarks of partial FJSSP were used, which are the most difficult within the set of FJSSP problems. The group of test problems was taken from those proposed by Brandimarte (1993).

For tuning the parameter T_o, of SA, a sample space of feasible solutions for each problem is generated (see Table 5) using the local search algorithm with partial scheduling (see Fig. 3) and using the neighborhood structure proposed in Cruz-Chávez et al. (2006). Table 4 shows the standard deviation values that were generated for test instances using a sample space of feasible solutions. The set Ω consists of 65,000 solutions. Thirty sets Ω were generated for each of the problems. Table 4

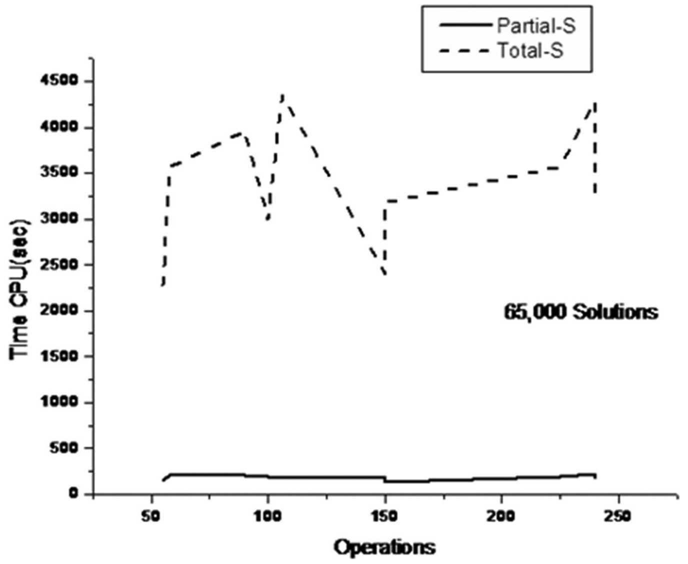


Fig. 5. Efficiency of S-Partial versus S-Total.

shows the average deviation of 30 sets Ω for each problem. According to Section 4, the initial value for the control parameter of SA is twice the standard deviation.

To compare the efficiency and efficacy of the proposed scheduling mechanisms, two strategies were used to generate schedules for each of the test instances. The first strategy is called S-Total; it uses the classic scheduling algorithm proposed in Nakano and Yamada (1991), Yamada and Nakano (1992), and Zalzala and Fleming (1997). This procedure requires obtaining the total scheduling of all operations in an FJSSP to evaluate the quality of the solution. The strategy S-Total is used in many of the algorithms presented in the literature for FJSSP. The second strategy, called S-Partial (proposed mechanism), obtained the scheduling for some of the operations involved in the instance of the problem to evaluate the quality of the solution. Figure 5 shows the average time required for S-Partial to generate 65,000 solutions in 30 executions of each problem presented in Table 5. It also shows the average time required for S-Total to generate 65,000 solutions for all of test instances used. In Fig. 5, it can be observed that S-Partial has a better efficiency in comparison to S-Total for each of the test instances. S-Partial's largest average time does not exceed 300 seconds, while S-Total's shortest average time is 2500 seconds. It is demonstrated that S-Partial works with greater efficiency and requires an average of one-tenth of the time required by S-Total to evaluate a set Ω of solutions.

Figure 6 shows the degree of dispersion of the solutions for the evaluation of the Ω of the mk07 instance. This is one of the most intractable of all the used test instances. If the average of the resulting solutions is added to about two times the standard deviation, the probabilistic principle would accept 75% of all solutions generated in the process of SA; the process follows a Poisson distribution. In processes with a normal distribution, two times the standard deviation leads to an acceptance of 95% of the solutions evaluated (Kendall and Stuart, 1958). In SA, when solutions are bad, these solutions can be evaluated with the Boltzmann acceptance criteria, only

Table 5
Results for 10 test instances of FJSSP using controlled and noncontrolled SA-Partial

			SA-Partial controlled			SA-Partial noncontrolled		
Problem	Job/Mach/Op	Mastrolilli and Gambardella (2000) taboo search	MS	RE	Av(t) (seconds)	MS	RE	Av(t) (seconds)
Mk01	10/6/55	40	40	0	3200	46	15.0	8705
Mk02	10/6/58	26	28	3.44	4600	31	6.89	9260
Mk03	15/8/90	204	216	5.88	6786	217	6.37	14,371
Mk04	20/5/100	66	60	0	5467	76	15.15	15,678
Mk05	15/4/106	173	168	0	6781	177	2.31	9786
Mk06	15/8/150	58	59	0	3783	64	10.34	8765
Mk07	10/15/150	144	147	2.08	7526	159	10.41	13,940
Mk08	20/10/225	523	524	0.10	8795	534	2.01	17,867
Mk09	20/10/240	307	307	0	4563	317	3.2	12,338
Mk10	20/15/240	198	197	0.50	7865	205	3.53	15,896

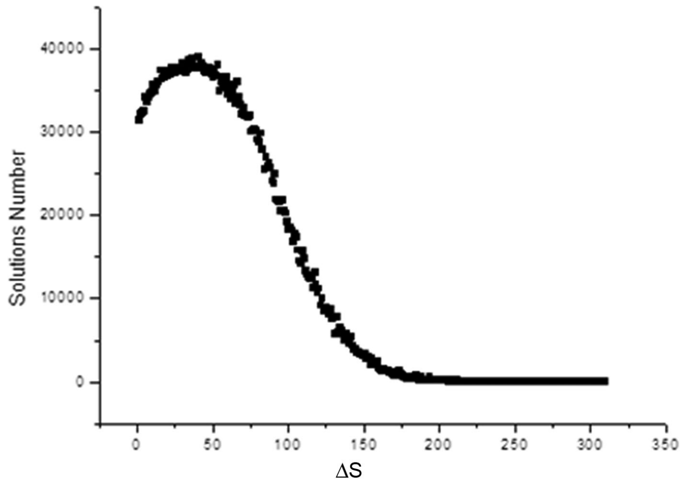


Fig. 6. Dispersion of the generated neighborhood for the Mk07 problem (20 × 5).

when $\Delta S \leq 2 \times \sigma$. Figure 6 shows that the maximum value of ΔS is about 312, so the solutions to be evaluated with the Boltzmann criteria must satisfy the condition $\Delta S \leq 234$.

If SA is controlled, 75% of all solutions within the search space of feasible solutions are delimited, which must fulfill the condition that $\Delta S \leq 2 \times \sigma$ (see Fig. 4). In addition, if the initial temperature is tuned so $T_o = 2 \times \sigma$, this accelerates the convergence toward a better solution. The partially controlled SA assists as well, where the partial term refers to the partial scheduling procedure (see Section 3). The result of an SA-Partial controlled is shown in Fig. 7, where the SA-Partial controlled process converges to a better makespan value of 41 in an average of 30 executions of SA for the Mk01 instance, while the SA-Partial noncontrolled process for the same instance at the same time achieves a makespan of 52 after an average of 30 executions. In real terms, the SA-Partial controlled process is 26% more effective than the SA-Partial noncontrolled process; the same phenomenon was observed in all other instances where the proposed algorithm was tested.

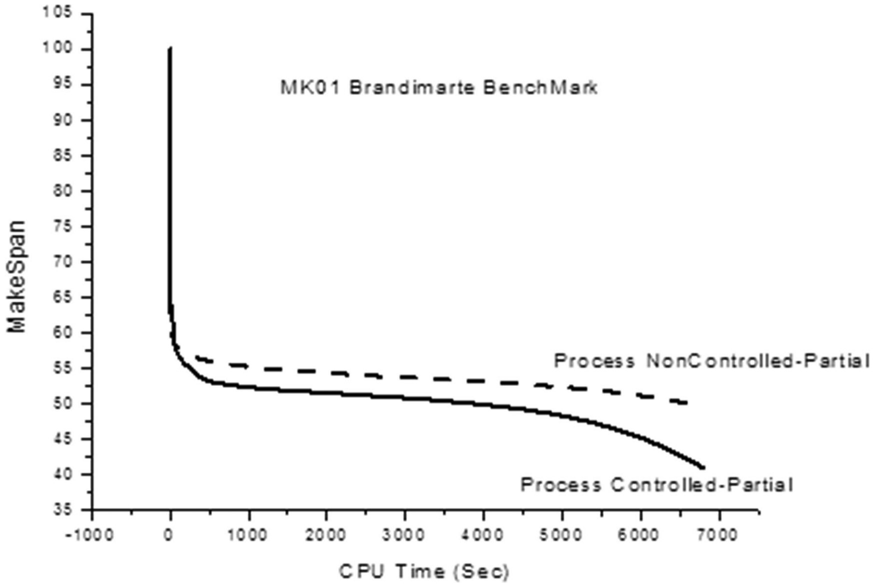


Fig. 7. Performance comparison for the Mk01 problem.

As explained in the SA-Partial controlled process, $T_o = 2 \times \sigma$, the value σ is obtained by running the local search algorithm 30 times with the partial scheduling presented in Fig. 3, using a population of 65,000 solutions. According to the above problem for Mk01 = 11.9, so that $T_o = 2 \times 11.9$ (Table 4). The value of T_o for the noncontrolled process used values ranges from 10 to 2000, which varied at random in each of the 30 executions.

Of the set of 10 test instances shown in Table 4, only those that could be compared with other studies referenced in the literature were chosen to show the effectiveness of the proposed algorithm. Table 5 shows evidence of efficiency and efficacy of the algorithm SA-Partial controlled and SA-Partial noncontrolled, using the average results of 30 runs for the test problems. The data shown in Table 5 correspond to the relative error (RE) for the best solution found (MS = makespan) and the time required to find the solution for the four test cases. According to the results shown in this table, the effectiveness of the algorithm with a controlled process is much better than the algorithm without controlled process, since the RE and time required to find the solution in all test cases is much higher in the noncontrolled process. Table 5 presents the RE with respect to the results of taboo search (Mastrolilli and Gambardella, 2000).

Table 6 shows a comparison of the efficacy of the proposed algorithm SA-Partial controlled with other studies in the literature and application of other heuristics for instances of the Flexible-JSSP for Brandimarte benchmarks. The RE corresponds to the best solution found (MS = makespan). Compared with taboo search, the proposed algorithm obtains the same or improved results in 40% of the problems evaluated. Compared with GA, the proposed algorithm obtains the same or improved results in 67% of the problems evaluated. Compared with the Ant System algorithm, the proposed algorithm obtains the same or improved results in 40% of the problems evaluated. Compared with variable neighborhood search, the proposed algorithm achieves the same results in one of four problems evaluated. According to the results shown in Table 6, the effectiveness of the

Table 6
RE for Flexible-JSSP for Brandimarte benchmarks

Benchmarks	Job/Mach/Op	UB-LB	SA-Partial controlled	Mastrolilli and Gambardella (2000) tabu search	Ho and Tay (2004) genetic algorithm	Gao et al. (2007) Ant system	Amiri et al. (2010) variable neighborhood search
mk01	10/6/55	36–40	0	0	0	0	0
mk02	10/6/58	24–232	17	8.3	21	8.3	9.2
mk03	15/8/90	204–211	5.9	0	n.d.	0	0
mk04	20/5/100	48–81	25	38	40	25	n.d.
mk05	15/4/106	168–186	0	3	4.8	2.4	n.d.
mk06	15/8/150	33–86	79	76	103	76	n.d.
mk07	10/15/150	133–147	11	8.3	11	4.5	5.9
mk08	20/10/225	523	0.2	0	0	0	n.d.
mk09	20/10/240	299–369	2.7	2.7	1	2.7	n.d.
mk010	20/15/240	165–296	19	20	39	19	n.d.

n.d. = no data.

Table 7
RE for Flexible-JSSP for Barnes and Chambers benchmarks

Benchmarks	Job/Mach	SA-Partial controlled simulated annealing	Mastrolilli and Gambardella (2000) taboo search	Oddi et al. (2011) iterative flattering search
mt10x	10/11	0	0	0
mt10xx	10/12	0	0	0
mt10xxx	10/13	0	0	0
mt10xy	10/12	0	0.1	0
mt10xyz	10/13	0	0	0
mt10cl	10/11	0.1	0.1	0
mt10cc	10/12	0	0.2	0
setb4x	15/11	0	0	0
setb4xx	15/12	0	0	0
setb4xxx	15/13	0	0	0
setb4xy	15/12	0.2	0.7	0
setb4xyz	15/13	0	0	0

algorithm with a controlled process appears competitive with respect to other algorithms reported in the literature.

Table 7 shows a comparison of the efficacy of the proposed partial algorithm SA-controlled with other studies in the literature and application of heuristics for instances of the Flexible-JSSP benchmarks of Barnes and Chambers. The RE corresponds to the best solution found (MS = makespan). Compared with taboo search, the proposed algorithm obtains the same or improved results in 100% of the problems evaluated. Compared with flattering iterative search, the proposed algorithm achieves the same result in 67% of the problems evaluated. According to the results shown in Table 7, the effectiveness of the algorithm with a controlled process appears competitive with respect to other algorithms reported in the literature.

6. Conclusions

Experimental results show that the scheduling algorithm, Partial-S, proposed for FJSSP has better efficiency than Total-S, These experimental results were obtained in a shorter time due to the use of partial scheduling instead of full scheduling. Logically, this allows the SA to accelerate its local search and thus it can explore a larger solution space in FJSSP. Moreover, the tuning of the control parameter T_0 by the standard deviation allows an accelerated convergence of the SA-Partial controlled, obtaining better solutions faster, as shown in the experimental results. The combination of these two techniques, the partial scheduling and tuning by the standard deviation, improves the efficacy and efficiency in the process of SA and produces results that compete with those reported in the literature for instances of different sizes of FJSSP.

References

Aarts, E.H.L., Van Laarhoven P.J.M., 1985. Statistical cooling: a general approach to combinatorial optimization problems. *Philips Journal of Research* 40, 4, 193–226.

Alba, E., Luque, G., Nesmachnow, S., 2013. Parallel metaheuristics: recent advances and new trends. *International Transaction in Operational Research* 20, 1–48.

Amiri, M., Zandieh, M., Yazdani M., Bagheri, A., 2010. A variable neighborhood search algorithm for the flexible job-shop scheduling problem. *International Journal of Production Research* 48, 19, 5671–5689.

Applegate, D., Cook, W., 1991. A computational study of the job shop scheduling problem. *ORSA Journal on Computing* 3, 149–156.

Babayan, A., He, D., 2004. Solving the n-job 3-stage flexible flowshop scheduling problem using an agent-based approach. *International Journal of Production Research* 42, 4, 777–799.

Bozejo, W., Uchronski, M., Wodecki, M., 2014. Multi-GPU taboo search metaheuristic for the flexible job shop scheduling problem. *Advanced Methods and Applications in Computational Intelligence* 6, 43–60.

Brandimarte, P., 1993. Routing and scheduling in a flexible job shop by taboo search. *Annals of Operations Research* 2, 158–183.

Cerney, V., 1985. Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications* 45, 1, 41–51.

Chen, J., Chen, F., 2008. Adaptive scheduling and tool flow control in flexible job shops. *International Journal of Production Research* 46, 15, 4035–4059.

Cruz-Chávez, M.A., Frausto-Solís, J., Cora-Mora, J.R., 2006. Experimental analysis of a neighbourhood generation mechanism applied to scheduling problems. *Proceeding of CERMA 2006*, Morelos, Mexico, 26–29 September, pp. 226–229.

Cruz-Chávez, M.A., Martínez-Rangel, M.G., Hernández-Perez, J.A., Zavala-Díaz, J.C., Díaz-Parra, O., 2007. An algorithm of scheduling for the job shop scheduling problem. *Proceeding of CERMA 2007*, Morelos, Mexico, 25–28 September, pp. 336–341.

Dash, G.H. Jr., Kajiji, N., 2014. On multiobjective combinatorial optimization and dynamic interim hedging of efficient portfolios. *International Transactions in Operational Research* 21, 6, 899–918.

Gao, J., Sun, L., Gen, M., 2007. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operation Research* 35, 9, 2892–2907.

Gao, K.Z., Suganthan, P.N., Pan, Q.K., Chua, T.J., Cai, T.X., Chong, C.S., 2014. Pareto based grouping discrete harmony search algorithm for multi-objective flexible job shop scheduling. *Information Sciences* 289, 24, 76–90.

Gu, F., Chen, H.P., Lu, B.Y., 2006. The solution for multi-objective flexible job shop scheduling based on genetic algorithm. *Operations Research & Management Science* 2006, 2, 134–139.

Gu, P., Balasubramanian, S., Norrie, D.H., 1997. Bidding-based process planning and scheduling in a multi-agent system. *Computers and Industrial Engineering* 32, 2, 477–496.

Hansmann, K.W., Hoeck, M., 1997. Production control of a flexible manufacturing system in a job shop environment. *International Transactions in Operational Research* 4, 5/6, 341–351.

Ho, N.B., Tay, J.C., 2004. GENACE: an efficient cultural algorithm for solving the flexible job-shop problem. *CEC2004: Evolutionary Computation,* Vol. 2, IEEE, New York, pp. 1759–1766.

Ida, K., Oka, K., 2011. Flexible job-shop scheduling problem by genetic algorithm. *Electrical Engineering in Japan* 177, 3, 28–35.

Kacem, I., Hammadi, S., Borne, P., 2002a. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions* 32, 1, 1–13.

Kacem, I., Hammadi, S., Borne, P., 2002b. Pareto-optimality approach for flexible job-shop scheduling problem: hybridization of evolutionary algorithms and fuzzy logic. *Journal of Mathematics and Computer in Simulation* 60, 245–276.

Kendall, M.G., Stuart, A., 1958. *The Advanced Theory of Statistics*, Vol. 1. Charles Griffin, London.

Kirkpatrick, S., Gelatt, S.D., Vecchi, M.P., 1983. Optimization by simulated annealing. *Science* 220, 4598, 671–680.

Knopp, S., Dauzere-Peres, S., Yugma, C., 2014. Flexible job-shop scheduling with extended route flexibility for semiconductor manufacturing. Winter Simulation Conference (WSC). IEEE Conference Publications, IEEE, New York, pp. 2478–2489.

Liouane, N., Saad, I., Hammadi, S., Borne, P., 2007. Ant systems & local search optimization for flexible job shop scheduling production. *International Journal of Computers, Communications & Control* 2, 174–184.

Liu, J., Zhang, Ch., Gao, L., Wang, X., 2010. Research on flexible job-shop scheduling problem under uncertainty based on genetic algorithm. *Sixth International Conference on Natural Computation (ICNC) 2010*, Shandong, China, 10–12 August, pp. 2462–2467.

Ma, J., Lei, Y., Wang, Z., Jiao, L., Liu, R., 2014a. A memetic algorithm based on immune multi-objective optimization for flexible job-shop scheduling problems. *IEEE Congress on Evolutionary Computation (CEC) 2014*, Beijing, 6–11 July, pp. 58–65.

Ma, W., Zuo, Y., Zeng, J., Ling, S., Jiao, L., 2014b. A memetic algorithm for solving flexible job-shop scheduling problems. *IEEE Congress on Evolutionary Computation (CEC) 2014*, Beijing, 6–11 July, pp. 66–73.

Martínez-Rangel, M.G., Cruz-Chávez, M.A., Zavala-Díaz, J.C., Juárez-Romero, D., Díaz-Parra, O., 2007. Analysis of the simulated annealing convergence in function of the standard deviation and the Boltzmann quotient for scheduling problems. *Research in Computing Science* 32, 1, 282–293.

Mastrolilli, M., Gambardella, L.M., 2000. Effective neighbourhood functions for the flexible job shop problem. Technical Report, Instituto Dalle Molle Di Studi Sull Intelligenza Artificiale (IDSIA), pp. 45–98.

Mati, Y., Lahlou, Ch., Dauzère-Pérès, S., 2011. Modelling and solving a practical flexible job-shop scheduling problem with blocking constraints. *International Journal of Production Research* 49, 8, 2169–2182.

Nakano, R., Yamada, T., 1991. Conventional genetic algorithm for job-shop problems. In Belew, R.K., Booker, L.B. (eds) *Proceedings of the 4th International Conference on Genetic Algorithms and their Applications*, San Diego, pp. 474–479.

Niroomand, S., Vizvar, B., 2015. Exact mathematical formulations and metaheuristic algorithms for production cost minimization: a case study of the cable industry. *International Transactions in Operational Research* 22, 3, 519–544.

Nouiri, M., Jemai, A., Ammari, A.C., Bekrar, A., Niar, S., 2013. An effective particle swarm optimization algorithm for flexible job-shop scheduling problem. *Industrial Engineering and Systems Management (IESM), Proceedings of 2013 International Conference on Publication*, Rabat, 28–30 October, pp. 1–6.

Oddi, A., Rasconi, R., Cesta, A., Smith, S.F., 2011. Iterative flattering search for the flexible job shop scheduling problem. *Proceeding of the Twenty-Second International Journal Conference on Artificial Intelligence*, AAAI Press, Menlo Park, CA, pp. 1991–1996.

Papadimitriou, C.H., Steigliths, K., 1998. Combinatorial optimization. *Algorithms and Complexity*. Dover Publications, New York.

Rahimi, M., Fallah, E., Amiri, M., 2015. Optimization using simulation and response surface methodology with an application on subway train scheduling. *International Transactions in Operational Research*, DOI: 10.1111/itor.12150.

Roy, B., Sussman, B., 1964. Les problèmes d'ordonnancement avec contraintes disjonctives, Note D.S. No. 9 bis, SEMA, Paris.

Shaw, M.J., 1988. Dynamic scheduling in cellular manufacturing systems: a framework for networked decision making. *Journal of Manufacturing Systems* 7, 2, 83–94.

Shivasankaran, N., Senthilkumar, P., Venkatesh, R.K., 2014. Hybrid non-dominated sorting simulated annealing algorithm for flexible job shop scheduling problems. *Advances in Intelligent Systems and Computing* 248, 101–107.

Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. *Operations Research* 40, 113–125.

Wang, Y.H., Yin, C.W., Zhang, Y., 2003. A multi-agent and distributed ruler based approach to production scheduling of agile manufacturing systems. *International Journal of Computer Integrated Manufacturing* 16, 2, 81–92.

Wu, T.-Y., Wu, I.-C., Liang, C.-C., 2013. Multi-objective flexible job shop scheduling problem based on Monte-Carlo tree search. *Conference on Technologies and Applications of Artificial Intelligence (TAAI) 2013*, Taipei, 6–8 December.

Yamada, T., Nakano, R., 1992. A genetic algorithm applicable to large-scale job-shop. In Männer, R., Manderick, B. (eds) *Parallel Problem Solving from Nature 2*, North-Holland, Amsterdam, pp. 281–290.

Yuan, Y., Xu, H., 2015. Multiobjective flexible job shop scheduling using memetic algorithms. *IEEE Transactions on Automation Science and Engineering* 12, 1, 336–353.

Zalzala, A.M.S., Fleming, P.J. (eds), 1997. *Genetic Algorithms in Engineering Systems*. Institution of Electrical Engineers, London.

Zhang, C.Y., Rao, Y., Li, P.G., Shao, X.Y., 2007. Bilevel genetic algorithm for the flexible job-shop scheduling problem. *Chinese Journal of Mechanical Engineering*, 4, 119–124.

Zhang, Q., Manier, H., Manier, M., 2013. Metaheuristics for job shop scheduling with transportation. In Jarboui, B., Siarry, P., Teghem, J. (eds) *Metaheuristics for Production Scheduling*, Hoboken, NJ, John Wiley & Sons, 465–493.

Zheng, Y., Lian, L., Fu, Z., Mesghouni, K., 2015. Evolutional algorithm in solving flexible job shop scheduling problem with uncertainties. *LISS 2013: Proceedings of 3rd International Conference on Logistics, Informatics and Service Science*, Springer Verlag, Berlin, pp. 1009–1015.

Zhou, W., Bu, Y., Zhou, Y., 2014. Combining CA and PSO to solve flexible job shop scheduling problem. *The 26th Chinese Control and Decision Conference (CCDC) 2014*. Changsha, China, 31 May–2 June, pp. 1031–1036.