Evolutionary Multitask Optimisation for Dynamic Job Shop Scheduling using Niched Genetic Programming

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
[1]{John.Park,Yi.Mei,Aaron.Chen,Mengjie.Zhang}@ecs.vuw.ac.nz
[2]p.nguyen4@latrobe.edu.au

Abstract. Dynamic job shop scheduling (DJSS) problems are combinatorial optimisation problems where dynamic events occur during processing that prevent scheduling algorithms from being able to predict the optimal solutions in advance. DJSS problems have been studied extensively due to the difficulty of the problem and their applicability to real-world scenarios. This paper deals with a DJSS problem with dynamic job arrivals and machine breakdowns. A standard genetic programming (GP) approach that evolves dispatching rules, which is effective for DJSS problems with dynamic job arrivals, have difficulty generalising over problem instances with different machine breakdown scenarios. This paper proposes a niched GP approach that incorporates multitasking to simultaneously evolve multiple rules that can effectively cope with different machine breakdown scenarios. The results show that the niched GP approach can evolve rules for the different machine breakdown scenarios faster than the combined computation times of the benchmark GP approach and significantly outperform the benchmark GP's evolved rules. The analysis shows that the specialist rules effective for DJSS problem instances with zero machine breakdown have different behaviours to the rules effective for DJSS problem instances with machine breakdown and the generalist rules, but there are also large variance in the behaviours of the zero machine breakdown specialist rules.

1 Introduction

Job shop scheduling (JSS) problems [1] are combinatorial optimisation problems with significant importance in operation research and artificial intelligence [2]. JSS also has applications to real-world manufacturing environments and production scheduling [3]. Because of this, JSS problems have been extensively studied over the past 60 years by both academics and industry experts [4]. A JSS problem instance consists of a *shop floor* with a limited number of *machine* resources that is used to process incoming *jobs* [1]. To process a job, a job's *operations* need to be processed in a specific sequence, and each operation require a specific machine to process the operation. In addition, a machine can only process one operation at a time. The goal of JSS is to make intelligent decisions during processing to optimise a given *objective function*. Finally, in a real-world scenario, there are unforeseen events that can occur which can affect the

properties of the shop floor [5]. JSS problems with unforeseen events are called *dynamic* JSS (DJSS) problems, and have been studied extensively in the literature [5].

For this paper, we deal with a DJSS problem with the mean weighted tardiness (MWT) objective [1], dynamic job arrivals and machine breakdowns [3, 5]. This means that the jobs' properties are unknown they reach the shop floor, and unforeseen breakdowns of machines occur during processing where the machines need to be repaired for durations of times before they are available to process the jobs' operations. The most prominent method of handling DJSS problems with dynamic job arrivals is to evolve effective dispatching rules using evolutionary computation (EC) techniques such as genetic programming (GP) [3, 2]. In general, the rules evolved by the EC techniques generally outperform the man-made dispatching rules [3] given that the training set used to evolve the GP rules is appropriate for the DJSS problem that the rules are applied to [6]. On the other hand, predictive-reactive approaches are extensively applied to DJSS problems with machine breakdowns [5], and attempt to generate schedule that are robust as possible to disruptions caused by machine breakdowns. They often focus on small DJSS problem instances with fixed number of job arrivals (e.g. up to 80 jobs [7]). Both dynamic job arrivals and machine breakdowns have been studied extensively in the literature, but there has only been a limited number of GP approaches to DJSS problems with both dynamic job arrivals and machine breakdowns [8].

By investigating the two types of dynamic events simultaneously, it is likely that we can extend the scope of research into DJSS problems, and better emulate real-world scenarios where a large numbers of unforeseeable events are likely to occur. However, preliminary investigation by Park et al. [8] showed that it is too difficult for the evolved rules to generalise effectively over the different machine breakdown scenarios, and showed that the GP rules evolved using instances from all machine breakdown scenarios (i.e. "generalist" rules) were more biased towards the DJSS problem instances with no machine breakdowns than problem instances with machine breakdowns. Therefore, a GP approach that handles the DJSS problem by allowing the GP individuals to focus on the different machine breakdown scenarios as much as possible may be more effective than a standard GP approach. By focusing on specific machine breakdown scenarios, useful features can be discovered by the GP that can be shared during the GP process to improve the overall qualities of the output rules [9]. The idea of decomposing a problem to smaller subproblems has parallels to *multitask learning* [9], where multiple *tasks* are solved simultaneously. Evolutionary multitasking techniques [10] have been effectively applied to solve multiple optimisation problems concurrently [11], but they have not been applied to DJSS problems.

1.1 Goal

The goal of this paper is to develop a multitask GP approach that is able to cope with a DJSS problem with dynamic job arrivals and various severity of machine breakdowns that occur during processing. To do this, we propose a niched GP approach that evolves two types of rules: "generalist" rules that are effective over the entire DJSS problem and "specialist" rules that specifically handle the designated machine breakdown scenario that they are specialised for. Specialist rules are useful when the overall machine breakdown properties of the problems (e.g. the distribution of repair times [12]) are known in

advance, and the generalist rules are useful otherwise. Compared to a standard GP approach that evolves GP rules for the different machine breakdown scenarios separately, a niched GP approach that evolves rules simultaneously has the potential to improve on the effectiveness of the evolved rules by sharing useful properties of rules effective on other machine breakdown scenarios, and have better performances overall than the standard GP approach. In addition, by analysing the rules evolved by the niched GP approach, e.g., by determining the behaviours of different specialist rules, we can observe the overlap between the machine breakdown scenarios based on the behaviours of the evolved specialist rules.

2 Niched GP Approach to Handling DJSS Problems with Machine Breakdown

This section covers the niched GP approach that is used to evolve a generalist rule and specialist rules for the different machine breakdown scenarios simultaneously. First, we give the framework of the niched GP process, then provide the details on the niched GP's representation, the terminal and the function sets.

2.1 Overall Framework

The niched GP approach in this paper is extended from a niched GP approach proposed by Mei et al. [13] used to evolve a diverse set of rules. The niched GP keeps track of the GP individuals that are the best for the different machine breakdown scenarios during training. By doing this, GP may be able to retain useful features from rules which may not have the best overall performance which can then be shared with other GP individuals.

For the niched GP process, the GP individuals are first randomly initialised and the set of specialist rule $\mathcal{S}$ is empty. For each generation when a GP individual p is being evaluated, the individual is first evaluated to the "general" training set $\mathcal{T}$ to calculate its fitness $f(p)$. The problem instances in training set $\mathcal{T}$ consist of N machine breakdown scenarios. If an individual p has the best performance for problem instances in niche n (i.e. under a specific machine breakdown scenario) in the training set $\mathcal{T}$, then current generation niche individual t_I for niche n is updated to individual p. After all GP individuals in the current population have been evaluated, the current generation niched individuals $t_1, \ldots, t_N$ are then compared against the overall niched individuals $s_1, \ldots, s_N$. To compare the current generation niched individuals t_n to overall niched individual s_n, the individuals are evaluated on a niched training set $\mathcal{V}_n$, separate from the general training set $\mathcal{T}$, that only consists of problem instances with the specific machine breakdown scenario (i.e. the niched training sets are validation sets specifically for the niched individuals). If the $f'(t_n)$ of the current generation niched individual t_n is better than the fitness $f'(s_n)$ of the overall niched individual s_n over the niched training set $\mathcal{V}_n$, then s_n is updated to the current generation's niche individual t_n. Otherwise, the individual t_n is kept the same.

After the set of niched individuals has been updated, the clearing algorithm (denoted as $\texttt{Clearing}(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$) is carried out before the individuals undergo the standard

Algorithm 1 $\mathcal{S} \leftarrow$ `NichedGP`(G)

Output: The set of specialist rules $\mathcal{S}$ and the generalist rule g.
Initialise GP population $\mathcal{P}$;
Initialise specialist rule set $\mathcal{S} \leftarrow \{s_1, \ldots, s_N\}$ for the N niches;
for $gen \leftarrow 1$ ***to*** G **do**
  Set $t_1, \ldots, t_N \leftarrow \varnothing$ and $f_1, \ldots, f_{|\mathcal{P}|} \leftarrow 0$;
  for *each individual* p *in GP population* $\mathcal{P}$ **do**
    for *each problem instance* I *in the training set* $\mathcal{T}$ **do**
      Apply individual p to I to calculate normalised objective $Obj'(p, I)$;
    end
    Update $f(p)$ and g;
    Update t_n if p is better on problem instances in niche n;
  end
  for t_n *in* $t_1, \ldots, t_N$ **do**
    Apply t_n to problem instances in niched training set $\mathcal{V}_n$ and calculate the performance $f'(t_n)$ over the niched training set;
    Update $s_n \leftarrow t_n$ if $f'(t_n) < f'(s_n)$;
  end
  $\mathcal{P}' \leftarrow$ `Clearing`$(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$;
  Apply the breeding procedure using $\mathcal{P}'$ and update the population $\mathcal{P}$;
end
Output the set of specialist GP rules $\mathcal{S}$ and the best overall GP rule g;

tournament selection procedure. The clearing algorithm is modified from the algorithm used by Mei et al. [13]. However, unlike Mei et al.'s clearing procedure, where all GP individuals have a niche radius, only the specialist GP rule from the rule set $\mathcal{S}$ (i.e., niched individuals that perform the best on the different machine breakdown scenarios) and the best individual found so far are used as the niches in our niched GP approach. Afterwards, the individuals with poor performances within distance σ from the best niched individuals are removed from the GP population if the niche has reached its capacity κ. This continues until the maximum number of generations has been reached. Finally, the algorithm reports the best overall rule as the generalist rule g and the set of specialist rules $\mathcal{S}$. The pseudocode that summarises the niched GP process is shown in Algorithm 1.

Given that the same training set $\mathcal{T}$ is used, the niched GP approach will likely have a greater computation time than a standard GP approach that uses a single population because it further evaluates the niched individuals on the niched training sets on top of the standard evaluation procedure. When evolving dispatching rules for DJSS problems using GP, the evaluation procedure and the application of the individuals on the training instances is the most computationally intensive step of the GP process [2]. As the niched GP approach will have additional *# of niches* $\times$ *niched training sets sizes* simulation runs, the niched GP approach requires a total of $|\mathcal{P}| \times |\mathcal{T}| + N \times |\mathcal{V}|$ simulation runs. However, the niched GP approach will still have significantly shorter computation time compared to evolving generalist and specialist rules using a standard GP separately,

Table 1: Terminal set for GP, where a job j is waiting at the available machine m at a decision situation.

Terminal	Description
RJ	operation ready time of job j
PT	operation processing time of job j
RO	remaining number of operations of job j
RT	remaining total processing times of job j
RM	machine m's ready time
WINQ	work in next queue for job j
DD	job's due date d_j

Terminal	Description
SL	slack of job j
W	job's weight w_j
NPT	next operation processing time of job j
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

which requires $2 \times |\mathcal{P}| \times |\mathcal{T}|$ simulation runs to evaluate all the GP individuals per generation.

2.2 GP Representation, Terminal Set and Function Set

The GP representation, terminals and function sets are adapted from the GP approach used by Park et al. [8] to investigate the DJSS problem with dynamic job arrivals and machine breakdowns. For the niched GP approach, the GP individuals are arithmetic function trees that are used to calculate the priorities of jobs waiting at an available machine m^* during a decision situation [3]. The terminals listed in Table 1 for a GP individual's tree correspond to job, machine and shop floor attributes. The non-terminals consist of arithmetic operators $+$, $-$, $\times$, protected $/$, binary operators $\max$, $\min$ and a ternary operator `if`. Protected $/$ returns 1 if the denominator is zero, and returns the output of a standard division operator otherwise. `if` operator returns the value of the second child branch (representing the "then" condition) if the first child branch (representing the input into the "if" condition) is greater than or equal to zero, but returns the value of the third child branch (representing the "else" condition) otherwise.

2.3 Evaluation Procedure

The GP individual p is applied to the DJSS problem instances in the training sets as a *non-delay* [1] dispatching rule. The individual p is applied to a problem instance I to generate a schedule. Afterwards, the MWT value $Obj(p, I)$ of the schedule is normalised using a *reference rule* to reduce bias towards specific DJSS problem instances [14]. The reference rule R, which is the weighted apparent tardiness cost (wATC) rule [1], is applied to problem instance I to generate a schedule with $Obj(R, I)$. Afterwards, the normalised MWT value is calculated as $Obj'(p, I) = \frac{Obj(p,I)}{Obj(R,I)}$. From the normalised objective values, the fitness of individual p is given by $f(p) = \frac{1}{|\mathcal{T}|} \sum_{I \in \mathcal{T}} Obj'(p, I)$ after the individual has been applied to all problem instances in the training set $\mathcal{T}$.

Table 2: The parameters used for simulating a DJSS problem instance.

Parameters		Value
	Number of machines	10
	Warm up jobs	500
	# completed jobs before simulation termination	2500
Shop floor parameters	Utilisation rate	90%
	Job arrival rate (λ)	$\lambda \sim Poisson(13.5)$
	Operation processing times (o_{ij})	$o_{ij} \sim Unif[1, 49]$
	# operations per job (N_j)	$N_j \sim Unif[2, 10]$
	Job weight	Random from 1, 2, 4 with probabilities 20%, 60%, 20%
	Due date tightness	3.0 or 5.0
Machine breakdown parameters	Breakdown level	0%, 2.5% or 5%
	Mean repair time	25, 125 or 250

3 Experimental Design

This section describes the simulation model used to evaluate the specialised rules for the niched GP approach, followed by a description of benchmark GP used for comparison during evaluation. Afterwards, detailed parameter settings for GP and niching are provided.

3.1 DJSS Simulation Model

Discrete-event simulations are the standard method of simulating job shop scheduling problem instances [3]. A discrete-event simulation stochastically generates the dynamic events, i.e., the job arrivals and the machine breakdowns. The simulation model is adapted from the simulation model used by Park et al. [8], which is a modification of Holthaus's [12] simulation model. In the simulations, *machine breakdown level* (the proportion of simulation duration the machines are broken down [12]) and *mean machine repair time* parameters are used to stochastically generate machine breakdowns [12]. There are three different parameter values for machine breakdown level, three different values for mean times required to repair the machines, and two different values for the *due date tightness*. Due date tightness is a simulation parameter used to determine how the due date is generated for a job arrival [12]. Since the repair times are not a factor when the breakdown level is zero, i.e., there is no machine breakdown, the DJSS problem instances can be generated from $2 \times 3 \times 3 = 14$ different scenarios. There is no re-entry for the arriving jobs, i.e., a job has at most one operation on a machine [1]. These parameters are listed below in Table 2.

At each generation, the training set $\mathcal{T}$ simulates a DJSS problem instance from each simulation configuration scenarios (i.e. different combinations of due date tightness, breakdown level and mean repair time), resulting in a GP individual being applied to 14 DJSS problem instances. The simulations used in training set $\mathcal{T}$ are grouped up into the seven groups of two problem instances based on their breakdown level and mean repair time, e.g., a group with breakdown level of 2.5% and mean repair time of 25 is denoted

as $\langle 2.5\%, 25\rangle$. The group with no machine breakdowns (i.e. has a breakdown level of 0%) is simply denoted as $\langle 0\rangle$. In other words, the seven groups are the "niches" that are filled up by GP individuals that perform the best for the different machine breakdown level and mean repair time parameter values (i.e. $N = 7$). The seed used to simulate the problem instances from the simulation configurations are rotated every generation to help improve the generalisation ability of the evolved rules [13].

After the current generation niched individuals have been found, they are further evaluated on the niched training sets to update the set of specialist GP rules $\mathcal{S}$. A niched individual $t_{\langle b,r\rangle}$ from the current generation for the scenario $\langle b, r\rangle$ is applied to the niched training set $\mathcal{V}_{\langle b,r\rangle}$. The niched training set $\mathcal{V}_{\langle b,r\rangle}$ has the configurations with due date tightness of 3.0 or 5.0. In other words, further two simulation runs are used per niched individuals. Finally, to ensure that the comparisons between the rules between different generations are kept consistent, the niched training sets are fixed over every generation for the niched GP.

3.2 GP Benchmarks

To evaluate the niched GP's evolved rules, we use a standard single-tree GP representation [3, 2] with the same terminal and function set used by the niched GP for consistency (Table 1). Afterwards, the benchmark GP is applied to the machine breakdown scenarios independently to evolve the generalist and the specialist rules. The entire training set $\mathcal{T}$ is used to evolve the generalist rules from the benchmark GP approach. To evolve the specialist rules, instead of using the entire training set $\mathcal{T}$ described above, the benchmark GP only uses specific machine breakdown scenarios to evolve dispatching rules, e.g., a GP process is run with training instances being generated from $\langle 2.5\%, 25\rangle$. Since there are two possible due date tightness parameters, an individual in the benchmark GP process is applied to two training problem instances during the evaluation procedure. The best individual of the last generation before maximum number of generations is reached is the output dispatching rule for the benchmark GP process.

3.3 GP and Niching Parameters

The niched and the benchmark GP approaches follow parameters used by existing GP approaches for DJSS problems [8]. The GP 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 the individuals during initialisation is 4, and 8 across all generations of the GP process. Tournament selection of size 7 for both the two GP approaches. For the clearing algorithm $\texttt{Clearing}(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ used by the niched GP approach, the two parameters niche radius σ and niche capacity κ are kept consistent as the parameters used by Mei et al. [13], i.e., $\sigma = 1$ and $\kappa = 1$. Finally, $k = 3.0$ is used for the wATC reference rule used for the fitness calculation (Section 2.3).

Table 3: Comparison of the computation time required to evolve the rules for the GP approaches (in seconds).

Approach		Computation Time ($\times 10^4$ s) GP	NGP
Generalist		2.17 ± 0.35	$-$
Specialist	$\langle 0\%, 0 \rangle$	0.20 ± 0.02	$-$
	$\langle 2.5\%, 25 \rangle$	0.25 ± 0.03	$-$
	$\langle 2.5\%, 125 \rangle$	0.30 ± 0.05	$-$
	$\langle 2.5\%, 250 \rangle$	0.26 ± 0.04	$-$
	$\langle 5\%, 25 \rangle$	0.31 ± 0.05	$-$
	$\langle 5\%, 125 \rangle$	0.42 ± 0.07	$-$
	$\langle 5\%, 250 \rangle$	0.37 ± 0.07	$-$
	Specialist Combined	2.10 ± 0.15	$-$
Total		4.27 ± 0.39	2.32 ± 0.34

4 Experimental Results

To compare the two GP approaches, the GP process is run 30 times over each machine breakdown scenarios to obtain sets of independent rules for the different machine breakdown scenarios. The computation times of the runs is also recorded and compared against each other before comparing the performances of the generalist and the specialist rules. One GP approach is significantly better than the other GP approach either in terms of computation time or performance if it can be verified by the two sided Student's t-test at $p = 0.05$. After the comparisons, we analyse the behaviours of the rules evolved by the niched GP approach.

4.1 Computation Costs

Both the niched and the benchmark GP approaches are implemented in a Java program ran on Intel(R) Core(TM) i7 CPU 3.60GHz. The time taken to evolve the rules is given in Table 3, where the computation time is measured in seconds. In the table, "Specialist Combined" denotes the sum of the times required to evolve the specialist rules with the benchmark GP approaches over the different machine breakdown scenarios. This is to compare the overall computation time required to evolve the specialist rules with the benchmark GP approach to the niched GP approach, as the niched GP approach evolves the generalist rule and the specialist rules simultaneously over a single run. "Total" denotes the sum of the time required to evolve the generalist and the specialist rules for the niched and the benchmark GP approaches. Since niched GP approach evolves the generalist and the specialist rules simultaneously, its time is only given in the "Total" category.

From the tables, compared to the combined amount of time required to evolve the specialist rules or the generalist rules individually using the benchmark GP approach, the niched GP approach takes significantly longer amount of time. This is due to the additional evaluation required to further evaluate the niched individuals in the niched GP approach after the individuals have been evaluated over the training set. However,

for evolving all rules, i.e., both the generalist and the specialist rules, the niched GP approach is significantly faster than the benchmark GP approach.

An interesting observation is that the additional computation time required by the niched GP approach does not exactly correspond with the theoretical amount of time required to further evaluate the niched GP individuals (Section 2.1). From the GP and the DJSS parameters, the number of simulation runs required per generation for each specialist rules for the benchmark GP approaches is the population size times the number of configurations per machine breakdown scenario, i.e., $1024 \times 2 = 2048$. Combined together, the total number of simulation runs required by the benchmark GP approach to evolve the specialist rules require $2048 \times 7 = 14336$. This is equivalent to the number of simulation runs required by the niched GP approach to evaluate the GP population minus the additional runs required to further evaluate the niched GP individuals, which requires $7 \times 2 = 14$ simulation runs. This means that the additional simulation runs should approximately add $14/14336 \times 100\% = 0.1\%$ overhead to the niched GP approach compared to evolving the specialist rules separately. However, the experiments show that the niched GP approach takes $\sim 10\%$ longer computation time to evolve the rules compared to the combined time required by the benchmark GP approach to evolve the specialist rules. Instead, the additional computation time is likely be due to the fitness adjustments made to GP individuals that are close to the niched individuals in the clearing algorithm (Section 2.1). It may also be likely that the evolved GP rules for the niched GP approach is also bigger, which results in longer computation time required to calculate the priorities of jobs during the simulation.

4.2 Performance Comparison of Evolved Rules

The performances of a set of rules are calculated by applying the evolved rules to simulation models generated from the simulation configurations provided in Section 3.1. A simulation model in the test set uses a new seed so that the exact times of the job arrivals and machine breakdowns (and their properties) that are generated by the simulation differs from the simulations during training. An evolved rule is applied to DJSS simulation model to generate a schedule and get a MWT objective value. This is then repeated 30 times with different seeds for the simulation model to get an average MWT performance of the evolved rule over the simulation configuration. The evolved generalist rules are applied to all simulation configurations, whereas the evolved specialist rules are applied to the machine breakdown scenarios they are designed for. The performances of the specialist and the generalist rules are given in Table 4. In the table, $\mu \pm \sigma$ for each set of rules denotes that the mean MWT performance is μ and the standard deviation is σ. In addition, $\langle b, r, h \rangle$ in the tables denotes that the particular simulation model has b breakdown level, r mean repair time and h due date tightness factor.

From the tables, we can see that the niched GP approach generally outperforms the benchmark GP approach in terms of both the specialist rules performances and the generalist rules performances. The only configuration scenarios where the benchmark specialist rules significantly outperform the niched specialist rules are on the simulations with zero machine breakdowns. In addition, the benchmark specialist rules are slightly better than the niched specialist rules for the scenarios $\langle 5\%, 25, 3 \rangle$, $\langle 5\%, 250, 5 \rangle$ and $\langle 5\%, 250, 3 \rangle$, but the differences are not significant.

Table 4: Comparison showing the mean and the standard deviation of the MWT performances for the specialist and the generalist rules evolved by the niched and the benchmark GP approaches over the test simulation runs.

Approach		Specialist		Generalist	
		NGP	GP	NGP	GP
MWT ($\times 10^2$)	$\langle 0\%, 0, 5\rangle$	2.26 ± 0.44	1.94 ± 0.22	2.16 ± 0.13	2.21 ± 0.10
	$\langle 0\%, 0, 3\rangle$	4.68 ± 1.35	3.43 ± 0.17	3.25 ± 0.06	3.29 ± 0.11
	$\langle 2.5\%, 25, 5\rangle$	3.43 ± 0.21	3.55 ± 0.23	3.37 ± 0.12	3.46 ± 0.13
	$\langle 2.5\%, 25, 3\rangle$	4.60 ± 0.21	4.72 ± 0.22	4.45 ± 0.06	4.52 ± 0.12
	$\langle 2.5\%, 125, 5\rangle$	4.50 ± 0.10	4.69 ± 0.17	4.45 ± 0.13	4.54 ± 0.14
	$\langle 2.5\%, 125, 3\rangle$	5.81 ± 0.08	6.10 ± 0.16	5.82 ± 0.08	5.93 ± 0.16
	$\langle 2.5\%, 250, 5\rangle$	6.12 ± 0.17	6.28 ± 0.15	6.06 ± 0.14	6.18 ± 0.20
	$\langle 2.5\%, 250, 3\rangle$	7.55 ± 0.11	7.71 ± 0.15	7.51 ± 0.11	7.62 ± 0.18
	$\langle 5\%, 25, 5\rangle$	4.50 ± 0.13	4.58 ± 0.19	4.40 ± 0.15	4.52 ± 0.21
	$\langle 5\%, 25, 3\rangle$	6.50 ± 0.26	6.42 ± 0.18	6.33 ± 0.17	6.47 ± 0.27
	$\langle 5\%, 125, 5\rangle$	6.40 ± 0.15	6.51 ± 0.20	6.42 ± 0.16	6.56 ± 0.26
	$\langle 5\%, 125, 3\rangle$	8.42 ± 0.19	8.45 ± 0.26	8.55 ± 0.21	8.74 ± 0.35
	$\langle 5\%, 250, 5\rangle$	8.77 ± 0.33	8.74 ± 0.35	8.95 ± 0.27	9.20 ± 0.46
	$\langle 5\%, 250, 3\rangle$	11.20 ± 0.29	11.15 ± 0.30	11.43 ± 0.32	11.65 ± 0.46

Another interesting observation is that the generalist rules for the benchmark GP approach performs better than the specialist rules for a number of simulation configurations (from the simulation configuration $\langle 0\%, 0, 3\rangle$ to the simulation configuration $\langle 2.5\%, 250, 3\rangle$). This is likely attributed to the number of simulation runs each GP individual during the GP process undergoes during the evaluation procedure. The generalist rules are applied to 14 different simulation runs with different machine breakdown scenarios, whereas a specialist rule is only applied to two simulation runs over the specific machine breakdown scenario. In other words, the GP individuals in the benchmark GP process may not have had enough training instances to effectively evaluate the qualities of the individuals, resulting in underperforming specialist rules. To verify this, we ran additional experiments for the benchmark GP process where the GP individuals are applied to simulations under a specific machine breakdown scenario runs 14 times instead of two times, using different seeds for each simulation. The specialist rules evolved using the additional simulation runs for the GP individuals performs significantly better than the generalist rules.

4.3 Diversity Analysis

For the analysis procedure, the goal is to find differences in terms of the rules' behaviours that have been evolved using different machine breakdown scenarios. To do this, we calculate the phenotypic distances between the rules evolved by the niched GP approaches using the job rank distance measure proposed by Hildebrandt and Branke [14] and used by the clearing algorithm $\texttt{Clearing}(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ [13]. The distances between a single rule in a rule set are compared against the 30 rules of another rule set to obtain an average distance of the single rule to the rule set. The means and the standard deviations of the average distances of the rule for the generalist and the specialist rule

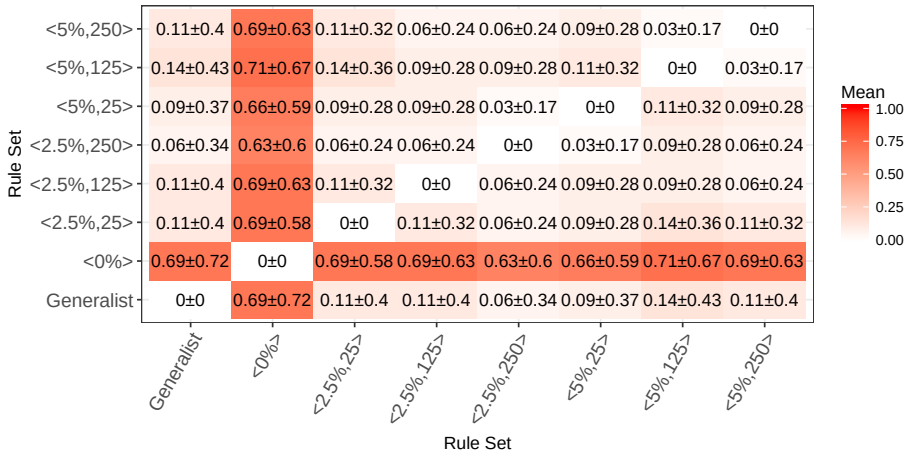

Fig. 1: Pairwise mean and standard deviations of the average distances between the rules evolved by the niched GP approach using Hildebrandt and Branke's ranked distance measure [14]. First table compares the specialist rule sets against each other, and the second table compares the generalist rule set against the specialist rule sets.

sets are shown in Fig. 1. In the figure, we provide a heat map of the average distances of two sets of rules as visual aids.

Compared to the other scenarios, the specialist rules that specialise on the scenario with zero machine breakdown has a higher average distances from the specialist rules evolved on other machine breakdown scenarios and the generalist rules. This implies that the rules that are effective on DJSS problems with only dynamic job arrivals are very different from the rules that are effective on DJSS problems with both dynamic job arrivals and machine breakdowns. However, the large standard deviation in the average distances between the behaviours of the rules evolved on zero machine breakdown and the other sets of evolved rules means that the differences in the distances are not statistically significant. Therefore, further experiments that isolate individual rules and analyse their behaviours may be required.

5 Conclusions and Future Work

This paper proposes a novel niched GP approach that incorporates multitasking [9] to evolve effective dispatching rules for a DJSS problem with dynamic job arrivals and machine breakdowns. The proposed niched GP approach evolves a generalist and multiple specialist rules for the different machine breakdown scenarios simultaneously. Evolving the generalist rules and the specialist rules for niched GP approach is significantly faster than sequentially evolving the rules using a benchmark GP approach. In addition, the evolved rules from the niched GP approach generally outperform the rules evolved by the benchmark GP approach.

For the future work, it may be promising to further investigate the behaviours of the rules evolved on the different machine breakdown scenarios, to determine why the

rules evolved for the niched GP approach performs significantly better. The preliminary comparison shows that the rules that are effective for DJSS problem instances with no machine breakdowns behave differently than the rules that are effective for DJSS problem instances with machine breakdown, but the large variance in the behaviours of the rules means that this difference is not significant. In addition, further experiments that apply the rules to DJSS problem instances with unseen machine breakdown scenarios (e.g. a DJSS problem instance with machine breakdown level of 10%) may further be able to test generalisation ability of the generalist rules evolved by the niched GP approach.

References

1. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. 4 edn. SpringerUS (2012)
2. 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
3. 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
4. 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
5. Ouelhadj, D., Petrovic, S.: A survey of dynamic scheduling in manufacturing systems. Journal of Scheduling **12**(4) (2009) 417–431
6. 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
7. Yin, W.J., Liu, M., Wu, C.: Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In: Proceedings of IEEE Congress on Evolutionary Computation (CEC 2003). (2003) 1050–1055
8. Park, J., Mei, Y., Chen, A., Zhang, M.: Investigating the generality of genetic programming based hyper-heuristic approach to dynamic job shop scheduling with machine breakdown. In: Proceedings of the 2017 Australasian Conference on Artificial Life and Computational Intelligence. Volume 10142 of Lecture Notes in Artificial Intelligence., Springer International Publishing (2017) 301–313
9. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering **22**(10) (2010) 1345–1359
10. Ong, Y.S., Gupta, A.: Evolutionary multitasking: A computer science view of cognitive multitasking. Cognitive Computation **8**(2) (2016) 125–142
11. Gupta, A., Ong, Y.S., Feng, L.: Multifactorial evolution: toward evolutionary multitasking. IEEE Transactions on Evolutionary Computation **20**(3) (2016) 343–357
12. Holthaus, O.: Scheduling in job shops with machine breakdowns: an experimental study. Computers & Industrial Engineering **36**(1) (1999) 137–162
13. Mei, Y., Nguyen, S., Xue, B., Zhang, M.: An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. IEEE Transactions on Emerging Topics in Computational Intelligence **1**(5) (2017) 339–353
14. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evolutionary Computation **23**(3) (2015) 343–367