2022 IEEE 18th International Conference on Automation
Science and Engineering (CASE)
August 20-24, 2022. Mexico City, Mexico

2022 IEEE 18th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-9042-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASE49997.2022.9926615

A Simulation Optimization-Aided Learning Method for Design Automation of Scheduling Rules

Hang Ma, Cheng Zhang and Zhongshun Shi

Abstract— **Intelligent manufacturing systems require real-time optimization algorithms for daily operations management. Scheduling rules have been proven to be efficient and commonly used in plenty of practical production scenarios, especially for the large-scale problems. However, almost all the scheduling rules are manually designed, which is time consuming and also results in the large loss of accuracy for complex problems. This paper proposes a new simulation optimization-aided learning method, denoted by SOaL, for design automation of scheduling rules. The proposed SOaL method treats the automated design of scheduling rules as a simulation optimization problem, where we use genetic programming algorithm to guide the rule generation and introduce ranking and selection algorithm to improve the rule evaluation accuracy. Using dynamic job shop scheduling problem as the simulation testbed, numerical results show the superiority of the proposed method.**

I. Introduction

Production scheduling is critical for the daily operations of complex manufacturing systems. Real-time and high-accuracy are two important elements of the scheduling algorithms design for use in practice. The main design challenges arise from the extremely large-scale decision variables, heavily constrained search space, and the dynamic nature of real-world production scenarios including rush order, uncertain processing times, machine breakdowns, etc. Therefore, scheduling rules have been widely used in practice due to their low computational complexity and high capability of coping with these dynamic changes. However, the current design of scheduling rules is mostly constructed by experts and practitioners manually. The design process is customized and time-consuming, and does not have the good scalability for various production scenarios. Therefore, transforming the traditional design manually to design automation of scheduling rules will enable the manufacturing systems to be smart, and has significant impacts to modern industry.

Recent advances in machine learning and artificial intelligence have offered fresh perspectives for the automated design of scheduling rules to accelerate the design process. The main machine learning algorithms used include genetic programming [1], neural networks [2] and reinforcement learning [3]. Considering the advantages of genetic programming (GP) on the representation and interpretability of scheduling rules [4]–[6], we utilize GP as a part of the proposed learning method in this paper. GP starts from an initialization stage of randomly creating a population of candidate rules using feature set (problem related attributes, e.g. due date of a job) and function set (arithmetic operator, logic operator and mathematical function). Then, GP iteratively evolves a population of new candidate rules by using genetic operators such as mutation and crossover to generate high-performance rules. In each generation of the evolution process, each rule is evaluated by a fitness function or a simulation model, which estimates the performance of the evolved scheduling rules to solve a specific scheduling problem. With its ability to represent arbitrary structure and evolve interpretable rules, GP can automatically discover effective rules that outperform those previously proposed by academia and industry [7]–[10].

The automatic design of scheduling rules can be summarized as rule *generation* and rule *evaluation*. For rule generation, a set of high quality scheduling rules is the foundation of solving scheduling problems. Most previous research mainly focuses on rule generation and has proven the superiority of scheduling rules generated by GP [11]–[14]. Rule evaluation is another important issue. Hildebrandt et al. (2010) conducted simulation experiments on GP to show the possibility to use fewer replications with a new random seed in each generation to achieve equivalent performance [15]. Shi et al. (2019) proposed a new evaluation method for GP by simulation optimization [10]. The main idea is to develop a new ranking and selection algorithm for ranking candidate rules in each generation. Then the rules with higher rank have more chance to be selected, which guides the search in better direction. Most related research focuses on improving evaluation efficiency in each generation within the learning process. It seems to be promising to improve the evaluation accuracy in the final stage by using GP via simulation optimization techniques. However, no research explores this. This paper will fill this gap.

In this paper, we propose a new simulation optimization-aided learning method, denoted by SOaL, for design automation of scheduling rules. The proposed SOaL method treats the automated design of scheduling rules as a simulation optimization problem, where we use genetic programming algorithm to guide the rule generation and introduce ranking and selection algorithm to improve the rule evaluation accuracy. Using dynamic job shop scheduling problem as the simulation testbed, numerical results show the superiority of the proposed method.

The remainder of this paper is organized as follows. In Section II, we introduce the dynamic job shop scheduling problem. In Section III, we propose the SOaL method for the automated design of dispatching rules for dynamic job shop scheduling. Section IV presents the design of compu-

H. Ma, C. Zhang and Z. Shi are with Intelligence, Dynamics, Emulation and Automation for Systems Laboratory, Department of Industrial and Systems Engineering, University of Tennessee, Knoxville, TN 37996, USA (e-mail: hma19@vols.utk.edu, czhang86@vols.utk.edu, tzshi@utk.edu)

tational experiments, numerical results and analysis. Section V concludes this paper.

II. Dynamics Job Shop Scheduling

In this paper, we focus on using proposed method to facilitate the task of automatically learning dispatching rules for dynamic job shop scheduling (DJSS) problem, which is a typical problem in complex manufacturing systems. For DJSS, a job set J of n jobs needs to be processed by a set M of m machines. Each job j has an arrival time and is composed of a sequence of operations O_j. Let $O = \{(j,i)|j \in J,\ i \in O_j\}$. Each operation in O_j can only be processed by one of the m machines for a specified processing time, p_{ji}. Once an operation starts processing on a given machine, it cannot be interrupted until it is completed. The order of operations for each job is predefined, and one cannot start processing an operation until all its precedent operations have been processed.

In the studied DJSS problem, jobs arrive over time according to some stochastic process, and there is no knowledge about the job until it arrivals. Once a job enters the shop, it joins the queue at the machine which is required to process its first operation. Upon completion of an operation it moves to the next available machine, or exits the shop if its last operation has been completed. Generally, there are no optimal solutions as all information about jobs are not known until the end of the scheduling period, therefore methods that construct complete schedules in advance are not able to be used. The objective of the scheduling is to make sequencing decision on each machine for processing jobs to minimize/maximize some objective functions.

As a sort of scheduling rules, dispatching rules (DRs) have been used consistently in practice because of their low computation cost and ability to cope with the dynamics of shop changes. Whenever a machine is available and jobs are waiting in the queue, a DR assigns a priority value to each queued job, and the job with the highest priority is selected to be processed next. In next section, we propose a simulation optimization-aided learning method for the automated design of dispatching rules for DJSS problems.

III. Proposed SOaL Method

In this section, the main idea of the proposed SOaL method is described first. Then we present the detailed designs of important components. At the end, we present the complete algorithm procedure of the proposed SOaL method.

A. Overview of SOaL Method

The main idea of the SOaL method is to use simulation optimization technique with GP to automatically generate and select effective dispatching rules through an offline learning manner. Although automatic design of DRs using GP is well developed, there are still some drawbacks in its rule evaluation. Firstly, the best DR in each generation is compared with current best one and updated if wins, which is deemed to be short-sighted. For example, assume the probability of selecting the best DR in each generation is 95% and there are 50 generations, thus the probability to select the true best DR after all the 50 generations is $0.95^{50} \approx 7.69\%$. Obviously, this selection mechanism makes it difficult to select and retain the true best DR to the last generation. Secondly, for the DRs in dynamic job shop scenario, each DR needs to run multiple simulation replications to estimate its quality. The whole GP run requires an intensive computational cost. This motives us to develop new evaluation mechanism to improve the whole performance.

In fact, the rule evaluation can be regarded as a simulation optimization problem. Given N simulation budget, the problem is how to effectively allocate N simulation budget to competing T DRs so that the probability of correct selection of best rule is maximized. Therefore, we introduce a new evaluation mechanism called global evaluation to enhance evaluation accuracy in our proposed SOaL method. Many advanced techniques [16]–[19] developed in simulation optimization community can be applied. We select the well-known Optimal Computing Budget Allocation (OCBA) procedure in [16], as the ranking and selection algorithm, with an aim to improve the rule evaluation accuracy.

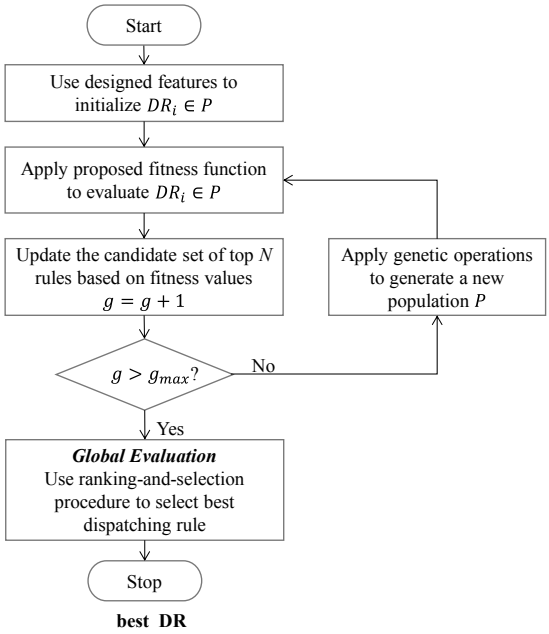

Fig. 1. The main flowchart of SOaL method

We present the detailed flowchart of the proposed SOaL method in Figure 1. The proposed SOaL method starts to generate an initial population $\mathcal{P}$ of dispatching rules randomly using predefined features and functions for DJSS problem. Then all the dispatching rules are evaluated using a simulation model to obtain fitness and the performance is recorded for each rule. The proposed method executes genetic operations to produce new candidates for next generation and the algorithm iterates. When the stopping rule of maximum generation holds, the global evaluation screens out inferiors ones and performs the OCBA procedure to evaluate the candidates and the best DR is selected as the final output. Then, the learned best DR can be applied online to solve new

instances of DJSS problem. Next, we present the important components of the proposed SOaL method.

B. Representation

We employ tree representation, which is common in GP, to represent dispatching rules. Table I and Table II are the feature set and function set in the proposed SOaL method, which were also the best settings identified for the tree representation in [20]. In addition, we divide feature set into 2 subsets. The first 7 basic features belong to basic set called F_B while *SLACK* and *TD* are due date related features in subset F_D. When dealing with different objectives, we will combine one or more subsets as feature set. The size of the search space depends on the feature set and function set. The function set consists of the basic arithmetic operations, the maximum function and a ternary version of if-then-else.

TABLE I
FEATURE SET OF SOAL

	Feature	Description
F_B	PT	processing time of current operation
	NPT	processing time of the next operation
	WINQ	work in next queue
	RPT	sum of processing times of all remain operations
	OpsLeft	the number of all remain operations within a job
	TIQ	current time since arrival in current queue
	TIS	current time since arrival in system of job
F_D	SLACK	the slack of current operation
	TD	time to due date ($due\ date - t$)
	Constant	random sample from uniform distribution in $[-1, 1]$

TABLE II
FUNCTION SET OF SOAL

Function	Description
$+$	Addition, binary operator
$-$	Subtraction, binary operator
$\times$	Multiplication, binary operator
$\div$	Protected division, binary operator
$\max(a, b)$	Maximum of a and b, binary operator
if-then-else(a,b,c)	if $a \geq 0$ then b else c, ternary operator

Figure 2 shows an example of tree representation of the well-known Holthaus rule 2PT+WINQ+NPT for DJSS [21]. In this example, 2, PT, WINQ and NPT at the leaf nodes of the tree are features. Operators "$+$" and "$\times$" at the internal nodes are functions. Thus, the dispatching rules can be represented as trees with proper feature set and function set.

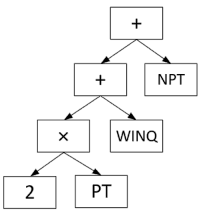

Fig. 2. An example on tree representation of rule 2PT+WINQ+NPT

C. Fitness Function

For the population $\mathcal{P}$ of dispatching rules generated by SOaL, we measure the quality of a dispatching rule $DR_i \in \mathcal{P}$ by the relative deviation of its objective value from the corresponding reference objective value using below equation:

$$dev(DR_i, I_n) = \frac{Obj(DR_i, I_n) - Ref(I_n)}{Ref(I_n)},$$

where $Obj(DR_i, I_n)$ is the objective value obtained when applying DR_i to simulation I_n, and $I_n \in \{I_1, \cdots, I_K\}$ is a set of K simulations for evaluation. $Ref(I_n)$ is the objective value for simulation I_n with a reference method. Then, we define the fitness function $f(DR_i)$ as the average value of $dev(DR_i, I_n)$ using the set of K simulations as follows:

$$f(DR_i) := \frac{1}{K} \sum_{I_n \in \{I_1, \cdots, I_K\}} dev(DR_i, I_n).$$

D. Algorithm Procedure of SOaL Method

Based on the above developed components, we present the detailed procedure of SOaL in Algorithm 1. Using the problem features and functions designed in Section III-B, the DRs in population $\mathcal{P}$ are randomly initialized. Then, we use the fitness function developed in Section III-C to evaluate the performance of all DRs based on simulation replications. Afterwards, the DRs are selected randomly based on their fitness to reproduce new candidates by crossover and mutation operators. After the stopping rule of maximum generation is reached, we apply global evaluation to evaluate all the evolved rules in $\mathcal{S}$ and obtain the final best DR.

Algorithm 1 SOaL

Input: g_{max}: the maximum generation for a whole run
K: the number of simulations for rule evaluation
Output: the best dispatching rule DR^*

```
Initialize P by using designed features and functions
Set S ← ∅; g ← 0
while g < g_max do
    f(DR_i) ← run K simulations for each DR_i ∈ P
    S ← S ⋃ P
    P ← reproduce using crossover and mutation operators from P of current generation
    g ← g + 1
end while
DR* ← apply Algorithm 2 to do global evaluation to select DR* from S
return DR*
```

The pseudo-code of the proposed global evaluation algorithm is presented in Algorithm 2. Since each DR is first evaluated by K simulation replications with common random number, those rules with same fitness can be treated as identical. For example, rule PT and rule $PT + PT$ are identical though they have different tree structures. Therefore, the proposed global evaluation algorithm will screen out identical rules at first step.

Algorithm 2 Global Evaluation

Input: $\mathcal{S}$: the set of all evolved dispatching rules
T: the number of candidate rules for screening out
N: total simulation budget for final evaluation
Output: the best dispatching rule DR^*

Set $\mathcal{C} \leftarrow \emptyset$
for $DR_i \in \mathcal{S}$ **do**
  if $f(DR_i) \notin \{f(DR_j)|DR_j \in \mathcal{C}\}$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{DR_i\}$
  end if
end for
$\mathcal{T} \leftarrow$ select the best T rules $\{DR_1, DR_2, \cdots, DR_T\}$ in $\mathcal{C}$ by increasing order of $f(DR_i)$
Set $l \leftarrow 1$; let N^l be the total accumulated number of simulations used for all $DR_i \in \mathcal{T}$ by the l-th iteration of final evaluation, and N_i^l that used for DR_i;
Set $N^0 = K \cdot T$; set $N_i^0 = K \ \forall \ i = 1, \cdots, T$
Let μ_i and σ_i^2 be the mean and variance of $f(DR_i)$ by N_i^0 simulations for $DR_i \in \mathcal{T}$
while $\sum_{i=1}^{T} N_i^{l-1} < N_0 + N$ **do**
  $N^l \leftarrow N^{l-1} + \Delta$
  Get $N_1^l, \cdots, N_T^l$ by Algorithm 3 with N^l, μ_i and σ_i^2
  Evaluate $DR_i \in \mathcal{T}$ by performing additional $\max(0, N_i^l - N_i^{l-1})$ simulations
  Update μ_i and σ_i^2 for $f(DR_i)$
  $l \leftarrow l + 1$
end while
$DR^* \leftarrow \arg\min_{DR_i \in \mathcal{T}} f(DR_i)$
return DR^*

Algorithm 3 OCBA

Input: N^l: total available simulation budget
μ_i, σ_i^2: mean and variance of $f(DR_i)$, $i = 1, \cdots, T$
Output: N_i^l: simulation budget allocated to $DR_i \in \mathcal{T}$

Set $r_1, r_2, \cdots, r_T \leftarrow 0$
Get the best mean $b \leftarrow \arg\min_i \mu_i$
Get the second best mean $s \leftarrow \arg\min_{i \neq b} \mu_i$
$r_s \leftarrow 1$
for $i = 1$ to T **do**
  if $i \neq b$ and $i \neq s$ **then**
    $r_i \leftarrow (\frac{\mu_b - \mu_s}{\mu_b - \mu_i})^2 \frac{\sigma_i^2}{\sigma_s^2}$
  end if
end for
$r_b \leftarrow \sigma_b \sqrt{\sum_{i \neq b} r_i^2 / \sigma_i^2}$
set $M \leftarrow \sum r_i$
for $i = 1$ to T **do**
  Obtain the ratio of samples for each $DR_i \in \mathcal{T}$: $ratio_i \leftarrow r_i / M$
end for
Get $N_1^l, N_2^l, \cdots, N_T^l$ by rounding of $ratio_i \times N^l$
return $N_1^l, N_2^l, \cdots, N_T^l$

In the global evaluation, we iteratively assign in total N simulations with increment quantity Δ to the selected non-identical and best T rules. We use OCBA procedure presented in Algorithm 3 to allocate additional simulation samples to competing DRs in each iteration. Here the simulations allocated during GP optimization, namely N_i^0, are used as the initial samples of OCBA for evaluating the initial μ_i and σ_i. Note to obtain N_i^l, appropriate rounding policy is applied for $ratio_i \times N^l$ in OCBA depending on parameters Δ and T. After the given simulation budget N is exhausted, the algorithm outputs the dispatching rule with best performance.

IV. Computational Experiments

In this section, we first present the simulation model, the parameter setting of the SOaL method and other methods for comparison. Then we present the results and analysis.

A. Simulation Model of DJSS

All experiments are based on the simulation model of dynamic job shop environment, which has been widely used in previous studies [15], [20], [22]. There are 10 machines on the shop floor. A random routing is assigned when job is arriving. The processing times follow a discrete uniform distribution U[1, 49], ranging from 1 to 49 minutes. Job arrivals follow a Poisson process. The total-work-content (TWK) method of due date setting [23] is used in the experiments with the allowance factor $\alpha = 4$. The machine utilization level u of the shop is set to 95%. At beginning the shop is empty, and the period until the 500-th job is used as warm-up. We collect data of jobs numbering from 501 to 2500, which will be used to calculate the performance.

B. Parameter Setting of SOaL

The parameter setting of the SOaL method is presented in Table III. The population size of 500 ensures enough diversity in the population. We use the standard crossover operator, which produces two offspring candidates by exchanging a randomly selected subtree between the two selected parents. The mutation operator is subtree mutation, which randomly selects a node of the tree and replaces it by a new, randomly created subtree. As shown in Table III, we use different features for two objectives: minimizing the mean flowtime $\bar{F}$ and the number of tardy jobs $\bar{T}$. For minimizing $\bar{F}$, we use the Holthaus rule 2PT+WINQ+NPT as reference rule, which is the best manually designed rule so far for $\bar{F}$. For minimizing $\bar{T}$, we use the SPT rule as reference rule.

In addition, we set K=15 simulation replications for each candidate rule in fitness evaluation, which means 7500 simulations in total for each generation. We perform a screen out procedure to select the best T=1000 candidate rules after GP optimization, resulting in N_0=15000. We set an additional N=15000 replications with the simulation increment Δ=15 for global evaluation.

C. Comparison Methods

In order to show the effectiveness and robustness of our proposed SOaL method, we choose a set of 12 benchmark dispatching rules proposed in the literature. We briefly summarize the benchmark dispatching rules as follows.

TABLE III
PARAMETER SETTING FOR THE PROPOSED SOAL METHOD

Population size	500
Generations	50
Crossover rate	90%
Mutation rate	5%
Elites	10
Selection Method	Tournament Selection (Size:7)
Initialization	Ramped Half-and-Half (Min:2 Max:6)
Maximum tree depth	17
Feature Set	$\bar{F} : F_B$ $\bar{T} : F_B \cup F_D$
Function Set	See Table II

1) *FIFO*: the job that has entered the queue at the earliest is processed first.
2) *SPT*: the job with shortest processing time is processed first.
3) *ERD*: the job with earliest release date is processed first.
4) *EDD*: the job with earliest due date is processed first.
5) *SRPT*: the operation belonging to the job that has the shortest remaining processing time is processed first.
6) *MDD*: the job with the minimum modified due date is processed first.
7) *MOD*: the operation with the minimum operation modified due date is processed first.
8) *ATC*: the operation with the minimum apparent tardiness cost is processed first.
9) *WINQ*: jobs are ranked in the order of an estimation of their waiting time in the queue before processing on the next machine can start.
10) *PT+WINQ*: the job with the minimum sum of WINQ and the processing time is processed first.
11) *PT+WINQ+Slack*: the job with the minimum sum of WINQ, the processing time of current operation and slack is processed first.
12) *2PT+WINQ+NPT*: the job with the minimum sum of WINQ, twice the processing time of current operation and that of next operation is processed first.

Meanwhile, we will compare the proposed SOaL with TGP [20] to demonstrate its effectiveness. TGP is a traditional tree-based GP method proposed by Branke et al. (2015) in the literature [20]. The rule with best fitness is selected as final output after GP optimization. We use the same parameters as shown in Table III for TGP and SOaL. All the methods are coded in Java and run on a Intel Core i7-10700H 2.90 GHz CPU and 16 GB RAM.

D. Numerical Results and Analysis

In the experiments, we use the tuple $\langle m, \alpha, u\rangle$ to represent the different settings of simulation scenarios. Here, m is the number of machines, α is the allowance factor and u is the utilization level of the shop. We use 3 dynamic job shop scenarios, $\langle 10, 6, 85\%\rangle$, $\langle 10, 4, 95\%\rangle$ and $\langle 50, 4, 95\%\rangle$, to test performance. Here, we can see that $\langle 10, 4, 95\%\rangle$ is also the setting for training and applied to generate new instances for testing. $\langle 10, 6, 85\%\rangle$ and a more complex scenario $\langle 50, 4, 95\%\rangle$ with 50 machines are used to evaluate the generalization performance. In each scenario, the average performances of 1000 testing replications of all comparison methods are recorded to evaluate the effectiveness.

Table IV shows the mean value (column "Mean") and standard error (column "SE") of the objective obtained by the methods under comparison on 1000 unseen simulation instances in different simulation scenarios. The DR obtained by TGP is denoted by TGP-DR while that of SOaL is by SOaL-DR. For minimizing mean flowtime (minutes), it can be seen that SOaL-DR significantly outperforms all benchmark rules in each simulation scenario. Moreover, the mean of SOaL-DR is generally better than that of TGP-DR by two standard errors, and thus is statistically significant. The numerical results show that simulation optimization technique could find better DR which is not selected by traditional GP method. For the complex scenario $\langle 50, 4, 95\%\rangle$ with 50 machines, we notice that the $\bar{F}$ obtained by SOaL-DR is 5213.70 while that of the TGP-DR is 5519.966, which is worse than 5518.68 obtained by the best manually developed rule 2PT+WINQ+NPT. It shows that global evaluation is very effective and can be used to improve effectiveness and robustness for GP optimization.

For minimizing the number of tardy jobs, it is seen that SOaL-DR still significantly outperforms all benchmark rules. SOaL-DR is also better than TGP-DR with statistical significance by generally two standard errors. Compared to reference rule SPT, SOaL-DR improves 99.2%, 85.5% and 94.4% on scenarios $\langle 10, 6, 85\%\rangle$, $\langle 10, 4, 95\%\rangle$ and $\langle 50, 4, 95\%\rangle$, respectively. It is interesting that SOaL-DR can achieve about 0.79 tardy jobs out of 2000 jobs in average, which means that the dispatching rule automatically learned by SOaL method could find the near optimal solution for some instances. It should be noted that the average computation time of SOaL-DR for each instance is nearly 0.3 s on scenarios $\langle 10, 6, 85\%\rangle$ and $\langle 10, 4, 95\%\rangle$, and about 1.4 s on scenario $\langle 50, 4, 95\%\rangle$ for minimizing both two objectives. All above results show that the proposed SOaL method is capable of automatically designing effective and robust dispatching rules for DJSS problem.

V. CONCLUSIONS

In this paper, we explore efficient machine learning method for design automation of scheduling rules from the perspective of simulation optimization. We develop a new simulation optimization-aided learning (SOaL) method by integrating traditional genetic programming algorithm with advanced ranking and selection technique to improve the design quality. The proposed SOaL method is then used for automatically designing dispatching rules for dynamic job shop scheduling problems. Numerical results show that the learned dispatching rules outperform the existing ones in all testing scenarios. Future work should focus on developing

TABLE IV

NUMERICAL RESULTS OF MEAN FLOWTIME AND NUMBER OF TARDY JOBS IN DIFFERENT PRODUCTION SCENARIOS

Method	Mean Flowtime: $\bar{F}$						Number of Tardy Jobs: $\bar{T}$					
	$\langle 10, 6, 85\% \rangle$		$\langle 10, 4, 95\% \rangle$		$\langle 50, 4, 95\% \rangle$		$\langle 10, 6, 85\% \rangle$		$\langle 10, 4, 95\% \rangle$		$\langle 50, 4, 95\% \rangle$	
	Mean	SE	Mean	SE	Mean	SE	Mean	SE	Mean	SE	Mean	SE
FCFS	1036.83	3.71	2295.05	16.87	8897.92	28.05	277.87	5.07	1933.05	2.82	1985.74	0.93
SPT	722.58	1.83	1312.14	8.78	5942.90	8.78	92.52	1.20	733.74	3.69	1148.32	5.52
ERD	977.31	2.98	1885.13	10.91	6952.04	10.91	158.96	3.52	1927.41	3.06	1915.83	2.53
EDD	936.10	2.90	1853.19	10.84	6912.74	10.84	55.36	2.71	1938.19	3.15	1926.84	2.67
SRPT	962.60	3.23	1875.15	9.46	6631.32	9.46	232.10	2.38	1119.88	2.79	1395.89	3.11
MDD	936.38	2.91	1891.15	10.14	6835.66	10.14	51.06	2.53	1912.25	3.20	1918.15	2.72
MOD	880.54	2.35	1413.85	8.59	6268.81	8.59	18.09	1.01	1324.39	7.80	1681.01	5.23
ATC	875.99	2.33	1422.78	8.17	6331.26	8.17	15.62	0.82	1012.98	5.73	1132.79	3.87
WINQ	811.16	2.21	1548.67	10.86	6190.47	10.86	127.39	2.01	1427.22	7.46	1469.73	7.32
PT+WINQ	724.09	1.89	1337.82	9.24	5851.63	9.24	78.68	1.31	1061.94	7.07	1298.56	8.13
PT+WINQ+Slack	726.91	1.92	1438.72	10.24	5553.38	10.24	98.14	1.89	1678.11	8.35	1682.94	7.67
2PT+WINQ+NPT	708.14	1.72	1245.39	8.06	5518.68	8.06	61.73	1.06	928.38	6.63	1135.62	8.40
TGP-DR	690.53	1.56	1132.36	5.97	5519.96	17.77	0.90	0.06	108.16	0.85	66.34	0.81
SOaL-DR	**687.01**	1.54	**1124.90**	5.62	**5213.70**	14.09	**0.79**	0.06	**106.71**	0.86	**64.44**	0.81

theoretical and algorithmic foundations for design automation of scheduling rules and applying the proposed method to complex production scheduling problems in practice.

REFERENCES

[1] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2015.

[2] J. N. Gupta, A. Majumder, and D. Laha, "Flowshop scheduling with artificial neural networks," *Journal of the Operational Research Society*, vol. 71, no. 10, pp. 1619–1637, 2020.

[3] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 1621–1632, 2020.

[4] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational Intelligence*. Springer, 2009, pp. 177–201.

[5] M. Qin, R. Wang, Z. Shi, L. Liu, and L. Shi, "A genetic programming-based scheduling approach for hybrid flow shop with a batch processor and waiting time constraint," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 1, pp. 94–105, 2019.

[6] Z. Shi, H. Ma, M. Ren, T. Wu, and J. Y. Andrew, "A learning-based two-stage optimization method for customer order scheduling," *Computers & Operations Research*, vol. 136, p. 105488, 2021.

[7] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2013.

[8] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter, "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems," *International Journal of Production Economics*, vol. 145, no. 1, pp. 67–77, 2013.

[9] R. Hunt, M. Johnston, and M. Zhang, "Evolving" less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 2014, pp. 927–934.

[10] Z. Shi, S. Gao, J. Du, H. Ma, and L. Shi, "Automatic design of dispatching rules for real-time optimization of complex production systems," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019, pp. 55–60.

[11] K. Miyashita, "Job-shop scheduling with genetic programming," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2000, pp. 505–512.

[12] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2013.

[13] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling," in *European Conference on Evolutionary Computation in Combinatorial Optimization (Part of EvoStar)*. Springer, 2020, pp. 214–230.

[14] F. Zhang, Y. Mei, N. Su, and M. Zhang, "Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552–566, 2021.

[15] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*. ACM, 2010, pp. 257–264.

[16] C.-H. Chen, J. Lin, E. Yücesan, and S. E. Chick, "Simulation budget allocation for further enhancing the efficiency of ordinal optimization," *Discrete Event Dynamic Systems*, vol. 10, no. 3, pp. 251–270, 2000.

[17] F. Gao, Z. Shi, S. Gao, and H. Xiao, "Efficient simulation budget allocation for subset selection using regression metamodels," *Automatica*, vol. 106, pp. 192–200, 2019.

[18] Z. Shi, S. Gao, H. Xiao, and W. Chen, "A worst-case formulation for constrained ranking and selection with input uncertainty," *Naval Research Logistics (NRL)*, vol. 66, no. 8, pp. 648–662, 2019.

[19] Z. Shi, Y. Peng, L. Shi, C.-H. Chen, and M. C. Fu, "Dynamic sampling allocation under finite simulation budget for feasibility determination," *INFORMS Journal on Computing*, 2021.

[20] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, "Hyper-heuristic evolution of dispatching rules: A comparison of rule representations," *Evolutionary Computation*, vol. 23, no. 2, pp. 249–277, 2015.

[21] O. Holthaus and C. Rajendran, "Efficient jobshop dispatching rules: further developments," *Production Planning & Control*, vol. 11, no. 2, pp. 171–178, 2000.

[22] C. Rajendran and O. Holthaus, "A comparative study of dispatching rules in dynamic flowshops and jobshops," *European Journal of Operational Research*, vol. 116, no. 1, pp. 156–170, 1999.

[23] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *The International Journal of Production Research*, vol. 20, no. 1, pp. 27–45, 1982.