Niching Genetic Programming based Hyper-heuristic Approach to Dynamic Job Shop Scheduling: An Investigation into Distance Metrics

ABSTRACT

Dynamic job shop scheduling (DJSS) problems are combinatorial optimisation problems which have uncertain elements that are present in dynamic environments. This paper focuses on the DJSS problem with unforeseen job arrivals, where a job's attributes are only revealed after it arrives on the shop floor. Genetic Programming based Hyper-heuristic (GP-HH) approaches have been proposed to automatically evolve dispatching rules for DJSS problems. Recent approaches have proposed coevolutionary GP to evolve ensembles of dispatching rules for DJSS problems. These approaches have found that evolved ensembles perform better than single rules evolved by standard GP-HH approaches. However, the diversity of the members of an ensemble is a significant factor in the quality of the ensemble, and the previous coevolutionary GP approaches to DJSS lack effective methods of controlling diversity in the population. Therefore, this paper adapts a fitness sharing algorithm to promote the evolution of a diverse set of high quality GP individuals to be used in the evolved ensemble. In addition, this paper also investigates four different distance measures that are used for the fitness sharing algorithm. The results show that the niched GP approach evolves smaller sized rules than the base GP approach, but have similar performances.

CCS Concepts

•Computing methodologies → Heuristic function construction; *Ensemble methods;* **•Applied computing → Operations research;**

Keywords

Time-tabling and scheduling, Genetic programming, Heuristics, Combinatorial optimization, Robustness of solutions

1. INTRODUCTION

Job shop scheduling (JSS) problems are combinatorial optimisation problems that have been studied extensively over the past 50 years [1]. A job arriving on the shop floor needs to be processed on a specific sequence of machines. During the time a machine is processing a job, the machine cannot start processing any other jobs. To generate a solution to a JSS problem instance, all jobs that arrive on the shop floor must be completed, i.e., their operations must all be processed. The sequence and the times in which the jobs are processed on the machines is jointly called a *schedule* [2]. The quality of a schedule for a problem instance is given by an *objective function* [2].

For this paper, we focus on dynamic JSS (DJSS) problems with unforeseen job arrivals. This means that the arrival time and the properties of a job are unknown until the job enters the shop floor. Mathematical optimisation techniques are not suitable for DJSS problems, and therefore *dispatching rule* heuristics have been proposed in the literature for DJSS problems with unforeseen job arrivals [3, 4]. A dispatching rule is a heuristic which iteratively select a job to be processed on a machine when the machine becomes available. This decision process is called a *dispatching decision* [2]. Dispatching rules are effective for DJSS problems due to their short reaction times and are able to cope with the dynamic environments [5]. However, aside from the problem that it is designed for, a dispatching rule is not guaranteed to be effective for different JSS problem instances. Therefore, Genetic Programming based Hyper-heuristic (GP-HH) approaches have been proposed in the literature to automatically evolve dispatching rules for DJSS problems [6]. A GP-HH bypasses the need of human experts and extensive trial-and-error testings required to manually develop an effective dispatching rule. Recent GP-HH approaches have evolved dispatching rules that have outperformed man-made dispatching rules for DJSS problems [6].

However, there are limitations to existing GP-HH approaches. A large number of the approaches evolve single dispatching rules to handle JSS problems. Evolving a single dispatching rule may result in the rule bloating and being too specific (i.e. overfitting) to the JSS training instances. Therefore, there are GP-HH approaches [7, 8] that evolve *ensembles* [9] of dispatching rules using cooperative coevolutionary GP [10, 11]. The research has shown that evolved ensembles of dispatching rules generally outperform single dispatching rules evolved by standard GP-HHs for DJSS problems with unforeseen job arrivals [7, 8]. However, coevolutionary GP approaches to evolving ensembles of dispatching rules are fairly recent, and more research is necessary in order to evolve effective ensembles with significant practical value.

One particular concept that the previous coevolutionary GP approaches have not fully explored is diversity. Diversity is a key concept for generating effective ensembles [9]. Rules that make up an ensemble needs to be able to cover for each other's errors, and an ensemble that lacks diversity will consist of rules with potentially bad coverage. In other words, the rules that make up an effective ensemble needs to both have high performance and be diverse. In

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

GP, *niching* algorithms have been proposed in the literature for GP to encourage *exploration* of uncrowded regions of the search space to potentially find high quality solutions. Niching algorithms have the potential to address the diversity issue with the current coevolutionary GP approaches for DJSS. Therefore, we use a popular type of niching method called fitness sharing [12], where fitnesses for GP individuals are adjusted based on their proximity to other individuals in the GP population. An effective distance measure that compares the distances between GP individuals is required for any successful fitness sharing algorithms. The distance measure can either be *genotypic* [13], where the structures of the individuals are compared against each other, or *phenotypic* [13], where the behaviours of the individuals are compared against each other. Previous studies have investigated various distance measures for GP individuals, and have showed that phenotypic distance measures are generally better than genotypic distance measures [13].

1.1 Goal

The goal of this paper is to develop an effective fitness sharing algorithm for a GP-HH to DJSS problem with unforeseen job arrivals and total weighted tardiness (TWT) minimisation objective. The DJSS problems with the TWT minimisation objective and other due date related objectives are popular in the literature for evaluating GP-HHs [6]. In particular, this paper will focus on coevolutionary GP that evolves ensembles of dispatching rules [7, 8]. No research has seriously considered fitness sharing algorithms for a coevolutionary GP approach to DJSS, and it is possible that a niched coevolutionary GP approach that uses a fitness sharing algorithm may generate more effective ensembles than the base coevolutionary GP approach. To do this, we investigate multiple phenotypic distance measures. For a DJSS training instance, the distance measures will utilise the behaviours of the individuals in the ensemble during the dispatching decisions. For evaluation, the niched coevolutionary GP approach with the different distance measures will be compared against the base coevolutionary GP approaches. Both novel distance measures and distance measures adapted from the literature are considered. Overall, this paper has the following three objectives:

(a) Layout a framework to incorporate a fitness sharing algorithm to a coevolutionary GP process.

(b) Incorporate four different distance measures for determining the crowding density of an individual.

(c) Compare the effectiveness of the distance measures against each other, and the niched GP-HHs against the base GP-HHs.

1.2 Organisation

The organisation of the paper is as follows. Section 2 gives the problem definition for DJSS and related GP-HH approaches to DJSS problems. Section 3 defines a framework for incorporating a fitness sharing algorithm into a coevolutionary GP approach that evolves ensembles of dispatching rules. Section 4 covers our main contribution, which are the different methods of calculating distances between individuals in an ensemble. Section 5 describes the dataset, the coevolutionary GP and the parameters used, and Section 6 evaluates the niched coevolutionary GP approach against the base coevolutionary GP approach. Finally, Section 7 gives concluding remarks and discusses future works.

2. BACKGROUND

This section covers the notations used for DJSS problems, definition of *active* and *non-delay* scheduling algorithms, and dispatching rule approaches proposed in the literature. The GP-HH approaches which evolve dispatching rules for DJSS problems is then covered, along with niching algorithms and cooperative coevolution.

2.1 Problem Definitions for DJSS

We use the following notations to describe various properties in a JSS problem instance. The number of machines on the shop floor is M, and N jobs arrive on the shop floor. A job has a sequence of N_j operations $\sigma_{1j}, \ldots, \sigma_{(N_j)j}$. An operation σ_{1j} must be processed first, and an operation σ_{ij} can only be processed after operation $\sigma_{(i-1)j}$ is processed. In addition, operation σ_{ij} needs to be processed on a specific machine $m(\sigma_{ij})$ for duration $p(\sigma_{ij})$ (abbreviated to p_{ij}), which is called *processing time* [2]. The time when operation σ_{ij} is ready to be processed is called *operation ready time*, and is denoted as $r(\sigma_{ij})$. This means that the time when the job arrives on the shop floor is $r(\sigma_{1j})$ (abbreviated to r_j), and is called the *release date* of job j. For a DJSS problem with the TWT minimisation objective, the two additional attributes are the due date d_j of a job j, and the weight w_j of job j. If a job j is completed after its due date d_j, then job j is tardy. The tardiness T_j of job j is the positive difference between the completion time C_j and the due date, i.e., $T_j = \max\{C_j - d_j, 0\}$. From this, the TWT minimisation objective is given below.

$$\min \qquad \sum_{j=1}^{N} w_j \times T_j \tag{1}$$

A dispatching rule is applied to a JSS problem instance as follows. During a dispatching decision when a machine m^* becomes available, the dispatching rule first determines which jobs should be considered for selection. One possible method is to only consider jobs that are already waiting at machine m^*. This minimises the idle time of machine m^*, and the dispatching rule is said to generate a non-delay schedule [2]. Another possible method is to consider jobs which arrive later down the line. However, the jobs considered have to arrive before the earliest time a job can be processed and completed by machine m^*, i.e., the earliest projected completion time. This is because a job could have been processed on the machine during the time the machine is idle waiting for the job to arrive. The dispatching rule is then said to generate an active schedule [2]. After the jobs to select from have been considered, the dispatching rule uses the attributes of the jobs and the shop floor to determine which job should be selected by machine m^* to be processed [2]. A selection procedure for the dispatching rule can range in complexity, from using basic job attributes, such as processing times and due dates of jobs to determine which job to process [2], to using meta-attributes that aggregate multiple job and machine attributes together [2, 3, 4]. An example of a complex man-made dispatching rule is Holthaus's priority-based dispatching rule [4]. In a priority-based dispatching rule, a priority function is used to aggregate the attributes together into a single priority function for a job waiting at the machine during a dispatching rule. The job with the highest priority is then selected to be processed.

2.2 Related Works

To evolve high quality dispatching rules using Genetic Programming based Hyper-heuristic (GP-HH), several different design decisions need considered. One possible consideration is the choice of terminal and function sets. Most GP-HH approaches to DJSS problems evolve single priority-based dispatching rules [15, 16, 5, 17], where the individuals represent priority function trees that can calculate the priorities of the jobs waiting at a machine during a dispatching decision. In a GP that evolves priority-based dispatching rules, the terminals consist of job, machine and shop floor attributes and the function set consist of arithmetic operators. Earlier GP-HH approaches have used basic attributes such as processing times and due date (for due date related DJSS problems) [15]. Later approaches have proposed GP-HHs which incorporates meta-attributes into the terminal set, such as slack [16] (the positive difference between the due date of a job and the current time), and total remaining processing time [16]. In particular, Hunt et al. [17] have proposed "less-myopic" terminals for a DJSS problem, and found that "less-myopic" evolved rules performed better than benchmark evolved dispatching rules.

In addition to the terminal and non-terminal considerations, overall the GP process can be modified [7, 8]. Park et al. [7] proposed the Ensemble of Genetic Programming for Job Shop Scheduling (EGP-JSS) approach, which incorporates Potter and De Jong's cooperative coevolution [10] to the GP process to evolve ensembles of dispatching rules. Afterwards, they incorporated a diversity measure to further improve the quality of the ensembles, but no noticeable improvements were observed. Park et al. [8] then proposed Multilevel Genetic Programming for Job Shop Scheduling (MLGP-JSS), an adaptation of Wu and Banzhaf's cooperative coevolutionary GP [11], for a DJSS problem to evolve ensembles of dispatching rules. They showed that MLGP-JSS evolves better rules than a benchmark GP-HH approach which evolves single dispatching rules with insignificant differences in the evolution times. They also compared MLGP-JSS approach against the EGP-JSS approach, and showed that EGP-JSS approach evolves better rules, but takes longer to do so. However, Park et al. do not use niching for MLGP-JSS.

3. FRAMEWORK FOR NICHED COEVOLUTIONARY GP TO DJSS

This section covers the framework required to incorporate a fitness sharing algorithm to a coevolutionary GP. First, the GP representation that is used for the coevolutionary GP is covered. Afterwards, an overview of the coevolutionary GP process is described, along with how fitness sharing is incorporated into the GP process. Finally, a sharing function that is commonly used for fitness sharing is covered.

3.1 GP Representation, Terminal Set and Function Set

In this paper, following a common approach of evolving single priority-based dispatching rules [6], we use a tree-based GP where the individuals are priority function trees. The representation, terminal and function sets in our study is the same as the ones used by Park et al. for the MLGP-JSS approach [8]. The terminal set consists of job, machine and shop floor attributes as shown in Table 1, which includes basic job and machine attributes and "less-myopic" attributes proposed by Hunt et al. [17] that have evolved high quality rules in their GP-HH approach. In addition, the function set contains the arithmetic operators $+$, $-$, $\times$, protected $/$ [6], $\max$ and $\min$.

3.2 Coevolutionary GP Process with Fitness Sharing

There are a various niching algorithms which have been proposed in the literature to promote the diversities of individuals in a GP population. However, the focus of this paper is to investigate effective phenotypic distance measures used for niching. Therefore, in our study we focus on a specific niching algorithm called fitness sharing which have been used extensively in the literature [18].

Table 1: The terminal set used for the GP representation, where job j is one of the job waiting at the machine m^* to process operation σ_{ij}.

Terminal		Description
Basic	RJ	Operation ready time
	RO	Remaining number of operations of job j
	RT	Remaining total processing times of job j
	PR	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
	W	Tardiness penalty of job j
	#	Constant
Less-myopic	NPR	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
Function		$+, -, \times, /, \texttt{if}, \max, \min$

Figure 1 shows a general overview of a coevolutionary GP process, and how the fitness sharing algorithm is incorporated. First, an initial population of GP individuals is generated randomly. Afterwards, a grouping procedure is carried out. The grouping procedure groups the individuals into ensembles temporary, i.e., the ensembles are discarded after the evaluation procedure [10], or peramently, e.g., the ensembles belong part of the GP population [11]. After the ensembles have been generated, non-delay dispatching rules are formed from the ensembles and potentially the GP individuals, where the individuals represent single priority-based dispatching rules. The rules are then applied to the DJSS training instances to generate schedules. The performances of the ensembles either directly [7] or indirectly [8] affect the chances of the individuals surviving to the next generation. In one approach [7], the fitness of an individual directly calculated from the performance of the ensemble over the training instances that the individual belong to. In another approach [8], the "fitness" of the ensembles are not used to calculate the fitnesses of the individuals, but the probability of the individual being selected as a parent depends on the fitnesses of the ensembles that it belongs to. However, when fitness sharing is incorporated into the coevolutionary GP process, the fitnesses of the individuals in an ensemble are adjusted based on the decisions they made as the ensemble is applied to the training instances. After the fitnesses of the individuals and the ensembles are calculated, the selection and the breeding procedures occur to produce the next generation of individuals, and the process is repeated until the termination criteria is reached.

3.3 Applying an Ensemble to a DJSS Problem Instance

To generate solutions to DJSS problem instances, the ensembles in the GP process are applied to the problem instances as a non-delay dispatching rule [7, 8]. When a machine m^* becomes available, an individual that is part of the ensemble first calculates the priorities of the jobs waiting at the m^*. The job with the highest assigned priority is given a "vote" from the individual. This process is repeated for all individuals in the ensemble. Afterwards, a

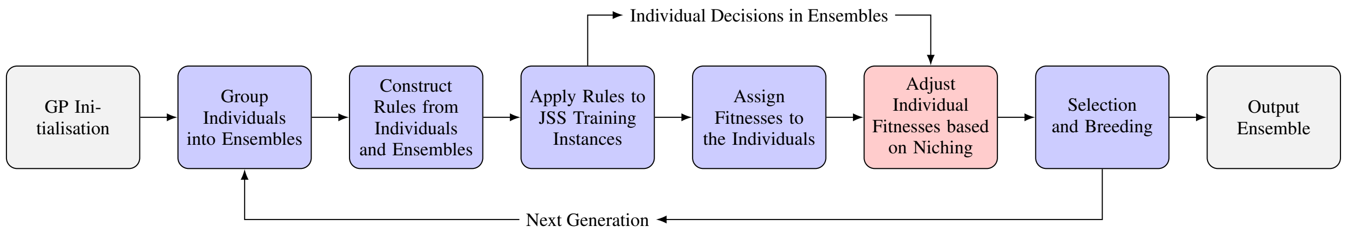

Individual Decisions in Ensembles
GP Initialisation
Group Individuals into Ensembles
Construct Rules from Individuals and Ensembles
Apply Rules to JSS Training Instances
Assign Fitnesses to the Individuals
Adjust Individual Fitnesses based on Niching
Selection and Breeding
Output Ensemble
Next Generation

Figure 1: An overview of a coevolutionary GP that uses fitness sharing.

majority voting procedure [9] occurs, and the job with the highest number of votes is selected by the ensemble to be processed by machine m^*.

3.4 Sharing Function and Fitness Adjustment

A key component of a fitness sharing algorithm is the sharing function [18]. A sharing function can be described as a method of "determining the degradation of an individual's payoff due to a neighbour at some distance as measured in some similiarity space" [18]. The sharing function is shown in Equation (2).

$$sh(d(\omega,\psi)) = \begin{cases} 1 - \left(\frac{d(\omega,\psi)}{\Delta(t)}\right)^{\alpha} & d(\omega,\psi) \leq \Delta(t) \\ 0 & \text{else} \end{cases} \tag{2}$$

In Equation (2), $d(\omega,\psi)$ is the distance between the individuals ω and ψ. $\Delta(t)$ is the "maximum distance that defines a niche" [12], and is a function of the number of generations t. The definition of $\Delta(t)$ depends on the problem, and is covered in Section 5.3. Finally, α is a control variable that determines how much the sharing function scales with the distances between individuals ω and ψ.

The fitness sharing algorithm uses a density function $\xi(\omega)$ for an individual ω around a grouping, e.g., such as the individuals in the population. For this paper, the individuals are members of an ensemble in the coevolutionary GP. Therefore, the diversity is maintained between the different members of the ensemble. The density function uses the values from the sharing function measures crowding of individuals from E around an individual $\omega \in E$. The density function is shown by Equation (3).

$$\xi(\omega) = \frac{1}{|E|} \sum_{\psi \in E, \psi \neq \omega} sh(d(\omega,\psi)) \tag{3}$$

After the density is calculated, the fitness f_ω of individual ω is adjusted so that individual ω is penalised if it is too close to other individuals in the ensemble. As the DJSS problem is a minimisation objective, the lower the f_ω value of individual ω, the better the individual. Therefore, the density value adjusts fitness f_ω so that a high density value results in a high adjustment to the fitness value and a low density value results in an adjusted fitness similar to the original fitness f_ω. The adjustment made to the fitness of an individual ω is shown in Equation (4).

$$f_\omega = f_\omega \times (1 + \xi(\omega)) \tag{4}$$

4. DISTANCE MEASURES FOR THE FITNESS SHARING ALGORITHM

This section describes the distance measures that are used in conjunction with the fitness sharing algorithm. Four distance measures for calculating distances between individuals for specific DJSS problem instances are investigated. The overall distance $d(\omega,\psi)$ is the average distance over all the problem instances in the training set, as shown in Equation (5). In the equation, $d(\omega,\psi,\gamma)$ denotes the distance between individuals ω and ψ over a training instance γ. $d(\omega,\psi,\gamma)$ is calculated from the decisions made by the individuals in the ensemble during dispatching decisions as it generates a schedule for instance γ.

$$d(\omega,\psi) = \frac{1}{\Gamma} \sum_{\gamma=1}^{\Gamma} d(\omega,\psi,\gamma) \tag{5}$$

4.1 Normalised Priorities Distance

The first distance measure compares the *priorities* assigned to the jobs by the individuals. For a dispatching decision i, the priorities assigned to the jobs by an individual in the ensemble is first normalised using a softmax function. Let $\delta_\omega(j_1), \ldots, \delta_\omega(j_K)$ be the priorities assigned by individual ω to jobs $j_1, \ldots, j_K$ waiting at the machine at a dispatching decision. Using the maximum and the minimum priorities assigned to the jobs (denoted as $\delta_\omega(j_{\max})$ and $\delta_\omega(j_{\min})$ respectively), we can first normalise the priority $\delta_\omega(j)$ for a job j to become

$$\delta'_\omega(j) = \frac{\delta_\omega(j) - \delta_\omega(j_{\min})}{\delta_\omega(j_{\max}) - \delta_\omega(j_{\min})} \tag{6}$$

Using the normalised priorities, the distance between individuals ω and ψ for a training instance γ is the root mean squared distance of normalised priorities assigned to the jobs selected to be processed at each dispatching decision. Therefore, if a total of L dispatching decisions are made for instance γ by the ensemble, then the distance $d_1(\omega,\psi,\gamma)$ between individuals ω and ψ is given in Equation (7).

$$d_1(\omega,\psi,\gamma) = \sqrt{\frac{1}{L} \sum_{k=1}^{L} (\delta'_\omega(j_k) - \delta'_\psi(j_k))^2} \tag{7}$$

4.2 Decision Vector Distance

Adapted from a distance measure proposed by Hildebrandt et al. [13], the second distance measure compares the *ranks* assigned to the jobs by the individuals. The ranks of jobs for an ensemble at a dispatching decision i is the ordering of the jobs from the most voted job to the least voted job. This means that the most voted job has rank 1, the second most voted job has rank 2, and so on. For an individual ω, the rank of the voted job is used to calculate individual ω's distance from the ensemble E at a particular dispatching decision i. An example of this is shown in Table 2. There are five jobs $(j_1, j_2, j_3, j_4, j_5)$ waiting at the machine. ω votes on job j_3, which is ranked second by the ensemble. Therefore, d_i is 2 for individual ω at decision i. This process is carried out for all L dispatching decisions made by the ensemble while generating the schedule, resulting in a "decision vector" [13] d^ω of length L. The values in the decision vectors are normalised using the number of

Table 2: Example of specific d_i calculation for an individual ω in the ensemble.

Dispatching Decision i	1
Number of Jobs	5
Jobs Waiting at Machine	$\langle j_1, j_2, j_3, j_4, j_5 \rangle$
Job Rankings by Ensemble	$\langle j_5, j_3, j_2, j_4, j_1 \rangle$
Job Selected by Rule	j_3
Distance d_i	2

Table 3: Example of overall distance d^ω calculation for dispatching decision i for an individual ω in the ensemble.

Dispatching Decision i	1	2	...	N
Number of Jobs	5	50	...	12
Job Selected by Rule	j_3	j_{12}	...	j_4
Ensemble E's Ranking of Jobs	2	3	...	4
Decision Vector d^ω	(2,	3,	...,	4)
Normalised Decision Vector $\overline{d^\omega}$	$(\frac{2}{5},$	$\frac{3}{50},$	...,	$\frac{4}{12})$

jobs waiting at the machine to get the normalised decision vector $\overline{d^\omega}$.

After the normalised decision vector $\overline{d^\omega}$ of individual ω and the normalised decision vector $\overline{d^\psi}$ of individual ψ are calculated, the distance $d(\omega, \psi, \gamma)$ between the individuals ω and ψ for a training instance γ is given by Equation (8).

$$d_2(\omega, \psi, \gamma) = \sqrt{\frac{1}{L}\sum_{k=1}^{L}(\overline{d^\omega}_{j_k} - \overline{d^\psi}_{j_k})^2} \tag{8}$$

4.3 Decision Overlap between Individuals Distance

The third distance measures *overlap* of the decisions between any two individuals in the ensemble. For a particular dispatching decision i, two individuals ω and ψ overlap each other if they have voted on the same job waiting at the machine. Therefore, the number of non-overlapping decisions is required to get the distance $d(\omega, \psi)$ between the individuals ω and ψ. This is shown in equation (9), where $Avg(\omega, \psi, L)$ is the average overlap between the two individuals ω and ψ over all L dispatching decisions.

$$d_3(\omega, \psi, \gamma) = 1 - Avg(\omega, \psi, L) \tag{9}$$

4.4 Decision Overlap against Ensemble Distance

The fourth distance measure is a modification of the third distance measure. Instead of comparing the overlap between two individuals in the ensemble, a decision of an individual ω is compared against the decision of the entire ensemble E. A possible limitation of comparing overlaps between individuals is that a particular individual ω may occupy a region of the heuristic space which is already covered by multiple individuals. By comparing the overlap between individual ω and the ensemble E, it is more likely that individual ω can form a niche away from other individuals. For the distance measure, ω is at a same distance from any individual ψ that is part of the ensemble E. This is shown in Equation (10), where $Avg(\omega, E, L)$ is the average overlap between individual ω and ensemble E over all L dispatching decisions.

$$d_4(\omega, \psi, \gamma) = 1 - Avg(\omega, E, L) \quad \forall \psi \in E, \psi \neq \omega \tag{10}$$

5. EXPERIMENTAL DESIGN

Table 4: Configurations used for the generating jobs in DJSS problem instances.

Parameter	$4op$	$8op$	Test
μ	25	25	$25, 50$
ρ	$0.85, 0.95$	$0.85, 0.95$	$0.90, 0.97$
h	$\{3, 5, 7\}$	$\{3, 5, 7\}$	$\{2, 4, 6\}$
# of operations per job	4	8	$4, 6, 8, 10, X \sim$ Unif $(2, 10)$
# of instances	2	2	20

This section covers the experimental setup. We describe the dataset used, the coevolutionary GP process for DJSS used for the niched GP approach, and then the parameter setting used to evolve the ensembles of dispatching rules.

5.1 Dataset

To train new heuristics from a hyper-heuristic approach for DJSS, long-running discrete-event simulations are commonly used as problem instances [16, 5, 17, 6]. Existing DJSS datasets are characterised by small number of problem instances with large number of job arrivals and a "warm up" period where completed jobs do not contribute towards the objective function evaluation. For this paper, we focus on a recent DJSS dataset proposed by Hunt et al. [17] which have been used to compare the performances of different GP-HH approaches [17, 8]. The dataset has 2 training sets: $4op$, where each job that arrive on the shop floor has 4 operations, and $8op$, where each job has 8 operations. Both training sets have 2 problem instances. Each problem instance in the dataset have 10 machines, jobs are randomly generated, and the problem instance requires 2500 jobs to be completed before jobs stop arriving. All jobs need to be completed before the problem instance is solved. In addition, the first 500 jobs are considered "warm-up".

The properties of an arriving job is also generated randomly. A job's operations have discrete processing times sampled uniformly at random from the interval $[1, 2\mu - 1]$. The job's arrival time follows a Poisson distribution with mean λ. The rate of arrival, λ, is configured so that it meets a desired utilisation rate ρ. Given that p_M is the expected proportion of the total number of machines that an arriving job's operations will be processed on, $\lambda = \frac{\rho \times p_M}{(1/\mu)}$. The due date d_j of a job j is given by $d_j = r_j + h\sum_{i=1}^{N_j} p(\sigma_{ji})$, where h is the tightness factor that is randomly selected from a list with equal probability. For the training sets $4op$ and $8op$, the h list consists of the values $3, 5, 7$. For the test set, h list consists of the values $2, 4, 6$, meaning that the due dates of jobs in test instances are more likely to be tighter than jobs in training instances. The weight w_j of a job j is selected from the values $1, 2, 4$ with the probabilities $0.2, 0.6, 0.2$ respectively. In addition, no two separate operations for a job require the same machine to be processed.

Each problem instance within each training sets $4op$ and $8op$ has different utilisation rate. The first instance in both training sets, denoted as γ_1, has $\rho = 0.85$ and the second instance, denoted as γ_2, has $\rho = 0.95$. All instances have $\mu = 25$, which means that the processing times of a job's operations follows a discrete uniform distribution with interval $[1, 49]$. On the other hand, the 20 problem instances in the test set have different configurations for μ, ρ and the number of operations per job. For a test instance, the possible values for μ is either 25 or 50, ρ is either 0.90 or 0.97. The number of operations per job can be either fixed 4, 6, 8, 10 operations, or be a random number between 2 and 10. The machines which an operation is carried out is random. The overall parameter settings for the dataset is shown in Table 4.

5.2 The Coevolutionary GP used for Evolving Ensembles of Dispatching Rules

The approach of using coevolutionary GP to evolve ensembles of dispatching rules for a DJSS problem is still fairly new in the literature. For this study, we use a recent coevolutionary GP approach to a DJSS problem called MLGP-JSS [8]. In MLGP-JSS, an individual ω in MLGP-JSS can potentially belong to multiple groups or ensembles. To adjust the fitness of an individual, we use the lowest density value calculated for the individual to adjust its fitness. This is to reduce the penalty to individuals that may occupy good niches in certain ensembles. However, the distance measures are designed to compare the distances between individuals in an ensemble. Therefore, we focus only on adjusting the fitnesses of the individuals that belong to ensembles, and do not adjust the fitnesses of the individuals that are not part of an ensemble.

5.3 Parameter Settings

The GP parameters used for MLGP-JSS is kept consistent as the parameters used by Park et al. [8]. This means that population size is 1024, and the number of generations is 51. The crossover, mutation and reproduction rates are 80%, 10%, and 10% respectively. The maximum depth of an individual during initialisation is 8. Tournament selection of size 7 is used during the selection process. There are additional parameters required for MLGP-JSS: the number of groups breed is 200, the number of groups retained is 100, the group cooperation, crossover and mutation rates are 31.25%, 50%, and 18.75% respectively.

The additional parameters for the niching algorithm are the maximum niching distance ($\Delta(t)$) and scaling factor (α). For this paper, we focus on tuning the maximum niching distance. The maximum niching distance is inversely proportional to the number of generations t, i.e., $\Delta(t) = 0.5/t$. This means that the maximum niching distance decreases as the number of generations increases. Therefore, $sh(d(\omega, \psi))$ is more likely to be zero, resulting in density $\xi(\omega)$ for an individual ω in an ensemble decreasing. This results in the smaller adjustment to the fitness of individual ω. In other words, we expect maximum niching distance to encourage exploration early on in the GP process and exploitation later on in the GP process (by reducing the influence of the niching algorithm) after individual niches have been established. On the other hand, the scaling factor is set to standard value of 1 ($\alpha = 1$) [18]. The overall parameter settings, including the GP parameters, are shown in Table 5.

Table 5: GP and niching algorithm parameters used by EGP-JSS and MLGP-JSS for evolving ensembles.

Approaches		MLGP-JSS
GP Parameters	Population size	1024
	Number of groups breed	200
	Number of groups retained	100
	Group cooperation rate	31.25%
	Group crossover rate	50%
	Group mutation rate	18.75%
	Generations	51
	Crossover rate	80%
	Mutation rate	10%
	Reproduction rate	10%
	Max initial depth	2
	Max depth	8
	Selection method	Tournament
	Selection size	7
Niching Parameters	Maximum niching distance $\Delta(t)$	$0.5/t$
	Scaling factor α	1

Table 6: The TWTs ($\times 10^5$) for the ensembles evolved by the coevolutionary GP approaches over the training and the test JSS problem instances. The ensembles are evolved from the training set $4op$.

Approach	Training, $4op$	Training, $8op$	Test
MLGP-JSS	5.61 ± 0.59	2.00 ± 0.36	26.32 ± 2.86
MLGP-D1	5.88 ± 0.66	2.10 ± 0.47	28.42 ± 8.78
MLGP-D2	5.93 ± 0.87	$2.46 \pm 0.64^{\downarrow}$	27.90 ± 5.19
MLGP-D3	5.76 ± 0.75	$2.44 \pm 0.58^{\downarrow}$	27.90 ± 5.19
MLGP-D4	5.70 ± 0.57	$2.55 \pm 0.55^{\downarrow}$	26.80 ± 3.91

Table 7: The TWTs ($\times 10^5$) for the ensembles evolved by the coevolutionary GP approaches over the training and the test JSS problem instances. The ensembles are evolved from the training set $8op$.

Approach	Training, $4op$	Training, $8op$	Test
MLGP-JSS	5.61 ± 0.50	2.47 ± 0.40	25.32 ± 2.31
MLGP-D1	5.77 ± 0.77	2.34 ± 0.65	$27.41 \pm 4.89^{\downarrow}$
MLGP-D2	5.66 ± 0.75	2.41 ± 0.68	$27.24 \pm 4.22^{\downarrow}$
MLGP-D3	5.78 ± 0.79	2.53 ± 0.70	$27.24 \pm 4.22^{\downarrow}$
MLGP-D4	5.76 ± 0.60	2.51 ± 0.71	26.80 ± 3.85

6. EVALUATION

This section covers the evaluation of the niched coevolutionary GP approach against the base coevolutionary GP approach (MLGP-JSS). The niched GP with the different distance measures are denoted as follows: normalised priority distance (Section 4.1) is MLGP-D1, decision vector distance (Section 4.2) is MLGP-D2, decision overlap between individuals distance (Section 4.3) is MLGP-D3, and decision overlap against ensemble distance (Section 4.4) is MLGP-D4. 30 independent ensembles are evolved for each niched GP and MLGP-JSS from each training set (resulting in 60 ensembles over the two training sets $4op$ and $8op$) so that their performances can be compared over both the training and the test sets. In addition, the sizes of the evolved ensembles and the sizes of the individuals which make up the ensembles are also compared.

6.1 Performance Evaluation

After a set of ensembles is evolved from a coevolutionary GP (e.g. MLGP-D1), it is applied to the test set to obtain a set of schedules for each JSS problem instance. Afterwards, the average TWT values of the schedules for each problem instance is used to compare the performances of the sets of evolved ensembles against each other. A set of ensembles evolved from a specific training set is compared against other sets of ensembles evolved from the same training set, e.g., MLGP-JSS rules evolved from $4op$ is compared against MLGP-D1, MLGP-D2, MLGP-D3, and MLGP-D4 rules evolved from $4op$. We test whether one set of ensembles is significantly better than another by comparing the differences in the TWT values of the solutions generated over the training and test sets, and by using an one sided Z-test at $p = 0.05$. Summaries of the results that compare ensembles evolved from $4op$ and ensembles evolved from $8op$ are shown in Table 6 and Table 7 respectively. If the rules evolved by the niched MLGP-JSS approach performs better than the rules evolved by the base MLGP-JSS approach over the training or the test sets, then they are marked with $\uparrow$. Otherwise, if they perform worse, then they are marked with $\downarrow$.

From the results in Table 6, we can see that the ensembles evolved by base MLGP-JSS have slightly better performances, i.e., lower average TWT values, than the ensembles evolved by the niched approaches over the test set. However, for the rules evolved over $4op$, the difference in performances over the training set $4op$ and the test set is not significant. On the other hand, the rules evolved by the niched MLGP-JSS perform similar to the rules evolved by the base MLGP-JSS over both training sets ($4op$ and $8op$), but do not perform as well over the test set.

There are two potential explanations for the similar performances of the MLGP-JSS approaches and the base MLGP-JSS approach. The first explanation is that the MLGP-JSS approaches are too biased towards exploration then exploitation. Too much bias towards exploration means that the individuals that make up the evolved ensemble are not effective and cannot cover for each other's errors due to their poor individual performances. For future niched coevolutionary GP approaches, the following major modifications can be investigated to balance the exploration/exploitation components of the GP process and improve the quality of the evolved ensembles: 1) the adjustment to the fitnesses of the individuals from the densities of the individuals, 2) the distance measures for calculating the distances between the individuals, and 3) the maximum niching distance and its relation to the number of generations.

Another possible explanation is that the distance calculations between individuals in the niched MLGP-JSS approaches are likely to scale poorly with the rate of job arrivals in the training instances. When generating a schedule for a training instance that has high utilisation rate, it is likely that there will be large numbers of jobs waiting machines during dispatching decisions. However, the the number of jobs waiting at a machine affects the distance between the individuals. Therefore, the problem instances with high job arrival rate will likely have a greater influence on the distance calculation, and therefore a greater influence on how the fitnesses of the individuals are adjusted by the fitness sharing algorithm. Overall, the poor scaling of the utilisation rates of the training instances on the fitness and the distance calculations may result in the niched MLGP-JSS approaches overfitting on problem instances with high job arrival rates instead of being effective over all training instances. It is likely that this issue can be addressed by further refining the distance measures used for the niched MLGP-JSS. For example, a sequence of dispatching decision scenarios can be sampled after a "reference rule" [13] is be applied to the DJSS training instances, and the decisions made by the individuals over the dispatching decision scenarios can be used to compare the distances between individuals.

6.2 Analysis of Ensemble and Individual Sizes

After the ensembles are evolved from the training sets $4op$ and $8op$, two size comparisons are carried out. First is the sizes of the evolved ensembles, i.e., the number single dispatching rules that make up an ensemble. Second is the sizes of the arithmetic function tree used by the priority-based dispatching rules which make up the ensembles, i.e., the number of operators and terminals. The results of the size comparison is shown in Table 8. If ensembles evolved by the niched MLGP-JSS approaches have significantly smaller sizes than the ensembles evolved by the base MLGP-JSS approach, then they are marked with $\uparrow$.

From the results, we can see that the niched MLGP-JSS approach evolved significantly smaller ensembles than the base MLGP-JSS approach. On the other hand, the average sizes of the individuals which make up the ensembles are very similar to each other, and the differences in the sizes of the individuals evolved are not statistically significant. Combined with the results from Section 6.1, this means that the niched MLGP-JSS approach evolves ensembles that are slightly faster (as less individuals vote on each dispatching decision) and are more *interpretable*, i.e., easier to understand by human operators, than the base MLGP-JSS approach with a minor difference in the performance.

Table 8: The number of individuals which make up the ensembles evolved by the coevolutionary GP approaches from training sets $4op$ and $8op$, and the average number of terminal and non-terminal nodes for these individuals.

Approach	$4op$		$8op$	
	Ensemble Size	Avg. Tree Sizes	Ensemble Size	Avg. Tree Sizes
MLGP-JSS	15.1 ± 5.1	112.3	24.2 ± 4.7	104.4
MLGP-D1	$6.9 \pm 2.1^{\uparrow}$	116.2	$9.3 \pm 2.8^{\uparrow}$	115.4
MLGP-D2	$9.2 \pm 3.1^{\uparrow}$	112.8	$11.2 \pm 3.2^{\uparrow}$	114.8
MLGP-D3	$10.9 \pm 2.1^{\uparrow}$	115.0	$10.3 \pm 2.6^{\uparrow}$	96.0
MLGP-D4	$10.8 \pm 1.9^{\uparrow}$	118.1	$10.8 \pm 2.0^{\uparrow}$	114.9

For further analysis, the sizes of the evolved ensembles are compared against the performances of the ensembles over the test set to determine whether there is a correlation between the two variables. This allows us to gauge the possibility of evolving high quality ensembles with small sizes using a coevolutionary GP approach. The results of the comparisons for the ensembles evolved from the training sets $4op$ and $8op$ are shown in Figure 2a and Figure 2b respectively.

The figures show that the correlation between the evolved ensemble sizes and their performance over the test set is very weak to moderate. The most negative Spearman's rank correlation coefficient between the ensemble sizes and ensemble performances for a set of ensembles evolved by a particular MLGP-JSS approach is approximately -0.48 (for the ensembles evolved by MLGP-D4). On the other hand, the correlation between the ensemble sizes and performances for the base MLGP-JSS approach is -0.08 and -0.19 respectively, which is very weak. Therefore, it may be likely that small high quality ensembles can be evolved by coevolutionary GP.

In summary, the following findings were made in this investigation:

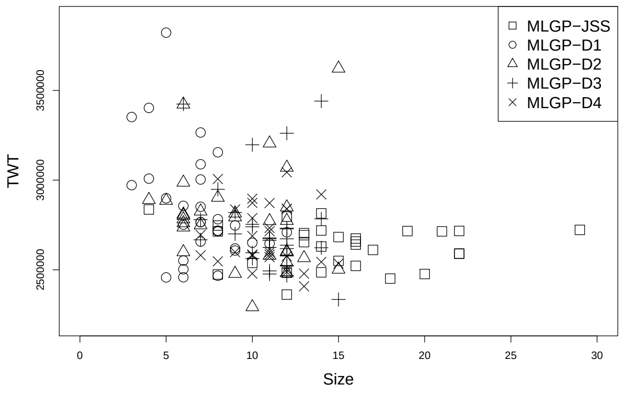
(a) Applying fitness sharing to a coevolutionary GP approach for DJSS is not a straight-forward procedure. The results shows that the niched GP approach does not significantly improve on the base coevolutionary GP approaach. However, niching has been shown to be effective in other problem domains, and there are other niching methods which could improve on a coevolutionary GP approach to evolve high quality ensembles. Therefore, this is an open area that shows promise, warranting future research.

(b) The niched coevolutionary GP approach evolved ensembles with significantly smaller number of members than the base coevolutionary GP approach, and the findings show that there are only moderate correlation between the sizes of the ensembles and the performances of the ensemble over the test set. Therefore, it may be possible to evolve more interpretable ensembles using niching that is competitive with larger ensembles evolved by the base coevolutionary GP approach.

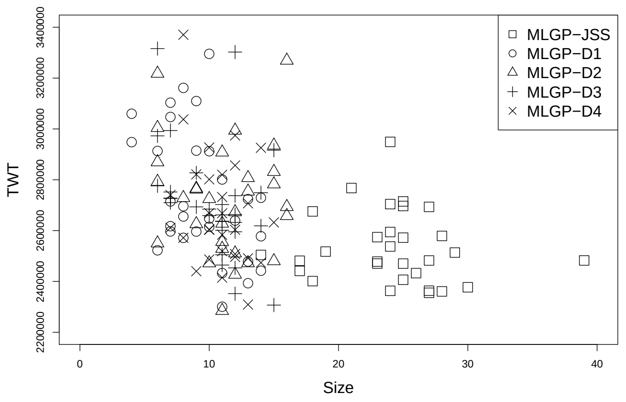
7. CONCLUSIONS AND FUTURE WORKS

In this paper, a fitness sharing algorithm is incorporated into coevolutionary GP approaches that evolve ensembles of dispatching rules for DJSS problems. The coevolutionary GP used is MLGP-JSS [8], an adaptation of the MLGP approach by Wu and Banzhaf [11]. Four different distance measures for the fitness sharing algorithm that use the decisions made by the individuals during dis-

Figure 2: Figures showing the comparison between the number of constituent components that make an ensemble, i.e., the size of an ensemble, against the performance of the ensemble over the test set.



(a) Comparison of the sizes of the ensembles evolved from $4op$ against the performances of the ensembles.



(b) Comparison of the sizes of the ensembles evolved from $8op$ against the performances of the ensembles.

patching decisions are investigated. Although the results show that the niched MLGP-JSS approach performs slightly worse than the base MLGP-JSS approach, it is possible that modifications to the fitness calculation and adjustment of the niched MLGP-JSS approach can significantly improve the overall qualities of the evolved ensembles. The results also show that the current niched MLGP-JSS approach evolves ensembles with significantly smaller number of constituent members than the base MLGP-JSS approach.

For future work, after fitness function and the distance measures have been modified to address the issues with the skewed individual fitnesses caused by the numbers of job arrivals and the job arrival rates for the problem instances, other coevolutionary GP that evolves ensembles of dispatching rules should be investigated to determine whether the fitness sharing algorithm significantly affects the quality of the evolved ensembles. One example is called Ensemble Genetic Programming for Job Shop Scheduling proposed by Park et al. [7], which adapts Potter and De Jong's cooperative coevolution [10] to evolve ensembles of dispatching rules for JSS problems.

8. REFERENCES

[1] Potts, C.N., Strusevich, V.A.: Fifty years of scheduling: a survey of milestones. Journal of the Operational Research Society **60** (2009) S41–S68

[2] Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. 4 edn. Springer (2012)

[3] Holthaus, O., Rajendran, C.: Efficient dispatching rules for scheduling in a job shop. International Journal of Production Economics **48**(1) (1997) 87–105

[4] Holthaus, O., Rajendran, C.: Efficient jobshop dispatching rules: Further developments. Production Planning & Control **11**(2) (2000) 171–178

[5] 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

[6] Branke, J., Nguyen, S., Pickardt, C., Zhang, M.: Automated design of production scheduling heuristics: A review. 10.1109/TEVC.2015.2429314 (2015)

[7] Park, J., Nguyen, S., Zhang, M., Johnston, M.: Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. Proceedings of the 2015 European Conference on Genetic Programming **9025** (2015) 92–104

[8] Park, J., Nguyen, S., Zhang, M., Johnston, M.: Genetic programming based hyper-heuristics to dynamic job shop scheduling: Cooperative coevolutionary approaches (2016) (to appear).

[9] Polikar, R.: Ensemble based systems in decision making. IEEE Circuits and Systems Magazine **6**(3) (2006) 21–45

[10] Potter, M.A., De Jong, K.A.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. Evolutionary Computation **8**(1) (2000) 1–29

[11] 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

[12] Pérez, E., Herrera, F., HernÃąndez, C.: Finding multiple solutions in job shop scheduling by niching genetic algorithms. Journal of Intelligent Manufacturing **14**(3-4) (2003) 323–339

[13] Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. http://dx.doi.org/10.1162/EVCO_a_00133 (2015)

[14] Geiger, C.D., Uzsoy, R.: Learning effective dispatching rules for batch processor scheduling. International Journal of Production Research **46**(6) (2008) 1431–1454

[15] 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. (2010) 257–264

[16] Hunt, R., Johnston, M., Zhang, M.: Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation. (2014) 927–934

[17] Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the Second International Conference on Genetic Algorithms. (1987) 41–49