An Adaptive Pursuit Genetic Algorithm for Solving Job-Shop Scheduling Problems

Guilherme S. Ferreira[1] and Heder S. Bernardino[2]

[1] Universidade Federal de Juiz de Fora (UFJF)
ferreira.guilherme@gmail.com
[2] Universidade Federal de Juiz de Fora (UFJF)
heder@ice.ufjf.br
WWW home page: http://www.hedersb.uk.to

Abstract. When a metaheuristic is used for solving Job-Shop Scheduling Problems (JSPs), ones need to select the correct movement operators and their parameters to improve the results for them. However, to make a correct setup of a problem is a hard work and it is problem-dependent. In this work, we propose the use of an Adaptive Genetic Algorithm (AGA) to automatically control the techniques contained in its framework while it is solving the problem. An Adaptive Pursuit Method with Extreme Credit Assignment is used to select the movement operators and their rates (crossover and mutation techniques), and it is used to select the Local Search rate. The algorithm is tested using instances provided by a well-known generator for JSPs. The results show superior performance and reliability when compared to a standard genetic algorithm.

Keywords: genetic algorithms, adaptive operator selection, adaptive parameter control, adaptive pursuit, job-shop scheduling

1 Introduction

Several operations research, production management, and computer engineering problems can be found in the literature and practical situations. One type of them is Job-Shop scheduling problem (JSP), which is a widely studied class of the combinatorial optimization problems classified as NP-Hard. Problems with a reduced number of jobs and/or machines, called small size JSPs, are solved using exact methods such as Branch & Bound algorithms [5] and linear programming algorithms [22]. However, these methods are inefficient to solve larger or more complex problems [20, 21]. Thus, a variety of metaheuristics techniques have been proposed to reach near-optimal solutions in a reasonable amount of processing time.

According to [6, 31], Evolutionary Algorithms (EA), more specifically Genetic Algorithms (GA), are the most chosen metaheuristics to solve JSPs, frequently applied in a hybrid way (Hybrid Algorithms) with Local Search (LS). Thus, this work was guided by those studies to select the search techniques.

However, the selection of suitable parameters for a GA can be a hard task, as they are problem-dependent. In fact, right parameter settings are usually obtained by trial and error process. An Adaptive Genetic Algorithm (AGA) for solving JSPs is proposed here to avoid these problems. The considered JSPs have identical parallel machines with the objective of reducing the makespan. The use of this type of algorithm is a trend in the evolutionary computation as they automatically choose the movement operators (such as crossover and mutation) and their parameters, reducing the time of setup and test of an algorithm.

According to [7] the approaches to set up the parameters can be classified as:

1. Parameter tuning - A person or a computer program makes tests to select the parameters values. After that, they run the EA with the most appropriate parameter found, a standard EA.
2. Dynamic control - the parameters are changed by some deterministic rule, which changes the parameter values without any feedback from the EA. Generally, these methods are focused on the evolutionary phases of an EA.
3. Adaptive control - In this context, there is some feedback from an EA, which is used to change the component or the parameter. An EA may use the Credit Assignment to define which operator or parameter will be used in the next step of an evolutionary process.
4. Self-adaptive control - The parameters are encoded on a chromosome and are susceptible to a crossover and mutation movements. This segment is not considered to evaluate the fitness of an individual. The main idea is that the best values produce best individuals and the elitism makes this propagate throughout the generations.

AGAs have already been established in the JSP in previous work. The approach proposed in [19] receives information of the evolutionary stage, classified as dynamic control, and different to the adaptive idea proposed here.

The next researches do not follow the adaptive methods of the literature such as Probability Matching or Adaptive Pursuit. These authors use simple adaptive methods. In [3], the crossover and mutation rates increases or decreases in fixed steps, according to the improvements of the offspring. A multi-objective JSP was presented in [12], even though Hongyan and Hong called it self-adaptive, the previously mentioned concept used here, calls it adaptive. The adaptation happens only in crossover and mutation rates, and the success of an operator is evaluated based on the average fitness of the population. A similar approach was proposed in [27], where the rates of the movements operators are changed based on the average, the maximum and the minimum fitness of the individuals.

The work of [15] proposed an adaptive selection for a cellular GA, which improves the probability of choosing the best individuals. In another study, Nalepa et al. [17] introduced a new method that sometimes generates offsprings of the same pair of parents, but differing about the applied crossover technique, and sometimes generates multiple children, applying the same crossover technique. Finally, in [26] a multi-population algorithm with adaptation in crossover and mutation rates was implemented. According to the authors, the algorithm was compared with other metaheuristics using many benchmark instances, and it demonstrated a reasonable efficiency, not being worse or better than the other techniques. A hybrid multi-objective backtracking search algorithm for solving scheduling problems, which uses adaptation to select an appropriate population to keep diversity, was developed in [16].

Lastly, these last ones are closer to our approach. Yan and Wu [29] used an adaptive procedure to select between movements operators. The approach used in that work is a roulette wheel based on the reward of the operators. The reward was tested with and without a time window. In that article, four crossover and four mutation techniques were implanted and the proposed outperformed conventional GA algorithms. An adaptive multimeme algorithm was proposed to solve the JSP in [30]. That work has a Multi-Armed Bandit technique integrating a stochastic variation and a local search procedure into a composite operator for each individual. The framework contains three LS techniques.

In this work, it was proposed an AGA for solving JSP to select between GA techniques (and their parameters) and an LS rate. Also, there is a constructive heuristic used to generate the initial population, if necessary it may be applied to improve the results. The AGA selects the parameters according to the feedback received in the evolutionary process, whereas in work of [19], the parameters are adjusted according to the number of generations, without any feedback. This proposal differs from the work of [15] as our proposal does not need a neighborhood structure and it selects the parameters (crossover and mutation), while that approach the adaptation scheme only happens in the selection mechanism. Finally, the work of [29] does not use a previously established method for adaptive evolutionary algorithms, and this proposal uses the Adaptive Pursuit Method proposed in [25]. So, Section 2 defines the studied problem. Section 3 explains the search techniques used here. Section 4 introduces the proposed algorithm. Finally, Section 5 and Section 6 show the results of our computational experiments and the conclusion about this work.

2 Job-Shop Scheduling Problems

The shop scheduling is a practical and relevant problem in several applications areas, such as flexible manufacturing system, production planning, logistics, communication and others. The problem consists in searching the best order to process a set of $\mathcal{N}$ jobs in $\mathcal{M}$ machines, and the processing of each job in each machine is also called operation.

The number of jobs, machines and constraint rules that indicate how the operations should be scheduled, defines the types of shop-scheduling, like Single Machine Shop (SMP), Parallel Machine Shop (PMP), Flow-Shop (FSP), Job-Shop (JSP), and Open-Shop (OSP) [10]. In the SMP, there is only a single machine, and each job has only one operation. The decision-maker has to find the optimal sequence of the jobs. The PMP allows the use of some machines working in parallel, doing the same work, where the jobs can be processed in all of these machines. In the FSP, the jobs have the same technological route, that is, the same machine order. The problem has $\mathcal{M}$ machines, and it is also possible to have parallel machines. The JSP, which is the focus of this study, is similar to the FSP, but the technological route may be different for each job. Finally, the OSP has no order constraints concerning the operations in each job.

As previously mentioned, in the context of JSP, the jobs do not have the same technological route and also have a predefined processing time. For standard problems, the number of operations is the same of the number of machines, but it can be smaller or greater (recirculation). Here we only considered the classical problems. In JSP, the best order to process the $\mathcal{N}$ jobs in the $\mathcal{M}$ machines is desired to achieve better results for the stipulated indicators. Formally, JSPs can be defined as [21]:

– each job $i \in \mathcal{N}$ can be processed by a machine $j \in \mathcal{M}$.
– the processing of the job $\mathcal{N}_i$ in the machine $\mathcal{M}_j$ is called operation $\mathcal{O}_{ij}$.
– operation $\mathcal{O}_{ij}$ requires exclusive use of the machine $\mathcal{M}_j$ by the time t_{ij}, called processing time.
– each job i has a technological route (operating sequence of x_i operations).
– $\mathcal{O}_{ij}$ can be processed by only one machine j at a time.
– each operation, after started, must be finished in the same machine and should not be interrupted until it is concluded.
– each machine runs one operation after another.

As indicated, rework or recirculation is not allowed. The objective of the JSP here is to reduce the makespan, defined as $C_{\max} = max_{i \in \mathcal{N}}\{f_i\}$, where f_i is the flowtime of the job i.

3 Search Techniques

There are different kinds of metaheuristics for solving optimization problems, and here GA, LS and the NEH heuristic are adopted. Thus, these techniques are detailed in sequence.

3.1 Genetic Algorithms

GA was proposed by Holland [11] and it is a search technique inspired by the evolution paradigm. One of its advantages concerning the other search techniques is since GA can easily apply other techniques on its framework [20], such as LS and others.

One way to represent the JSP is using a permutation coded-GA. In this work, we adopt the Job-based representation which was recommended in [13], wherein a chromosome is a vector with an equal length of the number of the jobs, and it represents the order of the jobs. This representation is illustrated by Figure 1.

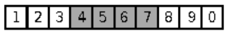

Fig. 1. Job-Based Representation

For solving JSPs, a large number of genetic operators have already been developed, and we adopted five available in [28], namely,

- The order-preserving one-point crossover sets up a cut point in the first parent and copies the genes until it. Then, the empty genes are filled with the other missing jobs, keeping the relative order of the second parent.
- The LOX (linear order crossover) is a procedure similar to order-preserving one-point crossover. First of all, it sets up two cut points and copies the genes between it to the same gene location in the offspring. The missing jobs in the offspring are filled with the relative order of the second father.
- The PMX (partially mapped crossover) selects two cut points, and the proto-offspring receives these genes from the first parent. The other jobs are taken from the second parent. The generated proto-offspring often has repeated jobs, and to turn it into a feasible offspring, the jobs that appear two times are replaced outside of the segment. To replace the jobs duplicity, they are mapped from parent 1 to parent 2, as illustrated in Figure 2. In the proto-offspring, the jobs numbered as 7 and 5 are duplicated. There a duplicity in the job 7. To solve this the job 7 was mapped to 5, and it results in another repetition. Then, we need one more step to solve that. For this, the job 5 was mapped to job 2. The other repetition is easier to solve, the job 4 was mapped to job 8, and finally, the offspring was generated without repetition.
- The Shift Mutation takes a job randomly and changes it to an arbitrary position.
- The Pairwise Interchange Neighborhood takes two jobs randomly and switch their positions.

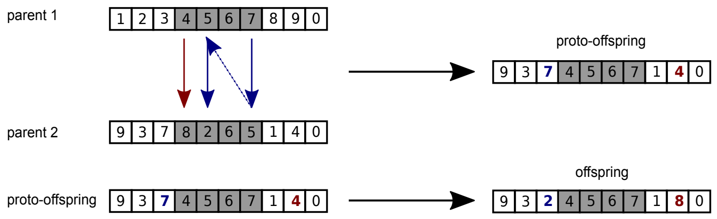

Fig. 2. PMX crossover

Although GAs can find promising regions in the solution space, they are not suitable to fine-tuning [8]. The algorithm is often applied with other search techniques. In our algorithm, it is possible to employ a constructive heuristic to generate some individuals and an LS in each generation.

3.2 Adaptive Genetic Algorithms

AGAs are a variant of a GAs that tries to adjust the algorithm during the search process. In this work, the implemented AGA is defined as adaptive dynamic. This algorithm requires components to evaluate the used components called by Credit Assignment, and components to select the operators and parameters on its framework, named respectively as Operator Selection or Parameter Selection [14].

The Credit Assignment compares the results of the application of the components and assigns an approximation of their quality. When an AGA is used for solving a mono-objective optimization problem, the fitness improvement is often applied to compare the differences of the components. The credit can be estimated using one of the following ideas [1, 14]:

1. Instantaneous reward – received after the last use of an operator/parameter.
2. Average reward – considers the average quality of an operator or parameter, received after few applications.
3. Extreme reward – considers the best quality received after a few applications of an operator or a parameter.
4. Learned quality attribution – whereas the quality values of operator or parameter are inferred using Machine Learning and other forecast techniques.

The literature provides some schemes for operator or parameter selection. However, the Adaptive Pursuit Method (AP) introduced in [25] is used here. This method was inspired by the field of learning automata, a type of Machine Learning algorithm, and belongs to a class of methods that fast reaches good solutions. In a non-stationary environment, we need to change swiftly to any change of the performance of the operators, and then AP quickly increases the performance of the best operator when compared to the traditional Probability Matching. The method calculates the probability for each operator and uses a roulette wheel to select one of them. All components, operators and parameters, have a lower bound probability. This characteristic allows the future applications of these components, as they can become more effective during the search process. Defining t as the current step, AP evaluates the quality of an operator as $\mathcal{Q}_a(t+1) = \mathcal{Q}_a(t) + \alpha[\mathcal{R}_a(t) - \mathcal{Q}_a(t)]$, where $\mathcal{R}_a(t)$ is the credit assignment of a operator. The probability of selecting an operator can be defined as:

$$\mathcal{P}_a(t+1) = \begin{cases} \mathcal{P}_a(t) + \beta[P_{\max} - \mathcal{P}_a(t)] & \text{if } a = a^*, \\ \mathcal{P}_a(t) + \beta[P_{\min} - \mathcal{P}_a(t)] & \text{otherwise} \end{cases} . \tag{1}$$

where K is the number of components, $P_{\min}$ and $P_{\max} = [1 - (K-1)P_{\min}$ are, respectively, the minimum and maximum probability values, and $a^* = max_a[\mathcal{Q}_a(t+1)]$ is the best operator at $t+1$.

In this work, we apply the Extreme Credit Assignment (EX) suggested in [9]. This method applies a time window W that keeps the last information of a component and uses the maximum success to evaluate the credit assignment for it. Defining $\delta(t)$ the fitness improvement at time t, then the reward for a operator is computed as $\mathcal{R}_a(t) = max\{\delta(t_i), i = 1 \cdots W\}$.

3.3 Constructive Heuristic

The NEH heuristic was applied to initialize some individuals of the initial population. Proposed by Nawaz et al. [18], this heuristic has the following steps:

1. For each job, one has to calculate the sum of the duration of all operations.

2. The jobs are sorted in descending order of its total processing time.
3. The first two jobs in the sorted sequence are selected and evaluated considering the makespan for all the possible sequences. The best order must be set, and its relative sequence is stored until the end of the algorithm.
4. After that, the next job in the list of Step 2 is taken, and it is required to find the best sequence placing it at all possible positions in the partial sequence found in the previous step. The best sequence is chosen and the relative order is kept.
5. The Step 4 is performed until all the jobs are allocated.

3.4 Local Search

The local search adopted here, proposed in [23], has two phases: the first one is a destructive phase, and the last one is a constructive phase. In the first phase, d jobs are randomly removed without repetition from an individual. These d jobs are stored in a list in the same order that they were removed. At this point, two sequences are generated, and the destructive phase ends. Then, the constructive phase uses the Steps 4-5 of the NEH heuristic, described in Section 3.3.

4 The Proposed Adaptive Genetic Algorithm

The AGA proposed here selects the movement operators (crossover and mutation), their parameters (such as crossover probability and mutation rate), and the parameter of the adopted LS. These components are automatically chosen in the evolutionary process and applied aiming to improve the results. Then, the quality and the probability of the selected operators and parameters are updated. The Credit Assignment is based on fitness differences between the best parent and the best-generated individual, in case of crossover, and based on the average fitness of the population and the generated individual, in the case of mutation and LS. The success rate of each operator is updated after the generation of each pair of individuals, and here we use the AP with EX, as previously explained. A binary tournament is used to select the parents. Finally, the replacement mechanism keeps the best individual of the generation g, discarding the worse created in the generation $g + 1$. A pseudo-code is presented in Algorithm 1.

5 Computational Experiments

A program was implemented using the C programming language[3]. A total of 125 instances with five different problems sizes, namely, 10×10, 20×10, 20×20, 50×10 and 50×50 (jobs $\times$ machines). The instances were generated by the process suggested by Taillard [24]. The stopping criterion was defined as the number of objective function evaluations equals to $(800 + 10\mathcal{N})\mathcal{N}^2$, as in [2], where $\mathcal{N}$ is the number of jobs.

[3] The source code is available at http://bitbucket.org/ciml-ufjf/ciml-lib

Algorithm 1: The Proposed Adaptive Genetic Algorithm for Job-Shop Scheduling

```
input : $NP$ (population size), $EVA$ (number of evaluations)
CreateInitialPopulation (NP);
for $i \leftarrow 1$ to $NP$ do
    Evaluate $f(\boldsymbol{x_{i,G}})$;   /* $\boldsymbol{x_i}$ is an individual in the population */
for $E \leftarrow 1$ to $EVA$ do
    for $i \leftarrow 1$ to $NP$ do
        crossover.op ← crossover-selectOperator();
        crossover.rt ← crossover-selectRate();
        $\boldsymbol{u}$ =
          crossover-generate-two-individuals($NP, \boldsymbol{x_G}, crossover.op, crossover.rt$);
        mutation.op ← mutation-selectOperator();
        mutation.rt ← mutation-selectRate();
        $u_1$ = mutate-first-individual($u_1, mutation.op, mutation.rt$);
        mutation.op ← mutation-selectOperator();
        mutation.rt ← mutation-selectRate();
        $u_2$ = mutate-second-individual($u_2, mutation.op, mutation.rt$);
        for $j \leftarrow 1$ to 2 do
            Evaluate($u_j$);
        localsearch.rt ← localsearch-selectRate();
    SelectPopulation($\boldsymbol{x_G}, \boldsymbol{u}$);
    AP-ApplyReward($\boldsymbol{x_G}, \boldsymbol{u}$);
```

The instances were divided into two groups. The first group is called learning phase. It has 25 instances, five for each problem size. In the learning phase, each algorithm was executed twice. The second is called the testing phase. This group has 100 instances, 20 for each problem size, and each algorithm ran five times.

The Table 1 presents the tested parameters, where P_S, C_R, M_R, LS_R are the population size, the crossover rate, the mutation rate, and the LS rate, respectively. The combination of these parameters with the movements operators, and with the possibility to use or not the NEH heuristic, generates 972 possibilities to set up a GA. The number of jobs removed in the LS destruction phase was set 10% of the number of jobs. All of these were submitted at the learning phase, and two setups for a GA are stored, the best to solve all problems and the best to solve problems with 10 jobs and 10 machines.

Table 1. Values allowed to be used for each GA parameter

P_S	C_R	M_R	LS_R
50	0.7	0.30	0.015
76	0.8	0.40	0.030
100	0.9	0.50	0.045

AGA automatically selects the movement operators and its parameters, but it also has four parameters described in Section 3.2. Here, we defined 30 parameter options to set up this algorithm. The values for α and β tested here are $[(2,2),(2,8),(5,5),(8,2),(8,8)]$. Also, the minimum probability $P_{\min}$ was set to 0.05. Thus, in average, one individual will be created with the worse settings in each generation. The population sizes are the same ones used in GA, and the same length was applied in the time window. Also, the parameter of the destruction phase of the LS was set 10%, as in GA. The AGA has in its framework the movement operators described in Section 3.1, and can choose the same probability or rate showed in Table 1. As in GA, we stored the best setup to solve all the problems and the best setup to solve the small problems when using AGA.

This work uses Performance Profiles (PPs) to compare the performance of the algorithms. Suggested in [4] to compare evolutionary computation techniques, they are an analytical resource for the visualization of the results of numerical experiments. Considering the set TA of problems instances ta_i, with $i = 1, 2, \cdots, n_{ta}$, a set A of algorithms a_j, with $j = 1, 2, \cdots, n_a$, and $C_{ta,a} > 0$ the makespan, the performance ratio can be defined as: $r_{ta,a} = \dfrac{C_{ta,a}}{min\{C_{ta,a} : a \in A\}}$. Thus, the performance profile of the algorithm is defined as: $\rho(\tau) = \dfrac{1}{n_{ta}}|\{ta \in TA : r_{ta,a} \leq \tau\}|$, where $\rho(\tau)$ is the probability that the performance ratio $r_{ta,a}$ of algorithm a is within a factor $\tau \geq 1$ of the best possible ratio. The algorithm with larger $\rho(\tau)$ are preferred.

Two analyses were performed to evaluate the proposed algorithm. In the first, we selected the best parameter settings provided during the learning phase and execute it on the test instances. Figure 3 presents the results of this analysis, where LOX_IN_NEH represents GA, and APEX represents the proposed AGA. One can observe the best-obtained parameter settings for GA was: LOX crossover, Interchange Mutation, adopting NEH heuristic to generate some individuals in the initial population. The best observed parameter values are 50, 0.7, 0.3 and 0.015, for P_{S}, C_{R}, M_{R}, and LS_{R}, respectively. For AGA, α and β are equal to 2, the NEH heuristic is used, and $P_{\mathrm{S}} = 50$, as in GA.

One can see that AGA is better than or equal to GA in approximately 70% of the test instances. In this case, the AGA can solve all instances within a factor 1.01 worse than those obtained by GA, showing a better reliability.

For a second analysis, we solved all the small problems of the learning instances, that is, 10 jobs and 10 machines. Then we select the best parameter settings, and we applied then to all test instances. This approach was selected due to the fact the small problems are solved faster and can reduce the amount of time spent to evaluate the possible operators and their parameter values. The results are presented in Figure 4. The best AGA algorithm was set with α and β equals 5, $P_{\mathrm{S}} = 100$ and did not use NEH. The best GA was using PMX crossover, Shift Mutation, and with NEH heuristic. The parameters selected were $P_{\mathrm{S}} = 76$, $C_{\mathrm{R}} = 0.9$, $M_{\mathrm{R}} = 0.5$, and $LS_{\mathrm{R}} = 0.045$.

We can see that AGA in Figure 4 outperformed GA in almost all instances. Also, the reliability of the algorithm is greater, as it can solve all the problems

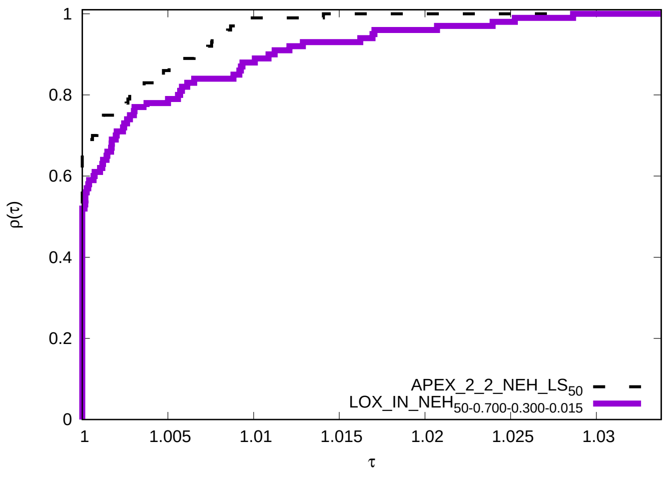

Fig. 3. PPs of GA and the proposed AGA for all instances. The parameters and operators are the best ones found when all instances (learning phase) are considered.

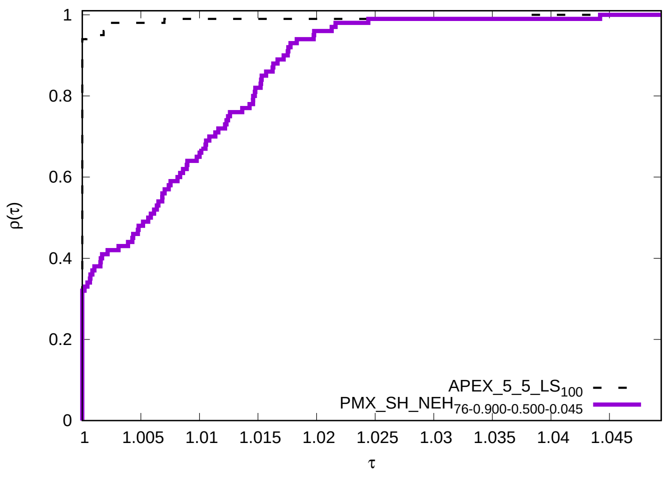

Fig. 4. PPs of GA and the proposed AGA for all instances. The parameters and operators are the best ones found when 10x10 instances (learning phase) are considered.

within an approximated factor of 1.025 from the best performance, and GA only can solve within a factor near 1.045.

6 Concluding Remarks and Future Works

In this paper, we proposed the use of an existing adaptive method applied to a GA for solving JSPs. The fitness values of the offspring are compared to those from their parents to identify the capacity of a crossover operator in generating better individuals. On the other hand, the fitness of the offspring is compared to the average performance of the current population for the mutation and LS procedures. Also, the algorithm uses AP with EX to selects between the movement operators from GA (and their parameters) and the LS rate, for the generation of each new individual.

The proposed algorithm was compared to a GA. AGA performed better than GA when: setting the best parameters for all problems; and setting the best parameters for small size problems, i.e., 10 jobs and 10 machines, and using these setups to solve the problems of all sizes. The results confirmed that the AGA is reliable to solve JSP problems.

For further works, Probability Matching or Dynamic Multi-Armed Bandit can be used aiming to improve the quality of the adaptive scheme. Also, the problem would be worked as a multi-objective optimization, or it would be introduced setup time, for example, turning the JSP problem more complex.

Acknowledgement

The authors thank the financial support provided by CAPES and FAPEMIG.

References

1. Aleti, Aldeida, Moser, I: A Systematic Literature Review of Adaptive Parameter Control Methods for Evolutionary Algorithms (2015)
2. Arroyo, J.E.C., de Souza Pereira, A.A.: A grasp heuristic for the multi-objective permutation flowshop scheduling problem. The International Journal of Advanced Manufacturing Technology 55(5), 741–753 (2011)
3. Asokan, P., Jerald, J., Arunachalam, S., Page, T.: Application of adaptive genetic algorithm and particle swarm optimisation in scheduling of jobs and AS/RS in FMS. International Journal of Manufacturing Research 3(4), 393–405 (2008)
4. Barbosa, H.J., Bernardino, H.S., Barreto, A.M.: Using performance profiles to analyze the results of the 2006 cec constrained optimization competition. In: Evolutionary Computation (CEC), 2010 IEEE Congress on. pp. 1–8. IEEE (2010)
5. Brucker, P., Jurisch, B., Sievers, B.: A branch and bound algorithm for the job-shop scheduling problem. Discrete applied mathematics 49(1-3), 107–127 (1994)
6. Chaudhry, I.A., Khan, A.A.: A research survey: review of flexible job shop scheduling techniques. Int. Trans. in Operational Research 23(3), 551–591 (May 2016)
7. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on evolutionary computation 3(2), 124–141 (1999)
8. Eiben, A.E., Smith, J.E., et al.: Introduction to evolutionary computing, vol. 53. Springer (2003)
9. Fialho, A., Da Costa, L., Schoenauer, M., Sebag, M.: Extreme value based adaptive operator selection. In: International Conference on Parallel Problem Solving from Nature. pp. 175–184. Springer (2008)
10. Hart, E., Ross, P., Corne, D.: Evolutionary scheduling: A review. Genetic Programming and Evolvable Machines 6(2), 191–220 (2005)
11. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992)

12. Hongyan, X., Hong, H.: Research on job-shop scheduling problem based on Self-Adaptation Genetic Algorithm. In: Int. Conf. Logistics Syst. and Intell. Manage. vol. 2, pp. 940–943. IEEE (2010)
13. Jorapur, V., Puranik, V.S., Deshpande, A.S., Sharma, M.R.: Comparative Study of Different Representations in Genetic Algorithms for Job Shop Scheduling Problem. Journal of Software Engineering and Applications 07(07), 571–580 (2014)
14. Krempser, E., Fialho, A., Barbosa, H.: Adaptive operator selection at the hyper-level. Parallel Problem Solving from Nature-PPSN XII pp. 378–387 (2012)
15. Li, Z.M., Zhang, Y., Zheng, X.D., Wan, X.Y.: Solving the Problem of General Job Shop Problem by Using Improved Cellular Genetic Algorithm. Advanced Materials Research 945-949, 3130–3135 (Jun 2014)
16. Lu, C., Gao, L., Li, X., Pan, Q., Wang, Q.: Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. Journal of Cleaner Production 144, 228–238 (Feb 2017)
17. Nalepa, J., Cwiek, M., Kawulok, M.: Adaptive memetic algorithm for the job shop scheduling problem. In: Int. Joint Conf. of Neural Networks (IJCNN). pp. 1–8. IEEE (2015)
18. Nawaz, M., Enscore, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 11(1), 91–95 (1983)
19. Pan, Y., Zhang, W.X., Gao, T.Y., Ma, Q.Y., Xue, D.J.: An adaptive Genetic Algorithm for the Flexible Job-shop Scheduling Problem. In: Int. Conf. of Comput. Sci. and Automation Eng. (CSAE). vol. 4, pp. 405–409. IEEE (2011)
20. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35(10), 3202–3212 (2008)
21. Ripon, K.S.N., Tsang, C.H., Kwong, S.: An Evolutionary Approach for Solving the Multi-Objective Job-Shop Scheduling Problem. In: Evolutionary Scheduling, vol. 49, pp. 165–195. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
22. Roshanaei, V., Azab, A., ElMaraghy, H.: Mathematical modelling and a meta-heuristic for flexible job shop scheduling. Int. J. of Prod. Research 51(20), 6247–6274 (2013)
23. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (Mar 2007)
24. Taillard, E.: Benchmarks for basic scheduling problems. european journal of operational research 64(2), 278–285 (1993)
25. Thierens, D.: An adaptive pursuit strategy for allocating operator probabilities. In: Genetic and Evolutionary Computation Conference. pp. 1539–1546. ACM (2005)
26. Wang, L., Cai, J.C., Li, M.: An adaptive multi-population genetic algorithm for job-shop scheduling problem. Advances in Manufacturing 4(2), 142–149 (Jun 2016)
27. Wang, L., Tang, D.b.: An improved adaptive genetic algorithm based on hormone modulation mechanism for job-shop scheduling problem. Expert Systems with Applications 38(6), 7243–7250 (Jun 2011)
28. Werner, F.: Genetic algorithms for shop scheduling problems: a survey. Preprint 11, 31 (2011)
29. Yan, J., Wu, X.: A genetic based hyper-heuristic algorithm for the job shop scheduling problem. In: Int. Conf. Intell. Human-Machine Syst. and Cybern. (IHMSC). vol. 1, pp. 161–164. IEEE (2015)
30. Zuo, Y., Gong, M., Jiao, L.: Adaptive multimeme algorithm for flexible job shop scheduling problem. Natural Computing pp. 1–22 (2016)
31. Çaliş, B., Bulkan, S.: A research survey: review of AI solution strategies of job shop scheduling problem. J. of Intelligent Manufacturing 26(5), 961–973 (2015)