Can Stochastic Dispatching Rules Evolved by Genetic Programming Hyper-heuristics Help in Dynamic Flexible Job Shop Scheduling?

Fangfang Zhang, Yi Mei and Mengjie Zhang
School of Engineering and Computer Science
Victoria University of Wellington
PO BOX 600, Wellington 6140, New Zealand
{fangfang.zhang, yi.mei, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Dynamic flexible job shop scheduling (DFJSS) considers making machine assignment and operation sequencing decisions simultaneously with dynamic events. Genetic programming hyper-heuristics (GPHH) have been successfully applied to evolving dispatching rules for DFJSS. However, existing studies mainly focus on evolving deterministic dispatching rules, which calculate priority values for the candidate machines or jobs and select the one with the best priority. Inspired by the effectiveness of training stochastic policies in reinforcement learning, and the fact that a dispatching rule in DFJSS is similar to a policy in reinforcement learning, we investigate the effectiveness of evolving stochastic dispatching rules for DFJSS in this paper. Instead of using the "winner-takes-all" mechanism, we define a range of probability distributions based on the priority values of the candidates to be used by the stochastic dispatching rules. These distributions introduce varying degrees of randomness. We empirically compare the effectiveness of GPHH in evolving the stochastic dispatching rules with different probability distributions, as well as evolving the deterministic dispatching rules. The results show that the evolved deterministic rules perform the best. We argue that this is because unlike the traditional reinforcement learning methods, the current GPHH does not store the quality (value function) of any particular state and action during the simulation, and thus cannot fully take advantage of the feedback given by the simulation. In the future, we will investigate better ways to make better use of the information during the simulation in GPHH to further improve its effectiveness.**

I. Introduction

Scheduling is one of the most important tasks that allocates resources to jobs, which benefits manufacturing a lot (e.g. reducing production costs, improving delivery speed and customer satisfaction). In job shop scheduling (JSS), machines are the most important resources in a job shop floor and a set of jobs (i.e. each job has a sequence of operations) are expected to be processed by a number of machines. JSS aims to maximise the efficiency of machines, thus to reduce the costs. An operation can only be processed on a specific machine in classical JSS.

In the literatures, different types of JSS such as dynamic JSS (DJSS) and flexible JSS (FJSS), which are the extensions of classical JSS, have been investigated. In DJSS, dynamic events such as machine breakdown [1] and job arrival over time [2], [3] are considered during the scheduling process. In FJSS, the machine resource is more flexible that an operation can be processed on a set of machines. This leads itself to a more complex problem. In FJSS, two decisions, which are machine assignment decision and operation sequencing decision, have to be decided. Machine assignment (*routing*) is to allocate a ready operation to an appropriate machine while operation sequencing (*sequencing*) aims to select one operation in its queue of a specific machine as the next operation to be processed. FJSS is an NP-hard problem [4]. The investigation on dynamic flexible job shop scheduling (DFJSS) is more practical. However, most of the related work about FJSS [5], [6] are conducted in static environments which are not the normal cases in real-world. The research on DFJSS is still on a very early stage. In this paper, for DFJSS, only job arrival event is considered because of it is the most frequent and common factor in the shop floor.

Many techniques to search for optimal solutions, which are known as *exact approaches* such as dynamic programming [7] and branch-and-bound [8], have been investigated to solve JSS problems. However, they are too time-consuming, especially when the problems are getting large. *Heuristic search methods* such as tabu search and genetic algorithms have been proposed to find "near-optimal" solutions for solving JSS problems [9], [10] with a reasonable time. However, it is hard for the above methods to handle dynamic events where a lot of real time decisions are needed to be made quickly. It is noted that *dispatching rules* have been widely adopted for solving DJSS problems due to their ability to handle dynamic events, ease of implementation and low time complexity. In general, a dispatching rule, as a priority function, is used to calculate the priority value of each candidate (e.g. each job in the queue when a machine becomes idle), and then decide the next candidate based on the priority values.

Manually designing dispatching rules is time consuming, and heavily rely on human expertise and domain knowledge. In the last decade, genetic programming (GP) has been the dominating technique to automatically evolve dispatching rules for JSS. A general GP-based hyper-heuristic (GPHH) framework was presented in [11]. The unique feature of GPHH approach is that the search space is heuristic rather solution. The main

goal is to improve the generalisation of the evolved dispatching rules. It has been successfully used in some applications such as timetabling [12] and job shop scheduling [13], [14]. For DFJSS, we have two types of rules to evolve, i.e. the routing rule and the sequencing rule. However, the previous studies mainly focuses on evolving sequencing rule by fixing the routing rule [6]. Actually, this can be done by either co-evolving them by cooperative co-evolution [15], or using multi-tree GP (MTGP) [2] that stores two trees in each GP individual. Thus, the routing rule and sequencing rule can be evolved simultaneously. This paper will conduct based on MTGP framework which is the state-of-the-art method for DFJSS.

Essentially, a dispatching rule plays a role similar to a policy in reinforcement learning, which decides the next action to take given any state during the simulation. In DFJSS, the simulation is a dynamic flexible scheduling process with unpredicted job arrivals. The goal of GPHH is to evolve the best dispatching rule that makes the best decision in each decision situation during the simulation, and finally leads to the solution (i.e. schedule) with the optimal objective value. In other words, to evolve the best dispatching rule, we also aim to identify the best decisions during the simulation. However, due to the dynamic nature of the simulation, it is impractical to know the best decision on-the-fly. Therefore, given any particular decision situation during the DFJSS simulation, it may be necessary to explore different decisions to gain a higher confidence on the best decision to make.

To address this issue, a commonly used strategy in reinforcement learning is to train a stochastic policy [16], which selects from the candidate actions based on some probability distributions. In this way, each candidate action has a non-zero probability to be selected, and thus the corresponding solution has a chance to be explored. However, in the context of evolving dispatching rules using GPHH, the previous studies mainly focused on evolving deterministic dispatching rules with the "winner-takes-all" strategy (i.e. always select the candidate with the best priority). Inspired by the success of training stochastic policy in reinforcement learning and the similarity between dispatching rules and policies, it is reasonable to believe that stochastic dispatching rules should be able to explore more potential solutions than deterministic dispatching rules.

This paper aims to use GPHH to evolve stochastic dispatching rules. It is worth mentioning that deterministic dispatching rules are a special type of stochastic dispatching rules. Different probability distributions (multinomial distribution and the distribution provided by softmax function, representing different degrees of introduced randomness) will be used by the stochastic rules. The GP-evolved stochastic rules will be analysed. Specifically, we aim to answer the following question.

Whether stochastic dispatching rule can perform better than deterministic dispatching rule?

Based on the question, in particular, it has the following research objectives in this paper.

1) Propose a new flexible GPHH framework to evolve stochastic dispatching rules.
2) Define a range of probability distributions to be used by the stochastic rules to make decisions (including the top-k multinomial distribution and softmax function).
3) Empirically compare the influence of different probability distributions (different degrees of randomness) on the performance of the GPHH.
4) Analyse the GPHH-evolved stochastic rules, in particular the convergence of the decisions they make.
5) Investigate the effect on the performance if we turn the stochastic dispatching rule into deterministic rule during testing.

The rest of the paper is organised as follows. In Section II, a brief introduction of the JSS problem, GPHH and the mechanism of dispatching rules for FJSS are given. Detailed descriptions of the proposed stochastic policy are given in Section III. The design of experiments is shown in Section IV and results with discussions are provided in Section V. Finally, Section VI concludes the paper.

II. Background

In this section, different types of JSS problems will be described firstly. Then, the GPHH approach will be illustrated followed by the mechanism of dispatching rule for solving the DFJSS problem.

A. Job Shop Scheduling

1) Classical Job Shop Scheduling: In the classical JSS, n jobs need to be scheduled on m machines, while trying to minimise the objective such as the makespan (i.e. the total time to completely process all jobs). For each job, there is a set of operations which need to be executed in a specific order and each operation can be processed at a specified machine. In essence, the classical JSS assumes that only one machine is able to process a particular operation. Some basic or commonly used definitions and notations are described as follows.

parameters:

- n: the number of jobs in the job shop
- m: the number of machines in the job shop
- M: the set of machines in the job shop
- N: the set of operations in the job shop
- i: index of job
- j: index of operation
- l_j: the number of operations for each job, $l_j <= m$
- $O_{ij} = (O_{i1}, O_{i2}, ..., O_{il_j})$: the set of operations of job_i
- w_i: the weights of job_i
- d_j: the due date of job_i
- $\delta(O_{ij})$: the processing time of operation O_{ij}
- $m(O_{ij})$: the machine that processes operation O_{ij}
- $\pi(O_{ij})$: the optional machines of O_{ij}, $\pi(O_{ij}) \in M$. This parameter is used in flexible JSS which will be described later.

variables:

- C_i: the completion time of job_i
- $r(O_{ij})$: the release time of operation O_{ij}. It is the time that jth operation of job_i is allowed to start. In our research, for the first operation of each job, it is set to zero. Otherwise, it is set to the completion time of its preceding operation.

constraints:

- The $(j+1)th$ operation of job_i (denotes by $O_{i(j+1)}$) can only be processed after its preceding operation O_{ij} has been processed.
- Each machine can only process at most one operation at a time.
- The scheduling is non-preemptive, i.e. the processing of an operation cannot be stopped or paused until it is completed.

objectives:

- Minimisation the maximum makespan: $C_{max} = max\{C_1, C_i, ..., C_n\}$
- Minimisation of mean tardiness: $\frac{\sum_{i=1}^{n} max\{C_i - d_i, 0\}}{n}$
- Minimisation the maximum flowtime: $max_{i\in\{1,2,...,n\}}\{C_i - r_i\}$
- Minimisation of mean flowtime: $\frac{\sum_{i=1}^{n}\{C_i - r_i\}}{n}$
- Minimisation of mean weighted flowtime: $\frac{\sum_{i=1}^{n} w_i * \{C_i - r_i\}}{n}$

The last three objectives are used as the measures of schedules obtained in this paper. Classical job shop scheduling implies that the information of all the jobs is known in advance for making decisions. With available information, this makes it easier for us to get a promising schedule. However, this is not practical in the real-world factories.

2) Dynamic Job Shop Scheduling: In practice, the environment is usually dynamic and jobs arrive in the job shop over time without prior information. In fact, the classical JSS problem is static JSS. In DJSS, at any given time point, only the information of the jobs that have arrived before the time point is available. This characteristic is opposite to static job shop scheduling.

3) Flexible Job Shop Scheduling: The FJSS problem is an extension to classical JSS problem. However, the machine resource is more flexible and an operation can be processed on more than one machine, which leads itself to a more complex problem. Thus, except for the constraints mentioned above, for FJSS, there is a relaxation of the machine resource constraint.

- Each operation O_{ij} can be processed on one of the corresponding set of machines $\pi(O_{ij}) \in M$ with $\delta(O_{ij})$.

In order to tackle the FJSS problem, two decisions, which are a machine assignment decision and an operation sequencing decision, have to be made. The machine assignment decision is to allocate a ready operation to an appropriate machine while the operation sequencing decision aims to select one operation as the next to be processed when a machine becomes idle and there are operations in its queue.

4) Dynamic Flexible Job Shop Scheduling: DFJSS considers both the characteristics of FJSS and DJSS. DFJSS is more challenging since not only does the specific machine need to be determined but also the processing sequence of operations should be decided simultaneously along with the new arrival jobs over time. DFJSS is strongly NP-hard [17].

B. Genetic Programming Hyper-heuristic

Hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems [11]. There are two main categories of hyper-heuristics, which are *heuristic selection* and *heuristic generation* [18]. Heuristic selection methodologies aim at choosing or selecting existing heuristics while heuristic generation methodologies aim at generating new heuristics from components of existing heuristics. Heuristic generation is popular adopted to generate more comprehensive rules for JSS. The area of automated heuristic or *hyper-heuristics* [18] have been proven to be promising for designing heuristics. The main difference between hyper-heuristic methods and other methods is that hyper-heuristic methods explore the "heuristic search space" rather than the solution space. A most state-of-the-art survey of existing studies on hyper-heuristics and its applications can be found in [19], [20].

Algorithm 1: Pseudo-code of GP to evolve dispatching rules for JSS

```
   Input  : Training instances
   Output: The best evolved rule ind*
1: while N_ind < Popsize do
2:     Initialisation: Randomly initialise each individual
3: end
4: set ind* ← null and fitness(ind*) ← +∞
5: gen ← 0
6: while gen < maxGen do
7:     Evaluation: Evaluate the individuals
8:     for i = 1 to |Individual| do
9:         if fitness_i < fitness_ind* then
10:            ind* ← ind_i
11:        end
12:    end
13:    Evolution: Generate new population by genetic operators
14:    gen ← gen + 1
15: end
16: return ind*
```

GP is a domain-independent approach that can automatically generates computer programs to solve problems. The flexible representation of GP makes it a good candidate to evolve dispatching rules (hyper-heuristic) for the JSS problems. As a population based evolutionary computation technique, the main steps of GP for evolving dispatching rules are shown in Algorithm 1.

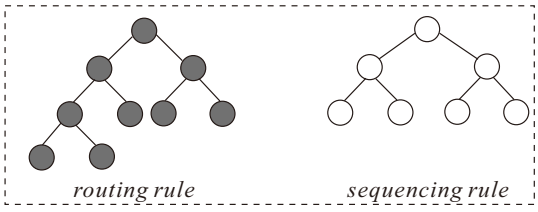


Fig. 1. An example of one individual of GP for DFJSS.

It is noted that in order to handle the DFJSS problems, each individual consists of two trees (i.e. one for evolving routing rule and the other for evolving sequencing rule). An example of one individual of GP for DFJSS is shown in Fig. 1.

C. Rules for Dynamic Flexible Job Shop Scheduling

In order to meet the sequence requirement of a job, only *ready operations* can be assigned to machines. Naturally, the ready operations have two sources. One is the first operation of each new job. The other is the operation whose proceeding operations have been just finished.

Routing rules and sequencing rules are needed in DFJSS. It is noted that the dispatching rule used in DJSS in previous research takes the same role as the sequencing rule in DFJSS. To avoid confusion, we use the term *dispatching rule* to contain two kinds of rules (i.e. routing rule and sequencing rule) in DFJSS. Once an operation becomes ready operation (i.e. routing decision situation), a routing rule will be triggered to decide which machine to allocate this operation. Sequencing rule will be triggered to determine which operation will be chosen to process next on a specific machine, when a machine becomes idle and its queue is not empty (i.e. sequencing decision situation).

III. Evolving Stochastic Dispatching Rule

The process of generating a schedule is similar to the process of finding the best trajectory among lots of operations. Deterministic dispatching rules always choose the operation with highest priority value as the next to be processed, which is too greedy to get a good solution in a stochastic environment. Probability is the measure of the likelihood that an operation will be chosen. Stochastic dispatching rule based on probability distribution evolved by GPHH might increase the diversity of trajectory and find a better schedule. Accordingly, we design a stochastic dispatching rule, and the pseudo-code of stochastic operation sequencing decision making process is shown in Algorithm 2.

Algorithm 2: Pseudo-code of the process of operation sequencing

Input : Sequencing decision situations
A sequencing rule
Output: The selected operation O^*

1: **while** *a machine becomes idle and its queue is not empty* **do**
2: Calculate the priority value for each operation in its queue
3: Calculate the **probability** for each operation based on its priority value
4: Choose the corresponding operation O^* based on its probability
5: **end**
6: **return** O^*

A key step in Algorithm 2 is the way to calculate the probability for each operation (line 3) based on its priority value. It is important to design a proper probability distribution to control the degree of introduced randomness. In this paper, we will investigate the effectiveness of a variety of mechanisms to set the probabilities. The probability should be positively correlated with priority value to make sure that operations with higher priority are more likely to be selected. A higher priority value gives the operation a higher precedence.

In this section, two kinds of probability distributions are investigated. Then, the designed top-k selection with proposed probability distributions is introduced followed by the summary of the characteristics of different strategies.

A. Multinomial Distribution

The most intuitive mechanism is to set the probability of each operation proportional to its priority value. The detailed steps of the multinomial distribution (i.e. also known as proportional distribution) strategy is given as follows.

Step 1: Normalise priority values of operations into the range [0, 1].

Normalisation is necessary because the priority values obtained by dispatching rule evolved by GPHH can be negative value, zero and positive value. This can be generalised to restrict the range of values using Eq. (1).

$$nPri_i = \frac{(Pri_i - Pri_{min})}{Pri_{max} - Pri_{min}} \tag{1}$$

For the convenience of explanation, Pri_i and $nPri_i$ are used to indicate the priority value of operation i obtained by sequencing rule and the normalised priority value of operation i. Pri_{max} and Pri_{min} are the maximal and minimal priority values of the operations in the queue, respectively.

It is worth mentioning that there are two special cases (i.e. both will cause the denominator in Eq. (1) to be zero) that we should take care when using Eq. (1). Firstly, if there is only one operation (i.e. $Pri_{max} = Pri_{min}$) in the queue of an idle machine, this unique operation will be chosen directly. Secondly, if there are more than one candidate operations and $Pri_{max} = Pri_{min}$, which means all the candidate operations have the same priorities, all the operations can be selected with same probability. For example, if there are six operations in the queue with same priority values, each operation will be chosen with a probability 1/6.

Step 2: Calculate the sum of priority values of candidate operations.

$$sumPri = \sum_{i=1}^{|N|} nPri_i \tag{2}$$

In Eq. (2), $sumPri$ stands for the sum of all the normalised priority values of candidate operations and $|N|$ is the number of candidate operations in the queue.

Step 3: Calculate the probability of each operation.

In Eq. (3), $Prob_i$ stands for the probability of operation i.

$$Prob_i = \frac{nPri_i}{sumPri} \tag{3}$$

Following these three steps, the probability of an candidate operation will be obtained. In general, the probability of an operation can be presented as Eq. (4). This is a multinomial distribution. The operation sequencing decision will be made based on the obtained probabilities.

$$Prob_i = \frac{nPri_i}{\sum_{i=1}^{|N|} nPri_i} \tag{4}$$

B. Softmax Function

In probability theory, the output of the softmax function can be used to represent a categorical distribution [21]. That is, a probability distribution over K different possible outcomes. In the field of reinforcement learning, it can be used to convert values into action probabilities. The main advantage of using softmax function is the output probabilities range. The range will be [0, 1], and the sum of all the probabilities will be equal to one.

$$Prob_i = \frac{e^{Pri_i}}{\sum_{i=1}^{|N|} e^{Pri_i}} \tag{5}$$

The softmax probability of an operation is shown as Eq. (5). It is worth mentioning that a high value will have a $higher$ probability than other values. Compared with multinomial distribution, the probability distribution provided by softmax function biases more on operations with higher priority values.

C. Top-k Selection with Probability Distributions

In order to bias more towards the operations with higher priority values, and remove the noise introduced by the less prioritised operations, we consider to only keep the top-k operations in terms of the priority value. Then, we can set the probability of each top operation based on the predefined distribution, such as the multinomial distribution and softmax function. Obviously, the aforementioned multinomial distribution and softmax function can be considered as a special type of the top-k selection, where k is a sufficiently large number.

D. Summary

Overall, the proposed probability distributions introduce different degrees of randomness.

- With the same distribution, a smaller k value in the top-k selection leads to a lower degree of randomness.
- The deterministic dispatching rule is a special type of stochastic dispatching rule with $k = 1$, and has the least degree of randomness.

IV. Experiment Design

We use the MTGP [2] as the framework for evolving the rules. The fitness of an individual in the GP population is evaluated using discrete-event simulations of problem instances of a job shop. In order to verify the performance and robustness of proposed stochastic policies, six scenarios are designed based on three objectives (e.g. max-flowtime (Tmax), mean-flowtime (Tmean) and mean-weighted-flowtime (WTmean)) and two utilisation levels (e.g. 0.85 and 0.95). In our experiment, 50 independent runs are executed, which assures that the results represent the average behaviour and not extreme situations.

A. Simulation Configuration

In the job shop, there are ten machines, which has been proven to be a good showcase for job shop environment. For dynamic simulation, commonly used configuration is adopted [22]. In this paper, the task is to process 5000 jobs by ten machines. Jobs with different weights will arrive stochastically according to a Poisson process with rate λ. Weight is a measure of the urgency or importance of jobs and 20%, 60% and 20% jobs will get weight 1, 2 and 4, respectively. The number of operations of a job and the number of candidate machines of an operation follows a uniform discrete distribution between one and ten. The processing time of an operation will be generated by uniform discrete distribution between one to 99. It is noted that the processing time of an operation is the same on all the candidate machines in this paper.

The utilisation is the proportion of time p that a machine is busy. The expression is shown in Eq. (6). In Eq. (6), μ is the average processing time of machines. P_M is the probability of a job visiting a machine. For example, P_M is 2/10 if each job has two operations.

$$p = \lambda * \mu * P_M \tag{6}$$

In order to get a steady state, a warm up period of 1000 jobs is used and we collect data from the next 5000 jobs. The new jobs keep coming until the 6000th job is finished.

B. Parameter Settings

In our experiment, the terminals are adopted in [22]. The details are shown in Table I. The function set is $\{+, -, *, /, max, min\}$, following the setting in [22]. The arithmetic operators take two arguments. The "/" operator is protected division, returning 1 if dividing by zero. The max and min functions take two arguments and return the maximum and minimum of their arguments, respectively.

TABLE I
THE TERMINAL SET.

Notation	Description
NIQ	The number of operations in the queue
WIQ	Current work in the queue
MWT	Waiting time of a machine
PT	Processing time of an operation on a specified machine
NPT	Median processing time for the next operation
OWT	The waiting time of an operation
WKR	Median amount of work remaining for a job
NOR	The number of operations remaining for a job
W	Weight of a job
TIS	Time in system

The initial population is generated using the ramped-half-and-half method with minimum depth of two and maximum depth of six. The population size is 1024 and the maximum generation is 51. The GP trees have a maximum depth of eight. For the genetic operators, the crossover, mutation and reproduction rates are 0.80, 0.15 and 0.05, respectively. The rates of terminal and non-terminal selection are 0.10 and 0.90. Tournament selection with a tournament size of seven is used to select individuals for genetic operators, which is a common setting in previous work [23].

V. Results and Analyses

For the convenience of description, the MTGP approach with multinomial distribution and softmax function are named as mMTGP and sMTGP, respectively. The approaches with respect to the test performance are investigated and discussed.

TABLE II
THE MEAN AND STANDARD DEVIATION OF THE OBJECTIVE VALUE OF MTGP WITH **MULTINOMINAL DISTRIBUTIONS** OVER 50 INDEPENDENT RUNS FOR SIX DYNAMIC SCENARIOS.

Scenario	Mean(StdDev)			
	k = all	k = 3	k = 2	k =1
<Tmax,0.85>	1098.14(73.20) **(+)**	1048.63(69.86) **(+)**	1011.68(52.83)	1008.95(54.91)
<Tmax,0.95>	1573.64(80.92) **(+)**	1418.82(51.56)	1392.88(51.21)	1405.26(44.30)
<Tmean,0.85>	365.42(2.83)	365.55(2.88)	365.71(3.07)	365.47(3.07)
<Tmean,0.95>	483.24(7.12)	485.43(7.89)	483.77(7.00)	482.52(4.91)
<WTmean,0.85>	791.93(7.17)	791.64(6.05)	790.64(6.12)	790.02(5.31)
<WTmean,0.95>	1007.39(17.38)	1003.88(14.67)	1008.16(18.04)	1003.84(17.44)

TABLE III
THE MEAN AND STANDARD DEVIATION OF THE OBJECTIVE VALUE OF MTGP WITH **SOFTMAX FUNCTION** OVER 50 INDEPENDENT RUNS FOR SIX DYNAMIC SCENARIOS.

Scenario	Mean(StdDev)			
	k = all	k = 3	k = 2	k =1
<Tmax,0.85>	1953.52(652.67)(+)	1068.48(68.37)(+)	1047.02(61.66)(+)	1008.95(54.91)
<Tmax,0.95>	6804.21(6157.54)(+)	1430.62(66.42)	1443.74(65.34)(+)	1405.26(44.30)
<Tmean,0.85>	381.57(8.82)(+)	365.50(2.74)	365.63(3.05)	365.47(3.07)
<Tmean,0.95>	587.87(161.43)(+)	485.31(9.28)	483.54(6.77)	482.52(4.91)
<WTmean,0.85>	833.44(23.77)(+)	792.19(6.90)	791.38(6.32)	790.02(5.31)
<WTmean,0.95>	1258.08(256.76)(+)	1004.93(16.09)	1003.15(16.15)	1003.84(17.44)

In the following results, "-, +" means the result is significantly better or worse than the counterpart in Wilcoxon rank sum test with a significance level of 0.05.

A. Test Performance of Evolved Stochastic Rules

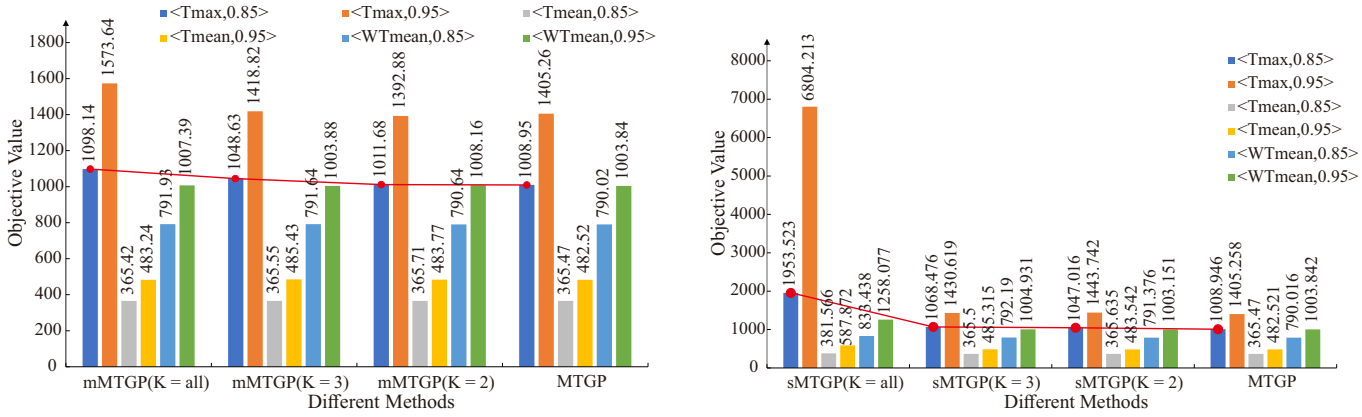
Table II and Table III show the mean and standard deviation of the objective value of the stochastic dispatching rule obtained by MTGP with multinominal distribution and softmax function, respectively. It is observed that the MTGP evolving deterministic dispatching rules is indicated as $k = 1$. Statistical analysis in Table II shows that MTGP with multinominal distributions are significantly worse than deterministic policy in three cases. In other cases, there is no significantly difference among them.

Table III shows that for MTGP with softmax function, in most cases of $k = 3$ and $k = 2$, the performance is not significantly worse than MTGP. From the perspective of mean values, they are slightly worse than MTGP in most scenarios (e.g. <Tmean,0.85>, <Tmean,0.95>, <WTmean,0.85> and <WTmean,0.95>). In scenario <WTmean,0.95>, when $k = 2$, the obtained mean and standard deviation are slightly better than that of deterministic policy. However, in the cases of $k = all$ of MTGP with proposed probability distributions, MTGP with softmax function performs significantly worse than MTGP and MTGP with multinomial distribution. That is, the performance of MTGP with softmax function becomes worse when taking all the operations as candidates. MTGP with softmax function losses its advantage in this case. *In general, in top-k strategies, GPHH with softmax function and GPHH with multinomial distribution have no significantly difference. However, taking all operations into account, GPHH with multinomial distribution is better than its counterpart.* This may be because multinomial distribution always ignore the worst one (zero probability), but softmax function still gives a positive probability to any of the operations. From this point of view, multinomial distribution could be less random than softmax and achieve better results in this case.

In addition, both Table II and Table III show that max-flowtime is more sensitive than mean-flowtime and mean-weighted flowtime when randomness is introduced, especially in the scenario <Tmax,0.95> ($k = all$) of MTGP with softmax function. The results also show that the performance becomes worse as the value of k increases. It means if the degree of randomness is too large, it will bring more noise and even behaves as a random schedule, which leads itself to get worse performance. *This suggests that using probability distribution with higher randomness is too random to get better performance than focusing on several top ranked operations.* The randomness involved should be well controlled.

In conclusion, deterministic dispatching rules are likely to be better (at least not worse) than stochastic dispatching rule in the scenarios we investigated. This may be because deterministic dispatching rules can assign each individual an exact fitness value rather than a stochastic fitness value, which is conducive to the execution of GPHH. This is much more obvious for the Tmax objective. The maximal flowtime is much more sensitive to the decisions than Tmean and WTmean, since it highly depends on the extreme case. Therefore, it is much more important to have an accurate evaluation for Tmax than the other two objectives.

Fig. 2 shows the test performance of different methods based on sMTGP and mMTGP. From the left to right, the methods perform better and better. On the other hand, a method more to the right tends to be focused more on exploitation (more greedy) than a method to its left. Therefore, one can see that introducing randomness in the decision making of dispatching rules in GPHH does not help at all. Overall, stochastic dispatching rules evolved by GPHH cannot help achieve better performance. It might because the underlining



(a) MTGP with multinomial distribution.

(b) MTGP with softmax function.

Fig. 2. The test performance of MTGP with multinomial distribution and softmax function.

training processes of GPHH and traditional reinforcement learning are so different. GPHH can generate new rules in a much more random way (i.e. crossover and mutation), but traditional reinforcement learning is much more greedy (i.e. based on gradient). Thus, we cannot see obvious benefit of introducing further randomness by using stochastic policy which can be great benefit to make traditional reinforcement learning less greedy.

On the other hand, the underlying difference between the training processes of GPHH and traditional reinforcement learning also suggests that there could be some ways to improve the current GPHH to further take advantage of the detailed feedback from the simulation, such as the value function of the states and actions during the simulation, and adjust the dispatching rules in a more intelligent way.

B. Insight of the Decision Making Process

In order to make a better understanding of the difference of making decisions between MTGP with multinomial distribution and softmax function, we choose mMTGP with $k = 3$ (i.e. the objective value is 1003.88) and sMTGP with $k = 3$ (i.e. the objective value is 1004.93) in scenario <WTmean,0.95> as an example. The probability calculations of the three candidate operations are shown in Table IV. The main difference is that

TABLE IV
THE DIFFERENCE OF PROBABILTY CALCULATIONS BETWEEN MTGP WITH MULTINOMIAL DISTRIBUTION AND SOFTMAX FUNCTION OF $k = 3$.

Operation	Priority Value	Probability	
		Multinomial Distribution	Softmax Function
O_{31}	159.65	0.7559	0.9812
O_{42}	155.56	0.2441	0.0165
O_{51}	153.61	0	0.0023

multinomial distribution still gives job with smaller priority value relatively larger chance than that of softmax function to be selected. It means softmax function is more greedy than multinomial distribution with the same k value. Thus, softmax function should be more similar with deterministic policy. However, it is very interesting that when taking all candidate operations into account, MTGP with softmax function perform worse than MTGP with multinomial distribution. As we can see from Table IV, even the three priority values are very close, multinomial distribution still ignore the worst operation, which will lead to less randomness than softmax.

C. Stochastic Dispatching Rule in Test Process

Another issue here is that whether probability distribution should be included in test process. The results above are conducted without randomness (i.e. probability distribution) in test process. In other words, the rules simply selects the operation with the highest priority value during the test simulations. Table V shows the test results with probability distribution and without probability distribution. In general, there is no significantly difference because testing with stochastic policy and testing without stochastic policy except for the scenario <Tmax,0.85> and <Tmax,0.95> in mMTGP($k = 3$). It can be seen that in test process of MTGP with softmax function, the mean values without probability distribution in test are slightly smaller than that with probability distribution in half scenarios of $k = 3$ and almost all scenarios of $k = 2$ (in bold). For MTGP with multinomial, it shows the same pattern.

In general, the results indicate that testing the performance of the evolved rules, we normally can safely remove the stochastic policy without causing performance degradation. However, it does not mean we cannot get better performance with probability distribution. A learned probability distribution might help, which will be investigated in the future.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed a flexible GPHH framework for evolving stochastic dispatching rules for DFJSS, with deterministic dispatching rules as a special case. The experimental results on a wide range of probability distributions showed that unlike in reinforcement learning, where it is effective to train stochastic policies, under the current GPHH framework, simply introducing more randomness in the decision making of the dispatching rules is not an effective strategy. We argue that this is due to the underlying difference between the training process of GPHH and traditional reinforcement learning. On

TABLE V
THE MEAN AND STANDARD ERROR OF THE OBJECTIVE VALUE OF sMTGP AND mMTGP (WITH AND WITHOUT PROBABILITY DISTRIBUTION IN TEST PROCESS) OVER 50 INDEPENDENT RUNS FOR SIX DYNAMIC SCENARIOS.

Scenario	sMTGP(k=3)		mMTGP(k=3))	
	with	without	with	without
<Tmax,0.85>	1075.44(54.53)	**1068.48(68.37)**	1090.22(41.48) **(+)**	1048.63(69.86)
<Tmax,0.95>	1437.20(50.48)	**1430.62(66.42)**	1545.43(40.22) **(+)**	1418.82(51.56)
<Tmean,0.85>	365.47(2.65)	365.50(2.74)	365.85(2.66)	**365.55(2.88)**
<Tmean,0.95>	485.11(8.65)	485.31(9.28)	485.98(7.38)	**485.43(7.89)**
<WTmean,0.85>	792.52(7.14)	**792.19(6.90)**	791.97(6.11)	**791.64(6.05)**
<WTmean,0.95>	1004.44(16.10)	1004.93(16.09)	1005.32(13.21)	**1003.88(14.67)**
Scenario	sMTGP(k=2)		mMTGP(k=2)	
	with	without	with	without
<Tmax,0.85>	1051.51(52.54)	**1047.02(61.66)**	1011.68(52.83)	1011.68(52.83)
<Tmax,0.95>	1446.89(60.59)	**1443.74(65.34)**	1393.02(51.16)	**1392.88(51.21)**
<Tmean,0.85>	365.68(2.97)	**365.63(3.05)**	365.70(3.07)	365.71(3.07)
<Tmean,0.95>	483.57(6.60)	**483.54(6.77)**	483.74(7.02)	483.77(7.00)
<WTmean,0.85>	791.41(6.12)	**791.38(6.32)**	790.65(6.13)	**790.64(6.12)**
<WTmean,0.95>	1003.15(16.15)	1003.15(16.15)	1008.16(18.04)	1008.16(18.04)

one hand, GPHH is gradient free, and generates new rules by random crossover and mutation. These genetic operators may already provide sufficient exploration capability to GPHH to explore different decisions during the simulation. On the other hand, GPHH does not store the intermediate feedback such as the value function of the states and actions during the simulation, thus cannot adjust the behaviour of the rules smoothly based on such feedback. As a result, it appears to be the most effective to evolve deterministic dispatching rules in GPHH, and the stochastic decisions will mainly introduce noise in the fitness evaluation during GPHH.

Overall, the results suggest that it is not worthy evolving stochastic dispatching rules under the current GPHH framework. The discussions and analyses also suggest that we can potentially improve the current GPHH framework by considering more detailed information during the simulation (such as the value function in reinforcement learning), and use it to adjust the dispatching rules in a more intelligent way. We will investigate this direction as our future work.

REFERENCES

[1] I. Sabuncuoglu and M. Bayız, "Analysis of reactive scheduling problems in a job shop environment," *European Journal of Operational Research*, vol. 126, no. 3, pp. 567–586, 2000.

[2] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 472–484.

[3] ——, "Surrogate-assisted genetic programming for dynamic flexible job shop scheduling," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 766–772.

[4] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[5] N. B. Ho and J. C. Tay, "Evolving dispatching rules for solving the flexible job-shop problem." in *IEEE Congress on Evolutionary Computation*, 2005, pp. 2848–2855.

[6] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[7] D. P. Bertsekas, *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[8] E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.

[9] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European journal of Operational Research*, vol. 167, no. 1, pp. 77–95, 2005.

[10] J. Hurink, B. Jurisch, and M. Thole, "Tabu search for the job-shop scheduling problem with multi-purpose machines," *Operations-Research-Spektrum*, vol. 15, no. 4, pp. 205–215, 1994.

[11] 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.

[12] M. B. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memetic Computing*, vol. 1, no. 3, pp. 205–219, 2009.

[13] 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*. ACM, 2014, pp. 927–934.

[14] 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, 2014.

[15] D. Yska, Y. Mei, and M. Zhang, "Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling," in *European Conference on Genetic Programming*. Springer, 2018, pp. 306–321.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[17] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 1, pp. 1–13, 2002.

[18] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[19] 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, 2016.

[20] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, 2017.

[21] N. Ahmed and M. Campbell, "Multimodal operator decision models," in *American Control Conference, 2008*. IEEE, 2008, pp. 4504–4509.

[22] Y. Mei, S. Nguyen, and M. Zhang, "Evolving time-invariant dispatching rules in job shop scheduling with genetic programming," in *European Conference on Genetic Programming*. Springer, 2017, pp. 147–163.

[23] 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.