A Genetic Programming Framework for Heuristic Generation for the Job-Shop Scheduling Problem

E. Lara-Cárdenas$^{[0000-0002-6357-4680]}$, X. Sánchez-Díaz$^{[0000-0003-2271-439X]}$, I. Amaya$^{[0000-0002-8821-7137]}$, J. M. Cruz-Duarte$^{[0000-0003-4494-7864]}$, and J. C. Ortiz-Bayliss$^{[0000-0003-3408-2166]}$

Tecnologico de Monterrey, School of Engineering and Sciences
Av. Eugenio Garza Sada 2501, Monterrey, NL 64849, Mexico
a00398510@itesm.mx, sax@tec.mx, iamaya2@tec.mx, jorge.cruz@tec.mx, jcobayliss@tec.mx

Abstract. The Job-Shop Scheduling problem is a combinatorial optimization problem present in many real-world applications. It has been tackled with a colorful palette of techniques from different paradigms. Particularly, hyper-heuristics have attracted the attention of researchers due to their promising results in various optimization scenarios, including job-shop scheduling. In this study, we describe a Genetic-Programming-based Hyper-heuristic approach for automatically producing heuristics (dispatching rules) when solving such a problem. To do so, we consider a set of features that characterize the jobs within a scheduling instance. By using these features and a set of mathematical functions that create interactions between such features, we facilitate the construction of new heuristics. We present empirical evidence that heuristics produced by our approach are competitive. This conclusion arises from comparing the makespan of schedules obtained from our proposed method against those of some standard heuristics, over a set of synthetic Job-Shop Scheduling problem instances.

Keywords: Hyper-heuristics · Job-Shop Scheduling · Genetic Programming

1 Introduction

Combinatorial optimization problems are widespread in everyday processes, liaised with both academic and industrial applications. A particular example of such combinatorial optimization problems resides in Job-Shop Scheduling (JSS), where the solving process requires to assign a set of tasks within jobs to a set of machines in such a way that they minimize the makespan (completion time). A JSS problem is inherent to any manufacturing process where it is imperative to schedule an optimal production plan.

Several variations of JSS have risen throughout the years [12, 30]. Likewise, different methods for solving them have appeared in the literature since 1956 [35].

For example, some authors have tackled the Dynamic Flexible JSS problem where there is uncertainty in processing times [22, 34, 36]. Others have considered the Deterministic No-Wait JSS version, whose primary goal is to find a schedule that minimizes the makespan [3, 23, 24]. Since the search space of such problems is usually huge, exhaustive exploration is impractical. Hence, solving JSS problems usually relies on approximation techniques. Some illustrative examples include the dispatching rules proposed by Blackstone *et al.* [17], and the *shifting bottleneck* procedure employed by Adams *et al.* [1]. Such dispatching rules refer to low-level heuristics that can either construct or modify a schedule. The main advantage of these approaches is their low computational cost, which means they deliver quick solutions. Nonetheless, their inability to guarantee optimality emerges as a side-effect. Thus, heuristics are usually applied in practice to solve hard combinatorial optimization problems, including the JSS problem [27].

One way of surmounting the optimality drawback is to incorporate more robust search techniques. There are several works following this path, including strategies such as Tabu Search [13, 26], and Guided Local Search with Branch-and-Bound [2]. However, there are also approaches based on Genetic Algorithms [19, 32], Genetic Search [31], and Genetic Programming [25]. Of course, these strategies also include hybrids [33] and other approaches [8, 18, 28]. Moreover, there is another strategy that emerged recently: Hyper-Heuristics (HHs) [9]. A HH extends the ideas proposed by Fisher and Thompson [14] and Crowston [10] in the early 1960s: a combination of priority dispatching rules should perform better than any of the rules in isolation. HHs represent a particular application where the idea is to automate the design or selection of the available heuristics. According to Burke *et al.* [5], HHs can be classified into two main categories: methodologies that select from a fixed set of heuristics and those that generate new heuristics. The former produces a mapping between the states of the problem and a feasible heuristic. Otherwise, generation HHs identify critical parts of existing heuristics to create new ones [6, 16].

Particularly, the literature contains some interesting works within the intersection of hyper-heuristics and the JSS problem. For example, Chaurasia *et al.* proposed a HH based on evolutionary algorithms and a guided heuristic for JSS problem instances [7]. Similarly, Garza-Santisteban *et al.* studied the feasibility of using the well-known Simulated Annealing to train HHs [15]. More recently, Lara-Cárdenas *et al.* implemented Neural Networks for improving the performance of existing HHs for the JSS problem [21].

In this work, we explore a novel idea for contributing to the state-of-the-art about generation HHs. Our model combines features that characterize the individual jobs within the instances, while other similar approaches from the literature reuse components from existing heuristics. The main benefit of this particularity is that we require no information from other heuristics or criteria to produce new competent ones. We tackle the JSS problem through an approach based on Genetic Programming (GP) for producing new heuristics. GP is an evolutionary algorithm that borrows ideas from the theory of natural evolution to build a program represented as a tree-like data structure [20]. The method evolves programs through three genetic operators: selection, crossover, and mutation. In GP, crossover and mutation are specifically designed to work with the tree-like data structure. This algorithm relies on an objective function to evaluate the programs, and then, the programs with the best objective values are more likely to survive to future generations. Thus, our hyper-heuristic model produces heuristics (or dispatching rules) by combining the features that characterize the jobs in a JSS problem. Merging this material allows to encapsulate significant human-derived expertise to reuse for improving performance. Our experiments offer empirical evidence that the newly-produced heuristics can be specialized for specific groups of instances, and that they outperform the best standard heuristics. Although our model is tested on JSS problems, it may be applied to other problem domains with ease, as long as they rely on heuristics that guide the search.

The remainder of this document is organized as follows. Section 2 presents the basic concepts related to this work. Subsequently, Section 3 describes the proposed approach and how it produces heuristics for the JSS problem. Section 4 discusses the experiments and main results achieved with such an approach. Finally, Section 5 remarks the most relevant conclusions and some future research directions.

2 Background

A Job-Shop Scheduling (JSS) problem is described by a set of jobs and a set of machines. Each job contains a list of tasks that must be processed in a specific order. Solving the JSS problem requires that the machines handle all the tasks in each job, in their corresponding order. Each task has a processing time on a particular machine since not all the machines can process all tasks. A feasible schedule must then satisfy that tasks of all jobs have been assigned to one suitable machine in a valid order of execution. The time needed to complete such a schedule is known as the *makespan*. All the solving methods considered in this work focus on yielding schedules that minimize such a value.

In the following lines, we briefly describe some relevant concepts related to this study: the available heuristics, the instances we used, the features to characterize the jobs within such instances, and the performance metrics for evaluating the methods under analysis.

2.1 Heuristics

For this work, we select heuristics based on reports in the same area [15,21]. Thus, we only consider those that build a solution from scratch by making one decision per step (*i.e.*, constructive heuristics). For the JSS problem, these heuristics decide which job to process next among all the available options. Such a decision is made at each stage of the search. Withal, we describe the selected heuristics as follows:

Shortest Processing Time (SPT) picks the activity with the shortest processing time, from the available activities to schedule.

Longest Processing Time (LPT) chooses the activity with the longest processing time, from the available activities yet to schedule.

Maximum Job Remaining Time (MRT) first takes the job that requires the most time to finish (*i.e.*, the one with the largest sum of the processing times of its activities yet to be scheduled). Then, it returns the first possible activity (in precedence order) that corresponds to such a job.

Most Loaded Machine (MLM) takes the machine with the maximum total processing time (the one with the largest sum of the processing times of the activities that it has allocated) from the available machines. Then, it selects the activity with the shortest processing time, from the activities that can be allocated on the selected machine.

Least Loaded Machine (LLM) works similar to MLM, but it chooses the machine with the minimum total processing time from the set of machines. Then, it selects the activity with the shortest processing time from the activities that can be allocated on the selected machine.

Earliest Start Time (EST) finds the job that has the earliest possible starting time at the current problem state, by considering the available activities yet to be scheduled. Then, it picks the activity corresponding to that job.

In all cases, these heuristics break ties by using the index of the job that coincides to its belonging activity; they prioritize small values.

2.2 Instances

All the instances considered for this research were synthetically produced by using the algorithm proposed by Taillard [29]. This algorithm produces JSS problem instances with a specific number of jobs and machines, specified by the user. The jobs' processing times are generated uniformly on an interval from 1 to 99, using the random generator proposed by Bratley *et al.* [4]. For this investigation, we set the parameters of the instance generator as recommended by Taillard [29]. We produced 25 JSS problem instances, with ten machines and ten jobs. We then split those instances into training and testing sets by using 15 and 10 instances. These sets are mutually exclusive, which means that each instance appears only in one of the two sets.

2.3 Job Characterization

In this work, the job characterization relies on six features that capture the state of the jobs and allow the creation of new heuristics. Such features are described below:

APT calculates the ratio between the sum of the processing times of the processed activities of the job and that of the whole list of the activities of the job. This feature pictures the completion percentage of the scheduling process for the job.

DPT determines the ratio between the standard deviation of the processing times of the processed activities and their mean value.

SLACK obtains the ratio between the slack (available machine time), and the makespan of the current schedule.

DNPT describes the ratio between the standard deviation and the mean processing times of all the unscheduled activities for each job.

NAPT is the complement of APT. It calculates the ratio between the sum of the processing times of unscheduled activities of the job and the sum of the processing times of the complete list of activities of the job (including scheduled ones).

NJT determines the sum of the processing times normalized for each job. Then, it divides such an amount by the number of pending jobs. It only applies to the pending jobs.

2.4 Performance Metrics

To evaluate the performance of the methods studied in this work, we employ two metrics. The first one is the makespan, which represents the value of the completion time of the schedule produced by a particular method on a specific instance. Then, the smaller the makespan, the better the performance. We have mainly used the total makespan determined as the sum of all the makespans per instance on the test set. The second metric is the success rate, which estimates the relative performance of the methods under study. The success rate represents the fraction of cases where one particular process 'succeeds' against another one by reducing the makespan in a specific instance. With the success rate, the closer the value to 1, the better the performance. In other words, the success rate indicates the proportion, from the total instances, where a technique (a hyper-heuristic, in our experiments) outperforms another model on the test set.

Using these metrics, we evaluated the performance of the heuristics produced and compared them against the human-made heuristics and a synthetic Oracle H*. Hence, H* represents the best possible schedule obtained utilizing the six heuristics to solve each particular instance. Since this value cannot be known in advance, H* is infeasible in practice. However, it is useful for comparison purposes, as we show in this investigation.

3 The Hyper-heuristic Approach

This model focuses on the generation of heuristics for the JSS problem utilizing Genetic Programming (GP). The heuristics produced by our approach contain an internal function that combines the features, which characterize a job in the JSS problem, by using a set of available operations. We consider this feature combination as a high-level feature that increases the discriminative power of the individual features. When the solving process summons a heuristic, such a heuristic evaluates all the available jobs employing its internal function. Then, the heuristic schedules the next available task in the job with the smallest evaluation. After scheduling a task, the solving process requires a new call to the heuristic (with its corresponding function evaluations) before scheduling another task. This process is repeated until the instance is solved.

As mentioned before, the evolution of such heuristics (actually, their internal functions) is achieved through a GP-based approach. A tree-like structure represents a function, as the example depicted in Figure 1. We can interpret such a structure as the function $f(x_1, x_2, x_3) = x_1 + (x_2 - x_3)$. In this function, x_1, x_2, and x_3 can be features that characterize the jobs in the instance. All the features considered for this work are dynamic, then the heuristics produced are dynamic as well. In our context, being dynamic means that the values change as the solving process takes place. Thus, one internal function may return different values for the same job at different moments of the solution process.

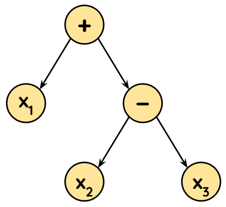

Fig. 1. A tree-like structure that encodes the function $f(x_1, x_2, x_3) = x_1 + (x_2 - x_3)$

The evolutionary process that powers the hyper-heuristic (HH) model is the component responsible for creating new heuristics by evolving their internal functions. The HH model requires three inputs: the training instances and the terminal and function sets. This model uses the training instances to evaluate the performance of the heuristics throughout the evolutionary process. The two last inputs (the terminal and function sets) are a requirement imposed by GP, which uses those sets to generate and modify the functions. In the GP's tree-like structures, leaf nodes always contain an element from the terminal set, while the rest of the nodes always contain an element from the function set. In this work, the terminal set is composed of features that characterize jobs, as described in Section 2.3 and the ephemeral constant $R \in [-1, 1]$. The function set contains six simple operations: addition (+), subtraction (−), multiplication (×), protected division (/), minimization (`min`), and maximization (`max`). All these operations take two operands each.

The evolutionary process starts with 30 randomly initialized individuals, using the ramped half-and-half method for this purpose. This process runs for 100 generations. For crossover and mutation, we considered rates of 0.9 and 0.05, respectively. Finally, for selecting the individuals for mating, we used a tournament selection of size two. At the end of the 100 generations, the model returns the individual with the smallest objective value as the resulting heuristic. At this point, the heuristic becomes ready to solve unseen instances.

The objective function within the evolutionary process estimates how well the heuristics will cope with unseen instances by considering two crucial components when dealing with a set of instances.

The function $f(\vec{x})$ regards both the time for completing all the schedules (f_a) and the variation in the results from one instance to another (f_b), as given

$$f(\vec{x}) = f_a(\vec{x}) + f_b(\vec{x}) = \frac{1}{n}\sum_{i=1}^{n} C_i + \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left|C_i - \bar{C}\right|^2}, \tag{1}$$

where C_i stands for the makespan of the schedule for instance i in the test set, and $\bar{C}$ is the average makespan of all the instances in the test set.

The rationale behind reducing the total makespan is evident: the sooner the schedules are completed, the better the solutions. Plus, the reason for minimizing the standard deviation of the results is more difficult to appreciate. We expect that, by also minimizing the deviation of the makespans of the schedules produced by the heuristics, we are likely to avoid extreme cases where one heuristic is desirable for some instances but entirely useless for others. The proposed fitness function then aims to render heuristics that show a good and steady performance in different instances.

4 Experiments and Results

In this work, we generated 30 heuristics for the JSS problem by using the GP-based hyper-heuristic approach. These heuristics are labeled with the prefix 'HGP'. Additionally, and for comparison purposes, we also built 30 hyper-heuristics by employing the model proposed by Garza-Santisteban *et al.*, which relies on Simulated Annealing (SA) to produce a method capable of switching heuristics as the solving process takes place [15]. These hyper-heuristics are labeled with the prefix 'HHSA'. The 30 HGPs and the 30 HHSAs were compared against the human-made heuristics defined in Section 2.1. To evaluate the performance of each method, we used two metrics: the success rate and the reduction in total makespan when solving the test set. Table 1 summarizes the performance of the methods produced. Due to space restrictions, we only show the results of the best three performers per method.

Based on the results from Table 1, we can observe that the best method for the test set was one of the heuristics produced with our approach: HGP22. This heuristic generated schedules that represent essential savings in time concerning the schedules provided by human-made heuristics. For example, when compared against SPT, it saves 132 hours to complete all the schedules for the instances in the test set. That is more than five days earlier than with the best heuristic for the test set. The reductions are even more significant for the rest of heuristics.

Although HGP22 proved to be a competent heuristic, not all the heuristics we produced seem to have the same quality. Regarding the results obtained by

Table 1. Success rate and time savings (in hours) of the three best heuristics produced with the proposed approach (HGPs) and the three best hyper-heuristics built with SA [15] (HHSAs) when compared against the human-made heuristics on the test set. An arrow (↑) before the time savings indicates that, for that particular case, the method required additional hours to complete the schedules of the test set (no savings were obtained). The best method is highlighted in bold.

Method	SPT	LPT	MRT	MLM	LLM	EST
HGP16	(0.4, ↑100)	(0.5, 613)	(0.7, 42)	(0.4, 156)	(0.8, 611)	(0.8, 1154)
HGP21	(0.4, ↑103)	(0.6, 610)	(0.6, 39)	(0.7, 153)	(0.9, 608)	(1.0, 1151)
HGP22	**(0.5, 132)**	**(0.7, 845)**	**(0.7, 274)**	**(0.6, 388)**	**(0.8, 843)**	**(1.0, 1383)**
HHSA03	(0.5, 51)	(0.6, 764)	(0.6, 193)	(0.5, 307)	(0.7, 762)	(1.0, 1305)
HHSA06	(0.4, 34)	(0.8, 747)	(0.6, 176)	(0.6, 290)	(0.9, 745)	(0.9, 1288)
HHSA11	(0.6, 72)	(0.7, 785)	(0.6, 214)	(0.6, 328)	(0.9, 783)	(0.9, 1326)

HGP16 and HGP21, they are, in all cases, outperformed by any of the HHSA methods. Then, it may be the case that the competent behavior of HGP22 may be an exceptional situation challenging to replicate if we rerun the model. Future work should include a more in-depth analysis of the model behavior.

So far, we have shown that it is possible to improve upon the results from human-made heuristics by employing the heuristics generated by the GP-based approach. However, we have not yet analyzed how these results look when comparing the heuristics and hyper-heuristics against the best possible outcome by using the human-made heuristics, *i.e.*, the Oracle (H*). The total makespan of H* on the test set was 7846 hours while the total makespans of HGP22, HGP16, and HGP21 were 8205, 8437, and 8440 hours, respectively. These makespans are similar for the hyper-heuristics produced by using Simulated Annealing, HHSA11, HHSA03, and HHSA06, which required 8265, 8286, and 8303 hours to finish their schedules. When we compare the best performer of each method, HGP22 and HHSA11, we observe a difference of 60 hours that favors HGP22. It is essential to mention that neither HGP22 nor HHSA11 can improve the total makespan of the Oracle, but they achieve it on a per-instance basis. The success rates of HGP22 and HHSA11 are 0.4 and 0.3, respectively, which indicates that they indeed improved the best possible result from the human-made heuristics for some specific instances in the test set.

Deepening on the reductions in the makespan for some specific instances, Figure 2 depicts the distribution of makespan differences (in hours) of the HGPs and HHSAs when they are compared against H* (drawn as a red line). The human-made heuristics are omitted in this chart since they are already represented in the Oracle's behavior. In this plot, values below zero mean that the method improved the best schedule produced with the human-made heuristics. It is interesting to notice that HGP16 has a considerable variation in the makespans of the schedules it produces. For some instances, the schedule obtained has a very small makespan. However, for some others, it pays the price, and their schedules are among the worst in the makespan. In other words, their results are not consistent, which results in an overall performance that sits below the other HGPs and HHSAs. But, it achieves significant savings in some isolated instances, savings that no other method can achieve.

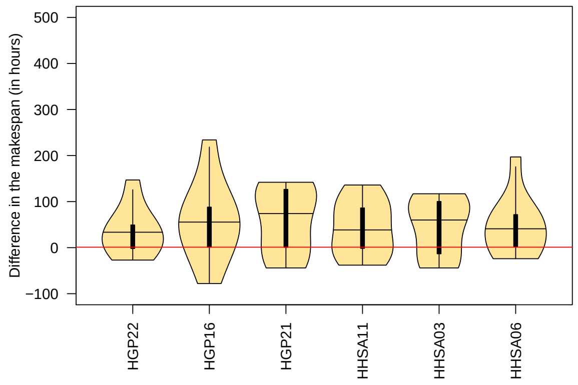

Fig. 2. Distribution of the makespan of the three best heuristics produced with the proposed approach (HGPs) and the three best hyper-heuristics produced with SA [15] (HHSAs) when compared against H* (red line) on the test set.

Finally, we conducted two pairwise two-tailed t-tests with a significance level of 5% to validate our results statistically. Although HGP22 and HHSA11 obtained good results on an individual level (some specific instances), we observed that they failed to reduce the total makespan of H* in the test set. Nonetheless, we scrutinized if the statistical evidence suggests differences in the performance of such methods. The tests were conducted on the pairs (HGP22, H*) and (HHSA11, H*). The p-values for these tests were 0.1323 and 0.1033, respectively. Then, the statistical evidence suggests that both HGP22 and HHSA11 are similar to the Oracle in terms of makespan for the test set.

5 Conclusions

In this study, we proposed an approach based on Genetic Programming (GP) for tackling the Job-Shop Scheduling (JSS) problem. The model produces new heuristics by combining features that characterize jobs within the instance being solved; this makes our model unique. To the best of our knowledge, other generation hyper-heuristic approaches rely on components of existing heuristics, which are reused. In our case, we do not have this limitation. Thus, the proposed model is free to explore and produce heuristics that explode a criterion that probably has never been seen before, since the evolutionary process creates a function that reflects such a criterion. However, the solution model now depends on the expressiveness of the features that describe the jobs.

In general, the GP-based approach favors the creation of heuristics that outperform the Oracle, at least for some instances in the test set. This was inferred from the results presented in Table 1 and Figure 2. However, we are aware that our training and test sets are quite short, and to validate our proposal thoroughly, we require to test it on a more extensive and diverse collection of instances. We plan to expand the set of training and test instances in the future. As future work, we would like to explore other features to characterize the jobs within the instance and compare the performance of the refined model against other existing generation hyper-heuristic models from the literature. Also, we consider that a combination of selection and generation hyper-heuristics might be possible by extending the model described in this document. We think that the model could use a hyper-heuristic that chooses among heuristics most of the time but generate completely new heuristics when the situation requires it. These new heuristics would be available for selection in future cases, extending the heuristic pool as the model deals with more instances. Furthermore, we plan to implement this GP-based approach in continuous optimization problems, for example, to create brand-new search operators [11].

Acknowledgments

This research was partially supported by CONACyT Basic Science Project under grant 287479 and ITESM Research Group with Strategic Focus on Intelligent Systems.

References

1. Adams, J., Balas, E., Zawack, D.: The shifting bottleneck procedure for job shop scheduling. Management Science **34**(3), 391–401 (1988)
2. Balas, E., Vazacopoulos, A.: Guided local search with shifting bottleneck for job shop scheduling. Management science **44**(2), 262–275 (1998)
3. Bozejko, W., Gnatowski, A., Pempera, J., Wodecki, M.: Parallel tabu search for the cyclic job shop scheduling problem. Computers & Industrial Engineering **113**, 512 – 524 (2017). https://doi.org/https://doi.org/10.1016/j.cie.2017.09.042
4. Bratley, P., Fox, B.L., Schrage, L.E.: A guide to simulation. Springer Science & Business Media (2011)
5. Burke, E.K., Hyde, M.R., Kendall, G.: Providing a memory mechanism to enhance the evolutionary design of heuristics. In: IEEE Congress on Evolutionary Computation. pp. 1–8. IEEE (2010)
6. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation. pp. 1559–1565 (2007)
7. Chaurasia, S.N., Sundar, S., Jung, D., Lee, H.M., Kim, J.H.: An Evolutionary Algorithm Based Hyper-heuristic for the Job-Shop Scheduling Problem

with No-Wait Constraint. In: Harmony Search and Nature Inspired Optimization Algorithms, vol. 741, pp. 249–257. Springer Singapore, Singapore (2019). https://doi.org/10.1007/978-981-13-0761-4_25
8. Chong, C.S., Low, M.Y.H., Sivakumar, A.I., Gay, K.L.: A bee colony optimization algorithm to job shop scheduling. In: Proceedings of the 2006 Winter Simulation Conference. pp. 1954–1961. Winter Simulatrion Conference, Monterey,California (Dec 2006). https://doi.org/10.1109/WSC.2006.322980
9. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: International Conference on the Practice and Theory of Automated Timetabling. pp. 176–190. Springer (2000)
10. Crowston, W.B., Glover, F., Trawick, J.D., et al.: Probabilistic and parametric learning combinations of local job shop scheduling rules. Tech. rep., CARNEGIE INST OF TECH PITTSBURGH PA GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION (1963)
11. Cruz-Duarte, J.M., Ivan, A., Ortiz-Bayliss, J.C., Conant-Pablos, S.E., Terashima-Marín, H.: A Primary Study on Hyper-Heuristics to Customise Metaheuristics for Continuous Optimisation. In: 2020 IEEE Congress on Evolutionary Computation (CEC). pp. 1–8 (2020)
12. Cunha, B., Madureira, A.M., Fonseca, B., Coelho, D.: Deep reinforcement learning as a job shop scheduling solver: A literature review. In: Madureira, A.M., Abraham, A., Gandhi, N., Varela, M.L. (eds.) Hybrid Intelligent Systems. pp. 350–359. Springer International Publishing, Cham (2020)
13. Fattahi, P., Messi Bidgoli, M., Samouei, P.: An improved tabu search algorithm for job shop scheduling problem trough hybrid solution representations. Journal of Quality Engineering and Production Optimization **3**(1), 13–26 (2018). https://doi.org/10.22070/jqepo.2018.1360.1035
14. Fisher, H.: Probabilistic learning combinations of local job-shop scheduling rules. Industrial scheduling pp. 225–251 (1963)
15. Garza-Santisteban, F., Sanchez-Pamanes, R., Puente-Rodriguez, L.A., Amaya, I., Ortiz-Bayliss, J.C., Conant-Pablos, S., Terashima-Marin, H.: A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 57–64. IEEE (jun 2019). https://doi.org/10.1109/CEC.2019.8790296, `https://ieeexplore.ieee.org/document/8790296/`
16. Grendreau, M., Potvin, J.: Handbook of metaheuristics. international series in operations research & management science, vol. 146 (2010)
17. H. Blackstone, J., T. Phillips, D., Hogg, G.: A state-of-the-art survey of dispatching rules for manufacturing job shop operations. International Journal of Production Research **20**, 27–45 (01 1982)
18. Huang, K.L., Liao, C.J.: Ant colony optimization combined with taboo search for the job shop scheduling problem. Computers & operations research **35**(4), 1030–1046 (2008)
19. dao-er ji, R.Q., Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research **39**(10), 2291 – 2299 (2012). https://doi.org/https://doi.org/10.1016/j.cor.2011.12.005
20. Koza, J.R., Koza, J.R.: Genetic programming: on the programming of computers by means of natural selection, vol. 1. MIT press (1992)
21. Lara-Cárdenas, E., Sánchez-Díaz, X., Amaya, I., Ortiz-Bayliss, J.C.: Improving hyper-heuristic performance for job shop scheduling problems using neural networks. In: Martínez-Villaseñor, L., Batyrshin, I., Marín-Hernández, A. (eds.) Ad-

vances in Soft Computing. pp. 150–161. Springer International Publishing, Cham (2019)
22. Lin, J.: Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time. Engineering Applications of Artificial Intelligence **77**(October 2016), 186–196 (2019). https://doi.org/10.1016/j.engappai.2018.10.008
23. Masood, A., Mei, Y., Chen, G., Zhang, M.: Many-objective genetic programming for job-shop scheduling. In: 2016 IEEE Congress on Evolutionary Computation (CEC). pp. 209–216. IEEE, Vancouver, Canada (July 2016). https://doi.org/10.1109/CEC.2016.7743797
24. Miyashita, K.: Job-shop scheduling with genetic programming. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation. pp. 505–512. GECCO'00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
25. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Genetic programming for job shop scheduling. In: Evolutionary and Swarm Intelligence Algorithms, pp. 143–167. Springer, Cham, Switzerland (2019)
26. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management science **42**(6), 797–813 (1996)
27. Sánchez, M., Cruz-Duarte, J.M., Ortiz-Bayliss, J.C., Ceballos, H., Terashima-Marín, H., Amaya, I.: A systematic review of hyper-heuristics on combinatorial optimization problems. IEEE Access **8**(1), 1–28 (2020). https://doi.org/10.1109/ACCESS.2020.3009318
28. Sha, D., Hsu, C.Y.: A hybrid particle swarm optimization for job shop scheduling problem. Computers & Industrial Engineering **51**(4), 791–808 (2006)
29. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research **64**(2), 278 – 285 (1993). https://doi.org/https://doi.org/10.1016/0377-2217(93)90182-M, project Management anf Scheduling
30. Türkyılmaz, A., Şenvar, Ö., Ünal, I., Bulkan, S.: A research survey: heuristic approaches for solving multi objective flexible job shop problems. Journal of Intelligent Manufacturing (feb 2020). https://doi.org/10.1007/s10845-020-01547-4, `http://link.springer.com/10.1007/s10845-020-01547-4`
31. Uckun, S., Bagchi, S., Kawamura, K., Miyabe, Y.: Managing genetic search in job shop scheduling. IEEE Intelligent Systems **8**(5), 15–24 (1993)
32. 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 (2016)
33. Wang, L., Zheng, D.Z.: An effective hybrid optimization strategy for job-shop scheduling problems. Computers & Operations Research **28**(6), 585–596 (2001)
34. Yska, D., Mei, Y., Zhang, M.: Feature construction in genetic programming hyper-heuristic for dynamic flexible job shop scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '18. pp. 149–150. ACM Press, New York, New York, USA (2018). https://doi.org/10.1145/3205651.3205741
35. Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J.: Review of job shop scheduling research and its new perspectives under industry 4.0. Journal of Intelligent Manufacturing **30**(4), 1809–1830 (Apr 2019). https://doi.org/10.1007/s10845-017-1350-2
36. Zhou, Y., Yang, J.J., Zheng, L.Y.: Hyper-Heuristic Coevolution of Machine Assignment and Job Sequencing Rules for Multi-Objective Dynamic Flexible Job Shop Scheduling. IEEE Access **7**, 68–88 (2019). https://doi.org/10.1109/ACCESS.2018.2883802