Automatic Design of Dispatching Rules with Genetic Programming for Dynamic Job Shop Scheduling

Salama Shady, Toshiya Kaihara, Nobutada Fujii, and Daisuke Kokuryo

Graduate School of System Informatics, Kobe University,
Kobe, Hyogo 6578501, Japan
shady.salama@kaede.cs.kobe-u.ac.jp

Abstract. Traditionally, scheduling experts rely on their knowledge and experience to develop problem-specific heuristics that require a considerable amount of time, experience, and code effort. Through this tedious process, experts must follow a trial-and-error cycle by evaluating the generated rules in a simulation model for the problem under consideration until achieving satisfactory results. Recently, hyper-heuristic approach has emerged as a powerful technique that uses artificial intelligence to automatically design efficient heuristics for various optimization problems. Genetic programming (GP) is the most popular hyper-heuristic approach to automate the design of production scheduling heuristics. In this paper, a genetic programming framework is proposed to generate efficient dispatching rules in a dynamic job shop. The proposed framework integrates the reasoning mechanism of GP with the ability of discrete event simulation in analyzing the performance of generated rules under dynamic conditions. Afterward, the evolved heuristics are compared to human-tailored literature rules under different dynamic settings using mean flow time and mean tardiness as performance measures. The achieved results prove the ability of the proposed approach in generating superior scheduling rules rapidly, within a few hours, compared to the conventional literature rules commonly adopted in the industry.

Keywords: Dynamic job shop, Scheduling, Dispatching rules, Hyper-heuristic, Genetic programming.

1 Introduction

Job Shop Scheduling Problem (JSSP) is a well-studied problem in combinatorial optimization literature that has proven to be NP-hard. JSSP can be defined as there are a number of jobs and each job has a predefined set of operations (tasks) that need to be processed for specific processing time on a limited number of machines (resources) while optimizing some performance criteria [1]. Due to the complex nature of the JSSP, several solution approaches have been developed to provide satisfactory solutions for both static and dynamic scenarios. Although many solution approaches are proposed for solving JSSP, most of these approaches mainly focus on relatively small problem instances or moderate to large instances with over-simplified assumptions. Dispatching Rules (DRs) have been widely employed for solving Dynamic Job

Shop Scheduling Problem (DJSSP) in a practical time due to its flexibility, scalability, and fast response to real-time events. There are several comparative studies in the literature for analyzing the performance of traditional scheduling rules under different conditions, for example, reference [2] and [3]. The common conclusion in these studies is that human-designed rules are developed to address specific job shop settings using a certain objective function, and usually perform poorly under other system configurations or performance measures. Moreover, literature rules experience a myopic nature by considering only local and current shop conditions that lead to poor quality solutions compared to global optimization methods.

In order to tackle these limitations, many scholars have used Artificial Intelligence (AI) techniques to automatically develop competitive scheduling rules that are trained under different scenarios. In recent years, Genetic Programming (GP) has proven its superiority compared to other AI techniques to generate DRs for different production scheduling problems [4]. Miyashita [5] is perhaps the first study that proposed GP approach to evolve DRs for job shop scheduling problems using a multi-agent model. Geiger et al. [6] employed the GP approach to obtain high-quality DRs for a single-machine scheduling problem. After evaluating the generated rules against 6 DRs from the literature, the generated rules revealed competitive results in minimizing completion time, total tardiness, and maximum tardiness. In the same context, the authors in [7] employed a decision tree to classify the available machines in a job shop as bottlenecks or not, then the GP approach is used to generate scheduling heuristic on a given machine based on its classification. The evolved rules provided better performance compared to seven literature rules. In [8], single-machine and static job-shop scheduling problems are considered. For the single machine problem, the authors examined the ability of the GP approach under both static and dynamic conditions as well as with or without setup times. While for JSSP, the GP approach used to minimize weighted job tardiness and total makespan. Their findings are consistent with previous studies in supporting the superiority of GP rules against literature rules. A two-stage hyper-heuristic is suggested in [9] by adopting the GP approach to generate DRs and an evolutionary algorithm to assign the developed rules to different work centers. In [10], the authors investigated the use of GP hyper-heuristic approach to evolve DRs for a two-machines job shop in both static and dynamic conditions. For the dynamic case, two GP representations were examined, including a single rule in both machines and evolving a specific rule for each machine. The GP approach achieved the optimal makespan in the static problem and good results in the dynamic problem.

The aim of this paper is to propose a GP framework for generating competitive DRs for DJSSP under different system scenarios. There are different sources of randomness considered in this study, such as job arrival, operations processing time, the number of operations per job, operation-machine compatibility, and due dates. To verify the effectiveness of the proposed framework, the GP evolved heuristics are compared to 12 human-tailored rules in minimizing mean flow time and mean tardiness under various dynamic settings. The remainder of this paper is organized as follows. The proposed framework is presented in Section 2. Also, the experimental details are given in Section 3. Section 4 provides a summary of the obtained results. Conclusions and some suggestions for future work are addressed in Section 5.

2 Genetic Programming Framework

The proposed framework consists of two main modules, a GP-based reasoning module, and a performance evaluation module to automatically generate dispatching rules for DJSSP. As shown in Figure 1, the GP module starts by generating an initial population of candidate heuristics using predefined functions, terminals (attributes), and a specific representation. In this study, we adopted a tree-based representation which is the most popular data structure to represent any priority function. Moreover, it is worth noting that there are three random techniques widely used to create the initial population, namely grow, full, and ramped-half-and-half. The grow method generates trees in variable sizes, while all trees have the same shape and size using the full method. Ramped-half-and-half combines both the full and the grow methods to produce a diverse population of trees with various shapes and sizes.

Afterward, the solution quality of generated heuristics will be assessed in the performance evaluation module using a Discrete Event Simulation (DES) model. The DES model includes a pre-defined set of training instances and a meta-algorithm. The meta-algorithm encapsulates the GP evolved rule and possible system constraints to produce valid schedules for a specific scheduling environment. The Giffler & Thompson (GT) meta-algorithm is widely used to create active and non-delay schedules [1]. The employed non-delay GT algorithm can be briefly described as follows. While there are unprocessed operations and there is an idle machine, the GT algorithm calculates the priority value for queued operations and schedules the operation with the highest priority. Then, the evolutionary process starts by selecting fit individuals from the population. The selection process determines which individuals can survive (copy the fittest heuristics to the new population) and which individuals are allowed to reproduce (apply genetic operations). After selecting the mating pool, genetic operators are employed to combine existing individuals (crossover operator) or introduce diversity (mutation operator) in order to form offspring for the next generation. The population evolves over a specified number of generations until the stopping criteria are satisfied. Finally, the algorithm terminates, and the best dispatching rule is obtained.

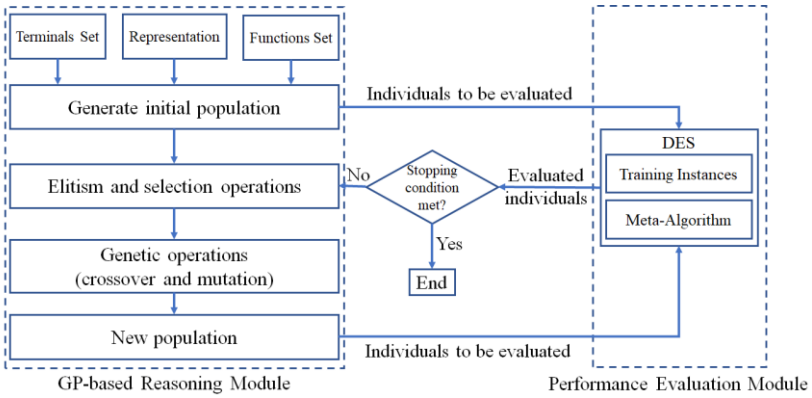

Fig. 1. The genetic programming framework

3 Numerical Experiments

3.1 The Genetic Programming Module

For the sake of evaluating the effectiveness of the proposed GP framework, it is used to generate competitive dispatching rules in minimizing mean flow time and mean tardiness for DJSSP. Then, the fittest rule for each objective is compared with 12 human-made rules under different system configurations. Tables 1 and 2 demonstrate the set of terminals and functions used in the GP module to create priority functions. The first part of the terminal set represents the local job shop attributes, while the second part refers to the global attributes. The ramped-half-and-half method is employed to generate the initial population with a maximum tree depth of 12. After executing multiple pilot experiments in order to fine-tune the GP parameters, the parameters presented in Table 3 are approved. The maximum number of generations is set to 50 generations each with a population size of 1000 dispatching rules. Using a double tournament selection technique, the GP individuals must pass two layers of tournaments, one by fitness and the other by size in order to be chosen. This selection procedure helps in mitigating the bloating effect which usually slows down the evolutionary search process, consumes extra memory, and restricts effective breeding [11]. Afterward, genetic operations are employed to create offspring.

Table 1. The terminal set for the GP module.

Node Name	Description
rJ	The release date of job (j)
RJ	The operation release date
RO	Number of remaining operations within the job
RT	Work remaining of the job
PR	Operation processing time
DD	Job due date
CT	Current time (Machine ready time)
WT	Current waiting time of the job (t - RJ)
#	Ephemeral constant (Random number between 0 and 1)
NPT	Processing time of next operation
WINQ	Work in the next queue
APR	Average operation processing time of jobs in the queue

Table 2. The function set for the GP module.

Node Name	Description
+	Binary addition operator
-	Binary subtraction operator
*	Binary multiplication operator
/	Protected binary division / (a, b) returns “1” if b = 0, else “a/b”
IF ternary	IF (a, b, c) returns “b” if a ≥ 0, else “c”
MIN	MIN (a, b) returns “a” if (a < b), else “b”
MAX	MAX (a, b) returns “a” if (a > b), else “b”

One-point leaf biased crossover is used to create children by randomly selecting a crossover point (with 10% probability to choose terminal nodes) in each individual and the parts below crossover points are exchanged. Also, uniform mutation operator randomly selects a point in an individual and replaces the subtree at this point with a randomly generated tree (with tree depth equals three). The best 5% of individuals are copied across generations to prevent the fittest heuristics from getting lost during the evolution process.

Table 3. The GP parameters.

Parameter	Description
Initialization	Ramped-half-and-half
Crossover rate	85%
Mutation rate	10%
Elitism	5%
Maximum depth	12
Number of generations	50
Population size	1000
Selection	Double tournament selection (size = 5)

3.2 The Performance Evaluation Module

A DES model for a commonly used symmetrical job shop was developed to assess the steady-state performance of the evolved heuristics [4]. The job shop has 10 machines and each machine can process only one operation at a time. The interarrival times between jobs are generated using an exponential distribution, and each job consists of a random number of operations between 2 and 14 operations. The processing time for each operation is randomly distributed U [1, 49] and U [1, 99]. The job due date is assigned using the total work content method, i.e. job due date = job release date + (allowance factor × total job processing time). The model is analyzed under three levels of utilization, including 70%, 80%, and 90%. In each simulation replication, the shop starts empty. Then, the obtained data between the beginning of the simulation until the arrival of the 1500th job is discarded, and the statistics are collected from the 1500th job to the next completed 5000 jobs. The triplet (p, u, c) represents the system settings by defining the average processing time, the job utilization level, and the tightness factor (to determine the tightness of job due date). Changing the values of "p" and "u" significantly affects the mean flow time, while varying the "c" values greatly affects the mean tardiness. The training and testing scenarios are illustrated in Table 4. Two simulation scenarios are employed to train the evolved rules in minimizing mean flow time and mean tardiness. Also, the 6 testing scenarios for each objective are used to compare the performance of the GP evolved rules with 12 rules from the literature. The benchmark rules are described as follows; SPT: Shortest Processing Time, LPT: Longest Processing Time, FIFO: First In First Out, LIFO: Last In First Out, WINQ: Work In Next Queue, SL: Slack, NSL: Negative Slack, SPT + WINQ + SL, MWKR: Most Work Remaining, COVERT: Cost Over Time, EDD: Earliest Due Date, and MOD: Modified Due Date, for more details please refer to [3].

Table 4. Simulation configurations for the training and testing sets.

Parameters	Training	Testing
Processing time	U [1, 49] and U [1, 99]	U [1, 49] and U [1, 99]
Shop utilization	70% and 90%	70%, 80%, and 90%
Tightness factor	3 and 5	3, 4, and 5
Mean flowtime scenarios	(25, 70, 3), (50, 90, 3)	(25, 70, 3), (50, 70, 3), (25, 80, 3), (50, 80, 3), (25, 90, 3), (50, 90, 3)
Mean tardiness scenarios	(25, 70, 5), (50, 90, 3)	(25, 70, 3), (50, 70, 3), (25, 80, 4), (50, 80, 4), (25, 90, 5), (50, 90, 5)

4 Results and Discussion

The proposed framework was executed on a PC with 2.7GHz CPU and 16 GB RAM. The GP module was trained on the training set for minimizing the mean flow time and the mean tardiness. Then, the fittest GP rule was compared with the 12 human-tailored rules on the 6 testing scenarios for each performance indicator. For the sake of convenience, we referred to the best GP rule for the mean flow time as FGP, and the best GP rule for mean tardiness as TGP. Tables 5 and 6 show the mean flow time and mean tardiness for each rule on different problem instances.

Table 5. Performance of all rules for mean flow time.

Rules	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Case 6**
SPT	394.2	792.58	526.28	1039.33	862.2	1687.42
LPT	636.37	1265.64	1073.51	2153.54	2634.44	5184.4
FIFO	482.54	961.23	689.75	1375.78	1278.46	2489.33
LIFO	479.55	945.38	676.84	1357.58	1266.19	2478.04
WINQ	437.35	870.25	598.22	1193.64	1032.13	2018.64
SL	457.15	906.1	654.36	1292.04	1233.28	2409.71
NSL	476.98	948.80	660.03	1323.17	1232.90	2389.51
SPT + WINQ + SL	440.3	888.38	631.62	1273.64	1210.19	2360.26
MWKR	567.54	1116.51	899.80	1761.97	1975.12	3847.52
COVERT	479.79	965.82	679.78	1371.5	1259.02	2483.99
EDD	454.16	897.46	644.57	1279.12	1225.65	2383.55
MOD	449.96	897.08	628.36	1247.36	1049.82	2043.90
FGP	**383.45**	**775.87**	**502.04**	**995.8**	**762.6**	**1519.96**
Relative change (%)	**2.73**	**2.11**	**4.61**	**4.19**	**11.55**	**9.92**

Regarding the mean flow time, it is obvious that the evolved GP rule outperforms the literature rules for all the test instances. The relative change was calculated using the following equation:

$$Relative\ change = \frac{P_{ref.} - P_{GP}}{P_{ref.}} * 100$$

Where $P_{ref.}$ denotes the performance value of the best literature rule in a specific problem instance, and P_{GP} denotes the performance value of the GP rule for the same instance. As shown in Table 5, the performance of the FGP dominates the best litera-

ture rule, and the average relative change is 5.85%. Moreover, the gap between the performance of FGP and the best literature rules in each instance widens by increasing the system usage and processing time interval i.e. when the problem instance becomes more complex and dynamic. The FGP rule was analyzed to understand how this role works and what parameters have the highest impact on the mean flow time. We noticed that all the global features are used in the evolved rule, while only some local features are considered. This finding manifests the importance of global attributes in mitigating the myopic nature of scheduling rules. Also, the top 4 features used in descending order based on the number of occurrences are the operation processing time, the work in the next queue, the processing time of the next operation, and the number of remaining operations within the job. This observation is consistent with the superiority of SPT and WINQ in being the best literature rules which proves the reasoning ability of GP to choose the most important job shop attributes for a specific objective. Excluded features include job release date, due date, and operation waiting time. Using these results, we concluded that the excluded features do not have a significant impact on average job flow time, which explains the disappointing performance of LPT, FIFO, and LIFO rules in reducing the mean flow time.

Regarding the mean tardiness, the TGP showed substantially better performance compared to the best literature rule with an average relative change of 45.56%. Also, the superiority of the GP approach was revealed as the job shop becomes more random, and due dates become tighter, as shown in case 1 compare to case 6. After investigating the features employed in the TGP rule and sorting them in descending order of importance, the following results were obtained. The most significant attributes for minimizing the mean tardiness are operation processing time, work in the next queue, number of operations within the job, and job due date. Furthermore, we gained a solid understanding of why the TGP achieved the best results, as well as why MOD, SPT+WINQ+SL, and SPT are the top literature rules in minimizing the mean tardiness. Similar to the FGP rule, all the global features are included in the TGP rule which demonstrates its great impact on enhancing the performance of DRs.

Table 6. Performance of all rules for mean tardiness.

Rules	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Case 6**
SPT	29.79	57.91	64.09	125.49	248.16	470.32
LPT	188.51	375.65	451.01	919.35	1780.28	3489.27
FIFO	45.02	86.82	85.75	166.14	393.78	731.4
LIFO	87.68	170.15	177.27	356.3	598.27	1153.94
WINQ	38.43	74.01	71.44	137.53	275.59	510.57
SL	19.58	38.57	39.14	71.87	306.65	562.62
NSL	24.04	46.03	39.8	79.03	282.02	524.05
SPT + WINQ + SL	15.95	32.12	33.58	63.39	288.09	526.25
MWKR	151.2	291.25	343.15	667.36	1222.67	2344.97
COVERT	42.71	84.99	82.73	161.73	381.02	727.08
EDD	23.17	44.05	42.45	81.3	308.73	575.72
MOD	21.78	42.21	38	72.52	240.47	448.61
TGP	**12.13**	**23.7**	**16.71**	**32.23**	**91.86**	**170.41**
Relative change (%)	**23.94**	**26.20**	**50.24**	**49.16**	**61.80**	**62.00**

5 Conclusions and Future Research

This paper proposed a GP framework to automatically generate scheduling policies in dynamic job shops. The main building blocks of this approach were presented with a detailed explanation of the internal mechanism. Also, the proposed approach was employed to minimize mean flow time and mean tardiness under different dynamic scenarios. The experimental results demonstrated the ability of the proposed framework to automatically design effective dispatching rules with respect to the considered performance measures. Moreover, the GP reasoning mechanism was able to select the most significant attributes to include them in evolved heuristics and exclude insignificant attributes. In future research, the proposed approach will be extended to generate scheduling rules that can handle multiple objectives. Also, the framework is expected to be evaluated using real-world data to assess its practical effectiveness.

References

1. Pinedo, M., Khosrow, H.: Scheduling: theory, algorithms and systems development. In: Springer, Berlin. Springer US (2012). https://doi.org/10.1007/978-1-4614-2361-4_1.
2. Rajendran, C., Holthaus, O.: Comparative study of dispatching rules in dynamic flowshops and jobshops. Eur. J. Oper. Res. (1999). https://doi.org/10.1016/S0377-2217(98)00023-X.
3. Sels, V., Gheysen, N., Vanhoucke, M.: A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. Int. J. Prod. Res. 50, 4255–4270 (2012). https://doi.org/10.1080/00207543.2011.611539.
4. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. 3, 41–66 (2017). https://doi.org/10.1007/s40747-017-0036-x.
5. Miyashita, K.: Job-shop scheduling with genetic programming. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation. pp. 505–512 (2000).
6. Geiger, C.D., Uzsoy, R., Aytuğ, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. J. Sched. 9, 7–34 (2006). https://doi.org/10.1007/s10951-006-5591-8.
7. Jakobović, D., Budin, L.: Dynamic scheduling with genetic programming. In: European Conference on Genetic Programming. pp. 73–84. Springer, Berlin, Heidelberg (2006). https://doi.org/10.1007/11729976_7.
8. Jakobović, D., Marasović, K.: Evolving priority scheduling heuristics with genetic programming. Appl. Soft Comput. J. 12, 2781–2789 (2012). https://doi.org/10.1016/j.asoc.2012.03.065.
9. Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B.: Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. Int. J. Prod. Econ. 145, 67–77 (2013). https://doi.org/10.1016/j.ijpe.2012.10.016.
10. Hunt, R., Johnston, M., Zhang, M.: Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In: Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014. pp. 618–625. Institute of Electrical and Electronics Engineers Inc. (2014). https://doi.org/10.1109/CEC.2014.6900655.
11. Luke, S., Panait, L.: Fighting bloat with nonparametric parsimony pressure. In: International Conference on Parallel Problem Solving from Nature. pp. 411–421. Springer Verlag (2002). https://doi.org/10.1007/3-540-45712-7_40.