Evolving Ensembles of Dispatching Rules Using Genetic Programming for Job Shop Scheduling

John Park[1(✉)], Su Nguyen[1,2], Mengjie Zhang[1(✉)], and Mark Johnston[1]

[1] Evolutionary Computation Research Group,
Victoria University of Wellington, PO Box 600, Wellington 6140, New Zealand
{John.Park,Su.Nguyen,Mengjie.Zhang}@ecs.vuw.ac.nz,
Mark.Johnston@msor.vuw.ac.nz

[2] International University - VNU HCMC, Ho Chi Minh City, Vietnam

Abstract. Job shop scheduling (JSS) problems are important optimisation problems that have been studied extensively in the literature due to their applicability and computational difficulty. This paper considers static JSS problems with makespan minimisation, which are NP-complete for more than two machines. Because finding optimal solutions can be difficult for large problem instances, many heuristic approaches have been proposed in the literature. However, designing effective heuristics for different JSS problem domains is difficult. As a result, hyper-heuristics (HHs) have been proposed as an approach to automating the design of heuristics. The evolved heuristics have mainly been priority based dispatching rules (DRs). To improve the robustness of evolved heuristics generated by HHs, this paper proposes a new approach where an ensemble of rules are evolved using Genetic Programming (GP) and cooperative coevolution, denoted as Ensemble Genetic Programming for Job Shop Scheduling (EGP-JSS). The results show that EGP-JSS generally produces more robust rules than the single rule GP.

Keywords: Genetic programming · Job shop scheduling · Hyper-heuristics · Ensemble learning · Cooperative coevolution · Robustness · Dispatching rules · Combinatorial optimisation · Evolutionary computation

1 Introduction

Job shop scheduling (JSS) problems are important optimisation problems that have been studied for over 50 years. JSS is still studied extensively due to its complexity and wide applications. JSS problems involve determining the optimal sequence to process jobs on the machines in a manufacturing system. For a JSS problem instance, each job has operations that need to be completed on different machines in a given sequence. However, a machine cannot process more than one job at a time. All operations must be processed by the machines to get a schedule, and the 'quality' of the solution generated for the JSS problem instance is given by the objective function. There are a number of existing approaches to solving JSS problems. Mathematical optimisation techniques give optimal solutions for

P. Machado et al. (Eds.): EuroGP 2015, LNCS 9025, pp. 92–104, 2015.
DOI: 10.1007/978-3-319-16501-1_8

static JSS problem instances. On the other hand, heuristic approaches, such as dispatching rules (DRs), have been applied to JSS to produce good solutions for large problem instances. Dispatching rules [12] are local decision makers which iteratively decide a sequence of jobs to be processed by a machine. In addition, meta-heuristic approaches [10,17] have also been applied to JSS. However, an issue with heuristic approaches to JSS is that they need to be carefully designed. Heuristic approaches also tend to be problem domain specific. Heuristics that are effective in one domain are not necessarily effective in other domains. Because of this, hyper-heuristics (HHs) [2] aim to automate the generation of heuristics such as DRs. However, DRs are limited as they make a single decision for choosing the next job to be processed by a machine. The myopic nature of DRs, combined with the fact that complex decisions need to be made for JSS problems, means that it is possible that DRs make bad decisions for certain situations within a particular JSS problem instance.

In classification, similar issues arise as single constituent rules cannot represent the noisy and complex decision boundaries between different classes sufficiently [13]. Because of this, ensemble approaches have been proposed [1,4], which have successfully been applied to difficult classification problems [13]. In an ensemble, a group of small constituent rules 'vote' on the outcomes. For example, the class labels represent the outcomes that can be 'voted' for in classification problem. It may be possible that ensembles of DRs can be used to deal with the complex decisions of selecting jobs better than single DRs, and improve the robustness of rules for JSS. However, ensemble approaches have not been seriously investigated for JSS.

The goal of this paper is to determine whether ensemble approaches can be used effectively for static JSS problem instances. An evaluation scheme is needed that allows a diverse set of rules to be evolved, as diversity is a cornerstone of ensemble approaches [13]. We denote this approach as Ensemble Genetic Programming for Job Shop Scheduling (EGP-JSS). This will be compared with an approach of evolving a single priority rule, denoted Genetic Programming for Job Shop Scheduling (GP-JSS). GP-JSS makes minor adjustments from a previous approach [11] of evolving DRs from GP by modifying the terminal set. Specific research objectives in this paper are:

(a) Developing a job selection procedure for the ensemble of rules for JSS.
(b) Developing a new fitness function for EGP-JSS to ensure that a diverse set of rules are evolved.
(c) Comparing the evolved ensemble rules by EGP-JSS and GP-JSS with the benchmark DRs.

2 Background

This section briefly describes some background on the JSS problem with previous approaches for JSS, and the hyper-heuristic approaches that have been applied to JSS.

2.1 Job Shop Scheduling Problem

A JSS problem instance consists of N jobs and M machines, and a list of operations for each job. Compared to dynamic JSS problems, static JSS problems have all attributes of jobs, machines and operations known from the beginning, and do not contain any stochastic elements. An operation σ_{ij} in a JSS problem instance is the i^{th} operation of job j, and $M(\sigma_{ij})$ denotes the machine that the operation is processed on. An operation σ_{ij} can only be carried out when operation σ_{i-1j} has been completed (with σ_{1j} being the first operation of a job j), and when the machine to be processed on $(M(\sigma_{ij}))$ is available. The time when a machine i is available is denoted as R_{M_i}. Each job j has a ready time $r(\sigma_{1j})$ for when its first operation is available, and each operation σ has processing time $p(\sigma)$, and setup time $s(\sigma)$. The number of operations for job j is N_j, and the total remaining processing time is $\sum_{k=i}^{N_j} p(\sigma_{kj})$. For this paper, we focus on the static JSS problem with makespan minimisation, i.e., minimising the maximum completion time $C_{\max}$. This is denoted as $Jm||C_{\max}$.

$Jm||C_{\max}$ for $M = 2$ machines can be solved optimally via Jackson's algorithm [12]. However, Garey et al. [5] showed that the JSS makespan minimisation problem is NP-complete for $M > 2$. In JSS problems with instances that have hundreds of jobs and a large number of machines [15], exact optimisation is too computationally expensive. For such JSS problem instances, the primary approaches use heuristics, such as DRs [12] and meta-heuristics. DRs range in complexity from basic first-in-first-out (FIFO) rules, which processes the jobs in the order they arrive, to more complex composite dispatching rules (CDRs) [9], which combine smaller heuristics to form custom made priority functions. On the other hand, a wide range of meta-heuristic approaches have been proposed in the literature. Meta-heuristic approaches include Simulated Annealing [10] and Genetic Algorithms (GA) [17].

2.2 Genetic Programming Based Hyper-Heuristic Approaches

In conjunction with heuristic and meta-heuristic approaches, hyper-heuristics (HHs) [2] have also been investigated for JSS. Instead of searching the solution space directly, HHs are given heuristic components to generate heuristics with, and a fitness measure to evaluate how well generated heuristics perform. It then searches for a good heuristic, optimising over the fitness measure. A number of HH approaches to JSS in the literature use Genetic Programming (GP) [2].

Dimopoulos and Zalzala [3] use GP to evolve priority based DRs for a single machine JSS problem. An arithmetic representation consisting of mathematical operators and job attributes are used to represent the individuals in the GP system. They showed that the evolved rules performed better than the man-made benchmark DRs. Geiger et al. [6] use GP to evolve priority based DRs for various single machine JSS problems in both static and dynamic environments. They showed that GP can evolve DRs that can generate optimal solutions for some special static single machine JSS problems with polynomial time exact algorithms, and evolve effective rules for NP-hard JSS problems.

Jakobovic et al. [8] proposed a GP based hyper-heuristic approach to evolving priority based DRs for the multi-machine static and dynamic JSS problems ranging from 3 to 20 machines. Tay and Ho [16] proposed a priority based GP approach to multi-objective flexible job-shop problems, and showed that the evolved rules outperformed other simple DRs. However, later examination [7] showed that Tay and Ho's approach [16] does not perform as well in different dynamic job shop scenarios. Nguyen et al. [11] compared three different representations for GP to evolve DRs for static JSS problems. The first representation they propose is a decision tree representation (R_1), where the individuals are given DRs and make decisions on which rule to use for dispatch jobs onto available machines. The second representation is an arithmetic representation (R_2) where the individuals represent priority function trees. The third representation (R_3) combines both R_1 and R_2 representations, where an individual can define its own priority function tree that is used in conjunction with the decision tree. They showed that out of the three GP representations, R_3 performed better than both R_1 and R_2. In addition, they showed that the evolved rules are competitive with meta-heuristics such as a hybrid GA [17] proposed in the literature.

3 The New Approaches

This section proposes two approaches. The first approach evolves simple priority based dispatching rules denoted Genetic Programming for Job Shop Scheduling (GP-JSS) approach. This extends Nguyen et al.'s [11] arithmetic representation for GP to evolve dispatching rules, and will be used as a benchmark. The second approach is EGP-JSS, which evolves an ensemble of priority rules simultaneously.

3.1 GP Representation

For both GP-JSS and EGP-JSS, the dispatching rules generated are non-delay. In a non-delay schedule, a job is selected to be processed on machine i as soon as machine i is ready to process a new job if there are any jobs waiting to be processed at that machine. We denote the number of idle jobs waiting at a machine i as W_i. Tree-based GP is used, and the individuals in the GP population represent arithmetic function trees. The function trees generate priorities for the jobs waiting to be processed by machine i. How these priorities are used to select the job to process differs between GP-JSS and EGP-JSS, and is discussed further below.

The terminal set consists of the properties of the job shop scheduling environment discussed in Sect. 2.1. These are shown in Table 1. These extend the terminal set used by Nguyen et al. [11] in their comparison of different GP representations. The new added terminals are the number of waiting jobs (NJ), and a sufficiently large value (LV). The function set consists of the operators $+$, $-$, $\times$, protected division $/$, and if. For the ternary if operator, the value of the second subtree `if` will be returned if the value of the first subtree representing the conditional is ≥ 0; otherwise, the value of the third subtree `else` is returned.

Table 1. The terminal set used for the GP representations, where job j is one of the job waiting to be processed as soon as machine i is ready.

Terminal	Description	Value
RJ	Operation ready time	$r(\sigma_{ji})$
RO	Remaining number of operations of job j	$N_j - i + 1$
RT	Remaining total processing times of job j	$\sum_{k=i}^{N_j} p(\sigma_{jk})$
PR	Operation processing time	$p(\sigma_{ji})$
RM	Machine ready time	R_{M_i}
NJ	Idle jobs waiting at machine	W_i
#	Constant	Uniform[0,1]
LV	Sufficiently large value	∞

To evaluate an individual x in the GP population, the individual is used as a non-delay dispatching rule to generate solutions on T_{train} sample training JSS problem instances. For each JSS problem instance I, a lower bound LB_I is calculated for the makespan as specified by Taillard [15]. From the solution, the makespan objective, $Obj(x, I)$, is calculated and the deviation dev_I of $Obj(x, I)$ from LB_I, as shown in Eq. (1), is used as the fitness value for individual x for the specific problem instance I. The average fitness $fitness_{avg}(x)$ of individual x over the entire training set T_{train} is given by Eq. (2).

$$fitness(x, I) = dev_I = \frac{Obj(x, I) - \text{LB}_I}{\text{LB}_I} \tag{1}$$

$$fitness_{avg}(x) = \frac{1}{T_{train}} \sum_{t=1}^{T_{train}} fitness(x, I_t) \tag{2}$$

For the EGP-JSS approach, we use two fitness functions. The first fitness function is simply the one used for GP-JSS, where $fitness(x) = fitness_{avg}(x)$. This is denoted as 'No Fitness Modification' (NFM). The second fitness function takes diversity of the indviduals in the ensemble into account by penalising similar individuals, and is denoted as 'With Fitness Modification' (WFM). WFM is covered in detail in Sect. 3.3.

3.2 Genetic Programming for Job Shop Scheduling (GP-JSS)

The GP-JSS approach uses a GP population of individuals to evolve a single tree as its output. GP-JSS is an extension of the R_2 representation proposed by Nguyen et al. [11] that uses the extended set of terminals provided in Table 1. How a job is selected in a non-delay priority based DR is illustrated in Fig. 1. When selecting which job to process for a free machine, an individual in the population is used to assign priority values to each of the idle jobs waiting to be processed by the machine. The job with the highest priority is then selected to be processed. This continues until all operations have been completed.

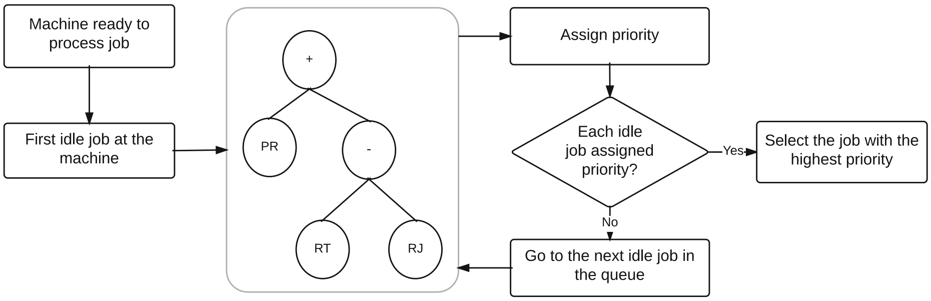
Machine ready to process job
First idle job at the machine
+
PR
-
RT
RJ
Assign priority
Each idle job assigned priority?
Yes
No
Select the job with the highest priority
Go to the next idle job in the queue

Fig. 1. Priority based dispatching rule job selection for available machine.

3.3 Ensemble Genetic Programming for Job Shop Scheduling (EGP-JSS)

EGP-JSS evolves dispatching rules which are used in an ensemble of priority rules to determine which job to process for a ready machine. However, using a single population for ensembles will require a carefully designed grouping scheme to group the individuals together, along with a complementary evaluation scheme to evaluate the grouped individuals. Instead of doing this, we consider an approach where we partition the population into S smaller subpopulations. Each subpopulation has size K. EGP-JSS groups the individuals from the different subpopulations together to form an ensemble. This approach of splitting the population into smaller subpopulations that work together to solve a problem is known as cooperative coevolution [14]. By using cooperative coevolution, we allow for the subcomponents of the ensemble to apply crossover, mutation and reproduction separately, and allow for diversity between the different subcomponents.

In cooperative coevolution, individuals in a subpopulation only interact with representatives of the other subpopulations when they are being evaluated for their fitness. A representative is defined as the individual with the best fitness in a subpopulation. Initially, before the first fitness evaluation, the representative of each subpopulation is chosen randomly. Unlike Potter and De Jong's [14] cooperative coevolution approach, we do not destroy unproductive subpopulations, as destroying and regenerating a new subpopulation of individuals will require a large number of generations for it to be effective.

The pseudocode of the EGP-JSS approach is shown in Algorithm 1. The job selection procedure and the fitness evaluation scheme is discussed further below.

Job Selection Procedure. As shown in Fig. 2, for rules evolved using EGP-JSS, the decision of choosing a job for a ready machine is carried out by the individual from the different subpopulations 'voting' on the jobs, and taking the job with the most votes. An individual 'votes' for the job if the job has the highest priority assigned to it by the individual. An individual's 'voting' procedure works similar to the job selection procedure for priority based DR described for GP-JSS.

```
Data: $S, K$, $T_{train}$, number of generations $G$, fitness evaluation scheme $eval$
Result: Representative individuals $x'_1, \ldots, x'_S$
Initialise GP subpopulations $\nabla_1, \ldots, \nabla_S$
for each subpopulation $\nabla_1$ to $\nabla_S$ do
    $x'_i \leftarrow$ random individual from $\nabla_i$
end
while $G$ number of generations has not yet passed do
    for each subpopulation $\nabla_1$ to $\nabla_S$ do
        for each individual $x$ in $\nabla_i$ do
            form an ensemble $E = \{x, x'_1, \ldots, x'_S\} - \{x'_i\}$
            for each instance $I$ in training $T_{train}$ do
                /* solve $I$ using $E$ as a non-delay dispatching rule */
                while leftover operations remaining do
                    if machine $i$ is available then
                        $j \leftarrow selection(E, j_1, \ldots, j_{W_i})$
                        process job $j$ on machine $i$
                    end
                end
                $fitness(x, I) \leftarrow$ fitness of solution
            end
            /* $eval$ denotes the fitness evaluation scheme */
            $fitness(x) \leftarrow eval(fitness(x, I_1), \ldots, fitness(x, I_{T_{train}}))$
            update $x'_i$ if $fitness(x) > fitness(x'_i)$
        end
    end
end
```

Algorithm 1. The pseudocode for the EGP-JSS approach.

If there is a tie in the votes, e.g., two jobs, j_1 and j_2 have the same number of votes as each other, a tie-breaker scheme is carried out. For an individual rule x, let $\delta_x(j_1), \ldots, \delta_x(j_{W_i})$ be the priorities assigned to jobs $j_1, \ldots, j_{W_i}$ waiting to be processed at a machine. The normalised priority of a job j, is defined by Eq. (3), where $f(j) = \frac{1}{1+e^{-\delta_x(j)}}$.

$$\delta'_x(j) = \frac{f(j)}{\sum_{r=1}^{W_i} f(j_r)} \tag{3}$$

Afterward, the job with the highest sum of priority values over all ensemble members out of the top voted jobs is then selected for processing.

With Fitness Modification (WFM) Evaluation Scheme. The WFM fitness function, which takes diversity of individuals into account, is defined as follows. To evaluate the diversity of an individual, the phenotype of individuals in a subpopulation are compared against the representative individuals of the other subpopulations. In this case, the phenotype is defined as the list (of length L_I) of all the priorities that are calculated for the jobs as the solution for

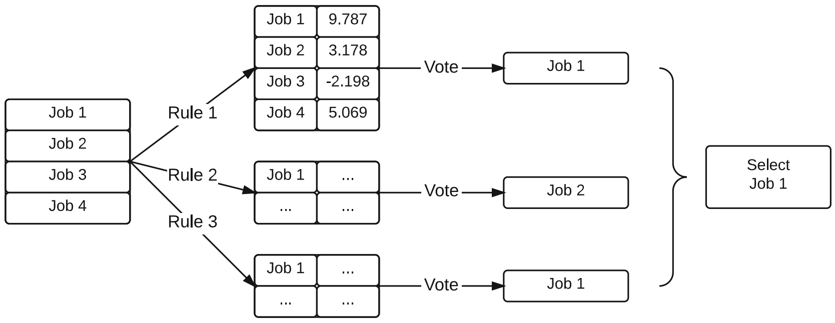

Fig. 2. Example of ensemble job selection process for an available machine.

the problem instance I is being generated. This includes the priorities that are calculated for waiting jobs that were not selected for processing at a particular moment of decision. These are normalised on the interval $[0, 1]$ using a sigmoid function $g(x, z) = \frac{1}{1+e^{-\delta_x(z)}}$, where x is the GP individual being evaluated, and $\delta_x(z)$ is the z^{th} priority calculated by x.

After all the priorities are normalised, the penalty is the average of the squared differences between the priorities of the individual x of a subpopulation to the representative individuals y of the other subpopulations, as shown by Eq. (4).

$$penalty(x, I) = 1 - \sum_{y=1, y \neq x}^{S} \sum_{z=1}^{L_I} \frac{(g(x, z) - g(y, z))^2}{(S-1)L_I} \tag{4}$$

To incorporate the penalty into the fitness evaluation of an individual x in the subpopulation, the average penalty of the individual $penalty_{avg}(x)$ is calculated over all problem instances by taking the mean of penalties. The average fitness from Eq. (2) is then multiplied by one plus the average penalty to get the final fitness $fitness(x) = fitness_{avg}(x)(1 + penalty_{avg}(x))$. This means that when an individual x from a subpopulation is very different from the representatives of the other subpopulations, the $penalty_{avg}(x) \approx 0$, and hence $fitness(x) \approx fitness_{avg}(x)$.

4 Experimental Design

For training and testing, we use the JSS benchmark dataset proposed by Taillard [15]. The dataset consists of 8 sets of 10 problem instances broken up by the number of jobs and the number of machines. All jobs in each problem instance have zero release times and setup times, and must be processed on all machines.

For training, we use three separate sets of JSS problem instances from the Taillard dataset. The first training set Δ_1 is the first five problem instances from the set of data containing $N = 15$ jobs and $M = 15$ machines. The second training set Δ_2 is the first five problem instances from the set of data containing $N = 30$ jobs and $M = 20$ machines. The third training set Δ_3 is the first five problem instances from the set of data containing $N = 100$ jobs and $M = 20$ machines. The standard GP-JSS approach has population size of 1024. For the EGP-JSS approach, given a fixed number of subpopulations S, the subpopulation size is given by $K = \lfloor \frac{1024}{S} \rfloor$. This gives us a total number of individuals in the EGP-JSS approach that is approximately equal to the population size of the GP-JSS approach. For $S = 3, 4, 5, 6, 7, 8, 9, 10$, this gives us $K = 341, 256, 204, 170, 146, 128, 113, 102$ respectively. These are shown in Table 2 with the notation $\langle S, K \rangle$, along with the other parameters used for GP. The GP-JSS and the EGP-JSS approaches were run over each training set 30 times using different seeds, resulting in 30 evolved dispatching rules over each training set. For testing, the problem instances that are not used in the training sets Δ_1, Δ_2 or Δ_3 are used, meaning that there are 65 problem instances in the test set.

Table 2. GP parameters used for evolving rules

Parameter	GP-JSS Value	EGP-JSS Value
⟨Subpopulations, Subpopulation sizes⟩	$\langle 1, 1024 \rangle$	$\langle 3, 341 \rangle$, $\langle 4, 256 \rangle$, $\langle 5, 204 \rangle$, $\langle 6, 170 \rangle$, $\langle 7, 146 \rangle$, $\langle 8, 128 \rangle$, $\langle 9, 113 \rangle$, $\langle 10, 102 \rangle$
Crossover rate	80 %	80 %
Mutation rate	10 %	10 %
Reproduction rate	10 %	10 %
Generations	51	51
Max-depth	8	8
Selection method	tournament selection	tournament selection
Selection size	7	7
Initialisation	ramped-half-and-half	ramped-half-and-half

The R_2 representation proposed by Nguyen et al. [11] is used as a benchmark for GP-JSS and EGP-JSS. R_2 will have the same parameter settings as the GP-JSS approach. As a benchmark, the R_2 representation proposed by Nguyen et al. [11] is used to compare the robustness of the rules evolved using GP-JSS and EGP-JSS. Afterward, the GP-JSS and the EGP-JSS approaches are compared against benchmark DRs. The first two benchmarks are simple non-delay schedules that select jobs to process on an available machine by the order of their arrival (FIFO); and selecting jobs by the shortest processing time (SPT). The other benchmarks are the best rules evolved by Nguyen et al. [11] for their R_1, R_2 and R_3 representations, which are used as non-delay dispatching rules. As evolved rules from EGP-JSS are not improvement heuristics, they are not compared against state-of-the-art meta-heuristic approaches to static JSS which compensate for long running time by producing very good solutions to static JSS problem instances.

5 Results

The solution's deviation dev_I (see Eq. (1)) from the lower bound is used for measuring the quality of the solution generated by the DRs. Afterward, the average of all the problem instances, denoted as dev_{avg}, is used for evaluating the DRs over the entire test set. In the tables that follow, sets of rules evolved by EGP-JSS that perform significantly better than the rules evolved by GP-JSS and R_2 are marked with †. The standard z-test is used to compare the DRs against each other. One set of evolved rules is considered significantly better than another if the obtained p-value under the statistical test is less than 0.05.

5.1 Parameter Settings Evaluation

First, the different ⟨Subpopulations, Subpopulation sizes⟩ in Table 2 are compared against each other to find the 'best' configuration. In addition, we compare WFM and NFM against each other to see whether the modified evaluation scheme (WFM) improves the performance of the evolved rules. The preferred configuration for EGP-JSS is used for comparison against the other benchmarks. This is shown in Table 3.

From the results of Table 3, we can see that the results of EGP-JSS under different parameter settings are similar to each other. No configuration is significantly better than other configurations. This means that when K scales with

Table 3. dev_{avg} of evolved rules from EGP-JSS for the Taillard's dataset for different $\langle S, K\rangle$ and for the fitness functions WFM and NFM

		Δ_1	Δ_2	Δ_3	Testing
WFM	$\langle 3, 341\rangle$	0.47 ± 0.07	0.36 ± 0.08	0.06 ± 0.05	0.28 ± 0.06
	$\langle 4, 256\rangle$	0.45 ± 0.03	0.33 ± 0.04	0.05 ± 0.02	0.26 ± 0.03
	$\langle 5, 204\rangle$	0.45 ± 0.04	0.34 ± 0.03	0.05 ± 0.02	0.26 ± 0.02
	$\langle 6, 170\rangle$	0.47 ± 0.04	0.35 ± 0.05	0.06 ± 0.04	0.27 ± 0.04
	$\langle 7, 146\rangle$	0.46 ± 0.04	0.34 ± 0.04	0.06 ± 0.03	0.27 ± 0.03
	$\langle 8, 128\rangle$	0.47 ± 0.05	0.35 ± 0.04	0.06 ± 0.03	0.27 ± 0.03
	$\langle 9, 113\rangle$	0.46 ± 0.03	0.35 ± 0.04	0.06 ± 0.03	0.27 ± 0.03
	$\langle 10, 102\rangle$	0.46 ± 0.03	0.35 ± 0.04	0.06 ± 0.02	0.27 ± 0.02
NFM	$\langle 3, 341\rangle$	0.49 ± 0.08	0.36 ± 0.08	0.07 ± 0.06	0.29 ± 0.07
	$\langle 4, 256\rangle$	0.47 ± 0.07	0.35 ± 0.07	0.06 ± 0.05	0.27 ± 0.06
	$\langle 5, 204\rangle$	0.46 ± 0.05	0.35 ± 0.05	0.06 ± 0.04	0.27 ± 0.05
	$\langle 6, 170\rangle$	0.46 ± 0.05	0.35 ± 0.05	0.06 ± 0.03	0.27 ± 0.04
	$\langle 7, 146\rangle$	0.45 ± 0.02	0.34 ± 0.03	0.05 ± 0.02	0.26 ± 0.02
	$\langle 8, 128\rangle$	0.45 ± 0.04	0.34 ± 0.04	0.05 ± 0.03	0.26 ± 0.03
	$\langle 9, 113\rangle$	0.45 ± 0.02	0.34 ± 0.02	0.05 ± 0.01	0.26 ± 0.01
	$\langle 10, 102\rangle$	0.46 ± 0.03	0.34 ± 0.03	0.05 ± 0.02	0.26 ± 0.02

S, the value of S is not significant to the performance of the evolved rules under the EGP-JSS approach.

5.2 GP-JSS and EGP-JSS

From the results of Sect. 5.1, we selected the configuration with $\langle 4, 256 \rangle$ that uses the modified fitness measure WFM to be compared against the GP-JSS and R_2 approaches. Although $\langle 4, 256 \rangle$ with WFM is not significantly better than the other configurations, it had the lowest mean deviation for the test set. For each approach, 30 rules are evolved using the training sets Δ_1, Δ_2 and Δ_3, and their performances over training runs and the test runs over the respective training and the test sets are used. This is shown in Table 4.

Table 4. dev_{avg} of evolved rules from GP-JSS and EGP-JSS for $Jm||C_{\max}$.

	Training			Testing		
	R_2	GP-JSS	EGP-JSS	R_2	GP-JSS	EGP-JSS
Δ_1	0.59 ± 0.15	0.57 ± 0.11	$0.45 \pm 0.03^\dagger$	0.37 ± 0.13	0.36 ± 0.12	$0.26 \pm 0.04^\dagger$
Δ_2	0.40 ± 0.15	0.40 ± 0.11	$0.33 \pm 0.04^\dagger$	0.32 ± 0.13	0.32 ± 0.10	$0.26 \pm 0.03^\dagger$
Δ_3	0.11 ± 0.10	0.12 ± 0.10	$0.06 \pm 0.01^\dagger$	0.32 ± 0.13	0.34 ± 0.12	$0.26 \pm 0.01^\dagger$

Although the GP-JSS extended the R_2 approach by adding more terminals to the terminal set, we can see in Table 4 that it did not improve on the original approach significantly. It is likely that the added terminals representing the number of idle jobs waiting at the machine (NJ) and sufficiently large value (LV) are not important to the sequencing decisions that are made by the DRs.

However, we can see that the rules evolved using EGP-JSS perform significantly better than the rules evolved using the GP-JSS and R_2 approaches, outperforming the other rules evolved under the three training sets Δ_1, Δ_2 and Δ_3. In addition, the rules evolved with EGP-JSS have much lower standard deviations, meaning that the evolved rules mostly performed similar to each other and are more stable than those evolved with GP-JSS and R_2. The results show that EGP-JSS can potentially produce more robust rules than the "standard" approach.

5.3 Evolved Rules and Benchmark Dispatching Rules

The final evaluation compares the best rules evolved from each training set using GP-JSS and EGP-JSS against other dispatching rules over the training and the test sets. The best evolved rules from GP-JSS and EGP-JSS are denoted as Θ_1^{GP} and Θ_1^{EGP} respectively, where the subscript on Θ denotes each training set (e.g. Θ_1 means best rule trained over Δ_1). The first two benchmarks are non-delay FIFO and SPT dispatching rules. The three other benchmarks are rules evolved by Nguyen et al. [11] using the three different representations for individuals in the GP population. The best rule from their R_1 representation, R_2 and R_3 are denoted as $\Theta^{c1}_{R_1}$, $\Theta^{c2}_{R_2}$ and $\Theta^{c3}_{R_3}$ respectively. This is shown in Table 5.

Table 5. Deviation of the DRs against the lower bound for the training sets (Δ_1, Δ_2, Δ_3) and the entire dataset.

Rule	Δ_1			Δ_2			Δ_3			Testing		
	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min	Avg	Max
FIFO	0.53	0.65	0.81	0.49	0.55	0.58	0.14	0.20	0.23	0.17	0.44	0.94
SPT	0.44	0.58	0.78	0.33	0.50	0.64	0.13	0.15	0.18	0.08	0.39	0.81
$\Theta^{c1}_{R_1}$	0.49	0.75	0.99	0.63	0.69	0.71	0.31	0.36	0.43	0.34	0.61	1.27
$\Theta^{c2}_{R_2}$	0.38	0.44	0.47	0.27	0.33	0.36	0.01	0.04	0.07	0.00	0.24	0.67
$\Theta^{c3}_{R_3}$	0.38	0.48	0.59	0.35	0.40	0.45	0.06	0.09	0.12	0.06	0.29	0.68
Θ^{GP}_1	0.35	0.44	0.56	0.27	0.31	0.34	0.03	0.05	0.09	0.01	0.24	0.63
Θ^{GP}_2	0.33	0.42	0.51	0.26	0.32	0.36	0.03	0.05	0.07	0.02	0.24	0.60
Θ^{GP}_3	0.38	0.44	0.47	0.28	0.31	0.36	0.01	0.03	0.07	0.02	0.25	0.57
Θ^{EGP}_1	0.37	0.43	0.47	0.30	0.32	0.37	0.03	0.05	0.08	0.01	0.24	0.58
Θ^{EGP}_2	0.38	0.44	0.47	0.24	0.31	0.36	0.01	0.04	0.07	0.00	0.24	0.67
Θ^{EGP}_3	0.38	0.44	0.47	0.28	0.33	0.36	0.01	0.04	0.07	0.03	0.24	0.62

From the results of Table 5, we can see that the best rules from the GP-JSS and the EGP-JSS approaches perform significantly better than the two simple DRs. This reinforces the idea that evolved rules outperform the simple DRs for JSS problems literature [3,6,11]. On the other hand, the best rules for GP-JSS and EGP-JSS perform similarly to $\Theta^{c1}_{R_1}$, $\Theta^{c2}_{R_2}$ and $\Theta^{c3}_{R_3}$.

6 Conclusions

In this paper, we proposed a novel approach (EGP-JSS) of evolving an ensemble of DRs using GP and cooperative coevolution. The experimental results show that the ensemble of rules evolved from the EGP-JSS approach perform significantly better than the benchmark GP-JSS and R_2 approaches. Including the two new terminals in GP-JSS does not significantly improve the performance over R_2. The rules evolved by EGP-JSS are more robust than the simple conventional rules FIFO and SPT.

For future work, extending the ensemble approach to dynamic JSS problem would be very interesting. In dynamic JSS problems properties of jobs are not known before they arrive at the shop floor. Because of this, global optimisation techniques used in static JSS do not work in dynamic JSS. Good robust dispatching rule approach will be required to handle the uncertainity in conjunction with the standard sequencing decisions in dynamic JSS. In addition, developing a GP based ensemble approach that uses a single population would also be very useful, as it removes the need to define the number of subpopulations and their respective sizes.

References

1. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996)
2. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
3. Dimopoulos, C., Zalzala, A.M.S.: Investigating the use of genetic programming for a classic one-machine scheduling problem. Adv. Eng. Softw. **32**(6), 489–498 (2001)
4. Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. In: Vitányi, P. (ed.) Computational Learning Theory. Lecture Notes in Computer Science, pp. 23–37. Springer, Berlin (1995)
5. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Math. Oper. Res. **1**(2), 117–129 (1976)
6. Geiger, C.D., Uzsoy, R., Aytu, H.: Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. J. Sched. **9**(1), 7–34 (2006)
7. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: 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, pp. 257–264 (2010)
8. Jakobovi, D., Jelenkovi, L., Budin, L.: Genetic programming heuristics for multiple machine scheduling. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) Genetic Programming. Lecture Notes in Computer Science, vol. 4445, pp. 321–330. Springer, Heidelberg (2007)
9. Jayamohan, M.S., Rajendran, C.: New dispatching rules for shop scheduling: a step forward. Int. J. Prod. Res. **38**(3), 563–586 (2000)
10. Kreipl, S.: A large step random walk for minimizing total weighted tardiness in a job shop. J. Sched. **3**(3), 125–138 (2000)
11. 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 Trans. Evol. Comput. **17**(5), 621–639 (2013)
12. Pinedo, M.L.: Scheduling: theory, algorithms and systems development. In: Gaul, W., Bachem, A., Habenicht, W., Runge, W., Stahl, W.W. (eds.) Operations Research Proceedings 1991. Operations Research Proceedings 1991, vol. 1991, 3rd edn, pp. 35–42. Springer, Heidelberg (2012)
13. Polikar, R.: Ensemble based systems in decision making. IEEE Circuits Syst. Mag. **6**(3), 21–45 (2006)
14. Potter, M.A., De Jong, K.A.: Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol. Comput. **8**(1), 1–29 (2000)
15. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Oper. Res. **64**(2), 278–285 (1993)
16. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput. Ind. Eng. **54**(3), 453–473 (2008)
17. Zhou, H., Cheung, W., Leung, L.C.: Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. Eur. J. Oper. Res. **194**(3), 637–649 (2009)