An Investigation of Ensemble Combination Schemes for Genetic Programming based Hyper-heuristic Approaches to Dynamic Job Shop Scheduling

John Park[1], Yi Mei[1], Su Nguyen[1,2], Gang Chen[1], Mengjie Zhang[1]

[1]Evolutionary Computation Research Group, Victoria University of Wellington, PO Box 600, Wellington, New Zealand
[2]La Trobe University, Melbourne, Australia
{John.Park,Yi.Mei,Su.Nguyen,Aaron.Chen,Mengjie.Zhang}@ecs.vuw.ac.nz

Abstract. Genetic programming based hyper-heuristic (GP-HH) approaches that evolve ensembles of dispatching rules have been effectively applied to dynamic job shop scheduling (JSS) problems. Ensemble GP-HH approaches have been shown to be more robust than existing GP-HH approaches that evolve single dispatching rules for dynamic JSS problems. For ensemble learning in classification, the design of how the members of the ensembles interact with each other, e.g., through various combination schemes, is important for developing effective ensembles for specific problems. In this paper, we investigate and carry out systematic analysis for four popular combination schemes. They are majority voting, which has been applied to dynamic JSS, followed by linear combination, weighted majority voting and weighted linear combination, which have not been applied to dynamic JSS. In addition, we propose several measures for analysing the decision making process in the ensembles evolved by GP. The results show that linear combination is generally better for the dynamic JSS problem than the other combination schemes investigated. In addition, the different combination schemes result in significantly different interactions between the members of the ensembles. Finally, the analysis based on the measures shows that the behaviours of the evolved ensembles are significantly affected by the combination schemes. Weighted majority voting has bias towards single members of the ensembles.

1 Introduction

Job shop scheduling (JSS) problems are types of combinatorial optimisation problems that model manufacturing environments [1]. In a JSS problem, there is a shop floor with machines that are used to process arriving jobs. Although both academics and industry experts have interest in JSS problems, there has always been a gap between the classical research to JSS (from an academic perspective) and its application (from an industrial perspective) [2]. In classical research to JSS, many approaches handle *static* JSS problems, where the properties of the shop are known a priori [3]. However, in practice the properties of the shop are extremely variable and it is commonly believed that any change to the shop floor can cause ripple effects [2]. To bridge the gaps between the static JSS problems that have been handled by academics (where the problems are predictable and can be optimised in advance) and unpredictable real-world scenarios encountered in the industry, researchers have focused on matching the problem

more closely with real-world manufacturing environments by incorporating unforeseen events into the problem [4]. JSS problems that have real-time unforeseen events that affect the properties of jobs, machines and shop floor are called *dynamic* JSS problems [4]. Examples of unforeseen events include dynamic job arrivals, where job arrivals are unknown until they reach the shop floor, and machine breakdowns [4–6]. In real-world manufacturing environments, it is likely that last minute (and potentially urgent) jobs can arrive that require attention [4, 5]. In general, dynamic JSS problems are much more difficult than static JSS problems, and conventional optimisation methods cannot solve dynamic JSS problems due to the unpredictable changes in the shop floor [7]. Instead, *dispatching rules* [3] are studied by both academics and industry experts for dynamic JSS problems due to their interpretability [6], short reactions times and their ability to cope well with the unforeseen events in dynamic JSS problems [8]. To automate the design of effective dispatching rules, many genetic programming based hyper-heuristic (GP-HH) approaches to dynamic JSS problems have been proposed in the literature [6]. GP-HH approaches have successfully evolved dispatching rules for various dynamic JSS problems which are more effective than the man-made counterparts [6].

In addition to the primary motivation of automatically generating effective dispatching rules for dynamic JSS problems [6], many GP-HH approaches have focused on evolving *robust* dispatching rules for the dynamic JSS problems [9], i.e., rules that function reliably and effectively despite noise and unexpected changes in the problem domain. However, many approaches focus on evolving dispatching rules with a single constituent component, and are often not sufficiently robust for dynamic JSS problems. This issue was addressed by evolving *ensembles* [13] of dispatching rules [10–12]. Ensemble learning has been shown to be effective at training robust high quality rules for JSS problems [10–12] and problems outside of JSS (e.g. classification problems [13]) because ensemble members are able to minimise errors made by other ensemble members [13]. This makes ensemble approaches a promising direction to improve the robustness of rules evolved by GP-HH for dynamic JSS problems. However, the existing ensemble GP-HH approaches to JSS [10–12] only use majority voting combination scheme [13] to combine the outputs of the subcomponents of the ensembles together. Design of interactions between the ensemble members is an important factor in ensemble learning [13]. It is clearly evidenced on some classification problems that different combination schemes, such as linear combination and weighted combination schemes, can be more effective than majority voting [13]. Therefore, it may be possible that better rules can be evolved by GP using combination schemes besides majority voting. In addition, by analysing the rules evolved by the different combination schemes, one can observe the behaviour of ensembles that are applied to dynamic JSS problem instances. This allows for future ensemble GP-HH approaches which may generate higher quality and more robust rules than the current state-of-the-art GP-HH approaches for dynamic JSS problems.

1.1 Goal

For the scope of this paper, the analysed combination schemes are majority voting, weighted majority voting, linear combination and weighted linear combination [13]. The goal of this paper is to investigate and analyse the combination schemes to further

improve the robustness of ensemble GP-HH approaches for dynamic JSS problems. Majority voting was previously used by the existing ensemble GP-HH approaches to dynamic JSS problems [10–12]. However, linear combination, weighted majority voting and weighted linear combination, which have extensively been used in the classification literature [13], have not been explored in any existing research on dynamic JSS. The members of the evolved ensembles are analysed by their behaviours and interactions on complex decision situations [14]. For this paper, we propose new analysis measures to compare specific behaviours between the evolved ensembles from the different combination schemes: the diversity in the decisions made by ensemble members, the bias towards specific ensemble members, and how the different members of the ensembles rank in the ensembles. Overall, this investigation into the combination schemes for GP-HH to dynamic JSS problems is broken down into the objectives given below.

(a) Extend an existing ensemble GP-HH approach by incorporating the four different combination schemes.
(b) Investigate the performances of evolved ensembles against each other and the state-of-the-art results [8, 15].
(c) Analyse in detail the behaviour of the evolved ensembles against complex decision situations from different perspectives.

1.2 Organisation

The organisation of the paper is as follows. Section 2 includes a background into dynamic JSS and related work which provide approaches to dynamic JSS problems. This section also includes a description of ensemble learning proposed in the literature. Section 3 describes the combination schemes investigated and modifications made to an existing ensemble GP approach to incorporate the combination schemes. Afterwards, Section 4 provides a description of the analysis procedure to measure how the rules behave in the problems. Section 5 covers the experimental design, Section 6 covers the evaluation procedure, the experimental results, the discussions of the results and the analysis. Section 7 gives the conclusions and the future works.

2 Background

This section gives the dynamic JSS problem definition investigated in this paper, and approaches in the literature for tackling dynamic JSS problems, including GP-HH approaches. It also gives a brief description of ensemble learning and their applications to JSS problems.

2.1 Problem Definitions for Dynamic JSS

In a dynamic JSS problem instance, there are M machines on the shop floor. An arriving job j has a sequence of N_j operations denoted as $\sigma_{1j}, \ldots, \sigma_{N_j j}$. The ith operation of job j, denoted as σ_{ij}, needs to be processed at machine $m(\sigma_{ij})$. The operations need to be processed in the order of their indices, e.g., operation σ_{2j} cannot start until

operation σ_{1j} has been processed and completed on machine $m(\sigma_{1j})$. The duration of time operation σ_{ij} is processed on the machine is called the *processing time* [3], and is denoted as $p(\sigma_{ij})$ (abbreviated to p_{ij}). In addition, a machine can only process one operation at a time. There is no re-entry (job has two or more operations on the same machine) and preemption (a job's operation being processed on a machine can be interrupted) [3]. The time when job j arrives at the relevant machine that the job's operation is currently up to is called the *operation ready time* [3], and is denoted as $r(\sigma_{ij})$ (abbreviated to r_{ij}). The time when the job j arrives on the shop floor is the operation ready time of the first operation of job j (denoted as r_j), and is called the *release time* of job [3]. The goal of JSS is to complete all arriving jobs by processing the operations of the jobs on the machines. The sequence of times and jobs that are processed on the machines is called a *schedule*, and the goal is to generate a schedule that is optimal given an *objective function* [3]. For this paper, we focus on the objective of minimising the mean tardiness (MT) of the schedule. In a problem instance with a tardiness related objective, a job j also has a due date d_j. The time when all of a job's operations have been completed is called the job *completion time*. If a job j's completion time C_j is greater than its due date d_j, then the job is considered *tardy* and has a tardiness $T_j = C_j - d_j$. Otherwise, a job completed before its due date has a tardiness value of zero, i.e., $T_j = 0$ if $C_j \leq d_j$. Afterwards, the mean tardiness of a schedule which has processed N jobs arriving on the shop floor is given by $\frac{1}{N}\sum_{j=1}^{N} T_j$, i.e., is the average tardiness over the N jobs completed in the schedule. JSS problems with tardiness related objective has been extensively investigated in the literature as they are strongly NP-hard [16]. In addition, we focus on dynamic JSS problems with dynamic job arrivals. This means that a job j's properties such as its operations and due date are unknown until it arrives on the shop floor at time r_j. In other words, a scheduling algorithm has limited information about properties of the shop floor during processing, i.e., the problems have constrained information horizon [17].

2.2 Related Work

Dispatching Rules for Dynamic JSS: Due to the constrained information horizon [17] in dynamic JSS problems, conventional optimisation techniques, which require exact information about the shop floor in advance, are completely impractical [7]. Therefore, heuristic approaches that make effective real-time decisions to unforeseen events are the prominent methods for handling dynamic JSS problems [6]. Many approaches focus on handling a very small number of dynamic events at a time, focusing on generating robust solutions to DJSS problem instances that are insensitive to the disruptions caused by the dynamic events [4]. However, we focus on dispatching rule based approaches to dynamic JSS problems. This is because dispatching rules can cope well with frequent dynamic job arrival events that occur in the dynamic JSS problem covered in this paper, and have been prominently applied to other DJSS problems with a large number of dynamic events in the literature [6, 18]. Dispatching rules are local decision makers that select jobs to be processed when the machines required by the jobs are available [3]. At every *decision situation* [14], when a machine m^* is available and there are jobs waiting at machine m^*, the dispatching rule selects a job out of the waiting jobs. The selection criteria for the jobs by the dispatching rules are based on the shop floor

attributes at the time of processing. The selected job is then processed by machine m^*. A partial schedule generated by dispatching rules during processing only consists of the operations that are currently being processed or have been completed. This means that dynamic events such as unforeseen job arrivals do not affect previously made decisions for the dispatching rules.

Dispatching rules range in complexity, and their relative effectiveness vary across JSS problems [19, 20]. The most simple dispatching rules incorporate single job attributes for the decision making procedure, e.g., shortest processing time (SPT), first-in-first-out (FIFO) [3]. On the other hand, composite dispatching rules [21–23] combine the decisions of multiple smaller dispatching rules to construct a dispatching rule that can handle a JSS problem more effectively than their component rules. However, dispatching rules have no guarantee that they remain effective for dynamic JSS problems outside of the specific problems they were designed for [8, 19, 20, 24], and there is a tradeoff in the performances of dispatching rules over the different dynamic JSS problems [19, 20]. Therefore, new rules often need to be designed as new dynamic JSS problems are encountered, which requires human experts to carry out extensive trial-and-error tests to verify the effectiveness of the dispatching rules [25].

Evolving Dispatching Rules using GP-HH: Because manually designing effective dispatching rules is difficult, a major motivation behind many GP-HH approaches is to develop an effective hyper-heuristic approach that can then automatically evolve high quality dispatching rules from building blocks [6]. Designing building blocks for specific dynamic JSS problems is significantly less difficult than designing complete dispatching rules [25]. The term "hyper-heuristics" have been frequently used by researchers in the literature to describe the automatic design of dispatching rules using evolutionary computation (e.g. GP) and machine learning techniques [5, 6, 12, 14, 18]. This includes the GP-HH approach proposed by Nguyen et al. [8], who investigated three different GP representations to evolve different types of dispatching rules. They found that the GP evolved rules performed competitively with the state-of-the-art heuristics in the literature for some static JSS problems, and outperform man-made dispatching rules. Hildebrandt et al. [5] and Hunt et al. [26] evolved dispatching rules for dynamic JSS problems and showed that they perform better than man-made dispatching rules. Branke et al. [6] and Nguyen et al. [18] provide comprehensive surveys of evolutionary scheduling approaches in the literature, including GP-HH approaches that evolve dispatching rules. Finally, Nguyen et al. [27] provides a unified framework for the GP-HH for production scheduling and link the existing GP-HH approaches to specific parts of the framework.

To evolve high quality dispatching rules using GP-HH, several different design decisions need to be made. This paper focuses on the design of the GP process and how its output is converted to a dispatching rule. In particular, we focus on cooperative co-evolutionary GP-HH approaches to dynamic JSS problems. In cooperative coevolution, multiple interacting subcomponents are evolved that can be combined together to form a rule [28]. To apply cooperative coevolutionary algorithms to a dynamic JSS problem, the problem is first decomposed into smaller subproblems, where each subproblem can be solved by specialised agents. Approaches that incorporate cooperative coevolution

aim to evolve different individuals that can fill different "ecological niches" [29] and can work together to solve difficult problems more effectively. Coevolutionary GP-HH approaches are not standard in the literature, and many GP-HH approaches evolve single dispatching rules that work independently [5, 6, 8, 15, 26, 30–32]. Although cooperative coevolutionary GP-HH approaches are often more complex and computationally intensive than the standard GP-HH approach of evolving single rules, various coevolutionary GP-HH approaches have produced high quality dispatching rules that are significantly better than the standard GP single rules. Nguyen et al. [9] proposed a coevolutionary GP-HH approach for a multi-objective JSS problem, where individuals from the first subpopulation predict the due dates of the jobs at decision situations, and the individuals from the second subpopulation incorporate the due date information to calculate the priorities of the jobs. They showed that the approach performs favourably compared to good multi-objective GP approaches such as NSGA-II [33] and SPEA2 [34]. In addition, there are cooperative coevolutionary GP approaches that evolve ensembles in the literature.

Ensemble Learning and Ensemble GP-HH in Dynamic JSS: Although many effective ensemble approaches have been applied to a number of problems outside of JSS (e.g. classification, regression [13]), only a few ensemble approaches have been proposed for JSS. Park et al. [10] proposed a GP-HH approach for evolving ensembles of dispatching rules using Potter and De Jong's Cooperative Coevolution framework [29], and found that the evolved ensembles outperform the standard GP-HH approach for a static JSS problem. Park et al. [11] then adapted multi-level genetic programming (MLGP) proposed by Wu and Banzhaf [35] to evolve ensembles for a dynamic JSS problem, which evolves better rules than standard GP-HH with comparable evolution times. Hart and Sim [12] proposed an ensemble approach for a static JSS problem where artificial immune network is used to evolve highly specialised dispatching rules. They showed that the ensembles evolved by this approach outperform the ensembles evolved by Park et al. [10] for the static JSS problems.

One major design consideration when developing an ensemble learning approach is the scheme used to combine the multiple outputs of the members of the ensemble into a final decision [13]. The evolutionary approaches that evolve ensembles described above [10–12] use majority voting [13] to combine the outputs of the ensemble members together. In classification, various studies have shown that the effectiveness of different aggregation techniques depend on the properties of the problem such as noise, the effectiveness of ensemble members, classifiers used to generate the ensemble members, etc. [36–38]. Overall, there are specific scenarios in classification where combining the outputs of the ensemble members discretely (e.g. majority voting) is more effective than a numeric combination (e.g. linear combination), and other scenarios where linear combination is more effective [36, 37, 13]. In other words, the effectiveness of a combination scheme is specific to the properties of the classification problem [13]. Therefore, it may be the case that a combination scheme that has not been investigated in dynamic JSS works better than majority voting for a specific dynamic JSS problem domain. Parallels can be made between applying ensembles to classification problems and to dynamic JSS problems, as handling a decision situation that occurs during processing

for a JSS problem instance shows similarities to classifying an instance in a classification problem as a specific job is selected off the list of waiting jobs to be processed at a machine. Because of this, it is unclear which combination scheme will likely work best for handling dynamic JSS problems. Another major design consideration is diversifying ensemble members, where a balance needs to be made between the members being able to cover for each others' errors, but also avoid making poor decisions as often as possible [13]. In summary, Polikar [13] provides a comprehensive survey of ensemble learning and the various ensemble learning approaches to classification.

Final Remarks on Related Work: In summary, many GP-HH approaches have been proposed for dynamic JSS that evolve high quality dispatching rules that generally outperform man-made dispatching rules [6]. However, they focus on evolving single dispatching rules, which may not be sufficiently robust to handle noise and unexpected changes in dynamic JSS problems. In addition, the existing ensemble GP-HH approaches to dynamic JSS that evolve ensembles have shown to outperform benchmark GP-HH approaches that evolve single dispatching rules [10–12]. On the other hand, existing ensemble GP-HH approaches to dynamic JSS have not investigated combination schemes (outside of majority voting) that may be more effective for specific dynamic JSS problem domains. In addition, they have not carried out extensive behavioural analysis, nor proposed analysis measures to compare the different ensemble GP-HH approaches. Therefore, comparing combination schemes over a dynamic JSS problem and developing various analysis measures may also allow us to understand ensemble GP-HH approaches for dynamic JSS and develop methods that can evolve higher quality and more robust rules.

3 Modified EGP-JSS Approach and Combination Schemes

The combination schemes investigated are majority voting, linear combination, weighted majority voting and weighted linear combination [13]. Ensemble GP-HH approaches that use linear combination, weighted majority voting and weighted linear combination schemes have not been investigated in the literature for dynamic JSS problems. An existing ensemble GP-HH called Ensemble Genetic Programming for JSS (EGP-JSS) adapted from the literature so that the investigated combination schemes can be incorporated to the GP's training process [10]. However, the original EGP-JSS approach needs to be modified to evolve the weighted majority voting and linear combination schemes. Therefore, the modified EGP-JSS (mEGP-JSS) extends the GP-HH approach by simultaneously evolving the weights for the weighted ensemble by incorporating a genetic algorithm (GA) [39] to the cooperative coevolutionary procedure. This section first provides descriptions of the combination schemes and how they are applied to the decision situations. Afterwards, we cover the changes made by mEGP-JSS for weighted combination schemes and to the fitness evaluation.

3.1 Job Selection by Combination Schemes

The combination schemes are used by the ensembles during job sequencing decisions at decision situations. The majority voting scheme is adapted from existing ensemble

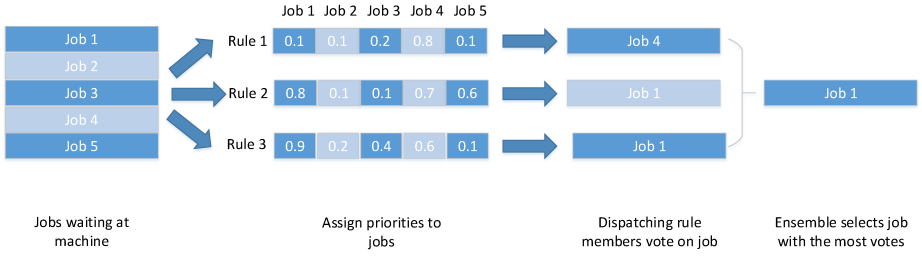

Fig. 1: Example of majority voting for ensembles being applied to a decision situation.

GP-HH approaches to dynamic JSS [10–12], and the apparent tardiness cost (ATC) rule [40] is used as a tiebreaker. Figure 1 shows an example of majority voting with three ensemble members being applied jointly to a decision situation with five jobs.

For the linear combination scheme, the members of the ensemble assign "scores" to the different jobs, and the job with the highest sum score is selected to be processed. The score of a job assigned by a member is calculated from the assigned priorities using min-max normalisation. Given that a member ω assigns priorities $\delta_{1,\omega}, \ldots, \delta_{L,\omega}$ to the L jobs waiting at the machine, the priority assigned to job j is converted into a score $s_{j,\omega}$ as shown in Equation (1). In the Equation, $\delta_{\min}$ and $\delta_{\max}$ are the minimum and maximum priorities assigned to any jobs waiting at the machine by member ω.

$$s_{j,\omega} = \frac{\delta_{j,\omega} - \delta_{\min}}{\delta_{\max} - \delta_{\min}} \tag{1}$$

For the weighted majority voting scheme, each member ω has a weight ν_ω. This means that the member assigns a score equal to its weight to the job that it votes for. For the weighted linear combination scheme, an intermediate value is first calculated for a job j by ensemble member ω by normalising the priorities (Equation (1)). Afterwards, the intermediate value is multiplied by the member's weight ν_ω to get the final score for the job.

When applied to a decision situation, different combination schemes can potentially select different jobs for processing. For example, Table 1 shows ensembles with combination schemes being applied to a decision situation with five jobs. In the tables, the six ensemble members have the weights $3.5, 1.0, 2.0, 1.2, 2.5, 6.5$. For the majority voting scheme, job 1 is selected as rules 2, 3 and 4 vote on job 1. For the linear combination scheme, job 4 is selected because rules 1 and 5 heavily favour job 4 and job 4 has the second highest priorities for rules 2 and 3, resulting in job 4 having a higher score than job 1 overall. For the weighted majority voting scheme, job 3 is selected because rule 6 has a high weight compared to the other rules, resulting in a higher score. Likewise, job 3 is selected for the weighted linear combination because of the high weight of rule 6 contributing towards the total score towards the job. The different outcomes for the decision situations encountered by the ensembles using the different combination schemes may lead to the ensembles generating significantly different schedules and performances overall. It may also lead to significantly different rule structures and behaviours being evolved by mEGP-JSS.

Table 1: Examples of the combination schemes for an ensemble with three members being applied to a decision situation with five jobs waiting at the machine. The weights of the members are $\nu = 3.5, 1.0, 2.0, 1.2, 2.5, 6.5$ respectively.

Rules/Members			Waiting Jobs					Selected Jobs
			Job 1	Job 2	Job 3	Job 4	Job 5	
Priorities		1	0.10	0.10	0.20	0.80	0.10	-
		2	0.80	0.10	0.10	0.70	0.60	
		3	0.90	0.20	0.40	0.60	0.10	
		4	0.60	0.50	0.10	0.20	0.30	
		5	0.20	0.10	0.30	0.90	0.70	
		6	0.10	0.10	0.90	0.10	0.10	
Job Scores	Majority voting	1	0.00	0.00	0.00	1.00	0.00	Job 1
		2	1.00	0.00	0.00	0.00	0.00	
		3	1.00	0.00	0.00	0.00	0.00	
		4	1.00	0.00	0.00	0.00	0.00	
		5	0.00	0.00	0.00	1.00	0.00	
		6	0.00	0.00	1.00	0.00	0.00	
	Linear combination	1	0.00	0.00	0.14	1.00	0.00	Job 4
		2	1.00	0.00	0.00	0.86	0.71	
		3	1.00	0.13	0.38	0.63	0.00	
		4	1.00	0.80	0.00	0.20	0.40	
		5	0.13	0.00	0.25	1.00	0.75	
		6	0.00	0.00	1.00	0.00	0.00	
	Weighted majority voting	1	0.00	0.00	0.00	3.50	0.00	Job 3
		2	1.00	0.00	0.00	0.00	0.00	
		3	2.00	0.00	0.00	0.00	0.00	
		4	1.20	0.00	0.00	0.00	0.00	
		5	0.00	0.00	0.00	2.50	0.00	
		6	0.00	0.00	6.50	0.00	0.00	
	Weighted linear combination	1	0.00	0.00	0.50	3.50	0.00	Job 3
		2	1.00	0.00	0.00	0.86	0.71	
		3	2.00	0.25	0.75	1.25	0.00	
		4	1.20	0.96	0.00	0.24	0.48	
		5	0.31	0.00	0.63	2.50	1.88	
		6	0.00	0.00	6.50	0.00	0.00	

3.2 Incorporating Weighted Combination Scheme to EGP-JSS

To evolve ensembles that use weighted combination schemes, EGP-JSS's GP process is modified to evolve weights for the members of the ensembles and the ensemble members simultaneously in a single evolutionary run. This allows us to evolve weighted ensembles with similar computation times as the computation times required to evolve unweighted ensembles, as other approaches (such as a two-step procedure) would require added computation after the GP process is completed. Given that there are Ψ GP subpopulations of size Φ, an additional $(\Psi + 1)$th GA subpopulation of size Φ_ν are also initialised along with the GP population. The GA individuals in the additional subpopulation are real-valued vectors of length Ψ, where the gene value at index i corresponds to the weight ν_i of an individual from subpopulation i. The gene values have a lower bound of zero, which prevents the member weights from being negative. On the other hand, from pilot experiments we found that the choice in the upper bound did not make a significant difference in the performance of the weighted ensembles during the preliminary experiments. Therefore, the upper bound of ten is selected as a rule of thumb. During the evaluation procedure, the GP individual being evaluated is grouped with the representatives from other subpopulations, i.e., the individuals with the best fitnesses

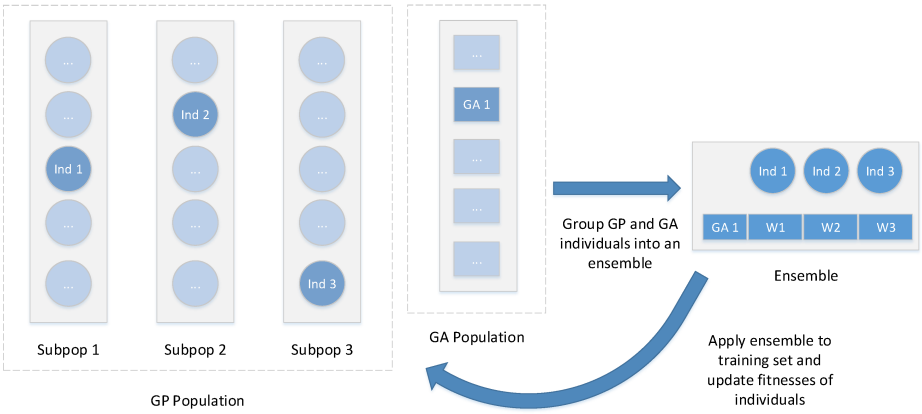

Fig. 2: Example of mEGP-JSS generating an ensemble with weighted members.

from each subpopulations [29], and the representative of the GA subpopulation to form a weighted ensemble.

Similar to how a GP individual in EGP-JSS is evaluated, a GA individual is evaluated by grouping it up with the representatives from the GP subpopulations to form a weighted ensemble. An example of this is shown in Fig. 2, where the ensemble consists of GP individuals Ind 1, Ind 2 and Ind 3 from the three GP subpopulations. Afterwards, Ind 1, Ind 2 and Ind 3 are weighted using the values from GA 1. The weighted ensemble is then applied to the training instances. However, the fitness calculation for the GP and the GA individuals is modified from the original EGP-JSS approach, and is described below (Section 3.3). After all GP and GA individuals have been evaluated, the GA representative is updated to the individual with the best performance. Afterwards, standard one point crossover operator and a gaussian mutation operator [39] are used for breeding the next generation of individuals. The output of the GP process is the representatives from the GP subpopulations that form an ensemble, along with the representative from the GA subpopulation that assigns the weights to the members of the ensemble.

3.3 mEGP-JSS Performance and Fitness Calculation

The fitness calculation in mEGP-JSS is modified from the original EGP-JSS. After an ensemble E containing a GP individual ω is applied to the dynamic JSS training instances, the fitness $f(\omega)$ of individual ω is calculated by normalising the performances of the ensemble over the problem instances using the ATC as a *reference rule*. First, both the ensemble E and the reference rule R is applied to a problem instance γ to obtain the schedules with the objectives values $\text{MT}(E,\gamma)$ and $\text{MT}(R,\gamma)$ respectively. Afterwards, the normalised objective value $\text{MT}'(E,\gamma)$ of the ensemble over the instance is given in Equation (2).

$$\text{MT}'(E,\gamma) = \frac{\text{MT}(E,\gamma)}{\text{MT}(R,\gamma)} \tag{2}$$

In the literature, normalisation of performances over the problem instances have been adopted in fitness calculation for GP-HH approaches to dynamic JSS to reduce the bias towards specific training instances [5, 41]. If a particular JSS problem instance has a greater optimal mean tardiness value than another JSS problem instance, we can expect many dispatching rules to obtain higher mean tardiness values on this problem instance. This results in the first problem instance having a greater effect on the fitness of an individual than the second problem instance if the mean tardiness values are not normalised. After the normalisation, the normalised performance of the ensemble E over the instances in the training set T_{train} is the average normalised mean tardiness values of the schedules as shown in Equation (3), which is also used as the fitness $f(\omega)$ of the individual ω.

$$Perf(E, T_{train}) = \frac{1}{|T_{train}|} \sum_{\gamma \in T_{train}} \text{MT}'(E, \gamma) \tag{3}$$

4 New Measures of Behaviour Analysis of the Evolved Ensembles

Based on the mEGP-JSS algorithm, we carry out several behavioural analyses of the evolved ensembles that are evolved with the different combination schemes. Many GP-HH approaches in the literature to dynamic JSS have both analysed the structures (i.e. genotype [42]) and the behaviours (i.e. phenotype [42]) of evolved rules [6]. However, due to the limited ensemble evolutionary scheduling approaches to dynamic JSS in the literature [10–12], the amount of analysis on the behaviours of evolved ensembles is limited. For example, Hart and Sim [12] have carried out both structural and behavioural analysis of ensembles evolved by their hyper-heuristic approach. They performed extensive analysis on ensembles' performances when they are limited (or not limited) in size and analyse the relation between the structural make-up of the ensembles (using the terminal distributions) and how well it solves specific problem instances. While the structural analysis is quite extensive, the analysis of the behaviours of the ensembles evolved by Hart and Sim is limited, with the scope of the paper being focused on the effectiveness of particular ensembles for specific problem instances.

Analysing and observing the behaviours of the evolved ensembles is important for understanding exactly how ensemble GP-HH can outperform standard GP-HH approaches, allowing us to exploit the advantages of ensemble based approaches while avoiding the disadvantages. In an ensemble, the dispatching rule members that collaborate with each other need to behave in specific ways for the ensemble to be effective. The members of the ensemble need to be able to sufficiently buffer for each other [13]. In other words, for complex decision situations, where the selection of a job that may lead to good solutions is ambiguous, the ensembles need to be able to make diverse sets of good decisions. In classification, complex decisions occur at the class boundaries, where it is ambiguous whether an instance belongs to one class or another [13]. Research in classification has shown that classifiers with single constituent components often cannot cope with complex decisions and is unable to map the class boundaries effectively [13]. If an ensemble in JSS is not diverse enough, it would perform no better than a single dispatching rule, and make potentially bad decisions that single constituent

rules are likely to make [13, 10]. However, it is unlikely that analysing the ensembles through structural analysis will give clear results on the interactions between the different members of the ensembles. The arithmetic trees in the GP evolved dispatching rules often have redundancies in the tree [6], and multiple different tree structures can potentially lead to rules with similar behaviour. Multiple sources in the literature have shown that comparing the behaviours of the trees is more effective than comparing the structures of the tree when used to calculate the fitnesses of the individual in the GP population for surrogate modelling [14] and diversity measures [43]. Therefore, we introduce and justify the following analysis measures to quantify the interactions between the members of an ensemble: the level of "conflicts" between the decisions made by the individuals, the highest level of "contribution" that a member of an ensemble makes towards the overall ensemble decisions (i.e. an ensemble's "bias" towards a specific member), and the spread of the members "importance" on the scale of the ensemble as a whole.

4.1 Measuring Behaviours of Evolved Ensembles

For calculating the different analysis measures for an evolved ensemble, the ensemble is first applied to decision situations directly sampled from the dynamic JSS problem instances will be performed together with evaluation. These decision situations are generated by applying a *sampling dispatching rule* over a problem instances. The selected decision situations encountered by the sampling rule have at least ϵ jobs waiting at the machines. Afterwards, out of the decision situations with at least ϵ jobs, D decisions are selected with equal probabilities to be the sample decision situations that the evolved ensembles are applied to when calculating the analysis measures. ϵ parameter allows us to tune the complexity of the decision situations (due to the greater number of jobs the ensemble needs to account for) and its potential impact on the quality of the schedule. It is possible that a decision situation with a greater number of jobs has a greater impact on the quality of the final schedule than a decision situation with a smaller number of jobs due to the greater number of potentially "bad" decisions that can be made by the ensemble. Directly sampling decision situations is advantageous because generating decision situations manually is difficult, and using a sampling rule allows us to sample decision situations directly from the problem instances that the evolved rules will be evaluated on. In addition, it is likely that the sampling method gives a better representation of the decision situations encountered by the evolved rules than manually generating decision situations. Given a set of decision situations, we design the following three new measures to analyse the behaviours of the ensemble.

4.2 New Measure 1 – Decision Conflict (M1)

M1 calculates the proportion of sampled decision situations where the members of the ensembles have assigned highest priorities equally between two or more jobs in decision situations, i.e., how often the top decisions "conflict" with each other. For a decision situation d out of D sampled decisions, suppose there are L_d jobs waiting at the machine and the member ω of ensemble E assigns the highest priority to a job $j_{d,\omega}$. Then the number V_{d,j^*} of members that assign the highest priority to job j^* at decision d is given

by Equation (4). From this, we get M1(E) in Equation (5), where it is the proportion of decisions out of D decisions where there are at least two jobs are tied for the highest number of top priority assignment.

$$V_{d,j^*} = |\{\omega \in E | j^* = j_{d,\omega}\}| \tag{4}$$

$$\text{M1}(E) = \frac{1}{D}|\{d \in [1,\ldots,D], \exists i,j[i \neq j \wedge \max_k\{V_{d,k}\} = V_{d,i} = V_{d,j}]\}| \tag{5}$$

For the majority voting combination scheme, M1 calculates the proportion of times when a tie in the number of votes occurs between the members of the ensemble. This results in the tiebreaker (the ATC rule) being used to resolve the tie between the top voted jobs for the majority voting scheme. For the other combination schemes, tiebreaker is unlikely to be used, as the jobs are selected based on the total scores that are real-number values. However, M1 measures how often members' "biases" towards specific jobs at the decision situations conflict with the other members of the ensembles. If the members of the ensemble are diverse, then it is likely that the different members of the ensembles are biased towards different jobs for a high number of complex decision situations.

4.3 New Measure 2 – High Contribution Members (M2)

M2 calculates the proportion of sampled decision situations where the decisions of the highest contributing member match-up with the decisions made by the ensemble E, i.e., member whose decisions most align with the decisions made by the ensemble. For a decision situation d out of D decisions, suppose that a member ω of an ensemble E assigns the highest priority to job $j_{d,\omega}$. If job is selected by ensemble E, then the decision between member ω and ensemble E itself can be considered to be "overlapping" for decision d. In other words, given that ensemble E selects job $j_{d,E}$ at decision d, the overlap $q^{\text{M2}}_{\omega,E}$ between member ω and ensemble E over D decisions is given by Equation (6). Afterwards, we get M2(E) in Equation (7), where it is equal to the member with the most overlap with the ensemble over D decisions.

$$q^{\text{M2}}_{\omega,E} = \frac{1}{D}|\{d \in [1,\ldots,D] | j_{d,\omega} = j_{d,E}\}| \tag{6}$$

$$\text{M2}(E) = \max_\omega\{q^{\text{M2}}_{\omega,E}\} \tag{7}$$

This measure is useful for determining the effectiveness of evolved ensembles which have strong biases towards specific members in the ensembles. For example, having ensembles with high M2 values but with poor test performances indicates that ensembles are biased towards single members and hence lose effectiveness. This would support the idea that the ensembles that behave similar to single rules and are not able to handle different types of complex decisions which may arise during a dynamic JSS problem instance by itself. On the other hand, the converse (i.e. high M2 and good test performance) will show that having a single highly biased member in the evolved ensembles may be more effective for scheduling.

Table 2: Normalised rank calculation for a member ω of an ensemble E over D decision situations.

Decision	Jobs	Priorities by Member ω	Rank by ensemble E	$r'_{d,\omega}$
1	1	0.9	1	$\frac{1}{2}$
	2	0.2	2	
2	1	0.1	4	$\frac{1}{5}$
	2	0.1	5	
	3	0.2	3	
	4	0.8	1	
	5	0.1	2	
...	...	...	...	...
D	1	0.2	1	$\frac{2}{3}$
	2	0.7	2	
	3	0.1	3	

4.4 New Measure 3 – Low Job Ranks Members (M3)

M3 calculates the worst average ranks of the ensemble members using a rule ranking system modified from Hildebrandt and Branke [14], i.e., the "spread" of the decisions made by the ensemble members. First, at decision d the jobs are ranked based on the scores assigned to them by an ensemble E. In other words, the top job, i.e., the job selected by the ensemble to be processed, is ranked 1, the second highest scored job ranked 2, and such. Afterwards, for a member ω of ensemble E it is assigned a "decision-rank", denoted as $r_{d,\omega}$ based on the rank of the job that it assigned the highest priority to. Member rank $r_{d,\omega}$ is then normalised based on the number of jobs waiting at the machine (L_d) at decision d, i.e., $r'_{d,\omega} = r_{d,\omega}/L_d$. An example of how a member's normalised ranks for the decision situations are calculated is shown in Table 2.

After the normalised member ranks are calculated, M3 is given by the member of the ensemble which has the worst average normalised member rank values, i.e., the member whose biases towards specific jobs in decision situations are ranked poorly by the ensemble. This is given in the equation as follows.

$$\text{M3}(E) = \max_{\omega}\{\frac{1}{D}\sum_{d=1}^{D} r'_{d,\omega}\} \tag{8}$$

M3 is proposed to measure the diversity in decisions made by the ensemble members. High M3 value for an ensemble implies a high distribution in the ranks, which may show that ensemble members are highly diversified. Combined with the results from the ensembles' performances, this may allow us to observe whether the diversity of the ensembles' decisions through combination schemes either positively or negatively correlate with the performances of the ensembles over the dynamic JSS problem instances.

5 Experiment Design

This section covers the experimental design to evaluate the mEGP-JSS approach with the different combination schemes. First, we introduce the GP-HH benchmark used for comparison and the parameter settings for the benchmark and mEGP-JSS approaches. The mEGP-JSS that uses a particular combination scheme is denoted as follows: mEGP-MV for majority voting, mEGP-LC for linear combination, mEGP-wMV for weighted majority voting and mEGP-wLC for weighted linear combination. Afterwards, the simulation model used to evolve and evaluate the GP rules is described.

5.1 Baseline Approach

We modify a GP-HH approach that has been proposed by Park et al. [11], denoted as GP-JSS, that evolves single dispatching rules as the baseline approach for evaluating the mEGP-JSS approach. Although EGP-JSS has been shown to outperform GP-HH that evolve single rules [10, 11], evolving single dispatching rules using GP-HH is very prominent for DJSS problems [27]. GP evolved single rules have also shown to consistently outperform man-made dispatching rules [6]. For consistency, the GP-JSS uses the same fitness function as the mEGP-JSS approaches (Equation (2)), and the same terminals, arithmetic operators and GP parameters provided below to evolve the single rules.

5.2 GP Terminal and Function Sets

The terminal set used by GP-JSS and mEGP-JSS approaches consist of a mixture of terminal sets used by existing GP-HH approaches in the literature [5, 8, 15]. These terminals range from common attributes (e.g. operation processing time PT) to more complex terminals that utilise multiple common attributes (e.g. remaining processing time of job RT) and the previous states of the shop floor (e.g. average wait time at next machine NQW). These terminals have been shown to evolve high quality dispatching rules in the literature [15]. The function set consists of the arithmetic operators $+$, $-$, $\times$, protected $/$, $\texttt{if}$, $\max$ and $\min$. The protected $/$ works as a division operator if the denominator is non-zero, but returns a value of 1 if the denominator is zero. $\texttt{if}$ is a ternary operator which returns the value of the second argument if the first argument is greater than or equal to zero, and the value of the third argument otherwise. The full list of terminal and function sets is given in Table 3.

5.3 GP and GA Parameter Settings

The parameters used for GP-JSS, mEGP-MV, mEGP-LC, mEGP-wMV and mEGP-wLC approaches are shown in Table 4. The GP parameters for mEGP-MV and mEGP-LC are kept consistent as the parameters used by Park et al. [10] for the mEGP-JSS approach, where the total population size (1024) is the same as the population size of GP-JSS. These parameters are modified from Koza's parameter setting [42], which is a standard set of parameters used for GP. For GP process in mEGP-wMV and mEGP-wLC, the four GP subpopulation sizes at 231 and the GA subpopulation size at 100

Table 3: The terminal and function sets used for EGP-JSS, where job j is one of the jobs waiting at the machine m^* to process operation σ_{ij}.

Terminal	Description
RJ	Operation ready time
RO	Remaining number of operations of job j
RT	Remaining total processing times of job j
PT	Operation processing time of job j
RM	Machine m^* ready time
NJ	Non-delay jobs waiting at machine m^*
DD	Due date of job j
NPT	Next operation processing time
NNQ	Number of idle jobs waiting at the next machine
NQW	Average waiting time of last 5 jobs at the next machine
AQW	Average waiting time of last 5 jobs at all machines
#	Constant real-value in the interval $[0, 1]$
Function	$+, -, \times, /, \texttt{if}, \max, \min$

add to total population size of 1024. Since the number of GP subpopulations is four, the sizes of the GA individuals are also four. The GA parameters for mEGP-wMV and mEGP-wLC are based on common GA parameters used in the literature [**?**] with slight adjustments, i.e., the lower and upper bound are set to 0 and 10 respectively.

5.4 Dynamic JSS Simulation Model

An existing simulation model proposed by Hunt et al. [15] is used in this paper, where discrete-event simulations are used to represent dynamic JSS problem instances. Discrete-event simulations have been used to evaluate various GP-HH approaches for dynamic JSS in the literature [5, 9, 44, 8, 26, 15]. In a discrete-event simulation, the jobs arriving on the shop floor are generated stochastically. This means that a dynamic JSS problem instance is generated from a seed value and a set of *simulation configurations*. In Hunt et al.'s simulation model [15], two training sets $4op$ and $8op$ are used to evolve the rules. At every generation, a different seed is used in conjunction with the training set's simulation configuration, resulting in different dynamic JSS training instances being used to evaluate the GP individuals. This has been shown to improve the quality of the rule output from the GP process compared to fixing the seed [5]. The parameter values for Hunt et al's simulation model [15] are given in Table 5.

6 Results and Analysis

This section covers the evaluation of mEGP-JSS approaches and the combination schemes integrated with the mEGP-JSS processes in this paper. The rules are evolved by the benchmark GP-JSS approach and the different combination schemes of the mEGP-JSS approach. First, we provide the plot of the fitnesses of the individuals in the different GP approaches as they are applied to the two training sets $4op$ and $8op$ to evolve the

Table 4: GP parameters used by the GP-HH approaches for evolving ensembles of dispatching rules

Approach	Parameters	Value
GP-JSS parameters	Population Size	1024
mEGP-MV & mEGP-LC parameters	Number of GP subpopulations	4
	Subpopulation size	256
mEGP-wMV & mEGP-wLC parameters	Number of GP subpopulations	4
	GP subpopulation size	231
	GA subpopulation size	100
	GA crossover rate	90%
	GA mutation rate	10%
	GA reproduction rate	0%
	GA genome value range	$[0, 10]$
	Crossover type	One-point
	Mutation distribution type	Gaussian
	Mutation distribution std.	0.5
Common parameters	Number of generations	51
	GP crossover rate	80%
	GP mutation rate	10%
	GP reproduction rate	10%
	GP initial depth	8
	GP maximum depth	17
	Selection method	Tournament selection
	Selection size	7

dispatching rules. This is done thirty times to evolve a set of dispatching rules that are compared based on their performance over the simulation model described in Section 5.4. Afterwards, the analysis procedure described in Section 4 is carried out on mEGP-JSS to measure the values of M1, M2 and M3 from the evolved rules. The discussion of the results from Sections 6.1 and 6.2 can also be found in Section 4, i.e., after all the results have been provided.

6.1 Training Fitness Convergence Curves

To get the training performances of the different GP approaches, we first get the training performances of the independent runs. For a GP-JSS run, the training performance at a specific generation is the individual with the best fitness in the population. For a mEGP-JSS run, the training performance at a specific generation is the average performance of the individuals with the best fitness in each subpopulation, i.e., the individuals that will be updated as the representative of the next generation. The training performances over the generations are then averaged out over the multiple independent runs, and are shown in Figure 3.

For the rules evolved over $4op$, we can see that the different GP approaches except mEGP-wMV roughly converge to the same output, whereas mEGP-wMV consistently has worse training performance over the generations than the other approaches. On the other hand, for the rules evolved over $8op$, mEGP-LC rules have significantly worse

Table 5: Simulation configurations used for the generating arriving jobs in dynamic JSS problem instances.

Parameter	$4op$	$8op$	Test
Warm-up period	500		
Max jobs completed	2500		
Mean processing time (μ)	25		$25, 50$
Utilisation rate (ρ)	$0.85, 0.95$		$0.90, 0.97$
Tightness factor (h)	$\{3, 5, 7\}$		$\{2, 4, 6\}$
# of operations per job (N_j)	4	8	$4, 6, 8, 10, X \sim \text{Unif}\,(2, 10)$
# of configurations	2	2	20

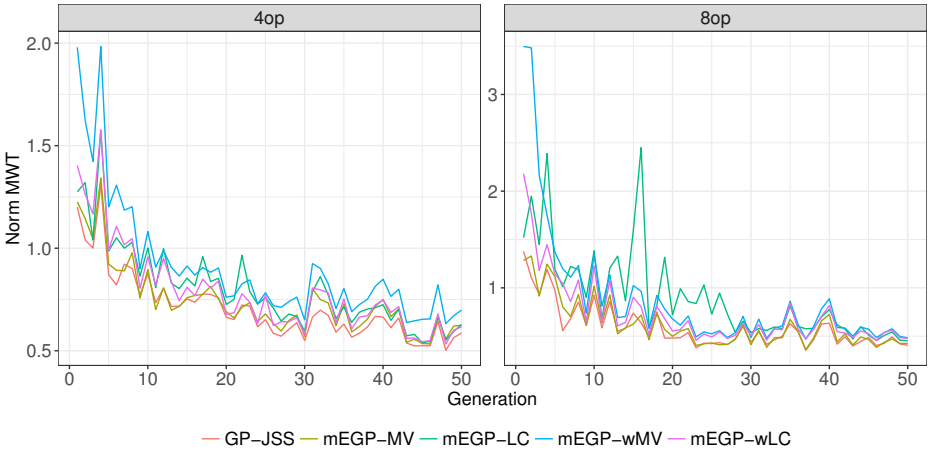

Fig. 3: The average fitnesses of the GP approaches over the training sets.

training performance than the other rules over generations between 10 and 30. However, mEGP-LC rules converge to similar training performances as the other GP rules after generation 30.

6.2 Test Performance

The sets of evolved rules are used to generate schedules for the instances in the training and the test sets. We examined 20 different test scenarios, each with different processing times (μ), utilisation rates (ρ) and number of operations per job (N_j). For each test scenario, thirty replications were randomly generated. The test performance of a rule on a test scenario is defined as the average mean tardiness obtained by applying the rule to the thirty corresponding replications. In addition, for each test scenario, we test whether one set of evolved rules is *significantly* better than another by using two tailed t-test at $\alpha = 0.05$ for the pair of results. The performances of rules over the entire simulation model is shown in Table 6 for the rules evolved from $4op$ and in Table 7 for the rules evolved from $8op$. In the tables, "Training Set" shows the average mean tardiness over the four simulation configurations from both $4op$ and $8op$. The test set is partitioned

Table 6: Comparison of the performances of mEGP-JSS (with the different combination schemes) and GP-JSS over the simulation model. Rules are evolved from $4op$.

Data Subset		mEGP-JSS				GP-JSS
		mEGP-MV	mEGP-LC	mEGP-wMV	mEGP-wLC	
MT ($\times 10^2$)	Training Set	1.78 ± 1.86	1.63 ± 1.70	1.89 ± 1.97	1.71 ± 1.79	1.73 ± 1.86
	$\langle 25, 0.90, 4 \rangle$	1.49 ± 0.06	1.39 ± 0.09	1.60 ± 0.11	1.44 ± 0.10	1.49 ± 0.20
	$\langle 25, 0.90, 6 \rangle$	2.22 ± 0.11	2.06 ± 0.12	2.45 ± 0.23	2.15 ± 0.19	2.38 ± 0.74
	$\langle 25, 0.90, 8 \rangle$	3.30 ± 0.22	2.96 ± 0.38	3.24 ± 0.61	3.08 ± 0.46	2.99 ± 0.56
	$\langle 25, 0.90, 10 \rangle$	6.03 ± 0.37	5.56 ± 0.66	6.17 ± 0.91	5.73 ± 0.65	5.99 ± 2.14
	$\langle 25, 0.90, X \rangle$	1.32 ± 0.07	1.23 ± 0.09	1.46 ± 0.15	1.30 ± 0.11	1.37 ± 0.29
	$\langle 50, 0.90, 4 \rangle$	2.86 ± 0.19	2.64 ± 0.20	3.31 ± 0.39	2.87 ± 0.29	3.25 ± 1.27
	$\langle 50, 0.90, 6 \rangle$	5.11 ± 0.31	4.69 ± 0.62	5.12 ± 0.72	4.84 ± 0.60	4.82 ± 0.84
	$\langle 50, 0.90, 8 \rangle$	7.95 ± 0.50	7.30 ± 0.84	8.27 ± 0.90	7.61 ± 0.65	8.11 ± 3.01
	$\langle 50, 0.90, 10 \rangle$	1.60 ± 0.11	1.49 ± 0.14	1.85 ± 0.24	1.62 ± 0.20	1.75 ± 0.47
	$\langle 50, 0.90, X \rangle$	3.30 ± 0.23	3.05 ± 0.29	3.79 ± 0.43	3.32 ± 0.42	3.84 ± 1.72
	$\langle 25, 0.97, 4 \rangle$	7.96 ± 0.55	7.35 ± 1.10	7.88 ± 1.22	7.52 ± 1.01	7.47 ± 1.50
	$\langle 25, 0.97, 6 \rangle$	13.18 ± 0.80	12.54 ± 1.66	13.90 ± 1.55	12.92 ± 1.20	13.90 ± 4.82
	$\langle 25, 0.97, 8 \rangle$	1.40 ± 0.12	1.33 ± 0.18	1.66 ± 0.26	1.45 ± 0.21	1.58 ± 0.49
	$\langle 25, 0.97, 10 \rangle$	2.27 ± 0.23	2.11 ± 0.28	2.73 ± 0.46	2.38 ± 0.45	2.89 ± 1.82
	$\langle 25, 0.97, X \rangle$	9.10 ± 0.67	8.64 ± 1.41	9.19 ± 1.25	8.76 ± 1.11	8.91 ± 1.77
	$\langle 50, 0.97, 4 \rangle$	13.95 ± 1.00	13.19 ± 1.80	14.96 ± 1.35	13.75 ± 1.36	15.16 ± 5.48
	$\langle 50, 0.97, 6 \rangle$	1.52 ± 0.09	1.44 ± 0.12	1.71 ± 0.17	1.54 ± 0.16	1.64 ± 0.33
	$\langle 50, 0.97, 8 \rangle$	2.54 ± 0.17	2.38 ± 0.22	2.95 ± 0.35	2.59 ± 0.28	2.95 ± 1.16
	$\langle 50, 0.97, 10 \rangle$	5.07 ± 0.37	4.69 ± 0.67	4.97 ± 0.85	4.79 ± 0.68	4.72 ± 0.91
	$\langle 50, 0.97, X \rangle$	9.20 ± 0.65	8.63 ± 1.12	9.40 ± 1.32	8.83 ± 0.99	9.28 ± 3.37

into four subsets of configurations based on the expected processing time (μ) and the utilisation rate (ρ). In the tables, $\langle x, y, z \rangle$ categories denote that the configuration has $\mu = x$, $\rho = y$ and $N_j = z$. The sections highlighted with *red* mean that the mEGP-JSS with the particular combination scheme is significantly worse than the benchmark GP-JSS approach for the simulation configuration, and the sections highlighted with *blue* mean that mEGP-JSS rules performed significantly better than the GP-JSS rules.

From the results of the test set, mEGP-LC rules evolved on both $4op$ and $8op$ outperform the benchmark GP-JSS approach, where they perform significantly better than the GP-JSS rules for many simulation configurations. For the $4op$ rules, mEGP-MV has comparable performance to GP-JSS. On the other hand, for the $8op$ rules, mEGP-MV is generally better than GP-JSS and mEGP-wLC is generally worse than GP-JSS. Finally, mEGP-wMV performs the worst out of the mEGP-JSS approaches, with poor performance in comparison to than the GP-JSS approaches.

In addition to the comparisons to the benchmark GP-JSS rules, results in Tables 6 and 7 also enables us to perform pairwise comparisons between the mEGP-JSS rules. The results of the pairwise comparison are given in Appendix A.1. The pairwise results show that the unweighted mEGP-JSS approaches (mEGP-MV and mEGP-LC) generally perform better than the weighted counterpart (mEGP-wMV and mEGP-wLC).

Table 7: Comparison of the performances of mEGP-JSS (with the different combination schemes) and GP-JSS over the simulation model. Rules are evolved from $8op$.

Data Subset		mEGP-JSS				GP-JSS
		mEGP-MV	mEGP-LC	mEGP-wMV	mEGP-wLC	
MT ($\times 10^2$)	Training Set	1.91 ± 2.01	1.81 ± 1.93	2.22 ± 2.36	1.98 ± 2.09	1.86 ± 1.97
	$\langle 25, 0.90, 4 \rangle$	1.57 ± 0.07	1.50 ± 0.16	1.80 ± 0.27	1.63 ± 0.18	1.56 ± 0.17
	$\langle 25, 0.90, 6 \rangle$	2.27 ± 0.10	2.16 ± 0.18	2.61 ± 0.33	2.33 ± 0.20	2.37 ± 0.22
	$\langle 25, 0.90, 8 \rangle$	3.83 ± 0.36	3.76 ± 0.90	4.77 ± 1.30	4.33 ± 1.04	3.67 ± 1.07
	$\langle 25, 0.90, 10 \rangle$	6.83 ± 0.54	6.67 ± 1.31	8.41 ± 2.01	7.66 ± 1.46	6.82 ± 1.60
	$\langle 25, 0.90, X \rangle$	1.31 ± 0.06	1.25 ± 0.12	1.48 ± 0.17	1.33 ± 0.11	1.29 ± 0.10
	$\langle 50, 0.90, 4 \rangle$	2.81 ± 0.12	2.70 ± 0.25	3.27 ± 0.39	2.89 ± 0.22	2.99 ± 0.33
	$\langle 50, 0.90, 6 \rangle$	5.66 ± 0.40	5.53 ± 1.03	6.70 ± 1.34	6.11 ± 1.13	5.47 ± 1.18
	$\langle 50, 0.90, 8 \rangle$	8.69 ± 0.55	8.29 ± 1.37	10.30 ± 1.92	9.28 ± 1.45	8.60 ± 1.61
	$\langle 50, 0.90, 10 \rangle$	1.50 ± 0.07	1.44 ± 0.13	1.66 ± 0.16	1.51 ± 0.11	1.52 ± 0.15
	$\langle 50, 0.90, X \rangle$	3.15 ± 0.14	3.02 ± 0.30	3.55 ± 0.37	3.21 ± 0.22	3.25 ± 0.36
	$\langle 25, 0.97, 4 \rangle$	8.73 ± 0.62	8.41 ± 1.42	9.92 ± 1.60	9.13 ± 1.51	8.21 ± 1.63
	$\langle 25, 0.97, 6 \rangle$	14.12 ± 0.84	13.63 ± 1.92	16.45 ± 2.21	14.95 ± 1.79	14.25 ± 2.00
	$\langle 25, 0.97, 8 \rangle$	1.27 ± 0.08	1.22 ± 0.13	1.41 ± 0.15	1.30 ± 0.11	1.30 ± 0.14
	$\langle 25, 0.97, 10 \rangle$	1.99 ± 0.15	1.94 ± 0.26	2.29 ± 0.28	2.10 ± 0.22	2.11 ± 0.34
	$\langle 25, 0.97, X \rangle$	9.79 ± 0.62	9.28 ± 1.33	10.70 ± 1.34	9.93 ± 1.43	9.29 ± 1.42
	$\langle 50, 0.97, 4 \rangle$	14.39 ± 0.81	13.64 ± 1.56	16.00 ± 1.52	14.59 ± 1.36	14.42 ± 1.52
	$\langle 50, 0.97, 6 \rangle$	1.48 ± 0.06	1.44 ± 0.12	1.68 ± 0.18	1.54 ± 0.12	1.54 ± 0.14
	$\langle 50, 0.97, 8 \rangle$	2.48 ± 0.11	2.40 ± 0.22	2.85 ± 0.32	2.59 ± 0.23	2.65 ± 0.29
	$\langle 50, 0.97, 10 \rangle$	5.67 ± 0.42	5.53 ± 1.05	6.68 ± 1.45	6.19 ± 1.19	5.42 ± 1.21
	$\langle 50, 0.97, X \rangle$	10.16 ± 0.67	9.76 ± 1.68	12.09 ± 2.26	10.98 ± 1.77	10.11 ± 2.00

6.3 Behavioural Analysis and Further Discussion

For each test simulation configuration in the simulation model, a problem instance is generated and M1, M2 and M3 are calculated for mEGP-JSS approaches using the procedure described in Section 4. The parameters that need to be set are the minimum number of jobs waiting at a decision situation (ϵ) and the number of decision situations (D). After parameter tuning, $\epsilon = 10$ and $D = 50$, i.e., 50 decision situations sampled from a problem instance are used to calculate M1, M2 and M3 and the decision situations have at least 10 jobs. The results of applying the analysis measures are shown in Fig. 4.

From the figure, the M1 values, which is used to measure the level of conflict between the different ensemble members, do not significantly differ from each other. In addition, with the exception of the mEGP-MV rules evolved over $4op$ and the mEGP-wMV rules evolved over $8op$, the GP rules evolved over both training sets have M1 values over 0.39 on average. This means that for all decision situations, the average proportions of conflicting decisions made by members of the ensembles are above 39%. For example, mEGP-MV rules evolved from $4op$ have an average M1 value of 0.39 and mEGP-MV evolved from $8op$ have an average M1 value of 0.43. This means that out of decision situations sampled for the $8op$ rules, the majority voting scheme resulted in a tie for approximately 43% of the decision situations, where this results in the ATC

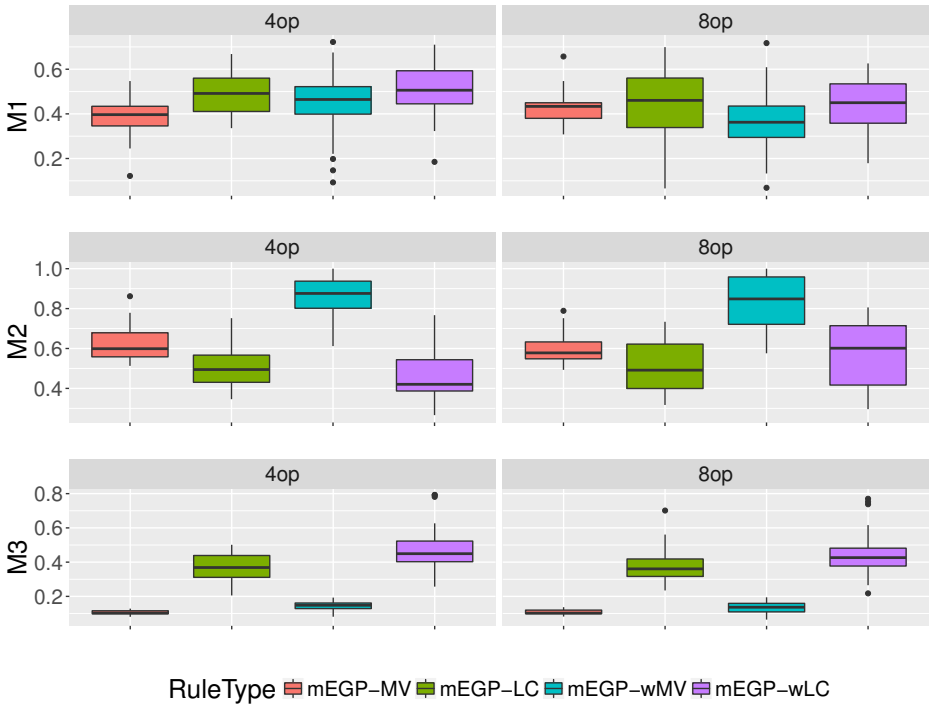

Fig. 4: The analysis measures for the mEGP-JSS approaches plotted against the performance over the problem instances. $4op$ and $8op$ denotes that the GP rules are evolved from the respective training sets.

tiebreaker rule being used. The relatively high M1 value is significant for the mEGP-MV rules compared to the other mEGP-JSS rules. mEGP-LC, mEGP-wMV and mEGP-wLC rules use numeric score values instead of discrete votes to determine which jobs are selected during decision situations. This means that the ties among the top scoring jobs are unlikely to happen in mEGP-LC, mEGP-wMV and mEGP-wLC even if there are conflicts between the decisions made by the ensemble members. Therefore, a decision situation with conflicting decisions made by the ensemble members would affect the decision making process of mEGP-MV rules more than the other mEGP-JSS rules. This may explain why the linear combination schemes generally perform better than the respective majority voting schemes for the dynamic JSS problem instances used in this paper.

On the other hand, mEGP-wMV rules have high M2 compared to other GP rules evolved over the training sets, whereas the other GP rules have similar M2 values. M2 is used to measure the bias of an ensemble towards a specific ensemble member. Therefore, for mEGP-wMV rules there is a single member that participates highly actively in the majority of the decision making process. Further considering the performance results from Section 6.2 (Tables 6 and 7), where the mEGP-wMV rules generally perform

worse than other mEGP-JSS combination schemes, it is clear that having high M2 value negatively affects the quality of the GP evolved ensembles. As expected, we found that the mEGP-wMV rules behave similarly to single rules given their high M2 values. It increases the difficulty for other members of an mEGP-wMV ensemble to cover for the "high-contribution" rules's errors. This is reflected in the performance results, where mEGP-wMV generally perform worse than the other GP evolved rules. In addition, it is likely that the unweighted combination schemes perform better than the weighted combination schemes because our combination of GP and GA is unable to explore the search space effectively and find a good configuration of weights and ensemble members simultaneously from the search space. This may potentially cause an underfitting problem, which is evidenced by the convergence curves for mEGP-wMV and the test performance of mEGP-wMV rules. In the results for the training performances of the GP individuals, mEGP-wMV shows generally worse training performances when trained over $4op$, and has slightly worse training performances near the end of the generation when trained over $8op$. In other words, the current results show that evolving individuals that contribute equally towards job selection may be more beneficial than having the individuals be weighted during the evolutionary process.

Finally, mEGP-LC and mEGP-wLC rules have high M3 values, i.e., likely have high "spread" in the decisions made by the ensemble members. In addition, the rules that use the weighted combination schemes (mEGP-wMV and mEGP-wLC) have higher M3 values than the unweighted counterpart (mEGP-MV and mEGP-LC respectively). The differences in the M3 values for the GP rules that use the linear combination and weighted linear combination schemes against the majority voting and weighted majority voting schemes is likely because of the information loss from converting priorities to scores in the decision making process. At a decision situation for mEGP-MV and mEGP-wMV rules, the number of jobs that are assigned votes is at most the number of members in the ensemble, i.e., at most four with the current GP parameter settings. Therefore, the rest of the waiting jobs that do not have a vote will have zero scores. This means that the worst rank for any given voted job (a job with non-zero score) will still be near one. On the other hand, at a decision situation for mEGP-LC and mEGP-wLC, the waiting jobs are likely to have been assigned non-zero score values by the members of the ensemble, meaning that a job assigned the highest priority by a member can still have a worse rank than a job which has not been assigned the highest priority by any members of the ensemble. This may then lead to more ensembles that can accommodate for a more diverse ensemble members. From the results, mEGP-LC rules perform well against mEGP-MV rules for $4op$ and comparably for $8op$. In addition, mEGP-wLC rules perform well against mEGP-wMV rules.

One additional observation made from the performance evaluation is that the performance of mEGP-MV rules have comparable performance to GP-JSS rules. This is contrary to previous results in the literature [10, 11], which state that EGP-JSS with majority voting generally outperform standard GP-HH approaches. However, this is due to the case that previous works considered mainly to train the GP rules on static JSS problem instances [10], or use the same dynamic JSS training instances in every generation [11]. Generating dynamic JSS training instances that are different for any consecutive generations have been shown to improve the generalisation ability of the rules evolved

by the GP-HH approaches in the literature [5], and may be the reason why mEGP-MV rules have comparable performances to GP-JSS rules. Additional supplementary experiments in Appendix A.2 confirms that using same dynamic JSS instances every generation results in better evolved mEGP-MV rules than GP-JSS rules.

6.4 Summary

From the results, the following findings were made in this paper:

(a) Out of the different combination schemes for mEGP-JSS, mEGP-LC rules generally perform better than mEGP-MV, mEGP-wMV and mEGP-wLC rules. In addition, mEGP-LC rules also outperform the benchmark GP-JSS rules. From the analysis results, the average M1 values for all mEGP-JSS approaches are quite high, which means that members assign high priorities to different jobs. This means that combination schemes that effectively exploit decision diversity among ensemble members by reducing information loss are more desirable. This may be likely the reason why mEGP-LC performs better at handling complex decision situations than other combination schemes.

(b) The rules that use weighted combination schemes (mEGP-wMV and mEGP-wLC) generally perform worse than the unweighted counterpart (mEGP-MV and mEGP-LC respectively). For example, the performance of mEGP-wMV is generally worse than the other mEGP-JSS rules and the benchmark GP-JSS rules. From the M2 analysis, it is likely that decisions made by mEGP-wMV rule ensembles are significantly biased by individual ensemble members. In other words, mEGP-wMV rules that are evolved by the mEGP-JSS approach may be behaving similarly to single dispatching rules, and that ensembles evolved by mEGP-JSS for dynamic JSS that use combination schemes with equal weights is more effective than ensembles evolved by mEGP-JSS for dynamic JSS that use weighted combination schemes.

(c) Finally, the analysis results show that mEGP-LC and mEGP-wLC have higher M3 values than mEGP-MV and mEGP-wMV. Therefore, it is possible that mEGP-LC and mEGP-wLC can produce more diverse ensemble members because less information is lost when the priority values for jobs are converted to scores. It may also be the case that higher M3 can also be partially correlated to better performances, as mEGP-LC rules have better performance than mEGP-MV rules for $4op$ and mEGP-wLC rules have better performance than mEGP-wMV rules.

In summary, this paper provides an investigation into combination schemes, which has not yet been carried out for ensemble GP-HH approaches (EGP-JSS) to dynamic JSS. We found that mEGP-LC can generate the best rules out of the four mEGP-JSS approaches to the dynamic JSS problem. In addition, it provides a comprehensive behavioural analysis which has not been carried out for ensemble GP-HH approaches to dynamic JSS by developing new behavioural measures and analysing the evolved ensembles using the measures.

7 Conclusions and Future Work

In this paper, we investigate the effect of combination schemes in evolving ensembles of dispatching rules for dynamic JSS using GP-HH. Four different combination schemes have been investigated: majority voting, linear combination, weighted majority voting and weighted linear combination [13]. The modified EGP-JSS, denoted as mEGP-JSS, which incorporate the combination schemes are denoted as mEGP-MV, mEGP-LC, mEGP-wMV and mEGP-wLC. The results show that the mEGP-JSS which uses the linear combination scheme (mEGP-LC) generally performs better compared to the other mEGP-JSS approaches and the baseline GP-HH approach. In addition, further analysis shows that the current weighted combination schemes have worse performance than the unweighted combination schemes. This is likely because the weighted ensembles are too biased towards specific ensemble members (particularly for mEGP-wMV rules), which results in the ensembles behaving like single dispatching rules.

There are many further directions that can be investigated from the findings made in this paper. To circumvent the issue where there can be too much bias towards specific members of an ensemble, a two-step approach where good members are first evolved before the weights are assigned to the members could be promising. Another method that may be promising is to assign weights depending on the properties of the shop floor environment at decision situations. Certain rules may be more useful in decision situations with a large number of urgent jobs than a decision situation with less number of jobs, and vice versa. This would likely require a function that takes into account properties of the shop floor as a whole. In addition, linear combination schemes may be effective for other ensemble GP-HH approaches to dynamic JSS problems outside of the EGP-JSS approach. For example, another ensemble GP-HH approach that has evolved effective rules for dynamic JSS is Multilevel Genetic Programming for Job Shop Scheduling (MLGP-JSS) [11], which uses the majority voting to combine the ensemble members' priority assignments. Finally, the analysis measures used in the experiments may potentially be used as a diversity measure in an ensemble GP-HH approach to improve the quality of the evolved ensembles.

A Auxiliary Results for the mEGP-JSS Approaches

This section covers auxiliary results that are used to supplement the results, analysis and discussion provided in Section 6. First, we provide the pairwise comparison of the performances between the mEGP-JSS rules from the results in Section 6.2, and is used as additional material for the analysis and discussion in Section 6.3. Afterwards, to supplement the discussion in Section 6.3, additional experiments are carried out to analyse why mEGP-MV does not evolve rules that are significantly better than the benchmark GP-JSS approach (which differs from the previous results in the literature [10, 11]).

A.1 Pairwise Comparison between the mEGP-JSS Rules

After the performance of the rules have been evaluated in Section 6.2, further pairwise comparisons are made between the rules evolved from the different combination

Table 8: The pairwise comparison between the rules evolved using the different mEGP-JSS combination schemes over $4op$.

	mEGP-MV	mEGP-LC	mEGP-wMV	mEGP-wLC
mEGP-MV	–	$(0, 20)$	$(12, 0)$	$(0, 7)$
mEGP-LC	$(20, 0)$	–	$(17, 0)$	$(12, 0)$
mEGP-wMV	$(0, 12)$	$(0, 17)$	–	$(0, 14)$
mEGP-wLC	$(7, 0)$	$(0, 12)$	$(14, 0)$	–

Table 9: The pairwise comparison between the rules evolved using the different mEGP-JSS combination schemes over $8op$.

	mEGP-MV	mEGP-LC	mEGP-wMV	mEGP-wLC
mEGP-MV	–	$(0, 8)$	$(20, 0)$	$(8, 0)$
mEGP-LC	$(8, 0)$	–	$(20, 0)$	$(20, 0)$
mEGP-wMV	$(0, 20)$	$(0, 20)$	–	$(0, 13)$
mEGP-wLC	$(0, 8)$	$(0, 20)$	$(13, 0)$	–

schemes for mEGP-JSS to determine whether one set of rules is significantly better than another using the results from Tables 6 and Tables 7. This is shown in Table 8 for $4op$ and Table 9 for $8op$. In the table, the number of times one combination scheme is better or worse than another combination scheme over the 20 simulation configurations in the test set is counted. (x, y) denotes that combination scheme 1 is better on x number of instances than combination scheme 2, and is worse on y number of instances. For example, $(0, 8)$ for mEGP-MV and mEGP-LC means that the mEGP-MV rules does not perform significantly better than mEGP-LC rules on any of the simulation configurations, but mEGP-LC rules perform significantly better on eight different simulation configurations. To assist in visualising the results, a mEGP-JSS approach that performs better than another mEGP-JSS approach on Δ different simulation configurations is highlighted in blue. Otherwise, if it is worse on average on at least Δ different simulation configurations, then it is highlighted in red. After some adjustments, $\Delta = 5$ is selected for the comparison to be highlighted either red or blue.

From the pairwise comparisons, mEGP-LC rules evolved over both training sets outperform all other rules. In addition, mEGP-wLC rules outperform mEGP-MV rules evolved over $4op$, but mEGP-MV rules outperform mEGP-wLC rules evolved over $8op$. Finally, mEGP-wMV performs poorly in comparison to the other mEGP-JSS approaches. The discussion of the pairwise results are covered in the analysis section that also provides the behavioural measures (Section 6.3).

A.2 Effects of Seed Rotation on mEGP-MV

To test the effects of using a fixed set of dynamic JSS training instances across generation on the qualities of mEGP-MV and GP-JSS rules, new sets of rules are evolved using the GP-HH approaches. During the evaluation procedure, instead of rotating the seed every generation as described in Section 5.4, a fixed seed is used to generate the training instances. The convergence curves that show the training performances of the GP

individuals over the training set is shown in Figure 5. Afterwards, the evolved mEGP-MV and GP-JSS rules are applied to the problem instances in the test set. The performance of the mEGP-MV and GP-JSS rules evolved without seed rotation and the rules evolved with seed rotation (i.e. the results shown in Tables 6 and 7) over test problem instances is given in Table 10. For the 'no seed rotation' columns, if the mEGP-MV rules performed significantly better than the GP-JSS rules over a simulation configuration, then the corresponding results is highlighted in blue. Otherwise, if the mEGP-MV rules performed significantly worse, then the result is highlighted in red. For the 'with seed rotation' columns, if the seed rotation has improved the rules significantly over a simulation configuration (e.g. mEGP-MV rules evolved with seed rotation performed better than mEGP-MV rules evolved without rotation), then the corresponding results is highlighted in blue. Otherwise, the result is highlighted in red.

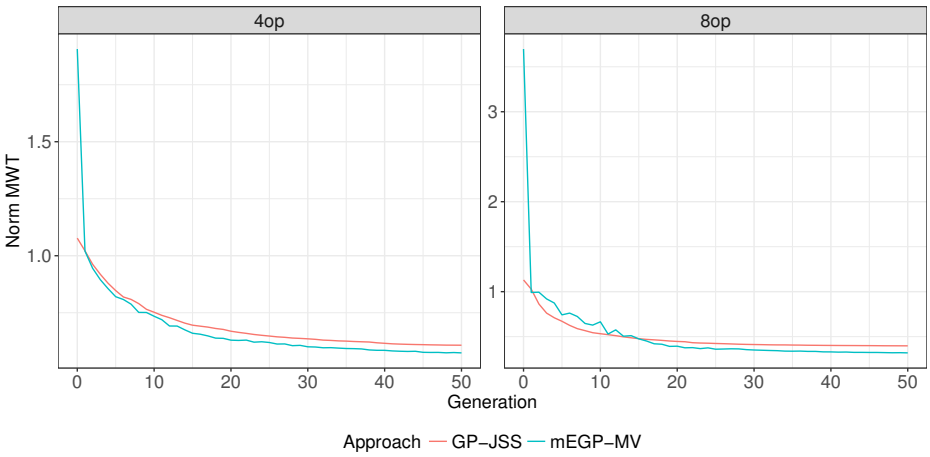

Fig. 5: The convergence curves showing the training performances of the GP-JSS and mEGP-MV individuals.

The convergence curves in the Figure 5 shows that although training performances of the mEGP-MV approach are initially worse than the training performances of the GP-JSS approach, they converge to a better training performance by the end of the GP processes. In addition, the results of the evolved rules over the test set show that the mEGP-MV rules perform significantly better than the GP-JSS rules. Finally, both GP-JSS and mEGP-MV rules trained on the training sets that rotate the seed perform significantly better in comparison to corresponding rules that are trained on fixed DJSS problem instances. In other words, the results shows that the rotating the problem instances can improve the GP-JSS rules significantly. This result is consistent with the findings in the literature [5], which states that rotating the seed used to generate the training instances improves the generalisation abilities of the evolved rules. However, an interesting observation is that the improvement of the qualities of the evolved rules is disproportionate for the GP-JSS and mEGP-MV rules. Particularly, GP-JSS evolved

Table 10: Comparison of the performances of mEGP-MV and GP-JSS over the test set.

Test MWT ($\times 10^2$)	No seed rotation				With seed rotation			
	$4op$		$8op$		$4op$		$8op$	
	mEGP-MV	GP-JSS	mEGP-MV	GP-JSS	mEGP-MV	GP-JSS	mEGP-MV	GP-JSS
$\langle 25, 0.90, 4\rangle$	1.49 ± 0.06	1.55 ± 0.18	1.73 ± 0.16	1.92 ± 0.32	1.49 ± 0.06	1.49 ± 0.20	1.57 ± 0.07	1.56 ± 0.17
$\langle 25, 0.90, 6\rangle$	2.18 ± 0.10	2.30 ± 0.27	2.45 ± 0.21	2.77 ± 0.48	2.22 ± 0.11	2.38 ± 0.74	2.27 ± 0.10	2.37 ± 0.22
$\langle 25, 0.90, 8\rangle$	3.64 ± 0.24	3.85 ± 0.80	4.71 ± 0.73	5.29 ± 0.94	3.30 ± 0.22	2.99 ± 0.56	3.83 ± 0.36	3.67 ± 1.07
$\langle 25, 0.90, 10\rangle$	6.53 ± 0.43	6.92 ± 1.39	8.16 ± 1.17	9.37 ± 1.66	6.03 ± 0.37	5.99 ± 2.14	6.83 ± 0.54	6.82 ± 1.60
$\langle 25, 0.90, X\rangle$	1.30 ± 0.07	1.42 ± 0.20	1.41 ± 0.12	1.58 ± 0.31	1.32 ± 0.07	1.37 ± 0.29	1.31 ± 0.06	1.29 ± 0.10
$\langle 50, 0.90, 4\rangle$	2.74 ± 0.15	3.00 ± 0.43	3.02 ± 0.24	3.56 ± 0.77	2.86 ± 0.19	3.25 ± 1.27	2.81 ± 0.12	2.99 ± 0.33
$\langle 50, 0.90, 6\rangle$	5.46 ± 0.30	5.82 ± 1.01	6.60 ± 0.73	7.40 ± 1.00	5.11 ± 0.31	4.82 ± 0.84	5.66 ± 0.40	5.47 ± 1.18
$\langle 50, 0.90, 8\rangle$	8.28 ± 0.51	8.74 ± 1.41	10.04 ± 1.12	11.44 ± 1.83	7.95 ± 0.50	8.11 ± 3.01	8.69 ± 0.55	8.60 ± 1.61
$\langle 50, 0.90, 10\rangle$	1.53 ± 0.10	1.71 ± 0.27	1.58 ± 0.15	1.79 ± 0.37	1.60 ± 0.11	1.75 ± 0.47	1.50 ± 0.07	1.52 ± 0.15
$\langle 50, 0.90, X\rangle$	3.19 ± 0.19	3.57 ± 0.56	3.32 ± 0.30	3.83 ± 0.85	3.30 ± 0.23	3.84 ± 1.72	3.15 ± 0.14	3.25 ± 0.36
$\langle 25, 0.97, 4\rangle$	8.53 ± 0.51	9.12 ± 1.65	9.92 ± 0.84	10.93 ± 1.24	7.96 ± 0.55	7.47 ± 1.50	8.73 ± 0.62	8.21 ± 1.63
$\langle 25, 0.97, 6\rangle$	13.72 ± 0.76	14.96 ± 2.63	15.83 ± 1.36	17.90 ± 2.22	13.18 ± 0.80	13.90 ± 4.82	14.12 ± 0.84	14.25 ± 2.00
$\langle 25, 0.97, 8\rangle$	1.35 ± 0.12	1.60 ± 0.31	1.31 ± 0.17	1.50 ± 0.35	1.40 ± 0.12	1.58 ± 0.49	1.27 ± 0.08	1.30 ± 0.14
$\langle 25, 0.97, 10\rangle$	2.18 ± 0.24	2.65 ± 0.60	2.07 ± 0.30	2.46 ± 0.81	2.27 ± 0.23	2.89 ± 1.82	1.99 ± 0.15	2.11 ± 0.34
$\langle 25, 0.97, X\rangle$	9.63 ± 0.54	10.42 ± 1.84	10.77 ± 0.70	11.80 ± 1.34	9.10 ± 0.67	8.91 ± 1.77	9.79 ± 0.62	9.29 ± 1.42
$\langle 50, 0.97, 4\rangle$	13.99 ± 0.73	15.37 ± 2.60	15.54 ± 0.92	17.41 ± 2.28	13.95 ± 1.00	15.16 ± 5.48	14.39 ± 0.81	14.42 ± 1.52
$\langle 50, 0.97, 6\rangle$	1.49 ± 0.07	1.65 ± 0.21	1.59 ± 0.14	1.79 ± 0.29	1.52 ± 0.09	1.64 ± 0.33	1.48 ± 0.06	1.54 ± 0.14
$\langle 50, 0.97, 8\rangle$	2.48 ± 0.15	2.80 ± 0.43	2.65 ± 0.24	3.08 ± 0.62	2.54 ± 0.17	2.95 ± 1.16	2.48 ± 0.11	2.65 ± 0.29
$\langle 50, 0.97, 10\rangle$	5.55 ± 0.35	5.90 ± 1.09	6.60 ± 0.74	7.38 ± 0.98	5.07 ± 0.37	4.72 ± 0.91	5.67 ± 0.42	5.42 ± 1.21
$\langle 50, 0.97, X\rangle$	9.86 ± 0.66	10.63 ± 2.06	11.71 ± 1.27	13.33 ± 1.92	9.20 ± 0.65	9.28 ± 3.37	10.16 ± 0.67	10.11 ± 2.00

over $4op$ is able to outperform the mEGP-MV rules for certain simulation configurations. This may be due to the fact that mEGP-MV approach has smaller subpopulation sizes, which may result in less genetic diversity among the individuals in a subpopulation to evolve more generalised rules. In addition, the individuals in a subpopulation for the mEGP-MV approach needs to interact with the representatives of the other subpopulations. This means that individuals that behave differently in a subpopulation may still result in similar behaving ensembles when grouped up with the representatives of the other subpopulations. Consequently, using different training instances every generation may have less of an influence on improving the generalisation abilities of the rules evolved by the mEGP-MV process than the GP-JSS process.

References

1. Potts, C.N., Strusevich, V.A.: Fifty years of scheduling: a survey of milestones. Journal of the Operational Research Society **60**(1) (2009) S41–S68
2. McKay, K.N., Safayeni, F.R., Buzacott, J.A.: Job-shop scheduling theory: What is relevant? Interfaces **18**(4) (1988) 84–90
3. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. 4 edn. SpringerUS (2012)
4. Ouelhadj, D., Petrovic, S.: A survey of dynamic scheduling in manufacturing systems. Journal of Scheduling **12**(4) (2009) 417–431
5. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios: A genetic programming approach. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2010), New York, NY, USA, ACM (2010) 257–264
6. Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M.: Automated design of production scheduling heuristics: A review. IEEE Transactions on Evolutionary Computation **20**(1) (2016) 110–124
7. Suresh, V., Chaudhuri, D.: Dynamic scheduling – a survey of research. International Journal of Production Economics **32**(1) (1993) 53–63
8. 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) (2013) 621–639
9. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2012). (2012) 1–8
10. Park, J., Nguyen, S., Zhang, M., Johnston, M.: Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In: Proceedings of 18th European Conference on Genetic Programming (EuroGP 2015). Volume 9025 of Lecture Notes in Computer Science., Cham, Springer International Publishing (2015) 92–104
11. Park, J., Mei, Y., Nguyen, S., Chen, G., Johnston, M., Zhang, M.: Genetic programming based hyper-heuristics to dynamic job shop scheduling: Cooperative coevolutionary approaches. In: Proceedings of 19th European Conference on Genetic Programming (EuroGP 2016). Volume 9594 of Lecture Notes in Computer Science., Cham, Springer International Publishing (2016) 112–127
12. Hart, E., Sim, K.: A hyper-heuristic ensemble method for static job-shop scheduling. Evolutionary Computation **24**(4) (2016) 609–635
13. Polikar, R.: Ensemble based systems in decision making. IEEE Circuits and Systems Magazine **6**(3) (2006) 21–45

14. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evolutionary Computation **23**(3) (2015) 343–367
15. Hunt, R., Johnston, M., Zhang, M.: Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2014), New York, NY, USA, ACM (2014) 927–934
16. Lawler, E.L.: A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In: Studies in Integer Programming. Volume 1 of Annals of Discrete Mathematics. Elsevier (1977) 331–342
17. Holthaus, O.: Scheduling in job shops with machine breakdowns: an experimental study. Computers & Industrial Engineering **36**(1) (1999) 137–162
18. Nguyen, S., Mei, Y., Ma, H., Chen, A., Zhang, M.: Evolutionary scheduling and combinatorial optimisation: Applications, challenges, and future directions. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2016). (2016) 3053–3060
19. Holthaus, O., Rajendran, C.: Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics **48**(1) (1997) 87–105
20. Holthaus, O., Rajendran, C.: Efficient jobshop dispatching rules: Further developments. Production Planning & Control **11**(2) (2000) 171–178
21. Hershauer, J.C., Ebert, R.J.: Search and simulation selection of a job-shop sequencing rule. Management Science **21**(7) (1975) 833–843
22. Jayamohan, M.S., Rajendran, C.: New dispatching rules for shop scheduling: A step forward. International Journal of Production Research **38**(3) (2000) 563–586
23. Jayamohan, M.S., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. European Journal of Operational Research **157**(2) (2004) 307–321
24. 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) (2012) 4255–4270
25. Burke, E.K., Hyde, M., Kendall, G., Woodward, J.: A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. IEEE Transactions on Evolutionary Computation **14**(6) (2010) 942–958
26. Hunt, R., Johnston, M., Zhang, M.: Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2014). (2014) 618–625
27. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems **3**(1) (2017) 41–66
28. Panait, L., Luke, S.: Cooperative multi-agent learning: The state of the art. Autonomous Agents and Multi-Agent Systems **11**(3) (2005) 387–434
29. Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation **8**(1) (2000) 1–29
30. Geiger, C.D., Uzsoy, R., Aytuğ, H.: Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. Journal of Scheduling **9**(1) (2006) 7–34
31. Jakobović, D., Budin, L.: Dynamic scheduling with genetic programming. In: EuroGP '06: Proceedings of the 9th European Conference on Genetic Programming. Volume 3905 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2006) 73–84
32. Dimopoulos, C., Zalzala, A.M.S.: Investigating the use of genetic programming for a classic one-machine scheduling problem. Advances in Engineering Software **32**(6) (2001) 489–498
33. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation **6**(2) (2002) 182–197
34. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength pareto evolutionary algorithm. In: Proceedings of Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems (EUROGEN 2001). (2001) 1–21

35. Wu, S.X., Banzhaf, W.: Rethinking multilevel selection in genetic programming. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. (2011) 1403–1410
36. Alkoot, F.M., Kittler, J.: Experimental evaluation of expert fusion strategies. Pattern Recognition Letters **20**(11) (1999) 1361–1369
37. Duin, R.P.W., Tax, D.M.J.: Experiments with classifier combining rules. In: Proceedings of International Workshop on Multiple Classifier Systems (MSC 2000). (2000) 16–29
38. Kuncheva, L.I.: Switching between selection and fusion in combining classifiers: An experiment. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics **32**(2) (2002) 146–156
39. Kramer, O.: Genetic Algorithm Essentials. Volume 679. Springer (2017)
40. Vepsalainen, A.P.J., Morton, T.E.: Priority rules for job shops with weighted tardiness costs. Management Science **33**(8) (1987) 1035–1047
41. Mei, Y., Zhang, M., Nguyen, S.: Feature selection in evolving job shop dispatching rules with genetic programming. In: Proceedings of the 2016 Conference on Genetic and Evolutionary Computation. (2016) 365–372
42. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
43. Nguyen, Q.U., Nguyen, X.H., O'Neill, M., Agapitos, A.: An investigation of fitness sharing with semantic and syntactic distance metrics. In: Genetic Programming. Lecture Notes in Computer Science. (2012) 109–120
44. Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B.: Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. International Journal of Production Economics **145**(1) (2013) 67–77