Solving flexible job-shop problem with sequence dependent setup time and learning effects using an adaptive genetic algorithm

Ameni Azzouz*, Meriem Ennigrou and Lamjed Ben Said

Institut Supérieur de Gestion,
SMART Lab,
Université de Tunis, Tunisia
Email: ameni.azzouz@isg.rnu.tn
Email: meriem.ennigrou@enit.rnu.tn
Email: Lamjed.bensaid@isg.rnu.tn
*Corresponding author

Abstract: For the most schedulling problems studied in literature, job processing times are assumed to be known and constant over time. However, this assumption is not appropriate for many realistic situations where the employees and the machines execute the same task in a repetitive manner. They learn how to perform more efficiently. As a result, the processing time of a given job is shorter if it is scheduled later, rather than earlier in the sequence. In this paper, we consider the flexible job-shop problem (FJSP) with two kinds of constraint, namely, the sequence-dependent setup times (SDST) and the learning effects. Makespan is specified as the objective function to be minimised. To solve this problem, an adaptive genetic algorithm (AGA) is proposed. Our algorithm uses an adaptive strategy based on: 1) the current specificity of the search space; 2) the preceding results of already used operators; 3) their associated parameter settings. We adopt this strategy in order to maintain the balance between exploration and exploitation. Experimental studies are presented to assess and validate the benefit of the incorporation of the learning process to the SDST-FJSP over the original problem.

Keywords: schedulling problem; genetic algorithm; adaptive strategy; learning effects.

Reference to this paper should be made as follows: Azzouz, A., Ennigrou, M. and Ben Said, L. (2020) 'Solving flexible job-shop problem with sequence dependent setup time and learning effects using an adaptive genetic algorithm', *Int. J. Computational Intelligence Studies*, Vol. 9, Nos. 1/2, pp.18–32.

Biographical notes: Ameni Azzouz received her BSc and MS from the University of Mannouba, Tunis, Tunisia, in 2010 and 2013, respectively and her PhD from the University of Tunis in 2017, all in Computer Science. She is currently a research member of SMART Lab. with University of Tunis,

Tunisia. Her current research interests include scheduling problems with time dependent processing times, learning effects, optimisation and evolutionary computation.

Meriem Ennigrou received her PhD in Computer Science from the Institut supérieur de gestion de Tunis, University of Tunis in 2010. She is an Assistant Professor and research member of SMART Lab. with University of Tunis, Tunisia. She is currently working on scheduling problems, evolutionary algorithms and vehicle routing problems.

Lamjed Ben Said received his BSc from the University of Tunis, Tunisia, in 1998, and his MSc and PhD from the University of Paris VI, Paris, France, in 1999 and 2003, respectively, all in Computer Science. He is currently a Professor of Computer Science with the Université de Tunis, where he is the Head of the SMART Laboratory and the Dean of Institut supérieur de Gestion (Université of Tunis). Author and co-author of 130 journal/patent/international conference/chapter book papers. His current research interests include multiagent systems, multicriteria decision making, evolutionary computation and behavioral economics.

This paper is a revised and expanded version of a paper entitled [title] presented at [name, location and date of conference].

1 Introduction

In classical schedulling problems, job processing times are assumed to be constant over time. This assumption might be unrealistic in many situations, because early empirical studies in several kinds of industries have demonstrate that the productivity of a production facility (a machine, a worker, etc.) improves continuously by performing similar tasks repeatedly. In others words, the learning effects take place in manufacturing systems when the workers are able to perform setup, to deal with machine operations and software, to read data or to handle raw materials. Motivate by this issue, Wright (1936) was first discovered this phenomenon in the aircraft industry and subsequently in different branches of industry, see, for example the literature review in Yelle (1979). Wright's formula has showed that a given operation is subject to a 20% productivity improvement each time the production quantity doubles. After the success of Wright's formula, many others form of learning models have been proposed in order to represent the learning effect as realistic as possible such as the Stanford-B model, DeJong model, S-curve model and plateau model (Badiru, 1992).

Although the impact of learning on productivity in manufacturing systems was discovered 80 years ago, only in last decades, Biskup et al. (1999) and Cheng and Wang (2000) were proposed this concept independently in the framework of schedulling problems. According to Biskup et al. (1999), the time needed to produce a unit decreases, usually following a negative exponential curve, as the number of repetitions of job. Hence, the actual processing time of job i if scheduled in position r, is given by: $P_{i,r} = P_i r^a$ where $j, r = 1, \ldots, n$, $P_{i,r}$ is the actual processing time of job i, $a < 0$ is a constant learning index, P_i is the normal processing time (i.e., the processing time without any learning effects) of job i and n is the total number of jobs.

As the production environments will keep changing constantly, several learning effect models have been proposed to make the formulation of problem as realistic as possible. For a comprehensive survey on schedulling problems with learning effects, the reader may refer to Biskup (2008) and Azzouz et al. (2018). Generally, researches on this context are almost deal with single and parallel machine schedulling problems. However, we are interested for the most complex combinatorial schedulling problem called the job-shop schedulling problem (JSSP). The JSSP consists of a set of n jobs that must be processed on a set of m specified machines. Each job consists on a specific set of operations, which have 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 generalisation 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. Accordingly, FJSP is more difficult than the classical JSSP. Recently, many researchers 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) and Azzouz et al. (2017).

Most job-shop schedulling 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 (independent sequence) and on the previous job that ran on the same machine (sequence dependent). Hence, reducing setup times is an important task for an efficient shop performance. The FJSP has been widely studied. However, a few papers have considered this problem with setup times. Among these, 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 minimise 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 neighbourhood search (VNS) based on integrated approach to minimise an aggregate objective function (AOF). Mousakhani (2013) formulate the SDST-FJSP as a mixed integer linear programming model to minimise total tardiness and propose a meta-heuristic based on iterated local search on the same problem. Oddi et al. (2011) consider the SDST-FJSP to minimise the makespan using the iterative flattering search (IFS) and propose a new benchmark which is denoted SDST-HU data. González et al. (2013) develops a memetic algorithm to minimise the makespan when the tabu search was applied to every chromosome generated by the genetic algorithm.They proved that the memetic algorithm has obtained the better result against the IFS. The most recent comprehensive survey of schedulling problem with setup times is given by Allahverdi (2015).

Most modern algorithms proposed to solve this kind of problem consider hybridisation between more than one algorithm. It is well known that the performance of a hybrid algorithm is heavily dependent on the setting of control parameters. For that, there is an increasing recent trend to consider adaptive mechanisms to alter operator choices and/or their parameters such as an adpative local search for a continuous dynamic optimisation problems (Riley et al., 2016; Yu et al., 2013; Mavrovouniotis

et al., 2015). This latter propose a single-solution-based metaheuristic that perturbs the variables separately in order to select the next search direction. Leon and Xiong (2015) propose a differential evolution algorithm based on greedy adaptation of control parameters. Voglis (2013) consider an adaptive memetic algorithm based on particular swarm optimisation with variable local search pool size. For that, the authors propose an adaptive selection strategy based on local search improvement scores. Kafafy et al. (2013) propose a hybrid evolutionary approach with search strategy adaptation for the continuous domains. This latter uses multiple search strategies such as genetic operators, differential evolution (DE), particle swarm optimisation (PSO) and guided mutation operator. In the initial step, all strategies have the same chance to be selected. Next, the more successfully strategy behaved in the previous iteration, the more probability it will be chosen in the current iteration to be used for generating the new solutions.

In this paper, we investigate the SDST-FJSP with learning considerations. Makespan is specified as the objective function to be minimised. There are two main motivations behind this work. In the one hand, a typical production system for the FJSP is the aircraft industry. In the other hand, learning phenomenon was first discovered in aircraft industry. Furthermore, learning effects exists in a manufacturing system characterised by a high level of human activity (Wright, 1936) (setting up the machine, cleaning the machines after the processing of a job, reading machine date, all kinds of handwork). To this ends, it is so interesting to study the SDST-FJSP with learning considerations. According to the best of our knowledge, there are no research which study the impact of learning effects on the FJSP. To study this problem, we propose a new adaptive genetic algorithm called AGA. Then, we show that our algorithm can be very effective with respect to the state-of-the-art. The remainder of this paper is organised as follow. In Section 2, we formulate the problem and we give an illustrative example. Section 3 presents the proposed algorithm to solve the SDST-FJSP. Section 4 describes the performance of our algorithm on a set of benchmark problems and explains the most interesting results. Conclusions and some future works are presents in Section 5.

2 Problem definition

The problem can be defined as follows: it consists in performing n jobs on m machines. The set of machines is noted $M, M = \{M_1, \ldots, M_k\}$. 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 machineM_k, the sequence dependent setup time is $S_{t,i,k} > 0$. The applied position-based learning effect is as follow:

$$S'_{t,i,k,r} = S_{t,i,k} r^a \qquad t, i, r = 1, \ldots, n \tag{1}$$

where $S'_{t,i,k}$ the actual sequence-dependent setup times (SDST) for i^{th} jobs on k^{th} machine and S_{tjk} is normal SDST. The current SDST-FJSP based on these assumptions is aimed to minimise the makespan objective function (i.e., the time required to complete all jobs). FJSP is classified as total FJSP (T-FJSP) and partial FJSP (P-FJSP) (Kacem et al., 2002). In T-FJSP, each operation can be processed by all machines. However, in P-FJSP, at least one operation may not be processed on all machines. In this paper, we consider the P-FJSP.

In the following, we briefly describe the specificity of our proposed algorithm.

3 The proposed AGA

Our main motivation behind this work is to solve the SDST-FJSP with learning consideration using an adaptive algorithm which update the parameters values regards to the requirement of the current search space. Tuning control parameters has a major impact on the performance of an algorithm. For that, we try to adapt the most important parameters in our model. In the following, we describe the details of our AGA components.

3.1 Genetic algorithm

Since the discovery of the genetic algorithms by Holland (1975), they have been recognised as a powerful methods for solving combinatorial optimisation problems such as schedulling 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.

3.1.1 Solution encoding

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:

$$\sum_{k=1}^{m} X_{ijk} = 1 \tag{2}$$

- 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. Figure 2 illustrate our encoding scheme.

Figure 1 A sample chromosome encoding by our representation

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

3.1.2 The initial population

Initial population plays a significant role in genetic algorithms in order to get good results. 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 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.

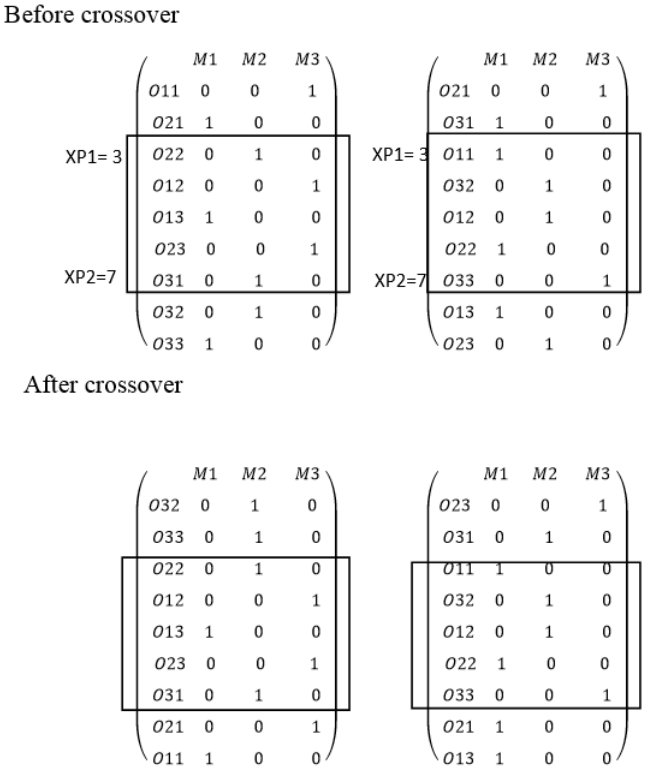
3.1.3 Selection

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.

3.1.4 Crossover operato

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).

Figure 2 Crossover operator



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 offspring1. The same steps are repeated for the second offspring by starting with the parent2. To more explain the crossover operator, we present an example in Figure 3. Note that in this example, we take XP1 = 3 and XP2 =7.

3.1.5 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.

3.2 Proposed adaptive strategy

Our adaptation strategy is based on three features:

1 a new strategy to initialise the parameter

2 an adaptive strategy based on the specificity of the current search space

3 an update function for all control parameters values based on the result of previous step.

This latter is the core of our adaptive strategy.

Table 1 Parameters configurations

	P_{cross}	P_{mut}	pop_{size}	*nbr_iter_max*
Low	0.2	0.2	50	50
Medium	0.3	0.3	60	80
High	0.6	0.6	80	120

3.2.1 Parameter initialisation

Most researches assume fixed parameter values with reference to values of the previous similar literature or after some preliminary experiments. However, the simplest way to improve the performance of an algorithm is to repeat the search from another starting point. For that, we propose, in each execution, a dynamic initialisation of parameters with random values. Nevertheless, each parameter has a large number of alternatives which can be an appropriate value. For this reason, we define three levels of values which can be affected to the parameter: *low*, *medium* and *high* level. In order to achieve more precise and stable results for our proposed algorithm, we have considered six parameters for tuning: P_{cross}, P_{mut}, pop_{size}, nbr_iter_max and sim.The details of these parameters with their level are described in Table 1. After initialising all parameters, we create the initial population of random individuals of size pop_{size}.

3.2.2 Adaptive strategy

As we previously mentioned, this strategy is based on the specificity of the current search space. How can we define this specificity and what is their impact in order to select the next search direction? In this context, in order to detect the specificity of the current search space, we propose two main functions: similarity and archiving strategy.

3.2.3 Similarity function

The similarity function calculates the percentage of similarity between all individuals in the current population. We called this percentage S_{pop}. For example, if S_{pop} is near to 100% that means that this population is completely identical. In this case, the next direction search space needs a more diversification technique than intensification. Otherwise, if S_{pop} is near to 30%, then, the next direction search space must be an intensification technique in order to maintain the balance between these two features. The pseudo code of similarity function is described in Algorithm 1. In line 8, we test the similarity between two individuals. If these latter are identical, then, this function return true. Otherwise, return false. Line 13 presents the percentage of similarity in the current population.

Algorithm 1 Similarity function

Input: pop population with size pop_{size}
Output: S_{pop} /*The percentage of similarity in pop*/
1: **Begin**
2: $i \leftarrow 0$; $j \leftarrow 1$
3: $nbr\_similar \leftarrow 0$
4: $nbr\_similar\_max \leftarrow determine\_nbr\_similar\_max(pop_{size}$
/*This function return the number of similarity in the case of 100% similarity*/
5: **For** each $indiv_i$ from pop **To** pop_{size} **do**
6: $j \leftarrow i + 1$
7: **For** each $indiv_j$ from pop **To** $pop_{size} - 1$ **do**
8: **If** $(Similar\_indiv(indiv_i, indiv_j) == True)$ **Then**
9: $nbr\_similar \leftarrow nbr\_similar + 1$
10: **End If**
11: **End For**
12: **End For**
13: $S_{pop} \leftarrow (nbr\_similar/nbr\_similar\_max) * 100$
14: **End**

3.2.4 Archiving strategy

The second function adopted in adaptive strategy is to save the successful move related to the selected operators. When one of the genetic operators described previously improve the quality of the initial solution, then, this successful selection is stored in the external archive. At the end of each iteration, we adapt all parameters according to this external archive. For example, if the number of successful moves of crossover is more than mutation, consequently, we make the probability to select crossover superior regards to the mutation. In fact, the objective of this step is to speed up the optimisation process.

3.2.5 Update control parameter

As previously stated, there are three levels of parameters values. Hence, we limit the variation of each parameter to three possible alternatives: high, medium or low values. Furthermore, we propose several kinds of scenario in which our model can update the parameter value in order to find the best solution in the search space. All proposed scenarios are summarised as follows:

- *Scenario 1:* If the similarity of the current population S_{pop} is more than a the *high level* of threshold sim, then, our algorithm adapt the parameters in order to augment the diversification feature as follows:

  1 P_{cross} increases to the *high level*

  2 P_{mut} decreases to the *low level*

  3 pop_{size} and nbr_iter_max increase to the *high level*.

- *Scenario 2:* If S_{pop} is less than the *low level* of the threshold sim, then, our algorithm performs the following steps:

  1 P_{mut} increases to the*high level*

  2 P_{cross}, pop_{size} and nbr_iter_max decrease to the *lower level*.

- *Scenario 3:* If S_{pop} is near to the *medium level* of the threshold sim, then, our algorithm perform the following steps:

  1 P_{mut} and P_{cross} change to the *medium level*

  2 pop_{size} and nbr_iter_max remains constant.

- *Scenario 4:* Here, we calculate the number of successful move for each algorithm and we save the largest one (i.e., $best\_succ\_move\_oper$). The operators which has the $best\_succ\_move\_oper$, increase its probability by one level.

We note here that in the case that the parameter values are already at the extremity level and they cannot be further increase, then, the value remains constant.

4 Experimental study

This section has two main objectives: First, we evaluate our proposed algorithm by comparing our result against the counterpart without adaptive strategy (static scheme) (Azzouz et al., 2016) and VNS proposed by Bagheri and Zandieh (2011). This comparison validates the benefits of our adaptive strategy. Secondly, we further analyse the benefit of incorporating each of the adaptive strategy components.

4.1 Used benchamarks and metrics

In this subsection, we describe the different benchmark problems used in our experimental study. In fact, we use commonly used test problems within the community that allow assessing the performance of any used algorithm with respect to different kinds of difficulties. We choose to use the benchmark proposed by Oddi et al. (2011) and González et al. (2013) denoted $SDST - HUdata$. It contains 20 instances derived from the first 20 instances of the FJSP benchmarks proposed by 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 considered for all benchmark instances. Each matrix has a size of $n \times n$, and the $S_{t,i,k}$ value indicates the setup time

needed to reconfigure the machine k when it switched from job t to job i. These setup times are sequence dependent and they fulfill the inequality triangle. To evaluate now the generated results and to compare the performance of the used algorithms we use the makespan which denotes the completion time of the last operation.

4.2 Parameter tuning and settings

This section is devoted to detail the parameter setting for each of the used algorithms under comparison. As we mentioned previously, we initialise AGA by choosing a random level for each parameter from Table 1. For the same algorithm without adaptive strategy, we consider the same probability for all algorithm (i.e., each algorithm has a probability equal to 0.33). Furthermore, we affect the medium value for pop_{size} and nbr_iter_max.

4.3 Results analysis and disscussion

Table 2 shows the results of the experiments in SDST-HU data benchmarks. The instance names are listed in the first column, the second column shows the size $(n \times m)$ of each instance. The remaining columns report the obtained results of of the used algorithms GA and AGA and VNS proposed by Bagheri and Zandieh (2011). We indicate the best makespan in 10 runs for each instance.

Table 2 Makespan values of GA and AGA on SDST-HU data benchmarks

Problem instance	*Size (n × m)*	*Flexibility*	*VNS*	*GA*	*AGA*
La01	10 × 5	1.15	721	721	709
La02			908	871	900
La03			828	768	764
La04			774	794	782
La05			726	613	621
La06	15 × 5	1.15	1,096	1,055	1,004
La07			1,155	1,034	1,094
La08			1,150	1,132	1,052
La09			1,106	1,053	1,069
La10			1,077	1,058	1,010
La11	20 × 5	1.15	1,446	1,431	1,446
La12			1,409	1,291	1,270
La13			1,354	1,300	1,255
La14			1,402	1,340	1,333
La15			1,629	1,626	1,616
La16	10 × 10	1.15	1,397	1,282	1,282
La17			1,059	999	991
La18			1,165	1,192	1,146
La19			1,224	1,174	1,168
La20			1,224	1,190	1,179

We observe from Table 2 that AGA presents the best performance on the majority of test problems. It outperforms GA on all 14 benchmark problems. This reflects the ability of AGA to control the search process by executing the same genetic operators at the appropriate time. To further evaluate the performance of our algorithm, we study the interaction between the performance of the algorithm and the problem size in Figure 3. We remark that the AGA keeps its robust performance in different problem sizes.

Figure 3 Makespan values of the used algorithms regarding to the number of jobs (see online version for colours)

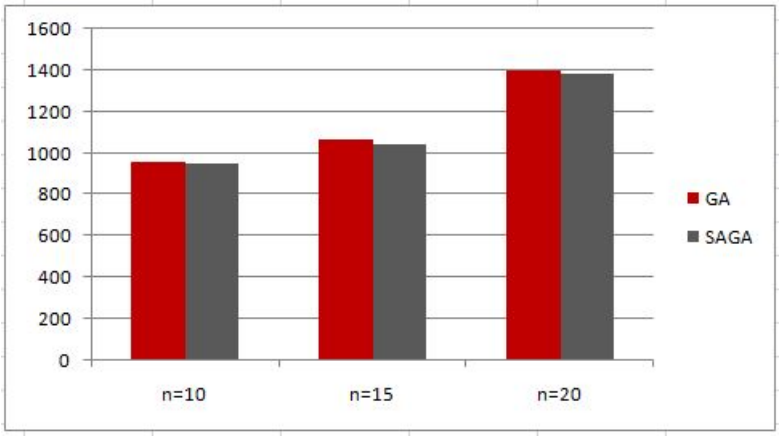
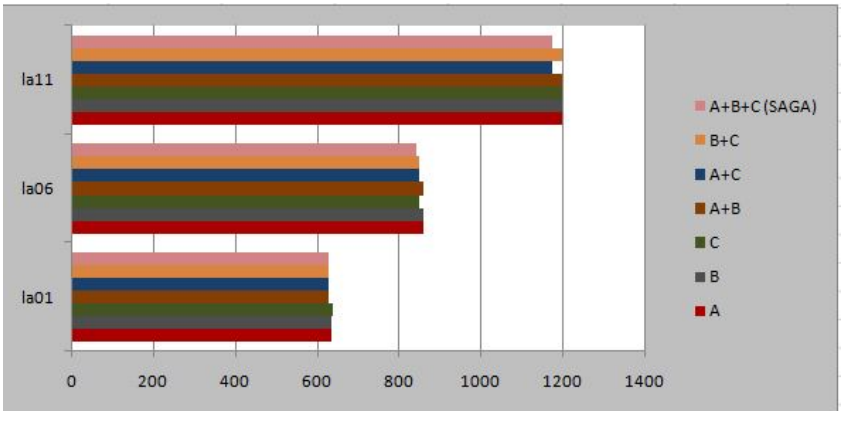


Figure 4 The impact of our adaptive strategy used in AGA (see online version for colours)



In order to further analyse the impact of our adaptive strategy, we evaluate each of its three components separately. We denote A, B and C for the parameter initialisation function, archiving function and similarity function respectively. Figure 4 presents the performance of the AGA for three different problem La01, La06 and La16 with n = 10, 15 and 20 respectively. Here, we state that archiving function and parameter initialisation function (A + C) achieves a better performance than other alternatives. However, our AGA with all its components still have the best results for all instances.

5 Conclusions

In this paper, we consider two realistic assumptions with the FJSP, namely, the SDST and the learning effects. For that, we have proposed an AGA to minimise makespan. AGA adopt a new adaptation strategy based on similarity function and archiving process. Results showed that the present algorithm is better than its counterpart without adaptive strategy. In future works, it will be interesting to study the impact of AGA on the dynamic SDST-FJSP with learning effects, in order to reflect as closely as possible the reality of the actual flexible manufacturing systems.

References

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

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 *Proceedings of the 11th Mexican International Conference on Artificial Intelligence(MICAI) (Special Sessions)*, pp.83–88.

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)*, Vol. 1, pp.94–101.

Azzouz, A., Ennigrou, M. and Ben Said, L. (2016) 'Flexible job-shop scheduling problem with sequence-dependent setup times using genetic algorithm', in *18th International Conference on Enterprise Information Systems (ICEIS)*, Vol. 2, pp.47–53.

Azzouz, A., Ennigrou, M. and Ben Said, L. (2017) 'A hybrid algorithm for flexible job-shop scheduling problem with setup times', *International Journal of Production Management and Engineering*, Vol. 5, No. 1, pp.23–30.

Azzouz, A., Ennigrou, M. and Ben Said, L. (2018) 'Scheduling problems under learning effects: classification and cartography', *International Journal of Production Research*, Vol. 56, No. 4, pp.1642–1661.

Badiru, A.B. (1992) 'Computational survey of univariate and multivariate learning curve models', *IEEE Transactions on Engineering Management*, Vol. 39, No. 2, pp.176–188.

Bagheri, A. and Zandieh, M. (2011) 'Bi-criteria flexible job-shop scheduling with sequence-dependent setup times – variable neighborhood search approach', *Journal of Manufacturing Systems*, Vol. 30, No. 1, pp.8–15.

Biskup, D. (1999) 'Single-machine scheduling with learning considerations', *European Journal of Operational Research*, Vol. 115, pp.173–178.

Biskup, D. (2008) 'A state-of-the-art review on scheduling with learning effects', *European Journal of Operational Research*, Vol. 188, No. 2, pp.315–329.

Cheng, T.C.E. and Wang, G. (2000) 'Single machine scheduling with learning effect considerations', *Annals of Operations Research*, Vol. 98, Nos. 1–4, pp.273–290.

Davis, L.D. (1985) 'Applying adaptive algorithms to epistatic domains', in *Proc. International Joint Conference on Artificial Intelligence*, pp.162–164.

Garey, M.R., Johnson, D.S. and Stockmeyer, L. (1976) 'Some simplified np-complete graph problems', *Theoretical Computer Science*, Vol. 1, No. 3, pp.237–267.

González, M.Á., Rodriguez Vela, C. and Varela, R. (2013) 'An efficient memetic algorithm for the flexible job shop with setup times', in *23rd International Conference on Automated Planning and Scheduling*, pp.91–99.

Holland, J. (1975) *Adaptation in Natural and Artificial System*, 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', *Operations-Research-Spektrum*, Vol. 15, No. 4, pp.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*, IEEE, Vol. 2, pp.1205–1210.

Kacem, I., Hammadi, S. and Borne, P. (2002) 'Approach by localization and multi-objective evolutionary optimization for flexible job-shop scheduling problems', *Syst. IEEE Syst. Man Cybern.*, Vol. 32, No. 1, pp.1–13.

Kafafy, A., Bonnevay, S. and Bounekkar, A. (2013) 'A hybrid evolutionary approach with search strategy adaptation for mutiobjective optimization', in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ACM, pp.631–638.

Kim, S.C. and Bobrowski, P.M. (1994) 'Impact of sequence-dependent setup time on job shop scheduling performance', *The International Journal of Production Research*, Vol. 32, No. 7, pp.1503–1520.

Leon, M. and Xiong, N. (2015) 'Greedy adaptation of control parameters in differential evolution for global optimization problems', in *2015 IEEE Congress on Evolutionary Computation (CEC)*, pp.385–392.

Mavrovouniotis, M., Neri, F. and Yang, S. (2015) 'An adaptive local search algorithm for real-valued dynamic optimization', in *2015 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, pp.1388–1395.

Mousakhani, M. (2013) 'Sequence-dependent setup time flexible job shop scheduling problem to minimise total tardiness', *International Journal of Production Research*, Vol. 51, No. 12, pp.3476–3487.

Nuijten, W.P. and Aarts, E.H. (1996) 'A computational study of constraint satisfaction for multiple capacitated job shop scheduling', *European Journal of Operational Research*, Vol. 90, No. 2, pp.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 *COPLAS 2011 Proceedings of the Workshop on 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', *Computers & Operations Research*, Vol. 35, No. 10, pp.3202–3212.

Riley, M., Mei, Y. and Zhang, M. (2016) 'Improving job shop dispatching rules via terminal weighting and adaptive mutation in genetic programming', *In Evolutionary Computation (CEC), 2016 IEEE Congress on*, pp.3362–3369, IEEE.

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

Turkyllmaz, A. and Bulkan, S. (2014) 'A hybrid algorithm for total tardiness minimisation in flexible job shop: genetic algorithm with parallel vns execution', *International Journal of Production Research*, Vol. 53, No. 6, pp.1832–1848.

Voglis, C., Hadjidoukas, P.E., Parsopoulos, K.E., Papageorgiou, D.G. and Lagaris, I.E. (2013) 'Adaptive memetic particle swarm optimization with variable local search pool size', in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ACM, pp.113–120.

Wright, T.P. (1936) 'Factors affecting the cost of airplanes', *J. Aeronautical Science*, Vol. 3, No. 2, pp.122–128.

Yelle, L.E. (1979) 'The learning curve: historical review and comprehensive survey', *Decision Sciences*, Vol. 10, No. 2, pp.302–328.

Yu, Y., Ma, H. and Zhang, M. (2013) 'An adaptive genetic programming approach to QoS-aware web services composition', in *2013 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, pp.1740–1747.

Zhang, G., Gao, L. and Shi, Y. (2011) 'An effective genetic algorithm for the flexible job-shop scheduling problem', *Expert Syst. Appl.*, Vol. 38, No. 4, pp.3563–3573.

Ziaee, M. (2014) 'A heuristic algorithm for solving flexible job shop scheduling problem', *Int. Adv. Manuf. Technol.*, Vol. 71, Nos. 1–4, p.519.