Genetic Programming with Multi-tree Representation for 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. Flexible job shop scheduling problem can be regarded as an optimization problem in production scheduling that captures practical and challenging issues in real-world scheduling tasks such as order picking in manufacturing and cloud computing. Given a set of machines and jobs, FJSS aims to determine which machine to process a particular job (by routing rule) and which job will be chosen to process next by a particular machine (by sequencing rule). In addition, dynamic changes are unavoidable in the real-world applications. These features lead to difficulties in real-time scheduling. Genetic programming (GP) is well-known for the flexibility of its representation and tree-based GP is widely and typically used to evolve priority functions for different decisions. However, a key issue for the tree-based representation is how it can capture both the routing and sequencing rules at the same time. In this paper, to address this issue, we proposed to use multi-tree GP (MTGP) to evolve both routing and sequencing rules together. In order to enhance the performance of MTGP algorithm, a novel tree swapping crossover operator is proposed and embedded into MTGP. The results suggest that the multi-tree representation can achieve much better performance with smaller rules and less training time than cooperative co-evolution for GP in solving dynamic FJSS problem. Furthermore, the proposed tree swapping crossover operator can greatly improve the performance of MTGP.

Keywords: Multi-tree representation · Flexible job shop scheduling · Dynamic changes · Genetic programming.

1 Introduction

The rapid development of globalization and information technologies has made our world a Global Village, where the interest of countries is interconnected. The core of the connection highly relies on international trade. Thus, it brings more opportunities and also thrives competition among companies. The study of allocating the jobs to machines and determining the order of processing the allocated jobs on each machine to optimize criteria such as flowtime, tardiness or customer satisfaction will benefit the companies by increasing their efficiency, profit or reputation.

Flexible job shop scheduling (FJSS) problem is an extension to classical Job Shop Scheduling (JSS) problem. The FJSS task, as its name suggests, assumes a more flexible situation. It reflects a production environment where it is possible to run an operation on more than one machine. This special trait causes the problem to become more complicated than classical JSS because we not only have to decide where to allocate jobs, but also need to decide which job to be processed next simultaneously. FJSS is an NP-hard problem [2].

In addition, dynamic changes are inevitable in the real-world applications. For example, it is obvious that job orders are unpredicted or cannot be accurately predicted for companies, especially taking uncertain factors such as price impact, asymmetric information, rush hours and indefinite events into consideration. That is to say, we could not know job information until the job arrives. Dynamic flexible job shop scheduling (DFJSS) was born for considering this situation.

All these characteristics make DFJSS problem much more challenging than standard JSS and FJSS problems. Thus, the exact optimization methods such as mathematic programming [15] are often inapplicable, especially to large scale instances. Under this circumstance, heuristic search methods such as tabu search [14], genetic algorithm [16], simulated annealing [18] become more and more popular. These methods can get better performance in achieving reasonable solutions in less time. However, the biggest drawback is their lack of capability to adapt to the dynamic environmental change.

In order to reduce computational complexity and cope with dynamic changes, Dispatching Rules (DRs) have been widely applied [6, 10, 13]. When a machine becomes idle and has waiting operations in its queue, DRs will be triggered to select the operation with highest priority to be processed next. In this way, computation is carried out only at each decision point and decisions can be made efficiently.

However, lots of DRs are designed manually [17] and manual design has its inherent weaknesses. For instance, it highly relies on domain knowledge and it is very demanding on labour and time. Fortunately, Genetic Programming (GP) has been proven to be an effective hyper-heuristic method, which can automatically design DRs for scheduling [1, 9, 10, 12] that are much better than the manually designed ones. However, the existing works mainly focus on evolving the *sequencing rule* (the rule to select which waiting operation will be processed next when a machine becomes idle) without considering the *routing rule* (the rule to select which machine will be chosen to allocate the ready operations).

For DFJSS, a crucial issue is how it can evolve both routing and sequencing rules simultaneously. The representation is the crux of the applicable algorithm. There are two main reasons. Firstly, an appropriate representation is definitely a rudimentary factor for an algorithm to build a solution. Secondly, the representation determines the size of the search space and there is a clear trade-off between the complexity of the representation and the ability of GP to explore the search space. These two facts foster the motivation to propose a more suitable representation for DFJSS. To our best of knowledge, Cooperative Co-evolution (CC) was firstly embedded into GP to evolve routing and sequencing rules together

[19]. The proposed CCGP in [19] is the current state-of-the-art algorithm of DFJSS. However, the CC approach cannot fully capture the interaction between the routing and sequencing rules. Research in this area is still in a very early stage and little work has been reported on this important aspect. Dealing with multiple interdependent decisions, especially in dynamic environment, is always difficult but also creates opportunities to find the real global optimal solution. This is particularly challenging when multiple decisions need to be made at the same time.

In this paper, GP with multi-tree representation is introduced to evolve routing and sequencing rules together and a novel tree swapping crossover operator is proposed to evolve more effective rules.

1.1 Goals

In this paper, we aim to find more effective routing and sequencing rules for DFJSS based on GP with a multi-tree representation. In particular, we have the following research objectives.

- Introduce GP with multi-tree representation (MTGP) for evolving the routing and sequencing rules simultaneously.
- Propose a novel tree swapping crossover operator for the MTGP algorithm according to the feature of DFJSS problem. The MTGP with the newly proposed tree swapping crossover is denoted as sMTGP.
- Compare the performance of MTGP, sMTGP and CCGP to verify the effectiveness of the multi-tree representation and the novel tree swapping crossover operator.
- Analyse the rules evolved by MTGP, sMTGP and CCGP.

1.2 Organization

The rest of the paper is organized as follows. Section 2 gives the background introduction. Then, the proposed MTGP and sMTGP are introduced in Section 3. Experiment is designed in Section 4, and results with discussions are shown in Section 5. Finally, conclusions and future work are given in Section 6.

2 Background

2.1 Dynamic Flexible Job Shop Scheduling

In the basic version of job shop scheduling (JSS) problem, n jobs need to be processed by m machines. Each job consists of a sequence of operations and a machine can process at most one operation at a time. For each operation, it can be processed at a specified machine. In essence, the JSS problem is based on the assumption that only one machine is able to run a particular operation.

FJSS breaks through the constraints of resources uniqueness: each operation can be processed by more than one machine and its processing time depends on the machine that processes it. Thus, FJSS can improve the production efficiency, shorten the ordering cycle and increase the rate of orders delivered on time.

In real life, industry is in a dynamic environment, for instance, in terms of a factory, the orders will arrive over time. Actually, there are some methods to predict the information of incoming jobs to reduce uncertainty, thus to improve the accuracy of decisions. However, the gap between prediction and reality is always inevitable and sometimes they have a very wide difference. It is indicated that when dealing with the real-world applications, dynamic changes should be taken into consideration.

2.2 Rules for Dynamic Flexible Job Shop Scheduling

This paper aims to involve two kinds of rules for DFJSS, which are routing and sequencing rules, to make decisions at decision points. A routing rule is triggered when a new job arrives or when an operation is completed and its next operation becomes ready to be processed to allocate ready operation to a particular machine. When a machine is free and there are operations waiting, an operation in its queue will be chosen by a sequencing rule to be processed next.

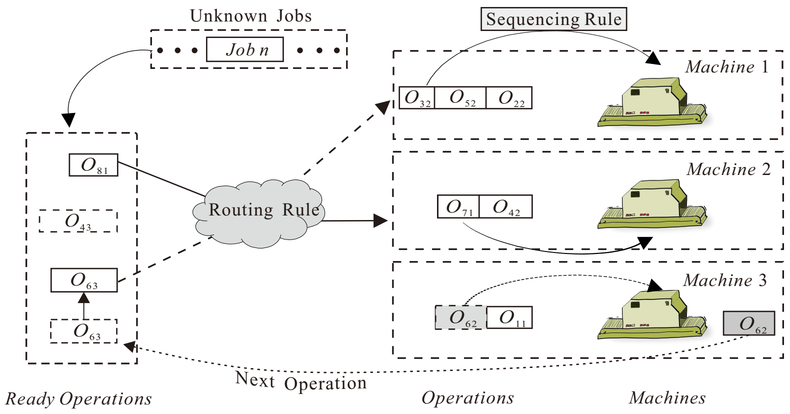

Fig. 1. An example of decision process of DFJSS.

Fig. 1 shows an example of decision process of DFJSS. In the figure, the solid lines stand for what is happening and the dotted lines indicate what will happen. There are three machines in the job shop and each job can be processed by any machine. Each job consists of several operations in a certain order. In the current system state, the operations (O_{32}, O_{52}, O_{22}, O_{71}, O_{42}, O_{62} and O_{11}) have been allocated to different machines by the routing rule. Then, each machine uses the sequencing rule to decide the next operation to be processed, e.g. *machine* 3 selects O_{62}. After O_{62} is completed, its subsequent operation O_{63} becomes ready, and will be allocated by the routing rule.

3 Genetic Programming with Multi-tree Representation

The choice of which representation to use when dealing with a problem using GP is vital. Tree-based GP is a popular way in previous research and multi-tree representation [7] as a special structure has been applied to classifier design [3, 11] and feature manipulation [8].

In multi-tree representation, each individual is represented as a list trees. Taking advantage of this feature to solve DFJSS problems, routing and sequencing rules can be denoted by different trees in one individual. According to this, multi-tree representation naturally lends itself to DFJSS. The pseudo-code of MTGP is given in Algorithm 1.

Algorithm 1: Pseudo-code of MTGP

```
// Initialization
1 while $N_{ind}$ < Popsize do
2     foreach individual
3         Initialize each tree //Randomly initialize each tree by ramp half-and-half
4 end
// Evolution
5 while Stopping criteria not met do
6     Evaluate the individuals
7     Copy the elites to the new population
8     Select individuals based on fitness value
9     Generate offsprings by applying crossover/mutation/reproduction operators
10 end
11 return best individual
```

In this paper, we use the multi-tree representation that one individual contains two trees to match our problems. To be specific, the first tree is used to indicate the sequencing rule and the second tree denotes the routing rule. The fitness of one individual depends on the two trees working together. In the case of multi-tree representation, the evolutionary algorithm must come to a decision as to which trees the genetic operator will be applied.

In classical multi-tree representation, the genetic operators are defined to act upon only one tree in an individual at a time. Other trees are unchanged and copied directly from the parents to the offsprings. Genetic operators are limited to a single type of trees at a time in the expectation that this will reduce the extent to which they disrupt "building blocks" of useful code. However, when coping with DFJSS problem, such a crossover operator has the following issues.

Firstly, the crossover operation only happens between one type of trees of the parents, therefore, the offsprings generated might not be substantially different from their parents. Thus, the population will lose its diversity and the ability of exploration will decrease.

Secondly, the crossover operation cannot improve the diversity of the combinations of routing and sequencing rules. In DFJSS, a good rule cannot be "good" by itself, but should behave well when collaborating with the other rule. Thus, the diversity of combinations is an important factor for achieving good solutions.

In order to overcome these shortcomings and make the algorithm more in line with the properties of DFJSS, a new tree swapping crossover operator is proposed. Fig. 2 shows the tree swapping crossover operator, which shares the same process with the classical crossover operator except that the unselected trees (the same type) are also swapped with each other. Fig. 2 shows two parents ($parent_1$ and $parent_2$) are selected to generate offsprings and the second type (T_2) of trees is selected for crossover. The dotted circles mean that the subtrees are chosen and will be swapped. The standard crossover operator will stop here. But for the tree swapping crossover operator, the other type of trees is also swapped. Thus, two offsprings ($Offspring_1$ and $Offspring_2$) are generated.

This will bring two benefits. The first is that useful blocks are not easily broken. The second is the pairs or combinations of routing and sequencing rules

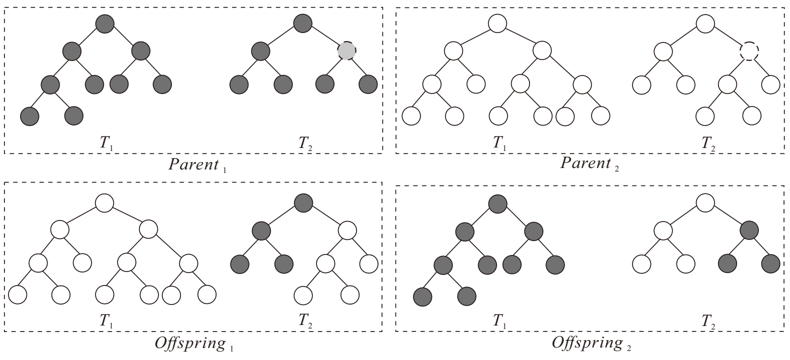

Fig. 2. Tree swapping crossover operator for multi-tree representation.

examined in sMTGP are much more diverse. That is to say, the population of sMTGP will become more diverse compared with MTGP. More importantly, this point matches well with the characteristics of the DFJSS.

4 Experiment Design

4.1 Parameter Settings

In our experiment, time-invariant terminals in [10], were adopted. The details are shown in Table 1. Six functions $\{+, -, *, /, max, min\}$ are selected in the function set, in which "/" is the protected division that returns the largest double positive number if divided by 0. All of them take two operands.

Table 1. The terminal set.

	Notation	Description
machine-related	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
job-related	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
system-related	TIS	Time in system

For fair comparison, the parameters in MTGP and sMTGP are the same as in [19]. 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 was set as parent selection method with a tournament size of 7.

The learning process continued until the generation met the maximum generation, which was set to 51. The 30 independent runs test results were reported as the system performance.

4.2 Simulation Configuration

For dynamic simulation, the configuration is given in Table 2, which has been commonly used in existing studies [5, 10]. In order to improve the generalization ability of the evolved rules, the seeds used to stochastically generate the jobs are rotated in the training process in each generation.

Table 2. Dynamic simulation configuration.

Parameter	Value
Number of machines	10
Number of jobs/warmup jobs	5000/1000
Number of operations per job	Uniform discrete distribution between 1 and 10
Available machines per operation	Uniform discrete distribution between 1 and 10
Job arrival process	Poisson process
Utilization level	0.85, 0.95
Processing time	Uniform discrete distribution between 1 and 99
Job weights	weight 1 (20%), weight 2 (60%), weight 4 (20%)
Due date factor	4

4.3 Comparison Settings

In our research, three algorithms were involved. CCGP [19] is built on GP with cooperative co-evolution and MTGP is the proposed algorithm that introduces GP with multi-tree representation to evolve routing and sequencing rules together. sMTGP is the improved MTGP with the tree swapping crossover. Moreover, a typical performance indicator for JSS is the flowtime, i.e., the sum of the total waiting time and the total processing time for one job. In this paper, we used three different kinds of variations of flowtime to measure the performance of the proposed algorithms, namely Max-Flowtime, Mean-Flowtime and Mean-weighted-Flowtime. Also, different scenarios were used to measure their robustness.

For the DFJSS problem, in our case, it is impossible to get the best known (lower bound) objective value of the instances. So, benchmark routing rule (LWIQ, Least Work in Queue, select the machine with the least work in its queue) and sequencing rules (SPT, Shortest Processing Time, choose the job with shortest processing time, for mean-flowtime; FCFS, First Come First Serve, the job comes first will be processed firstly, for max-flowtime and mean-weighted-flowtime) [4], were applied to get a baseline objective value for each instance. The reason for choosing them is that they show better performance than others in previous work [6] and often be chosen as benchmark rules [19]. Here, the relative performance ratio was defined as the average normalized objective value obtained by evolved rules over the counterpart got by benchmark rules. Thus, in our case, the smaller the fitness value, the better.

5 Results and Discussions

5.1 Optimization Performance

In our experiment, six scenarios were set to test the performance of MTGP, sMTGP and CCGP. The best pair of rules of the last generation was tested on test data set to measure its performance. The test data set consists of 50 dynamic simulations with different random seeds. In addition, Wilcoxon test at the 5% level was used for comparison between the three algorithms. First of all, MTGP and sMTGP were compared with CCGP respectively to measure the feasibility of multi-tree based GP. Then, sMTGP and MTGP were compared for analysing the effectiveness of proposed tree swapping crossover operator.

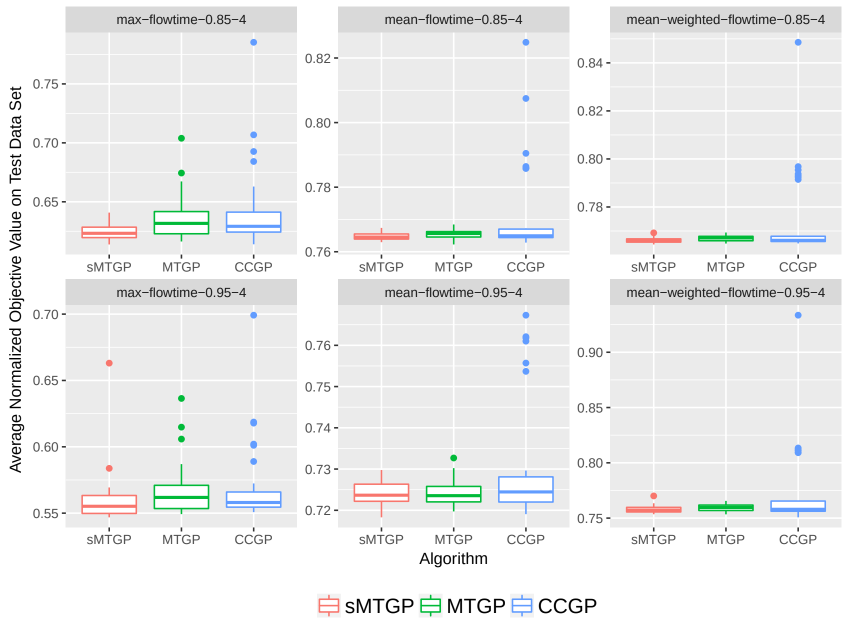

Fig. 3. The boxplot of average normalized objective value obtained by sMTGP, MTGP and CCGP on test data set.

All the mean value obtained by MTGP and sMTGP are better than CCGP and all the standard deviation value are smaller than the counterparts. Wilcoxon test results show that sMTGP is significantly better than CCGP only in two scenarios (Max-Flowtime-0.85-4, Mean-Flowtime-0.85-4). It is interesting that MTGP got better mean value than CCGP, but none of the instances of MTGP is significantly better than CCGP.

When further looking into the boxplot in Fig. 3, one can see that CCGP has many more outliers than MTGP and sMTGP. This is because CCGP cannot handle well the interactions between routing and sequencing rules directly, thus can be stuck into poor local optima more often. The reason why there is no statistical significance between MTGP and CCGP is that the two algorithms showed very similar performance except the outliers. Fig. 3 clearly shows that multi-tree representation managed to dramatically reduce the probability of outliers.

According to these observations, the performance of GP with the multi-tree representation is more stable than GP with cooperative co-evolution. Also, Wilcoxon test results show that sMTGP is significantly better than MTGP in four scenarios (Max-Flowtime-0.85-4, Mean-Flowtime-0.85-4, Mean-weighted-Flowtime-0.85/0.95-4). It means that the proposed tree swapping crossover operator can effectively improve the performance of MTGP.

Fig. 4. shows that the sizes of evolved best sequencing rules by sMTGP and MTGP are obviously and dramatically smaller than the best rules evolved by

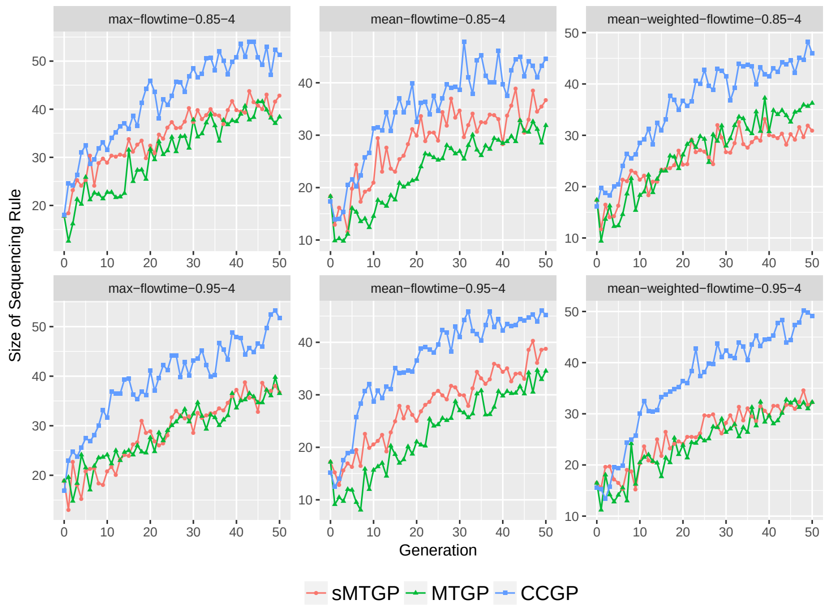

Fig. 4. The convergence curves of the average best sequencing rule size (30 runs) obtained by sMTGP, MTGP and CCGP in each generation.

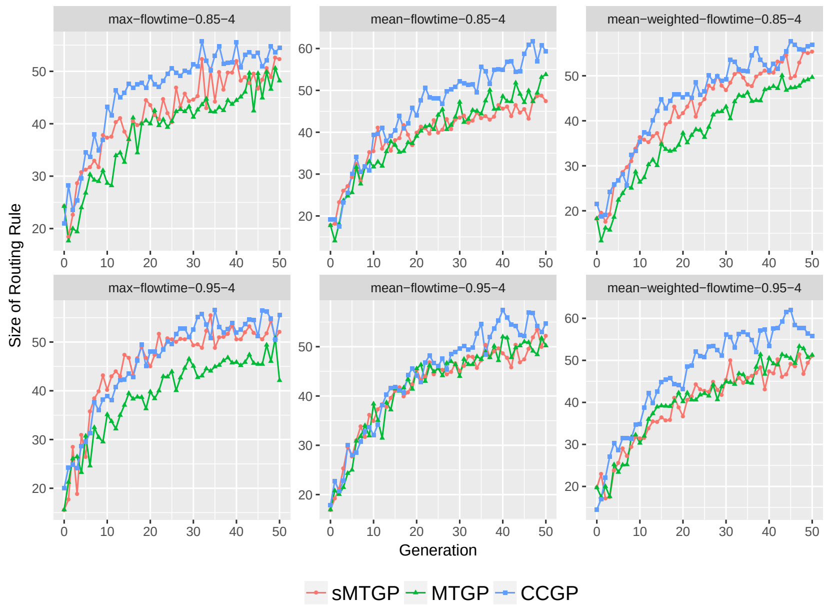

Fig. 5. The convergence curves of the average best routing rule size (30 runs) obtained by sMTGP, MTGP and CCGP in each generation.

CCGP. Also, Fig. 5 shows that the best routing rule sizes got by sMTGP and MTGP are smaller than that of CCGP. However, there is not so much difference compared with the changes of sequencing rule sizes. These observations confirm the potential of using multi-tree based GP to achieve smaller size rules.

From Table 3, it is clear that sMTGP and MTGP can evolve rules with lower time complexity than CCGP in all scenarios. In addition, for sMTGP, less training time is needed as compared to MTGP in three situations (scenario 3, 4, 6). This is a promising finding that the multi-tree representation is computationally cheaper than cooperative co-evolution.

Table 3. The average training time for each run of the three algorithms.

Index	Scenario	Training Time		
		sMTGP	MTGP	CCGP
1	Max-Flowtime-0.85-4	4459.9	4267.1	4642.8
2	Max-Flowtime-0.95-4	5057.2	4790.3	5144.9
3	Mean-Flowtime-0.85-4	4184.5	4278.0	4538.5
4	Mean-Flowtime-0.95-4	4667.6	4721.3	4849.9
5	Mean-weighted-Flowtime-0.85-4	4348.1	4181.7	4458.4
6	Mean-weighted-Flowtime-0.95-4	4585.7	4680.3	4957.0

Overall, MTGP and sMTGP (especially) undoubtedly show better ability to solve DFJSS problem. They can obtain better and smaller rules within a shorter training time.

5.2 Further Analysis

In the last section, the rule size relates to the best rule only. In order to explore whether the best rule is smaller by chance or the rules in the whole population generally become smaller, in this section, the average rule sizes in the whole population at each generation were investigated to get a clear vision of the changes of rule sizes.

We took the last scenario (Mean-Weighted-Flowtime-0.95-4) as an example to further investigate the changes of rule sizes. The details are shown in Fig. 6 and Fig. 7.

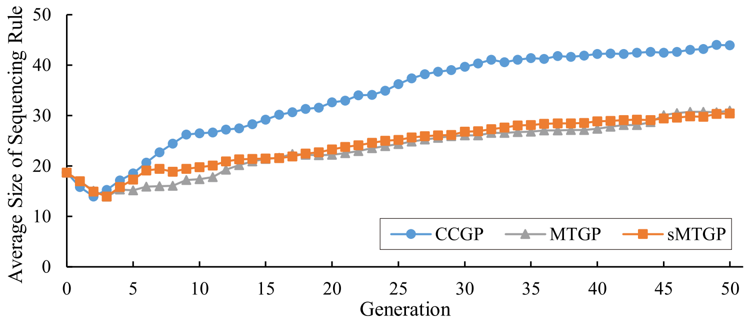

Fig. 6. The convergence curves of average sequencing rule size (30 runs) obtained by CCGP, MTGP and sMTGP in population in each generation.

As shown in Fig. 6 and Fig. 7, at the initial point, for all the three algorithms, the average sizes of both rules are about equal. However, the average sizes obtained by CCGP are larger than others over time. Maybe in multi-tree based GP, effective and smaller rules are more likely to be well preserved because there is at least one rule structure will not be changed by operator at each time during the evolution process. In addition, the average sizes obtained by MTGP and sMTGP show the same trend basically and routing rule sizes are bigger than sequencing rules. This is consistent with the observation in the last section.

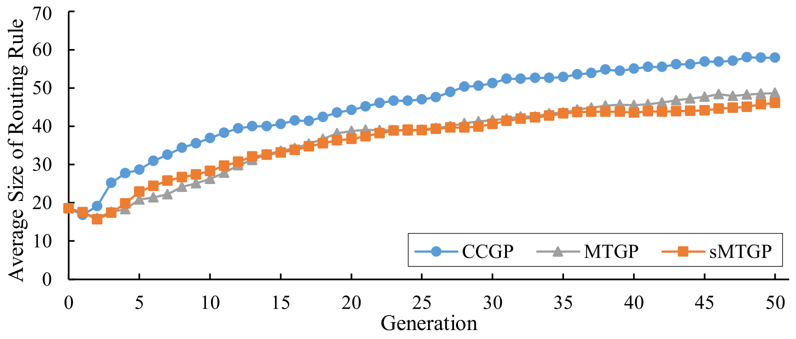

Fig. 7. The convergence curves of average routing rule size (30 runs) obtained by CCGP, MTGP and sMTGP in population in each generation.

6 Conclusions and Future Work

This paper tried to evolve routing and sequencing rules based on GP with multi-tree representation simultaneously, which is one of the very first piece of work in this field. From the experimental results, we got some interesting findings. Firstly, in addition to performance, both the routing and sequencing rules evolved by MTGP and sMTGP are much smaller than the rules built by CCGP, which provides valuable study materials for analysing the rules. And also, MTGP and sMTGP take less training time. This is an important merit because required-time consuming training is a big limitation for genetic programming. Secondly, the proposed tree swapping crossover operator can enhance the ability of MTGP from the perspective of performance, rule size and training time in general. Thirdly, for average normalized objective values on test data set, there are more outliers obtained by CCGP. That is to say, the assumption in CCGP that routing and sequencing rules are independent and can be involved separately, might be not true. This indicates that when we evolve two rules at the same time, we would better to take the interaction into consideration.

It is noted that the average rule size in the whole population becomes smaller with multi-tree representation. The reason will be further explored in the future. Also, it is worth interpreting and analysing the evolved rules to obtain further useful patterns. Moreover, more suitable representations for evolving two rules together will be investigated in the future.

References

1. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: A review. IEEE Transactions on Evolutionary Computation **20**(1), 110–124 (2016)

2. Brucker, P., Schlie, R.: Job-shop scheduling with multi-purpose machines. Computing **45**(4), 369–375 (1990)
3. Cordelia, L., De Stefano, C., Fontanella, F., Marcelli, A.: Genetic programming for generating prototypes in classification problems. In: 2005 IEEE Congress Evolutionary Computation. vol. 2, pp. 1149–1155 (2005)
4. Haupt, R.: A survey of priority rule-based scheduling. Operations-Research-Spektrum **11**(1), 3–16 (1989)
5. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evolutionary computation **23**(3), 343–367 (2015)
6. Holthaus, O., Rajendran, C.: Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics **48**(1), 87–105 (1997)
7. Langdon, W.B.: Genetic programming and data structures: genetic programming + data structures = automatic programming!, vol. 1. Springer (2012)
8. Lensen, A., Xue, B., Zhang, M.: Generating redundant features with unsupervised multi-tree genetic programming. In: European Conference on Genetic Programming. pp. 84–100. Springer (2018)
9. Mei, Y., Nguyen, S., Xue, B., Zhang, M.: An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. IEEE Transactions on Emerging Topics in Computational Intelligence **1**(5), 339–353 (2017)
10. Mei, Y., Nguyen, S., Zhang, M.: Evolving time-invariant dispatching rules in job shop scheduling with genetic programming. In: European Conference on Genetic Programming. pp. 147–163. Springer (2017)
11. Muni, D.P., Pal, N.R., Das, J.: A novel approach to design classifiers using genetic programming. IEEE Transactions on Evolutionary Computation **8**(2), 183–196 (2004)
12. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems **3**(1), 41–66 (2017)
13. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. IEEE Transactions on Evolutionary Computation **17**(5), 621–639 (2013)
14. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management science **42**(6), 797–813 (1996)
15. Papadimitriou, C.H., Steiglitz, K.: Combinatorial optimization: algorithms and complexity. Courier Corporation (1998)
16. Pezzella, F., Morganti, G., Ciaschetti, G.: A genetic algorithm for the flexible job-shop scheduling problem. Computers & Operations Research **35**(10), 3202–3212 (2008)
17. 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. International Journal of Production Research **50**(15), 4255–4270 (2012)
18. Van Laarhoven, P.J., Aarts, E.H., Lenstra, J.K.: Job shop scheduling by simulated annealing. Operations research **40**(1), 113–125 (1992)
19. Yska, D., Mei, Y., Zhang, M.: Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In: European Conference on Genetic Programming. pp. 306–321. Springer (2018)