Genetic Programming Hyper-heuristic with Cooperative Coevolution for Dynamic Flexible Job Shop Scheduling

No Author Given

No Institute Given

Abstract. Flexible Job Shop Scheduling (FJSS) problem has many real-world applications such as manufacturing and cloud computing, and thus is an important area of study. In real world, the environment is often dynamic, and unpredicted job orders can arrive in real time. Dynamic FJSS consists of challenges of both dynamic optimisation and the FJSS problem. In Dynamic FJSS, two kinds of decisions (so-called *routing* and *sequencing* decisions) are to be made in real time. Dispatching rules have been demonstrated to be effective for dynamic scheduling due to their low computational complexity and ability to make real-time decisions. However, it is time consuming and strenuous to design effective dispatching rules manually due to the complex interactions between job shop attributes. Genetic Programming Hyper-heuristic (GPHH) has shown success in automatically designing dispatching rules which are much better than the manually designed ones. Previous works only focused on standard job shop scheduling with only the sequencing decisions. For FJSS, the routing rule is set arbitrarily by intuition. In this paper, we explore the possibility of evolving both routing and sequencing rules together and propose a new GPHH algorithm with Cooperative Co-evolution. Our results show that co-evolving the two rules together can lead to much more promising results than evolving the sequencing rule only.

1 Introduction

In the modern industrial world, processing and manufacturing are global industries which are central to the economies of virtually every country. In a large factory setting, the efficient allocation of jobs to machines is therefore an extremely important concept that businesses must consider to increase throughput, decrease costs and increase profitability [15]. In a virtual setting, the idea of the efficient allocation of jobs to machines can also be applied to cloud resources. There are further applications to be found in timetabling, sports scheduling, health care scheduling and crew scheduling. This study of the allocation of jobs to machines is therefore a hugely important and relevant area of study to create more efficient outcomes in the modern world, saving time and resources.

In the Job Shop Scheduling (JSS) problem, there is a set of jobs to be completed, and a set of machines which can process the jobs [26]. A solution to this problem is an ordered schedule of assignments of jobs to machines, so that all

jobs are completed. These schedules are optimised relative to some objective, such as minimising the makespan and flowtime.

Flexible JSS (FJSS) problem is an extension of JSS. In JSS, each job operation only has one candidate machine to process it. In contrast, an operation of a job may have multiple candidate machines (*options*) in FJSS. As a result, FJSS involves allocating job operations to machines (i.e. *routing* problem) as well as selecting jobs from the queue of an idle machine to be processed next (i.e. *sequencing* problem). This makes FJSS more challenging than JSS.

FJSS is NP-hard since it has JSS as its special case (where all the operations have only one candidate machine). Thus, traditional optimisation methods such as branch-and-bound [18] is applicable when the problem size is not large. In this case, heuristic search methods such as simulated annealing [33], tabu search [25] and genetic algorithm [34] show promise in finding reasonably good solutions in a short time. However, in real world, the environment is usually *dynamic*, and unpredicted jobs can arrive at any time. The decisions made about which job to be processed next must be able to factor in the changing state of the system quickly and computationally cheaply. Therefore traditional optimisation techniques are infeasible for dynamic JSS due to their high computational complexity.

Dispatching Rules (DRs) have been used extensively in JSS (e.g. [3]) due to their computational efficiency. Whenever a machine becomes idle, a DR calculates a priority value for each job waiting in its queue and selects the most prior job to process next. Such computation is carried out at each decision point (e.g. when a machine becomes idle) and can be done efficiently. A variety of DRs have been designed manually to handle different scenarios. An overview of the manually designed DRs can be found in [29].

Manually designing DRs is time consuming and very demanding on domain expertise. The existing manually designed rules tend to be overly simplistic, with plenty of literature showing that many manually designed rules only perform well for certain objectives and in certain job shops [17, 28, 30]. Recently, Genetic Programming Hyper-heuristics (GPHH) has been successfully applied to automatically designing (evolving) DRs for scheduling [5, 19, 20], and the evolved DRs are much more effective than the manually designed DRs. However, the existing works mainly focused on evolving the *sequencing* rules, i.e. the rules selecting the operations from the queue of the idle machine to process next. The *routing* rule (i.e. the rule to select a candidate machine to process the given operation) is normally specified intuitive (e.g. selecting the machine with the least waiting time in [31]). Such simple routing rules are by no means the best and there is a potential to design routing rules that cooperates with the sequencing rules better in the given scheduling scenario. This motivates us to evolve the sequencing and routing rules simultaneously. To this end, we adopt the Cooperative Co-evolution (CC) [27] framework, which is a natural framework to evolve multiple components together. It has also been applied in JSS for co-evolving the DR and due date assignment rule [23].

1.1 Goals

In this paper, we aim to find more promising routing and sequencing rules for FJSS. Specifically, we aim to achieve the following research objectives.

1. Compare between different manually designed routing rules on different FJSS scenarios to understand which manually designed routing rule performs the best in general.
2. Propose a GP with Cooperative Co-evolution (called CCGP) for co-evolving the routing and sequencing rules simultaneously.
3. Compare CCGP with the GP that evolves sequencing rule with pre-specified routing rule (called SeqGP) to evaluate the performance of CCGP.
4. Conduct analysis on the characteristics of the rules evolved by CCGP to gain new knowledge about the structure of the effective routing rules for FJSS.

1.2 Organisation

The rest of the paper is organised as follows: Section 2 gives the problem description and related work. Then, the proposed CCGP is proposed in Section 3. Experimental studies are carried out in Section 4. Finally, Section 5 gives the conclusions and future work.

2 Background

2.1 Flexible Job Shop Scheduling

FJSS is to process a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ with a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j has an arrival time $t_0(J_j)$ and a sequence of operations $O_{1,j}, \ldots, O_{l_j,j}$. Each operation $O_{i,j}$ has a set of candidate machines $\boldsymbol{\pi}_{i,j} \subseteq \mathcal{M}$. It can be processed by any machine $\pi_{i,j,k} \in \boldsymbol{\pi}_{i,j}$. The duration of processing operation $O_{i,j}$ with machine $\pi_{i,j,k}$ is $\delta_{i,j,k}$. One cannot start processing an operation until its precedent operations have been completed. Each machine can process at most one operation at a time, and each operation is processed by exactly one machine without interruption. The goal of FJSS is to find a feasible schedule to optimise some objective(s). The commonly considered JSS objectives include minimising the makespan ($C_{\max}$), total flowtime ($\sum C_j$), total weighted tardiness ($\sum w_j T_j$), number of tardy jobs, etc [26].

JSS is a special case of FJSS, where for each operation $O_{i,j}$, $|\boldsymbol{\pi}_{i,j}| = 1$. In other words, each operation can be processed by only one machine. In this case, no routing decision needs to be made.

2.2 Related Work

The FJSS problem was first identified by Brucker and Schlie [6] in 1990, where a solution of a polynomial algorithm was suggested to solve each of the routing and sequencing sub-problems for a two job system. Early studies focused on

finding FJSS solutions using traditional optimisation approaches. Brandimarte [4] proposed using a hierarchical method to minimise the makespan for a FJSS system. In his work, he used a two-level tabu search algorithm in combination with the decomposition of FJSS into routing and job shop scheduling sub problems. In his work, Brandimarte also created a class of flexible job shops that would become used as a benchmark by future researchers [2]. Norman and Bean [24] developed a genetic algorithm with random key representation, elitist reproduction, immigration mutation as well as Bernoulli crossover to solve the FJSS problem with the objective of minimising total tardiness. In 2002, Kacem et al. [16] proposed a hybrid approach for solving the FJSS problem, using localisation for the routing component, and three manually designed dispatching rules for the sequencing component. An advanced genetic algorithm was proposed for evolving arrangements of jobs and machines.

In recent decades, hyper-heuristics [7] have attracted more and more research attention, as they can find heuristics (i.e. dispatching rules in JSS) rather than solutions, and thus are more flexible and scalable. More importantly, the evolved heuristics can handle dynamic environment much more effectively than traditional (re-)optimisation approaches. In 2001, Dimopoulos and Zalzal [9] used GP to evolve dispatching rules for JSS, for single machine scheduling with a terminal set of scheduling attributes (processing time, due date, number of jobs, release time, etc) with a standard function set. These evolved dispatching rules performed well and were better than traditional manually designed rules even for unseen and large instances. Then in 2006, Geiger et al. [10] presented a learning system which combined GP with a simulation model for an industrial facility. This proposed GP method creates a rule assigned priority to jobs on a single machine in both static and dynamic environments. This paper quickly produced many dispatching rules which rivalled results produced by rules found in past decades. A method for evolving dispatching rules for multiple machines was proposed, which used modified genetic operators. Miyashita [22] in 2000 developed an automatic method of evolved customised dispatching rules for a JSS environment, using GP. In his work, he considered the JSS problem as being a multi-agent problem, where each agent represents a resource (machine or work station). This multi-agent model was explored using GP, and produced good results, however prior knowledge of the JSS environment was required. This limits the application of this work to only static environments.

In 2007, Tay and Ho [32] proposed a GP method to evolve dispatching rules for a FJSS environment which were optimised for multiple objectives. These multiple objectives were treated as a single objective by linearly combined their objective functions. The proposed GP method can be thought of as a priority function which calculated the priority of operations in the queue of a single machine, based on static and dynamic attributes in the job shop. The dispatching rules evolved outperformed other manually designed dispatching rules, although the use of machine attributes was not considered. This system was assessed later in 2010 by Hildebrandt et al. [12] which showed that in some dynamic JSS instances, the evolved rules by Tay and Ho [32] performed only slightly better

than the earliest release date rule, and worse the than shortest processing time rule, which are very simplistic. Hildebrandt et al. [12] then used GP to evolve dispatching rules in four simulations (all with 10 machines, with a combination of two utilisation levels and two job types) for the single objective of mean flow time. Their evolved rules were robust, performing very well in both different environments (50 machines with varying processing time distributions) and the original training environments. In 2014, Nguyen et al. [23] used cooperative coevolution GP to evolve due date assignment rules and dispatching rules, for multi-objective JSS. In this work, Nguyen et al. showed that the evolved scheduling policies performed very well on unseen simulation scenarios, given different shop settings. In 2016, Mei et al. [21] used GP to evolve dispatching rules for JSS for a single objective. Feature selection was then performed on the terminal set of the dispatching rules, removing extraneous terminal attributes and reducing the problem search space. This led to significantly better dispatching rules evolved by GP on both training and test instances.

3 Genetic Programming with Cooperative Co-evolution

The pseudo-code of the proposed CCGP is described in Algorithm 1. In the proposed CCGP, there are two subpopulations $\mathbf{P}_r = \{P_{r,1}, P_{r,2}, \dots\}$ and $\mathbf{P}_s = \{P_{s,1}, P_{s,2}, \dots\}$, where $\mathbf{P}_r$ stands for the population of routing rules and $\mathbf{P}_s$ stands for the population of sequencing rules. In addition, a context vector $\mathbf{cv} = (cv_r, cv_s)$ is maintained for fitness evaluation. At first, the two populations are randomly initialised by ramp half-and-half method, and the context vector is randomly initialised from the populations. Then, at each generation, the routing rules and sequencing rules are evolved separately using the crossover/mutation/reproduction operator of GP. Then, each newly generated rule is evaluated by the `evaluate`$(\cdot)$ method. Finally, the context vector is updated by replacing the routing and sequencing components with the best individuals in the corresponding population, if they have better fitness values. In the minimisation case in FJSS (e.g. makespan and flowtime are to be minimised), a smaller fitness value is better.

The fitness evaluation procedure is given in Algorithm 2. It takes a routing rule p_r, a sequencing rule p_s, and a set of FJSS instances $\mathcal{I}_{\text{train}}$(i.e. *training set*), and returns a fitness value. For each training instance, it constructs a discrete event simulation based on the instance, the routing and sequencing rules, and run the simulation to generate a schedule.

At the beginning of the simulation, all machines are idle, and there may be some initial jobs ready to be processed (ready time 0). In the dynamic FJSS scenarios, unpredicted job arrival events are generated randomly as well. Then online decisions are made as follows until all the jobs have been completed.

- Whenever a job becomes ready to be processed, if its next operation has only one candidate machine, then place the job into the queue of the candidate machine. Otherwise, apply the *routing rule* to select the machine to process the job, and place the job to the queue of the selected machine.

Algorithm 1: Pseudo-code of the proposed CCGP

```
1  Randomly initialise P_r and P_s by ramp half-and-half;
2  cv_r ← P_{r,1}, cv_s ← P_{s,1};          // arbitrarily initialise context vector
3  while Stopping criteria not met do
       // Evolve the routing rules
4      P'_r ← elite(P_r);                    // copy the elites to the new population
5      while |P'_r| < popsize do
6          Generate offspring(s) by applying the crossover/mutation/reproduction
             operator to P_r;
7          Add the generated offspring(s) to P'_r;
8      end
       // Evolve the sequencing rules
9      P'_s ← elite(P_s);                    // copy the elites to the new population
10     while |P'_s| < popsize do
11         Generate offspring(s) by applying the crossover/mutation/reproduction
             operator to P_s;
12         Add the generated offspring(s) to P'_s;
13     end
       // fitness evaluation
14     foreach p ∈ P'_r do fit(p) ← evaluate(p, cv_s, I_train);
15     foreach p ∈ P'_s do fit(p) ← evaluate(cv_r, p, I_train);
16     P_r ← P'_r, P_s ← P'_s;                          // update subpopulations
       // update context vector
17     cv'_r ← arg min_{p∈P_r} fit(p), cv'_s ← arg min_{p∈P_s} fit(p);
18     if fit(cv'_r) < fit(cv_r) then cv_r ← cv'_r;
19     if fit(cv'_s) < fit(cv_s) then cv_s ← cv'_s;
20 end
21 return cv = (cv_r, cv_s);
```

– Whenever a machine is idle and its queue is not empty, apply the *sequencing rule* to select the next job from the queue, and start processing the next job.

A simulation essentially generates a schedule (with starting and finishing time of each job). Then, we can calculate the *normalised* objective value (e.g. makespan and flowtime) of the schedule. Finally, the fitness value is set to the average value of all the normalised objective values (line 7). Here, the normalisation (line 5) is with respect to a reference value $\texttt{obj}^*(I)$, which can be set to either the best known (lower bound of) objective value of the instance, or the objective value obtained by applying benchmark routing and sequencing rules.

As shown in Algorithm 1 (lines 14 and 15), for evaluating a routing (sequencing) rule, it is combined with the sequencing (routing) component of the context vector so that the discrete event simulation in Algorithm 2 can be constructed.

Algorithm 2: `evaluate`$(p_r, p_s, \mathcal{I}_{\text{train}})$

Input: A routing rule p_r, a sequencing rule p_s, a set of FJSS instances $\mathcal{I}_{\text{train}}$
Output: A fitness value

$f \leftarrow 0$;
foreach $I \in \mathcal{I}_{train}$ **do**
  Construct a discrete event simulation based on p_r, p_s and I;
  Generate a schedule $S(p_r, p_s, I)$ by running the discrete event simulation;
  $f \leftarrow f + \frac{\texttt{obj}(S(p_r,p_s,I))}{\texttt{obj}^*(I)}$; `// normalisation cross instances`
end
return $f/|\mathcal{I}_{train}|$; `// average over the training set`

4 Experimental Studies

To evaluate the proposed CCGP, we conducted experiments on both static and dynamic FJSS datasets. The static instances are commonly used in the evaluation of FJSS methods [2], and their lower and upper bounds of makespan are known. Specifically, there are 4 static FJSS datasets, namely the Barnes dataset [1], Brandimarte dataset [4], Dauzere dataset [8] and Hurink dataset [14]. The Barnes dataset consists of 21 instances with 10 or 15 jobs. Each job has 11 to 18 operations, and each operation has 1.07∼1.3 candidate machines. Thus, the Barnes dataset is small and has relatively low flexibility. The Brandimarte dataset has 10 small sized instances (no more than 20 jobs and 15 machines, each job has 5∼15 operations) and medium flexibility (each operation has 2∼6 machine options). The Dauzere dataset consists of 18 instances with similar size and flexibility as the Brandimarte dataset. There are 66 instances in the Hurink dataset, which can be divided into 4 subsets with increasing flexibility, namely sdata, edata, rdata and vdata. The sdata instances are essentially JSS instances, as no operation can be processed by more than one machine. In the most flexible vdata instances, all the operations can be processed by multiple machines.

For dynamic simulation, the configuration is given in Table 1, which has been used in previous studies (e.g. [20, 11]).

Table 1. The Dynamic JSS simulation system configuration.

Parameter	Value
#machines	10
#jobs (#warmup jobs)	5000 (1000)
#operations per job	Uniform discrete distribution between 1 and 10
#Machines per operation	Uniform discrete distribution between 1 and 10
Job arrival process	Poisson process
Utilisation level	{0.85, 0.95}
Processing time	Uniform discrete distribution between 1 and 99
Job weights	20% with weight 1, 60% with weight 2, 20% with weight 4

4.1 Parameter Settings

The parameter setting of CCGP is standard, as given in table 2. The terminal set of CCGP is described in Table 3. The terminals are adapted from the JSS terminals proposed in [20]. The terminals involving the future operations (e.g. NPT and WKR) are modified to take into account the machine-dependent processing times. For each future operation, the processing time is set to the median processing time of all the options.

The function set of CCGP is set to $\{+, -, *, /, \min, \max\}$, where "/" is the protected division that returns 1 if divided by 0. The "min" and "max" operators take two arguments, and return the minimal (maximal) value between them.

In the experiment, we will compare CCGP with the GP counterpart with routing rule fixed to LWQ, and evolving the sequencing rule only. For the sake of convenience, the counterpart will be denoted as SeqGP hereafter. For fair comparison, the population size of SeqGP is set to 1024 so that the number of fitness evaluations per generation is the same as CCGP. All the other parameters are the same for SeqGP and CCGP.

Table 2. The parameter setting of CCGP.

Parameter	Value
Number of subpopulations	2
Subpulation size (`popsize`)	512
Maximal depth	8
Crossover/Mutation/Reproduction rates	80% / 15% / 5%
Parent selection	Tournament selection with size 7
Elitism	2 best individuals
Number of generations	51

Table 3. The terminal set of CCGP.

Notation	Description
NIQ	Number of Operations in a Machine's Queue.
WIQ	Work In a Machine's Queue.
MWT	Waiting Time of a Machine.
PT	Processing Time of an Operation on a given Machine.
NPT	**Median** Processing Time for the Next Operation on Machine options.
OWT	The Waiting Time of an Operation.
WKR	**Median** Amount of Work Remaining for a Job.
NOR	The Number of Operations Remaining in a Job.
W	Weight of a Job.
TIS	Time In System.

4.2 Comparing Manually Designed Routing Rules for SeqGP

SeqGP requires a pre-specified routing rule for evaluating the evolved sequencing rules. In existing studies, only the *least waiting time assignment* routing rule was considered [13, 31] without investigating whether it is the best routing rule. In this paper, we first compare a set of commonly used manually designed routing rules on the static FJSS instances to identify the best routing rule for SeqGP.

Specifically, four manually designed routing rules are taken into account in the comparison. They are described as follows:

1. *Least Work in Queue* (LWQ): select the machine with the least work (total processing time) in its queue;
2. *Least Queue Size* (LQS): select the machine with the least queue size (number of operations in the queue);
3. *Earliest Ready Time* (ERT): select the machine that will become ready (idle) the earliest;
4. *Shortest Busy Time* (SBT): select the machine with the shortest busy time so far.

Among the above routing rules, the ERT is essentially the same as the least waiting time rule used in previous studies (e.g.[13, 31]).

For each routing rule, the SeqGP with that routing rule was run on each static instance for 30 times (except the 66 Hurink-sdata instances, which are essentially JSS instances). Then, a routing rule is considered as a "*winner*" of an instance if it achieved the best mean makespan over the 30 runs (there may be multiple winners). Then, we compare the number of instances where each routing rule was a winner.

Table 4 shows the number of instances in each static dataset where each routing rule was a winner. It can be seen that LWQ was a winner for most instances (127 out of 247), followed by ERT. More specifically, the advantage of ERT over LWQ was only on the Barnes dataset, which was the very inflexible. As the flexibility increases, the advantage of LWQ becomes more obvious.

The findings in this subsection is interesting as it identifies LWQ as a better routing rule than ERT, which has been used in previous studies, for static FJSS. In subsequent experiments, we set LWQ as the fixed routing rule for SeqGP.

4.3 Optimisation Performance on Static Instances

The first set of experiments aims to verify the optimisation performance of SeqGP and CCGP on the static FJSS instances, without a training and test (generalisation) process. This way, one can investigate the effectiveness of (co-)evolving dispatching rules as compared to directly optimising FJSS solutions.

For the static instances, the objective is to minimise the makespan. For each static instance, CCGP and SeqGP were run 30 times independently, and the normalised makespans (makespan over the known lower bound) of the best rules were recorded. In addition, two manually designed sequencing rules, i.e. First-Come-First-Serve (FCFS) and Shortest Processing Time (SPT), are also taken into comparison.

Table 4. The number of instances in each static dataset where each compared routing rule was a winner.

Dataset	#Instances	LWQ	LQS	ERT	SBT
Barnes	21	0	0	13	8
Brandimarte	10	8	0	2	2
Dauzere	18	16	0	1	1
Hurink-edata	66	31	3	27	9
Hurink-rdata	66	39	0	27	0
Hurink-vdata	66	33	0	33	0
Total	247	127	3	103	20

Table 5 shows the summary of the compared algorithms over 30 independent runs for the static datasets. FCFS and SPT are deterministic rules. Therefore, for each dataset, the average normalised makespan value cross all the instances of that dataset is shown. SeqGP and CCGP are stochastic algorithms. Therefore, for each dataset, the mean and standard deviation over the 30 runs are given. In addition, for each instance, Wilcoxon rank sum test with significance level of 0.05 was conducted between the 30 results obtained by CCGP and SeqGP. Then, for each dataset, the numbers of instances that CCGP performed significantly better than SeqGP ("W"), comparable with SeqGP ("D"), and significantly worse than SeqGP ("L") are given.

Table 5. The normalised makespan (MK/LB) with respect to lower bound of the compared algorithms over 30 independent runs for the static datasets.

Dataset	#Instances	FCFS	SPT	SeqGP	CCGP	W-D-L
Barnes	21	1.270	1.238	1.079(0.002)	1.065(0.003)	15-6-0
Brandimarte	10	1.431	1.501	1.229(0.004)	1.062(0.004)	8-2-0
Dauzere	18	1.244	1.227	1.086(0.001)	1.061(0.002)	16-2-0
Hurink-edata	66	1.247	1.241	1.070(0.001)	1.048(0.001)	50-15-1
Hurink-rdata	66	1.271	1.292	1.123(0.001)	1.062(0.001)	64-2-0
Hurink-vdata	66	1.312	1.324	1.215(0.000)	1.019(0.001)	64-2-0

From Table 5, it is obvious that both SeqGP and CCGP dramatically outperformed the manually designed rules (FCFS and SPT). In addition, CCGP performed much better than SeqGP. Overall, FCFS and SPT obtained solutions which are 25%–50% worse than the lower bound. SeqGP obtained solutions that are 7%–23% worse than the lower bound. All the solutions obtained by CCGP are less than 7% worse than the lower bound. The most obvious advantage of CCGP over SeqGP occurred on the Brandimarte and Hurink-vdata datasets, which have reasonable large problem sizes and flexibility.

More specifically, CCGP statistically significantly outperformed SeqGP on most static instances (e.g. 64 out of 66 of the Hurink-rdata and Hurink-vdata instances). CCGP was defeated by SeqGP on only one Hurink-edata instance out of the total 247 static instances. This clearly demonstrates the advantage of CCGP over SeqGP on solving static FJSS instances.

Figure 1 shows the convergence curves of SeqGP and CCGP on three representative instances (the ribbon is the standard deviation over the 30 runs), on which CCGP performed significantly better than, worse than, and comparable with SeqGP. All the other instances showed similar patterns. From the figure, it is clear that CCGP started from a much higher makespan due to the random initial routing rule. Then, it converged very fast, and achieved better results than SeqGP within 10 generations.

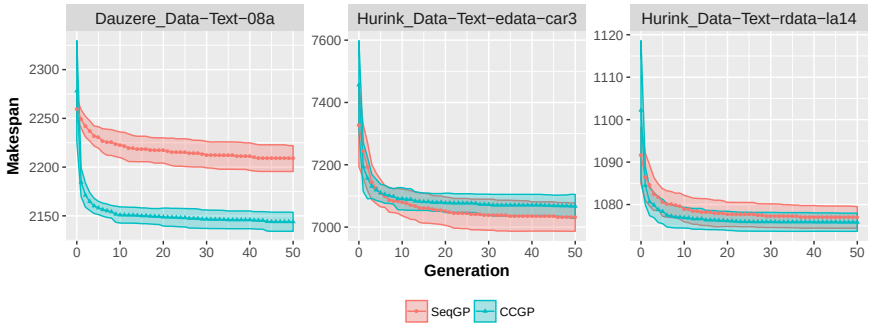

Fig. 1. The convergence curves of the makespan of the 30 runs of SeqGP and CCGP.

Finally, CCGP can obtain solutions that are less than 7% worse than the lower bound, which can be seen as a promising optimisation performance for static FJSS instances.

4.4 Generalisation Performance on Dynamic Instances

The experiments in the dynamic environment is to examine the generalisation performance of SeqGP and CCGP. We consider 2 utilisation levels (0.85 and 0.95) and 3 objectives in the dynamic environment. Specifically, we consider minimising (1) mean flowtime (`Fmean`), (2) max flowtime (`Fmax`) and (3) mean weighted flowtime (`MWF`). This results in $3 \times 2 = 6$ scenarios. For each scenario, SeqGP and CCGP were run 30 times independently over a training set. The training set consists of a single dynamic FJSS simulation. To improve generalisation, the random seed for generating the training simulation changes per generation. After the training process, the best rule of the last generation is then tested on an unseen test set to evaluate its test performance. The test set

consists of 50 dynamic simulations using the same configurations as the training set, but different random seeds.

For the dynamic simulations, the lower bound objective values are unknown. Therefore, the normalisation is with respect to the objective value obtained by a benchmark dispatching rule (routing plus sequencing rules). Here, the benchmark routing rule is fixed to LWQ for all the scenarios. The benchmark sequencing rule is specified depending on the scenario. Based on our preliminary work [20], we set the benchmark sequencing rule to FCFS for the scenarios minimising `Fmax`, to SPT for the scenarios minimising `Fmean`, and to WSPT for the scenarios minimising `MWF`.

Fig. 2 shows the convergence curves of the test fitness obtained by SeqGP and CCGP over the 6 dynamic scenarios. From the figure, it is obvious that CCGP significantly outperformed SeqGP in all the 6 scenarios. The Wilcoxon rank sum test with significance level of 0.05 also confirmed the significance. The convergence curves of CCGP are almost always below the curves of SeqGP. For the scenarios minimising `Fmean` and `MWF`, CCGP successfully initialised much more effective routing rules even from the first generation.

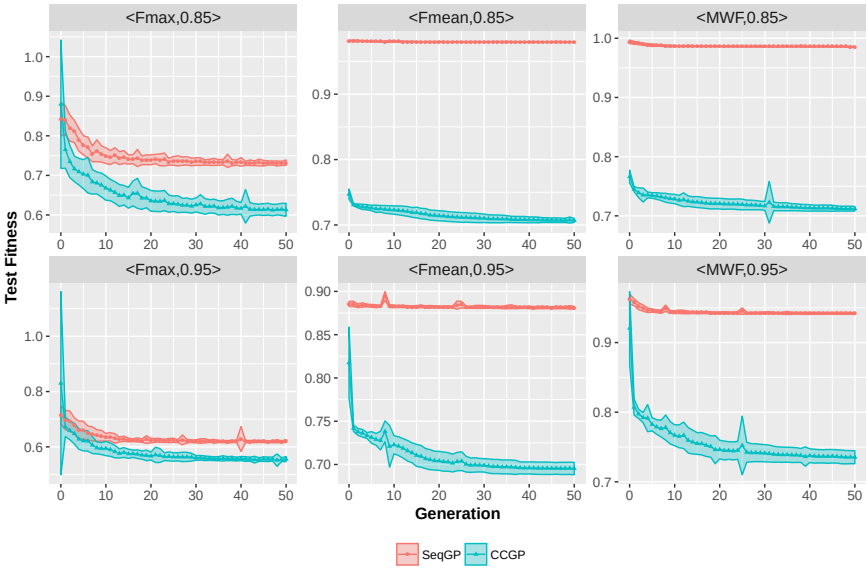

Fig. 2. The convergence curves of the test fitness obtained by SeqGP and CCGP.

Fig. 3 shows the convergence curves of the size of the sequencing rules obtained by SeqGP and CCGP. It can be seen that the two algorithms have similar convergence curves in terms of sequencing rule size, i.e. evolving routing rules does not seem to make the sequencing rule simpler or more complex.

Fig. 3. The convergence curves of the **sequencing** rule size obtained by SeqGP and CCGP.

In order to show the generalisation of SeqGP and CCGP, Fig. 4 shows the training fitness versus test fitness scatter plot based on the 30 final results of SeqGP and CCGP. From the figure, it is clear that both the training and test fitnesses of CCGP were much better than that of SeqGP. The generalisation of both algorithms are similar in terms of the correlation between training and test fitnesses. The generalisation of CCGP is poorer for the scenarios minimising `Fmax` than other scenarios. This may be because `Fmax` is a maximum function, which is not so smooth as the other objectives which are based on average as the sample size grows. Overall, the generalisation of CCGP is promising, as the test fitness is very consistent with the training fitness. On the other hand, one can see that for the dynamic scenarios with `Fmean` and `MWF` and low utilisation level (0.85), the pre-specified routing rule restricted the search space too much so that the evolved sequencing rules perform almost the same as the benchmark sequencing rules in both training and test instances.

4.5 Rule Analysis

Eq. (1) shows an example routing rule evolved by CCGP for the scenario $\langle$`MWF`, 0.95$\rangle$.

$$\min\{\text{NIQ} \times \text{PT}, \text{WIQ}\} + \frac{\text{W}}{\text{MWT} \times \text{PT}} - \min\{\text{MWT} \times \text{W}, \text{NIQ} \times \text{NOR}\}. \quad (1)$$

It mainly consists of three components. The first component $\min\{\text{NIQ}\times\text{PT}, \text{WIQ}\}$ is similar to WIQ, i.e. the number of operations in queue times the processing

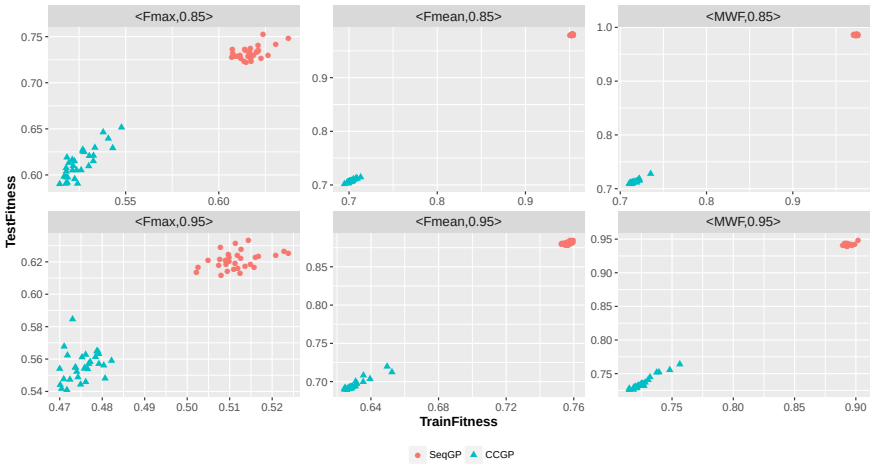

Fig. 4. The training fitness versus test fitness scatter plot based on the 30 final results of SeqGP and CCGP.

time of an operation is similar to the total processing time in queue. The second and third terms show that the routing rule prefers machines with larger MWT, i.e. the earliest available machine (MWT = current time − machine ready time). This preference is more obvious if the current job has a larger weight. That is, the routing rule tries to finish the more important jobs as early as possible. In summary, CCGP can automatically evolve routing rules that contain sensible patterns consistent with intuition for making routing decisions.

5 Conclusions and Future Work

In this paper, we propose to co-evolve the routing and sequencing rules together using a co-operative coevolution framework for flexible job shop scheduling (FJSS). Through comprehensive experiments, we had several interesting findings. First, we found that the commonly used pre-specified routing rule is not the best one for static FJSS. We found a better routing rule, which is LWQ (least work in queue). Then, we developed the GPHH with the routing rule fixed to LWQ (named SeqGP), and the Cooperative Co-evolution GP (CCGP) that co-evolves the routing and sequencing rules simultaneously. The results show that CCGP performed much better than SeqGP in both static and dynamic scenarios. This demonstrates that the routing rules evolved by CCGP are much better than the rules that are manually designed and fixed in SeqGP. In other words, there is a great potential to find much more effective routing rules for FJSS, especially in the dynamic environment.

In the future, we will focus on further improving the effectiveness of CCGP. In this paper, only a baseline CC framework is adopted. We will consider incorpo-

rating other domain specific strategies such as feature selection and construction to improve the effectiveness and efficiency of the GP search.

References

1. J. W. Barnes and J. B. Chambers. Solving the job shop scheduling problem with tabu search. *IEEE Transactions*, 27(2), 1995.
2. D. Behnke and M. J. Geiger. Test instances for the flexible job shop scheduling problem with work centers. Technical report, Helmut Schmidt Universitat, January 2012.
3. J. H. Blackstone, D. T. Phillips, and G. L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *International Journal of Production Research*, 20(1):27–45, 1982.
4. P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993.
5. J. Branke, S. Nguyen, C. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, 2016.
6. P. Brucker and R. Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1990.
7. E. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.
8. S. Dauzere-Peres and J. Paulli. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70(0):281–306, 1997.
9. C. Dimopoulos and A. Zalzala. Invesigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, 32:489–498, 2001.
10. C. D. Geiger, R. Uzsoy, and H. Aytug. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9:7–34, 2006.
11. T. Hildebrandt and J. Branke. On using surrogates with genetic programming. *Evolutionary computation*, 23(3):343–367, 2015.
12. 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 Conference*, 2010.
13. N. B. Ho and J. C. Tay. Genace: An efficient cultural algorithm for solving the flexible job-shop problem. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 1759–1766. IEEE, 2004.
14. J. Hurink, B. Jurisch, and M. Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *Operations-Research-Spektrum*, 15(4):205–215, 1994.
15. A. Jones, L. C. Rabelo, and A. T. Sharawi. Survey of job shop scheduling techniques. *Wiley Encyclopedia of Electrical and Electronics Engineering*, 1999.
16. 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*, 32(1):1–13, 2002.

17. A. S. Kiran. Simulation studies in job shop scheduling - i a survey. *Computers and Industrial Engineering*, 8(2):87–93, 1984.
18. A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, July 1960.
19. Y. Mei, S. Nguyen, B. Xue, and M. Zhang. 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.
20. 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*, pages 147–163. Springer, 2017.
21. Y. Mei, M. Zhang, and S. Nyugen. Feature selection in evolving job shop dispatching rules with genetic programming. *GECCO*, 2016.
22. K. Miyashita. Job-shop scheduling with gp. In *Genetic and Evolutionary Computation Conference*, 2000.
23. 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*, 18(2):193–208, 2014.
24. B. A. Norman and J. C. Bean. A genetic algorithm methodology for complex scheduling problems. *Naval Research Logistics*, 46(2):199–211, 1999.
25. E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813, 1996.
26. M. L. Pinedo. *Scheduling: theory, algorithms and systems*. Springer Science and Business Media, 2012.
27. M. A. Potter and K. A. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Comptuation*, 8:1–29, 2000.
28. R. Ramasesh. Dynamic job shop scheduling: A survey of simulation research. *Omega*, 18(1):43–57, 1990.
29. V. Sels, N. Gheysen, and M. Vanhoucke. 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.
30. V. Subramaniam, T. Ramesh, G. K. Lee, Y. S. Wong, and G. S. Hong. Job shop scheduling with dynamic fuzzy selection of dispatching rules. *The International Joural of Advanced Manufacturing Technology*, 16:759–764, 2000.
31. J. Tay and N. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.
32. J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Technical report, Evolutionary and Complex Systems Program, School of Computer Engineering, Nanyang Technological University, 2007.
33. P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113–125, 1992.
34. G. Zhang, L. Gao, and Y. Shi. An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4):3563–3573, 2011.