Flexible Job-shop Scheduling Problem with Sequence-dependent Setup Times using Genetic Algorithm

Ameni Azzouz, Meriem Ennigrou and Lamjed Ben Said
Lab. SOIE., Stratégies d'Optimisation et Informatique IntelligentE, ISG, Institut Supérieur de Gestion, Université de Tunis, Tunis, Tunisie

Keywords: Job-shop Scheduling Problem, Flexible Manufacturing Systems, Sequence-dependent Setup Times, Genetic Algorithms.

Abstract: Job shop scheduling problems (JSSP) are among the most intensive combinatorial problems studied in literature. The flexible job shop problem (FJSP) is a generalization of the classical JSSP where each operation can be processed by more than one resource. The FJSP problems cover two difficulties, namely, machine assignment problem and operation sequencing problem. This paper investigates the flexible job-shop scheduling problem with sequence-dependent setup times to minimize two kinds of objectives function: makespan and bi-criteria objective function. For that, we propose a genetic algorithm (GA) to solve this problem. To evaluate the performance of our algorithm, we compare our results with other methods existing in literature. All the results show the superiority of our GA against the available ones in terms of solution quality.

1 INTRODUCTION

Job-shop scheduling problem is one of the most important fields in manufacturing optimization where a set of n jobs must be processed on a set of m specified machines. Each job consists of a specific set of operations, which have to be processed according to a given order. This problem falls into the category of NP-hard problems (Garey et al., 1976). The Flexible Job Shop problem (FJSP), first introduced by (Nuijten and Aarts, 1996), is a generalization of the above-mentioned problem, where each operation can be processed by a set of resources and has a processing time depending on the resource used. Then, FJSP is more difficult than the classical JSP. Recently, many studies have been made to find the near optimal solution of FJSP using a varied range of tools and techniques such as (Zhang et al., 2011; Azzouz et al., 2012; Ziaee, 2014; Turkyllmaz and Bulkan, 2014; Azzouz et al., 2015). Most job-shop scheduling researches reported in the literature ignore the setup times or consider them as a part of the processing time. However, in many real-life situations such as chemical, printing, pharmaceutical and automobile manufacturing (Kim and Bobrowski, 1994), the setup times are not only often required between jobs but they are also strongly dependent on job itself (sequence independent) and the previous job that ran on the same machine (sequence dependent). Hence, reducing setup times is an important task to improve shop performance. The FJSP has been widely studied. However, few papers have considered this problem with setup times. In this paper, we propose a genetic algorithm for the FJSP with sequence-dependent setup times (SDST). Then, we show that our algorithm can be very effective with respect to the state of the art. The remainder of this paper is organized as follows. In section 2, we formulate the problem and we give an illustrative example. Section 3 contains some related works of the problem studied. Section 4 presents the proposed algorithm to solve the SDST-FJSP. Section 5 describes the performance of our algorithm on a set of benchmark problems and explains the most interesting results. Conclusions and some future works are presented in section 6.

2 PROBLEM DEFINITION

The SDST-FJSP can be defined as follows: this problem consists in performing n jobs on m machines. The set machines is noted M, $M = \{M_1, ..., M_k\}$. Each job i consists of a sequence of n_i operations (routing). Each routing has to be performed to complete a job. The execution of each operation j of a job i (noted O_{ij}) requires one machine out of a set of given machines $M_{i,j}$ (i.e. $M_{i,j}$ is the set of machines available to execute O_{ij}). The problem is to define a sequence of operations together with assignment of start times and machines for each operation. Assumptions considered in this paper are the following:

- jobs are independent of each other;
- machines are independent of each other;
- one machine can process at most one operation at a time;
- no preemption is allowed;
- all jobs are available at time zero;
- Setup times are dependent on the sequence of jobs. When one of the operations of a job t is processed before one of those of job i ($t \neq i$) on machine M_k, the sequence dependent setup time is $S_{t,i,k} > 0$.

The current SDST-FJSP based on these assumptions is aimed to minimize two kinds of objective functions:

- Minimize the makespan (i.e. the time required to complete all jobs)
- Minimize Aggregate objective function (AOF) where $AOF = \alpha F1 + (1-\alpha)F2$ and de-note the weight given respectively to makespan $(F1)$ and mean tardiness $(F2)$.

FJSP is classified as Total FJSP and Partial FJSP (Kacem and Borne, 2002). In Total FJSP (T-FJSP), each operation can be processes by all machines. However, in Partial FJSP (P-FJSP), at least one operation may not be processed on all machines. Several researches pointed out that the P-FJSP is more complex as compared to T-FJSP on the same scale. In this paper, we consider the P-FJSP.

Table 1: Processing times.

Job	Operation	M1	M2	M3
J1	O11	4	-	5
	O12	-	3	4
	O13	6	5	-
J2	O21	3	-	4
	O22	4	5	-
	O23	-	4	7
J3	O31	5	3	-
	O32	-	4	-
	O33	4	5	3

Table 2: Sequence-dependent setup times.

	Machine1			Machine2			Machine3		
	Job1	Job2	Job3	Job1	Job2	Job3	Job1	Job2	Job3
Dummy	3	2	1	2	4	1	4	3	2
Job1	0	1	3	0	2	4	0	2	3
Job2	1	0	2	4	0	3	2	0	4
Job3	1	3	0	2	1	0	3	2	0

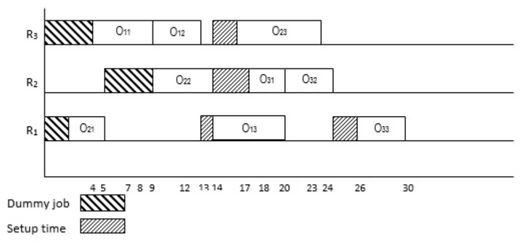

Figure 1: Gantt Chart of solution.

To illustrate this problem, we consider an instance with three jobs and three machines. In table1, we show the processing time of each operation. The symbol "-" means that the machine can not execute the corresponding operation. Table2 presents the sequence dependent setup times of each job. For instance, the setup time for *job*3 after *job*2 on machine M_1 is $S_{2,3,1} = 2$ and the setup time for *job*2 after *job*1 on machine M_3 is $S_{1,2,3} = 2$. Dummy job signifies the starting of a job on each machine. When one of the operations of a *jobi* is the first operation executed on machine M_k, the dummy job is $D_{i,k}$. For example, if $O_{3,1}$ is the first operation executed on machine M_2, then, dummy job value would be $D_{3,1} = 1$. In order to explain better this problem, we represent a Gantt Chart of solution on figure 1.

3 RELATED WORK

The majority of researches in scheduling problem with SDST are restricted to the flowshop problem. Due to the complexity of this problem, most of the literature based on meta-heuristic methods like genetic algorithms (Kaweegitbundit, 2011; Li and Zhang, 2012; Mirabi, 2014) tabu search (Varmazyar and Salmasi, 2012; Santos et al., 2014) and ant colony optimization (Gajpal and Ziegler, 2006; Mirabi, 2011).
Despite the relevance of the job shop in real manufacturing systems, not many papers consider both sequence-dependent setup times and job shop environments. Among these, (Cheung and Zhou, 2001) propose a hybrid algorithm based on genetic algorithm and heuristic rules to solve SDST-JSP with minimizing the makespan. For the same problem, (Zhou et al., 2006) propose an immune algorithm which certifies the diversity of the antibody. (Moghaddas and Houshmand, 2008) develop a mathematical and heuristic model based on priority rules. (Naderi and Ghomi, 2009) consider the job shop scheduling with sequence-dependent setup times and preventive maintenance policies using four meta-heuristics based on simulated annealing and genetic algorithms.
Considering the flexibility constraints, flexible job-

shop problem presents additional difficulty than the classical JSP and requires more effective algorithms. In recent decades, many attempts have been made to find the near optimal solution of SDST-FJSP using a varied range of tools and techniques. (Imanipour, 2006) was the first one who investigates the SDST-FJSP. The author modeled the problem as a non linear mixed integer programming model and proposes a tabu search for the same problem. (Saidi-Mehrabad and Fattahi, 2007) presented a Tabu Search for solving the SDST-FJSP to minimize makespan. They assumed in their research that each operation can be performed by two machine alternatives. They compared their obtained results with the results of the lingo software. (Bagheri and Zandieh, 2011) propose a variable neighborhood search (VNS) based on integrated approach to minimize an aggregate objective function (AOF) where $AOF = \alpha F1 + (1-\alpha)F2$ and α denote the weight given respectively to makespan $(F1)$ and mean tardiness $(F2)$. To evaluate this model, the authors generate randomly 20 problem instances under four different classes. Using the same AOF, (Sadrzadeh, 2013) present an artificial immune system algorithm (AIS) and a particle swarm optimization algorithm (PSO) and prove that both algorithms works better than VNS of (Bagheri and Zandieh, 2011). (Mousakhani, 2013) formulate the SDST-FJSP as a mixed integer linear programming model to minimize total tardiness and present a meta-heuristic based on iterated local search for the same problem. (Oddi et al., 2011) considers the SDST-FJSP to minimize the makespan using the iterative flattering search (IFS) and propose a new benchmark which is denoted SDST-HUdata. It consists of 20 instances produced as an extension of the existing well-known benchmarks of FJSP of (Hurink et al., 1994). (González and Varela, 2013) develop memetic algorithm to minimize the makespan which the tabu search was applied to every chromosome generated by the genetic algorithm. In order to evaluate their model, they used the same benchmark as in (Oddi et al., 2011) and prove that the memetic algorithm has obtained a better result than the IFS. Recently, (Rossi, 2014) investigate the SDST-FJSP with transportation times using ant-colony algorithm with reinforced pheromone. The most recent comprehensive survey of scheduling problem with setup times is given by (Allahverdi, 2015).

4 GENETIC ALGORITHM FOR SDST-FJSP

Since the discovery of the genetic algorithms by (Holland, 1975), they have been recognized as a powerful methods for solving combinatorial optimization problems such as scheduling problems. In our algorithm, we generate the initial population according several dispatching rules. After evaluating each solution in the population, if the stop criterion is not met, there are two choices. According to the probability $P_{crossover}$ and $P_{mutation}$, the current individual executes crossover operator or the mutation one respectively. The stop criterion is that a certain number of iteration is reached or the best solution has not been improved for a certain number of iteration. Next, we present the details of the implementation of our GA components.

4.1 Encoding Problem

For solving SDST-FJSP by GA, the first step is to represent a solution of a problem as a chromosome. We try to design an efficient coding of the individuals which respects the most important constraints of our problem in order to increase the number of feasible solutions produced after genetic recombination. Then, our chromosome is designed as a binary matrix, where:

- The rows correspond to all operations of jobs. Furthermore, the order in which they appear in the chromosome describes the sequence of operations present in the solution.
- The columns correspond to all machines.

Moreover, in our representation, we present a constraint described as follows:

$$\sum_{k=1}^{m} X_{ijk} = 1 \tag{1}$$

$X_{ijk} = 1$ when O_{ij} is assigned to resource M_k
$X_{ijk} = 0$ otherwise.

The sum of each row must be equal to one, to guarantee that each operation is assigned to only one machine. The order, in which the operations appear in this representation, is found according to the start times of the operations. When we have more than one operation executed in the same time, we choose the one which has the smallest number of jobs.

Figure2 illustrate our encoding scheme and represent the solution figured in the Gantt chart in Figure1.

4.2 The Initial Population

Initial population plays a significant role in genetic algorithms in order to get good result. Generally, the initial population is generated randomly in order to maintain the diversity of solutions. Therefore, to increase the diversity of the first generation and to

$$\begin{pmatrix} & M1 & M2 & M3 \\ O_{11} & 0 & 0 & 1 \\ O_{21} & 1 & 0 & 0 \\ O_{22} & 0 & 1 & 0 \\ O_{12} & 0 & 0 & 1 \\ O_{13} & 1 & 0 & 0 \\ O_{23} & 0 & 0 & 1 \\ O_{31} & 0 & 1 & 0 \\ O_{32} & 0 & 1 & 0 \\ O_{33} & 1 & 0 & 0 \end{pmatrix}$$

Figure 2: A sample chromosome encoding by our representation.

maintain a certain quality, we propose an improved function of initial population generation inspired from (Pezzella et al., 2008) which is based on three traditional dispatching rules as following:

- 20% using shortest processing time (SPT): Jobs are scheduled with this rule by sequencing them in ascending order of job processing times per process.
- 20% using longest processing time (LPT): Jobs are scheduled with this rule by sequencing them in descending order of job processing times per process.
- 20% using heuristic rules based on local search algorithm.
- The remaining with random solution.

4.3 Selection

The selection phase aims to choose the chromosomes for reproduction to create the next generation. In this study, we adopt our selection operator (Azzouz et al., 2015). Four individuals are randomly chosen from parent population and the fitness of each of them are compared in order to select the best and the worst solution for reproduction.

4.4 Crossover Operator

The goal of the crossover is to obtain better chromosomes to improve the result by exchanging information contained in the current good ones. As in (Azzouz et al., 2015), we have adopted the crossover operator ”order1” (Davis, 1985).
We adapted this crossover to our own coding described earlier. The idea of this operator is as follows: We randomly select two positions XP1 and XP2 in Parent1. The middle part is copied to the offspring1. The rest is filled from the parent2 starting with position XP2 + 1 and jumping elements that are already present in the offspring 1. The same steps are repeated

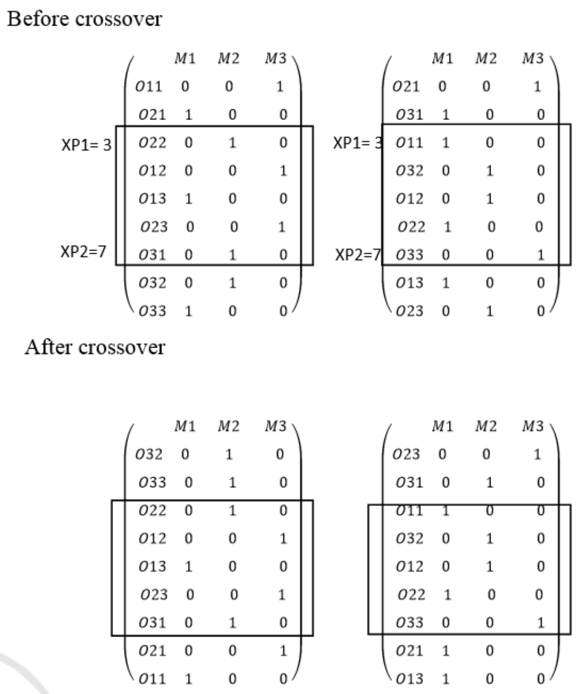

Figure 3: Crossover operator.

for the second offspring by starting with the Parent2. To more explain the crossover operator, we present an example in Figure3. Note that in this example, we take XP1 = 3 and XP2 =7.

4.5 Mutation

Mutation operator is used also to get a new individual having only one value different from an already existing one. In our work, we adopt intelligent mutation proposed by (Pezzella et al., 2008) in which we select an operation on the machine with the maximum workload (i.e. the amount of work that a machine produces in a specified time period), and assign it to the machine with the minimum workload if possible.

5 RESULTS AND EXPERIMENTS

This section evaluates the performance of our proposed genetic algorithm for two kinds of objective functions: makespan and Aggregate Objective Function (AOF).
For that, we compare our GA against the available algorithms in the literature including variable neighourbood search (VNS) proposed by (Bagheri and Zandieh, 2011), an adapted tabu search (TS) proposed by (Ennigrou and Ghedira, 2008), Artificial Immune System (AIS) and Particle swarm Optimization

(PSO) from (Sadrzadeh, 2013). Our proposed algorithm has been implemented using JAVA and run on PC with core2Duo, 2,6GHZ and 2GB RAM. In our experiment, we tested different values for our GA parameters, and computational experience proves that the following values are more effective:

- Population size: 150
- Crossover probability: 0.8
- Mutation probability: 0.2
- Number of iterations (stopping condition): 150
- Number of iterations with no improvement (stopping condition): 30.

For the makespan objective function, we consider the same benchmark as in (Oddi et al., 2011; González and Varela, 2013) which is denoted SDST-HUdata. It consists of 20 instances derived from the first 20 instances of the FJSP benchmark proposed in (Hurink et al., 1994). Each instance was created by adding to the original instance one setup time matrix$S_{t,k}$ for each machine k. The same setup time matrix was added for each machine in all benchmark instances. Each matrix has size nn, and the value $S_{t,i,k}$ indicates the setup time needed to reconfigure the machine k when switches from job t to job i. These setup times are sequence dependent and they fulfill the triangle inequality. The non-deterministic nature of our algorithm makes it necessary to carry out multiple runs on the same instance in order to obtain meaningful results.

After ten runs of each generated instance by the above-mentioned algorithms, the best solutions obtained for each instance (which is namedSol_{min}) are calculated. We use the relative percentage deviation (RPD) measure to compare the performance of algorithms. RPD is obtained as follows:

$$RPD = \frac{Sol_{algo} - Sol_{min}}{Sol_{min}} \times 100 \qquad (2)$$

where SOL_{algo} is the makespan of each algorithm. Table3 show the performance of the proposed GA compared with others algorithms. The instance names are listed in the first column, the second column show the size $(n \times m)$ of each instance. The third, fourth and fifth columns report the obtained results of VNS algorithm (Bagheri and Zandieh, 2011), TS algorithm (Ennigrou and Ghedira, 2008) and our GA.

The obtained results show that the proposed GA performs better than the others algorithms in 16 instances. Only in instance La02, La04 and La08 VNS has gained better results. However, in instance La15, TS obtained the better results. The proposed GA outperforms the others algorithms with average RPD of 1.40 while the worst performing algorithm is TS with average RPD of 4.15. Moreover, we notice that GA obtained the best average RPD of 0.71 in the largest number of machine against 4.58 and 6.48 for VNS and TS respectively. To further evaluate the performance of our algorithm, we study the interaction between the performance of the algorithm and the problem size in figure4. We remark that our algorithm keeps its robust performance in different problem sizes.

Table 3: Summary of results in the SDST-FJSP to minimize the makespan: SDST-HU data benchmark.

Instance	Size $n \times m$	VNS	TS	GA
La01	10×5	2.54	7.36	0.00
La02		1.58	8.53	2.57
La03		2.66	5.33	0.13
La04		3.50	3.66	4.07
La05		1.11	3.58	0.16
La06	15×5	4.45	2.79	2.04
La07		2.92	2.99	1.97
La08		2.37	2.41	2.86
La09		3.82	3.21	0.41
La10		1.84	4.02	0.00
La11	20×5	3.32	2.08	1.61
La12		5.33	2.71	2.38
La13		3.30	3.14	3.02
La14		1.70	1.76	1.26
La15		2.37	1.18	1.97
La16	10×10	2.37	4.04	0.00
La17		4.34	2.39	0.07
La18		1.86	5.81	0.91
La19		6.37	6.67	0.11
La20		4.84	9.36	2.48
Average		3.12	4.15	1.40

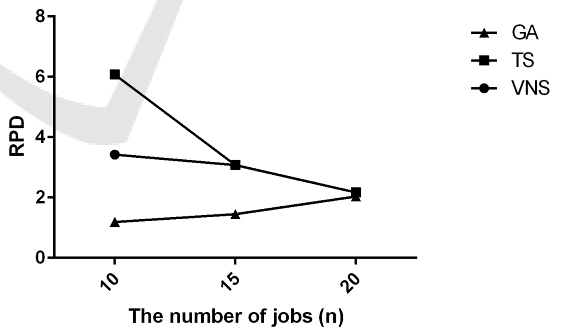

Figure 4: The average RPD of the algorithms versus the number of jobs.

Furthermore, for the AOF, we consider artificial benchmarks according to the function proposed by (Bagheri and Zandieh, 2011). We propose four classes of instances. These classes are different in number of jobs, n, number of operations for each job$i$, n_i, and number of machines, m, that are denoted as $(n \times n_i \times m)$. The generated instances have partial flexibility and the number of available machines for each operation (AMO) is generated randomly accord-

Table 4: The characteristics of the instances.

	$n \times n_i \times m$	AMO	Processing time	SDST	Dummy Jobs
Class1	$10 \times 5 \times 5$	U(1,5)	U(20,100)	U(20,60)	U(20,40)
Class2	$10 \times 5 \times 8$	U(1,8)			
Class3	$10 \times 10 \times 5$	U(1,5)			
Class4	$15 \times 10 \times 10$	U(1,10)			

ing to uniform distribution. Table 4 summarizes the characteristics of the artificial benchmarks used in this paper. In order to introduce due dates, we consider the same formula as in (Bagheri and Zandieh, 2011).

Overall, compared to VNS, AIS and PSO, our GA has a superiority result to minimize the *AOF* for all α values. Moreover, from the results shown in figure 5,6

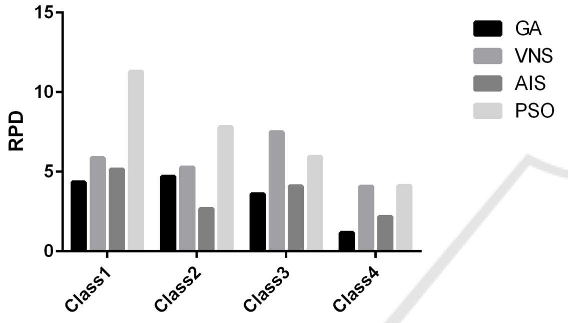

Figure 5: The average RPD of the algorithms of each type of problem class for $\alpha = 0.25$.

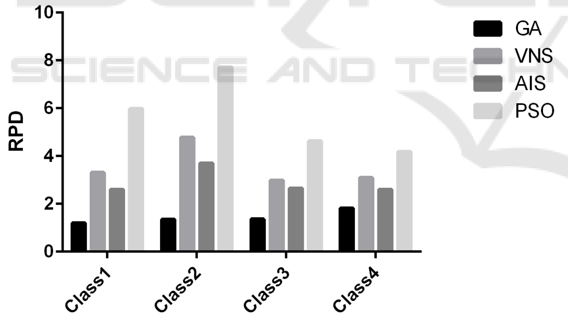

Figure 6: The average RPD of the algorithms of each type of problem class for $\alpha = 0.5$.

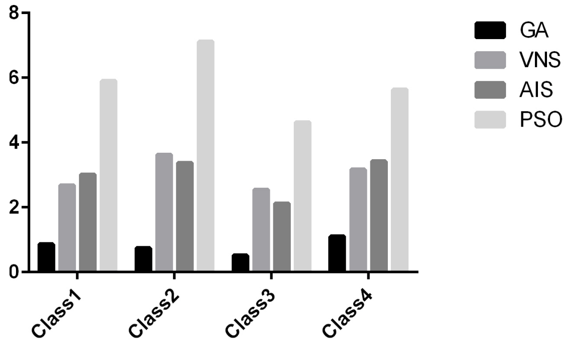

Figure 7: The average RPD of the algorithms of each type of problem class for $\alpha = 0.75$.

and 7, we remark that GA is more effective with $\alpha = 0.75$ then $\alpha = 0.25$. Otherwise, our algorithms have the best results with makespan against mean tardiness objective function.

6 CONCLUSION

In this paper, we focus on solving the flexible job shop scheduling problem where sequence dependent setup times are also taken into account. We have proposed genetic algorithm to minimize two kinds of objective functions: makespan and aggregate objectives function. For that, we tested GA on two kinds of benchmark. Results showed that the present GA is better than other algorithms. In future works, it will be interesting to investigate the dynamic scheduling problem to closely reflect the real flexible job shop scheduling environment. For the same reason, we will consider the multi-criteria scheduling problem and the scheduling problems with learning effects considerations.

ACKNOWLEDGEMENTS

The authors would like to say thanks to Miguel A. González for providing us with the SDST-FJSP instances.

REFERENCES

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. In *European Journal of Operational Research.*

Azzouz, A., Ennigrou, M., and Jlifi, B. (2015). Diversifying ts using ga in multi-agent system for solving flexible job shop problem. In *Proceedings of the 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO) vol1,pp 94-101.*

Azzouz, A., Ennigrou, M., Jlifi, B., and Ghedira, K. (2012). Combining tabu search and genetic algorithm in a multi-agent system for solving flexible job shop problem. In *In 11th Mexican International Conference on ArtificialIntelligence (MICAI) pp. 83-88.*

Bagheri, A. and Zandieh, M. (2011). Bi-criteria flexible job-shop scheduling with sequence-dependent setup times—variable neighborhood search approach. In *Journal of Manufacturing Systems, 30(1), 8-15.*

Cheung, W. and Zhou, H. (2001). Using genetic algorithms and heuristics for job shop scheduling with sequence-dependent setup times. In *Ann Oper Res 107:65.*

Davis, L. D. (1985). Applying adaptive algorithms to epistatic domains. In *In Proc.InternationalJoint Conference on Artificial Intelligence. 162-164.*

Ennigrou, M. and Ghedira, K. (2008). New local diversification techniques for flexible job shop scheduling problem with a multi-agent approach. In *In Autonomous Agents and Multi-Agent Systems, vol.17(2),270-287.*

Gajpal, Y.and Rajendran, C. and Ziegler, H. (2006). . an ant colony algorithm for scheduling in flowshops with sequence-dependent setup times of jobs. In *Int J of Advanced Manufacturing Technology 30 (5-6), 416-424.*

Garey, M. R., Johnson, D. S., and Stockmeyer, L. (1976). Some simplified np-complete graph problems. In *Theoretical computer science, 1(3), 237-267.*

González, M. Á.and Rodríguez Vela, C. and Varela, R. (2013). An efficient memetic algorithm for the flexible job shop with setup times. In *InTwenty-Third International Conference on Automated Planning and Scheduling pp 91-99.*

Holland, J. (1975). Adaptation in natural and artificial system. In *MIT University of Michigan Press, Ann Arbor.*

Hurink, J., Jurisch, B., and Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. In *Operations-Research-Spektrum, 15(4), 205-215.*

Imanipour, N. (2006). Modeling & solving flexible job shop problem with sequence dependent setup times. In *International Conference on Service Systems and Service Management (Vol. 2, pp. 1205-1210). IEEE.*

Kacem, I.and Hammadi, S. and Borne, P. (2002). Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. In *Syst IEEE Syst Man Cybern 32(1):1-13.*

Kaweegitbundit, P. (2011). Comparison of heuristic for flow shop scheduling problems with sequence dependent setup time. In *Advanced Materials Research 339 (1), 332-335.*

Kim, S. C. and Bobrowski, P. M. (1994). Impact of sequence-dependent setup time on job shop scheduling performance. In *The International Journal of Production Research, 32(7), 1503-1520.*

Li, X. and Zhang, Y. (2012). Adaptive hybrid algorithms for the sequence-dependent setup time permutation flow shop scheduling problem. In *IEEE Transactions on Automation Science and Engineering 9(3), 578-595.*

Mirabi, M. (2011). Ant colony optimization technique for the sequence-dependent flowshop scheduling problem. In *Int J of Advanced Manufacturing Technology 55 (1-4), 317-326.*

Mirabi, M. (2014). A novel hybrid genetic algorithm to solve the sequence-dependent permutation flow-shop scheduling problem. In *International Journal of Advanced Manufacturing Technology 74 (1-4), 429-437.*

Moghaddas, R. and Houshmand, M. (2008). Job-shop scheduling problem with sequence dependent setup times. In *Proceedings of the International MultiConference of Engineers and Computer Scientists, Vol II.*

Mousakhani, M. (2013). Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness. In *International Journal of Production Research, 51(12), 3476-3487.*

Naderi, B.and Zandieh, M. and Ghomi, S. F. (2009). Scheduling sequence-dependent setup time job shops with preventive maintenance. In *The International Journal of Advanced Manufacturing Technology, 43(1-2), 170-181.*

Nuijten, W. P. and Aarts, E. H. (1996). A computational study of constraint satisfaction for multiple capacitated job shop scheduling. In *European Journal of Operational Research, 90(2), 269-284.*

Oddi, A., Rasconi, R., Cesta, A., and Smith, S. (2011). Applying iterative flattening search to the job shop scheduling problem with alternative resources and sequence dependent setup times. In *In COPLAS 2011 Proceedings of the Workshopon Constraint Satisfaction Techniques for Planning and Scheduling Problems.*

Pezzella, F., Morganti, G., and Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. In *Computers & Operations Research,35(10), 3202-3212.*

Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. In *International Journal of Pro-duction Economics, 153, 253-267.*

Sadrzadeh, A. (2013). Development of both the ais and pso for solving the flexible job shop scheduling problem. In *Arabian Journal for Science and Engineering, 38(12), 3593-3604.*

Saidi-Mehrabad, M. and Fattahi, P. (2007). Flexible job shop scheduling with tabu search algorithms. In *The International Journal of Advanced Manufacturing Technology, 32(5-6), 563-570.*

Santos, N., Rebelo, R., and Pedroso, J. (2014). A tabu search for the permutation flow shop problem with sequence dependent setup times. In *International Journal of Data Analysis Tech-niques and Strategies 6 (3), 275-285.*

Turkyllmaz, A. and Bulkan, S. (2014). A hybrid algorithm for total tardiness minimisation in flexible job shop: genetic algorithm with parallel vns execution. In *International Journal of Production Research, 53(6), 1832-1848.*

Varmazyar, M. and Salmasi, N. (2012). Minimizing the number of tardy jobs in flow shop sequence dependent setup times scheduling problem. In *Applied Mechanics and Materials 110-116, 4063-4069.*

Zhang, G., Gao, L., and Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. In *Expert Syst. Appl. 38(4)3563-3573.*

Zhou, Y., Beizhi, L., and Yang, J. (2006). Study on job shop scheduling with sequence-dependent setup times using biological immune algorithm. In *Int J Adv Manuf Technol 30:105-111.*

Ziaee, M. (2014). A heuristic algorithm for solving flexible job shop scheduling problem. In *Int Adv Manuf Technol 71: 519.*