Evolving Dispatching Rules for Dynamic Job Shop Scheduling Problems using Genetic Programming

by

John Park

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2019

Abstract

Job shop scheduling (JSS) problems are difficult combinatorial optimisation problems that have been studied over the past 60 years. The goal of a JSS problem is to schedule the arriving *jobs* as effectively as possible on the limited *machine* resources on the *shop floor*. Each job has a sequence of operations that need to be processed on specific machines, but the machines can only process one job at a time. JSS and other types of scheduling are important problems in manufacturing systems, such as semiconductor manufacturing. In particular, this thesis focuses on *dynamic* JSS (DJSS) problems, where unforeseen events occur during processing that needs to be handled by the manufacturer. Examples of dynamic events that occur in DJSS problems are dynamic or unforeseen job arrivals, machine breakdowns, uncertain job processing times, and so on.

A prominent method of handling DJSS problems is to design effective *dispatching rules* for the DJSS problem handled by the manufacturer. Dispatching rules are local decision makers that determine what job is processed by a machine when the machine finishes processing the previous job and becomes available. Dispatching rules have been investigated extensively by both academics and industry experts due to their simplicity, interpretability, low computational cost and their ability to cope effectively in dynamic environments. However, dispatching rules are designed for a specific DJSS problem and have no guarantee that they retain their effectiveness on other DJSS problems. In a real-world scenario, the properties of a manufacturing system can change over time, meaning that previously effective dispatching rule may longer be effective. Therefore, a manufacturer may need to redesign a dispatching rule to maintain a competitive

edge on the market. However, designing an effective dispatching rule for a specific DJSS problem is expensive, and typically requires a human expert and extensive trial-and-error process to verify their effectiveness. To circumvent the manual design of dispatching rules, researchers have proposed *hyper-heuristic* approaches to automate the design of dispatching rules. In particular, various genetic programming based hyper-heuristic (GP-HH) approaches have been proposed in the literature to evolve effective dispatching rules for scheduling problems, including DJSS problems. However, there are many potential directions that have not been fully investigated.

The overall goal of this thesis is to develop new and effective GP-HH approaches to designing high-quality dispatching rules for DJSS problems that aim to improve beyond the standard GP approach while maintaining computational efficiency. The focus will be on developing approaches which can decompose complex JSS problems down to simpler subcomponents, evolving **multiple heuristics** to handle the subcomponents, and developing GP-HH approaches that can handle complex DJSS problems by exploiting the problem properties.

This thesis is the first to develop ensemble GP approaches that evolve *ensembles of dispatching rules* using cooperative coevolution. In addition, the thesis also investigates different combination schemes for one of the ensemble GP approaches to combine the ensemble member outputs effectively. The results show that ensemble GP approach evolves rules that perform significantly better than the rules evolved by the benchmark GP approach.

This thesis provides the first investigation into applying GP-HH to a DJSS problem with dynamic job arrivals and machine breakdowns. In addition, the thesis also develops machine breakdown GP approach to the DJSS problem by incorporating machine breakdown GP terminals. The results show that the standard GP do not generalise well over the DJSS problem. The best rules from the machine breakdown GP approach do

perform better than the best rule from the standard GP approach, and the analysis shows that the rules behaviour is similar to the shortest processing time rule in certain decision situations.

This thesis is the first to develop a multitask GP approach to evolve a *portfolio* of dispatching rules for a DJSS problem with dynamic job arrivals and machine breakdowns. The multitask GP approach improve on the standard GP approach either in terms of the effectiveness of the output rules or the computation time required to evolve the rules. The analysis shows that the difference between DJSS problem having no machine breakdowns and having machine breakdowns is a more significant factor than the difference between two DJSS problems with different frequencies of machine breakdown investigated.

Acknowledgements

I would like to thank my supervisors, Prof. Mengjie Zhang, Dr. Yi Mei, Dr. Gang Chen and Dr. Su Nguyen for providing me with constant support and encouragement before and over the duration of my PhD research. The feedback they provided on my research has been invaluable for my learning experiences as a PhD student and helped me improve various skills such as my analytical skills.

Thank you to the people in the Evolutionary Computation Research Group (ECRG) for their support and for providing a vibrant community at the university for the evolutionary computation (EC) technique.

I would also like to thank Victoria University of Wellington for financially supporting me through the Victoria Doctoral Scholarship and Prof. Mengjie Zhang for supporting through the Marsden Grant.

Finally, I would like to thank my parents for their constant support and encouragement through my studies.

List of Publications

- John Park, Su Nguyen, Mengjie Zhang, Mark Johnston. "Evolving Ensembles of Dispatching Rules using Genetic Programming for Job Shop Scheduling". *Proceedings of the 18th European Conference on Genetic Programming (EuroGP 2015). Lecture Notes in Computer Science.* **Vol. 9025.** Copenhagen, Denmark. 8–10 April 2015. pp. 92–104.

- John Park, Su Nguyen, Mark Johnston, Mengjie Zhang. "A Single Population Genetic Programming based Ensemble Learning Approach to Job Shop Scheduling". *Proceedings of 2015 Genetic and Evolutionary Computation Conference (GECCO Companion 2015).* Madrid, Spain. 11–15 July 2015. pp. 1451–1452.

- John Park, Yi Mei, Su Nguyen, Aaron Chen, Mark Johnston, Mengjie Zhang. "Genetic Programming based Hyper-heuristics to Dynamic Job Shop Scheduling: Cooperative Coevolutionary Approaches". *Proceedings of the 19th European Conference on Genetic Programming (EuroGP 2016). Lecture Notes in Computer Science.* **Vol. 9594.** Porto, Portugal. 30 March–1 April 2016. pp. 115–132.

- John Park, Yi Mei, Su Nguyen, Aaron Chen, Mark Johnston, Mengjie Zhang, "Niching Genetic Programming based Hyper-heuristic Approach to Dynamic Job Shop Scheduling: An Investigation into Distance Metrics". *Proceedings of the 2016 Genetic and Evolutionary Computation Conference (GECCO Companion 2016).* Denver, USA. 20–24 July 2016. pp. 109–110.

- John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Investigating the Generality of Genetic Programming based Hyper-heuristic Approach to Dynamic Job Shop Scheduling with Machine Breakdown". *Proceedings of the Third Australasian Conference on Artificial Life and Computational Intelligence (ACALCI 2017). Lecture Notes in Artificial Intelligence.* **Vol. 10142.** Geelong, Australia. 30 Jan–2 Feb 2017. pp. 301–313.

- John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "An Investigation of Ensemble Combination Schemes for Genetic Programming based Hyper-heuristic Approaches to Dynamic Job Shop Scheduling". *Applied Soft Computing*. **Vol. 63.** February 2018, pp. 72–86. DOI: 10.1016/j.asoc.2017.11.020.

- John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Investigating Machine Breakdown Genetic Programming Approach for Dynamic Job Shop Scheduling". *Proceedings of the 21st European Conference on Genetic Programming (EuroGP 2018). Lecture Notes in Computer Science.* Parma, Italy. 4–6 April 2018. pp. 253–270.

- John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Evolutionary Multitask Optimisation for Dynamic Job Shop Scheduling using Niched Genetic Programming". *Proceedings of the 31 Australasian Joint Conference on Artificial Intelligence (AI 2018). Lecture Notes in Computer Science.* Wellington, New Zealand. 11-14 December 2018. pp. 739–751.

- John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Multitask Genetic Programming for Dynamic Job Shop Scheduling Subject to Breakdowns". (in progress).

Contents

Chapter 1

Introduction

1.1 Problem Statement

Job Shop Scheduling (JSS) problems are a group of optimisation problems that have been studied extensively over the past 60 years [48, 128]. The problems have garnered attention from both academics and leading industry experts. From an academic perspective, JSS problems are difficult and computationally intensive to solve. Most JSS problems with fixed properties are NP-hard [16]. In the worst case scenario, the scaling of the computation times required to generate optimal solutions for JSS problem instances relative to the sizes of the problem instances is *super-polynomial* [125]. This results in an issue where large problem instances are too computationally intensive to generate optimal solutions using an exact optimisation technique. JSS problems are considered to be among the most difficult combinatorial optimisation problems [46]. On the other hand, JSS is a good model of many manufacturing scenarios ranging from semiconductor manufacturing to automobile assembly lines [125] from a manufacturing and production perspective. In the United States alone, manufacturers contribute trillions of dollars to the US economy [144].

A mathematical model of a JSS problem instance is as follows. A JSS problem instance usually has a fixed number of *machines* on the *shop floor*

that can be used to process arriving *jobs* [125]. A job has a predetermined sequence of *operations* which need to be processed in order for it to be completed. Each operation can only be processed on a specific machine. This means that a job is routed through a specific sequence of machines before it leaves the shop. However, a machine can only process one operation at a time. The goal of a JSS problem is to process all arriving jobs as effectively as possible based on the *objective* of the JSS problem. For example, an objective of a JSS problem can be to process all jobs so that the time spent by the jobs on the shop floor is minimised (i.e. minimise the flowtime [72, 73]), or to minimise the time when the last job in a batch is completed (i.e. minimisation the makespan [10, 141]). In addition, there are also JSS problems where the jobs are assigned *due dates* and *weights*, and the goal is to process the jobs based on their due date urgency [12, 72, 73, 146].

JSS problems are divided into two separate subsets of problems. The first subset of JSS problems are the *static* JSS problems [125]. In a static JSS problem instance, all properties of the jobs and the machines on the shop floor are known in advance. Approaches to static JSS problems can be broken down into two categories. In the first category, many papers have proposed *exact mathematical optimisation* techniques [1, 20, 25, 48, 125]. These techniques generate optimal solutions for JSS problem instances. An optimal solution for a JSS problem instance is feasible, i.e., satisfies all constraints of the problem instance, and has an equal or better objective function value than all other feasible solutions for the problem instance. For example, a JSS problem instance may have the objective of minimising makespan. An optimal solution to a makespan minimisation problem will have a makespan value where the makespan of all other feasible solutions will be equal to or higher than the makespan of that optimal solution. In the *worst case* scenario, the computational cost to generate an optimal solution for a JSS problem instance is superpolynomial [16], i.e., the search spaces for the problem instances scales very poorly to the size of the problem instances. Therefore, mathematical optimisation tech-

niques have been limited to very small problem instances. In one study in 2014, it took approximately 7.8 hours to find an optimal solution to a problem instance with 9 jobs and 5 machines [7]. For larger JSS problems, such as problem instances with up to 100 jobs and 20 machines [141], it is generally too computationally difficult to find optimal solutions to the problem instances using mathematical optimisation. In many real-world scenarios, although optimal solutions are preferred, they are often not necessary. Solutions that are "good enough" are sufficient to allow the manufacturer to have a competitive edge in the market. Because of this, it is sometimes more advantageous to trade-off computation time with the quality of solutions for JSS problem instances. This leads to the second category of approaches to static JSS problems, which are *heuristic* approaches. Heuristic approaches are "rules-of-thumb" which can generate "good", but not necessarily optimal, solutions [64]. In addition, for some heuristic approaches, there is also no guarantee that the solution will be feasible. The state-of-the-art approaches for static JSS problems with large problem instances use meta-heuristics. Meta-heuristics are higher level heuristics which provide a general framework to guide low-level heuristics that make local decisions [114]. Examples of meta-heuristics approaches to JSS include techniques such as Tabu Search [12] and Genetic Algorithm [158]. Specific meta-heuristic approaches [75] have been shown to handle large JSS problem instances very effectively.

The second subset of JSS problems are dynamic JSS (DJSS) problems [125]. Unlike static JSS problem instances, where the properties of jobs and machines are known *a priori*, unforeseen events occur in DJSS problem instances which affect the properties of the problem instance. In many DJSS problems, the job and the job's attributes are unknown before they arrive on the shop floor. Other DJSS problems can focus on unforeseen machine breakdowns [115], and changes in the job properties [115]. Developing approaches to DJSS problems are ways to address the issues bridging the gap between the theoretical approaches to JSS and the industrial applica-

tions of the various approaches [102]. In a real-world scenario, dynamic changes to the shop floor are unavoidable, and the rules that are applied in an industrial setting need to be robust to changes in the environment [103].

Approaches to solving DJSS problems differ from static JSS problems. Because DJSS problem instances have unforeseen events, it is not possible to determine whether a given schedule is optimal before processing occurs. Therefore, approaches such as mathematical optimisation techniques are not suitable for DJSS problem instances [140]. In addition, if the DJSS problem also contains stochastic elements, then planning for future events becomes even more difficult. Instead, many effective approaches to DJSS problems use heuristics that make local sequencing decisions, such as dispatching rules [17]. This is because dispatching rules have short reaction times and can cope with the unforeseen changes in the manufacturing environment [105]. When a machine is available and has jobs waiting at it to be processed, a dispatching rule determines which job should be selected to be processed next by the machine. For example, the SPT (shortest processing time) dispatching rule [125] selects the job with the shortest processing time waiting at the available machine. Dispatching rules generally do not attempt to predict and optimise the schedule in advance unlike exact optimisation and meta-heuristics, making them relatively simple compared to those approaches. Other approaches use a dispatching rule which combines multiple job attributes and machine attributes together to select a job. For example, SPT/FIFO combines the SPT heuristic with the FIFO (first-in-first-out) heuristic, which selects the job which arrived at the machine first. SPT/FIFO will first rank the jobs by the shortest processing time first, then by the arrival time at the machine, and select the top-ranked job. Dispatching rules which combine multiple attributes together are denoted as composite dispatching rules [72, 73]. A decision to select a specific job to be processed by the dispatching rule is denoted as a dispatching decision. Although dispatching rules are not

as effective as state-of-the-art meta-heuristics in static JSS problems [158], they are very effective for dynamic environments [105].

Although dispatching rules can be effective for specific problem domains, they are not guaranteed to be effective when applied to other problem domains. Heuristics are usually designed to be effective for a specific problem or class of problems [17], and no single heuristic is more effective than other heuristics for all JSS problems [150]. In a real-world scenario, the underlying properties of the problem are likely to change over time, e.g., machines are added or removed from the shop floor, or change in the objectives of the problems, and new technology allows certain machines to become more effective at processing jobs, etc. Therefore, heuristics need to be updated frequently for the manufacturer to keep up with other competitors. However, designing a new heuristic can be difficult, and requires human experts and extensive empirical testing to ensure that the new heuristic is effective. To circumvent the issue of manually designing dispatching rule heuristics, researchers have proposed the use of *hyper-heuristics* [21]. Hyper-heuristics are heuristics which generate heuristics that can generate solutions to DJSS problem instances instead of outputting the solutions directly. The heuristics generated are often dispatching rules [17], which can then be *reused* for further JSS problem instances in the problem domain. In essence, a new and effective heuristic can be developed autonomously for a specific problem domain using hyper-heuristics without the need for a human expert and a lengthy trial and error process. A significant number of hyper-heuristic approaches to JSS problems use Genetic Programming (GP) [80]. They are denoted as Genetic Programming based hyper-heuristic (GP-HH) approaches [17]. GP-HH approaches to JSS problems are popular compared to other hyper-heuristic approaches due to the representation of individuals in the tree-based GP populations intuitively being able to be interpreted as priority dispatching rules. GP also has a powerful search ability that can combine heuristic subcomponents together to form a cohesive rule effectively [103].

From the literature, GP-HH approaches generally outperform manually designed dispatching rules for both static and DJSS problems [17, 102], making it a very promising approach to investigate.

The overall goal of this thesis is to develop new *GP-HH* approaches to evolve effective dispatching rules for complex *dynamic job shop scheduling environments* with multiple types of dynamic events. The GP-HH approaches will aim to improve over the standard GP approach over the DJSS problems both in terms of the effectiveness of the evolved rules and the computation time required to evolve the rules. The focus will be on developing GP-HH approaches that can evolve *multiple heuristics* simultaneously so that the different rules can be combined to handle a DJSS problem more effectively, or so that the different rules can be applied to different sub-problems in a DJSS problem effectively.

1.2 Motivations

Dispatching rules are relatively simple heuristics in comparison to some meta-heuristics. However, dispatching rules can range widely in the complexity of the job selection procedure [125]. The most basic dispatching rules, such as FIFO, take a single attribute of the problem into account and uses it during the job selection procedure. Researchers have expanded on the idea of dispatching rules by combining multiple rules together to form more complex dispatching rules. Good combinations of rules have been shown to improve performance over individual dispatching rules for specific JSS problems [72, 73]. Because of this, most of the previous studies [40, 47, 60, 71] which utilise GP-HH focus on evolving single priority dispatching rules. In a priority dispatching rule, the attributes of jobs and machines are combined into an arithmetic function. The arithmetic function is then used to calculate the 'priorities' of jobs waiting at the machine. The job with the highest priority is then selected by the dispatching rule.

Although many effective GP approaches have been proposed for DJSS

problems with large number of dynamic job arrivals [17, 102, 103], only a limited number of GP approaches have been applied to DJSS problems with different types of dynamic events. For example, DJSS problem with *machine breakdowns* are an another type of dynamic event that are prominently investigated in the literature [115]. In a machine breakdown event, a specific machine becomes inactive for a period of time, and any job being processed at the time the machine becomes inactive needs to be resumed from the point where the operation was interrupted by the machine breakdown [61].

The most prominent approaches to DJSS problems with machine breakdowns are *predictive-reactive* or *robust pro-active* approaches [115]. Predictive-reactive approaches predicts the final schedule beforehand. They then reacts to machine breakdowns by updating the final schedule, while attempting to minimise changes which need to be made due to the disruption caused by the breakdown [152]. Robust pro-active approaches attempt to predict the machine breakdown and minimise the effect of the machine breakdown on the predicted schedule [94]. The approaches to DJSS problems with machine breakdowns focus on generating schedules to problem instances that are robust to the dynamic event [115]. The goal of many approaches that handle DJSS problems with machine breakdowns is to minimise the difference between the predicted schedule and final schedule after all jobs have been processed [8, 6]. However, the approaches to DJSS problems with machine breakdowns focus on problems with small problem instances (e.g. 20 total jobs on the shop floor [6]) compared to the GP approaches that evolve rules for DJSS problem with dynamic job arrivals (e.g. over 2500 job arrivals on the shop floor [66, 60]).

Limited number of approaches have been proposed handling large DJSS problems with both dynamic job arrivals and machine breakdowns simultaneously [61, 5, 4]. Handling DJSS problems with multiple types of dynamic events are important direction in the field of dynamic scheduling [102], as real-world manufacturing environments are likely to consist

of multiple unforeseeable events that the scheduling algorithm needs to cope with. For these types of problems, existing machine breakdown approaches generally are not suitable due to the large number of dynamic job arrivals. After an initial schedule has been predicted, rescheduling needs to occur frequently to accommodate for new job arrivals, which may likely be computationally infeasible for DJSS problems that are handled by dispatching rules. Adibi et al. [5, 4] have proposed a variable neighbourhood search (VNS) procedure to handle large DJSS problems with both dynamic job arrivals and machine breakdowns, but the computation time of VNS was not presented and VNS is compared to man-made dispatching rules which are outperformed by GP evolved rules [109]. However, very few GP approaches have been designed to handle machine breakdowns [155], and no GP-HH approach has been specifically designed to cope with both dynamic job arrivals and machine breakdowns in a DJSS problem.

A major limitation of single priority dispatching rules evolved by a standard GP-HH is that they only consider local decisions and are myopic in nature [18, 66]. In a DJSS problem instance, a scheduling algorithm needs to make multiple complex decisions to generate a schedule, and decisions made early in the schedule can significantly impact the overall quality of the schedule later down the line. It is likely having multiple different types of dynamic events add a significant challenge over the DJSS problems with one type of dynamic events, and a standard GP approach may not be general enough to handle the DJSS problem effectively. Therefore, an investigation into the generality of GP on the DJSS problem is required to determine whether a specialised GP approach is needed or, if the standard GP is sufficient to handle the DJSS problem.

Given that the standard GP is not sufficient for DJSS problems with dynamic job arrivals and machine breakdowns, then it is likely that we need to develop specialised approaches that improve on the existing GP-HH approaches in the literature to handle the DJSS problem. It may be possible to do this by considering methods that evolve *multiple rules* together

simultaneously. In problems outside of JSS, researchers have proposed effective *ensemble* approaches to better handle problems than single rules [126]. In classification, ensemble approaches can better represent decision boundaries than single classifiers [126]. The decision to label an instance with a class label in classification can be considered to have parallels to deciding which job is selected by the machine in scheduling, as the machine needs to be assigned a job to process. Therefore, a GP approach that can evolve ensembles for DJSS problem may be able to better handle DJSS problems than the standard GP approach that evolves single rules.

Another approach to handle the DJSS problem may be to incorporate methods of sharing domain knowledge in DJSS to improve the overall *generalisation* ability of GP. *Multitask learning* approaches have been applied effectively to classification problems in the literature [27, 116], and have also been applied to optimisation problems [35, 52] to improve the overall effective of evolutionary algorithms. In multitask learning, where multiple problem domains are handled simultaneously by a learning algorithm so that latent features learned from one problem domain can be transferred to other problem domains to improve the overall effectiveness of the learning algorithm on the domains [116]. A DJSS problem with dynamic job arrivals and machine breakdown is likely to be a complex problem, and handling the problem has multiple problem domains may result in more effective set of rules than handling it as a single problem. In other words, it may be more effective to evolve rules specific to the different problem domains in the DJSS problem, i.e., a *porfolio* of "specialist" rules, than to handle the problem using single generalised rules. Incorporating multitask learning with GP is novel and interesting area of research that have not yet been proposed in the literature. In addition, researchers have proposed evolutionary multitasking to various optimisation problems, but not to DJSS problems.

1.3 Research Goals

The overall goal of this thesis is to carry out an extensive investigation of **GP-HH** approaches to boost the effectiveness of GP on **dynamic job shop scheduling environments** in terms of the effectiveness of the evolved rules and the time required to evolve the rules. The focus of the thesis is to develop GP-HH approaches that evolve *multiple dispatching rules* simultaneously that can handle complex DJSS problems with different types of dynamic events. Evolving multiple dispatching rules to handle DJSS problems has an advantage over evolving single dispatching rules. Multiple dispatching rules can be combined to have better performance over single dispatching rules, or the multiple dispatching rules can be specialised to DJSS problem domains better than a single rule that needs to be effective over a diverse range of DJSS problem domains. In other words, the proposed GP-HH approaches have the potential to evolve *diverse* and *reusable* rules that can generalise more effectively over a diverse range of difficult DJSS problems compared to the existing state-of-the-art scheduling heuristics. This research will be broken down into the following key objectives.

1. Develop GP-HH approaches to DJSS problems that evolves high-quality *ensemble* rules effective on both the current problem domain and *unseen* problem domains. The GP evolved rules need to be able to generalise effectively on the problem domain that the rule is evolved on, and any further unseen problem domains that the rule may encounter in the future. However, information about these unseen domains is not always available in advance. Ensemble learning has been shown to improve the generalisation ability of algorithms in the literature when used to augment an existing approach [126]. Ensemble learning likely has the potential to also improve the quality of rules evolved by GP for the DJSS problems. The underlying principle behind ensemble learning is that multiple and diverse heuristics are developed in such a way that each

heuristic is covered by other heuristics in terms of their weaknesses, i.e., the heuristics cover for each other's errors [126]. This objective is broken down into the following sub-objectives. First, we need to investigate an ensemble algorithm that can be incorporated with GP to evolve effective ensembles of dispatching rules for large-scale DJSS problems with dynamic job arrivals handled by existing GP-HH approaches in the literature [59, 60, 124]. This is because applying ensemble GP approaches for JSS problem is a very new field of research, and only a few preliminary works have applied ensemble GP approaches to DJSS problems [42]. Many effective ensemble algorithms have been proposed in the literature for problems outside of JSS to address different research fields [126], but their effectiveness in JSS are relatively unknown. Second, we investigate the ensemble combination scheme used to combine the outputs of the ensemble members together, which is a key factor in the overall effectiveness of the ensembles [126]. Based on the effectiveness of the ensemble GP approaches over the DJSS problem, it may be possible to apply components of the ensemble GP to more complex DJSS problems, such as a DJSS problem with dynamic job arrivals and machine breakdowns.

2. Investigate the effectiveness of GP-HH approaches to DJSS problems subject to *machine breakdowns*. Note that GP-HH approaches to JSS with machine breakdown have not been covered significantly in the literature, and no GP-HH approaches have been proposed for DJSS problems with both dynamic job arrivals and machine breakdowns. In terms of man-made dispatching rules, there most effectiveness dispatching rules are based on the frequency of machine breakdowns [61]. Certain man-made dispatching rules perform better than other dispatching rules on problem instances with no machine breakdowns but perform worse on problem instances with a high frequency of machine breakdowns. In other words, there is a trade-off in the performances of the man-made dispatching rules depending on the frequency of machine breakdown.

On the other hand, the effectiveness of GP evolved rules have not been explored for DJSS problems with machine breakdowns. It is uncertain whether rules that are evolved for a DJSS problem domain with no machine breakdowns perform well on DJSS problem domains with machine breakdown, and whether GP that are evolved for multiple DJSS problem domains perform well on all domains. By investigating the efficacy of GP, we can determine which areas we can make improvements to the GP for the DJSS problem, or conclusively determine if GP can evolve rules that can cover for the entire problem effectively. In addition, analysing the structures and the behaviours of the rules evolved by the GP in the literature for different machine breakdown scenarios can provide insight into the differences in the properties of the rules. Afterwards, we will investigate which shop floor attributes are important when dealing with machine breakdowns and how they affect the quality of the rules evolved using GP-HH. It is likely that incorporating attributes specific to machine breakdown scenarios into the information for a GP-HH approach can improve the overall quality of the rules evolved for the problem. For example, by incorporating attributes such as the number of times a machine has failed, along with the average repair time of the machine, the evolved dispatching rule can potentially avoid processing jobs with long processing times just before a machine failure is likely to happen. Various manually designed features will be tested to improve the effectiveness of GP on the DJSS problem.

3. Develop *multitask* GP-HH approaches to DJSS problems that are able to evolve high-quality *portfolio* of rules for the DJSS problem with dynamic job arrivals and machine breakdowns. Multitask learning [27] is a subset of *transfer learning* [116] where multiple problem domains are solved simultaneously so that knowledge acquired from one problem domain can be shared to improve the machine learning algorithms's effectiveness. Multitask learning may also be able to better handle a complex DJSS problem by separating it out into smaller sub-problems

representing the multiple domains, and then solving the multiple domains as the tasks in a multitask learning algorithm simultaneously. By doing this, useful features learned from one task can be shared and used to boost the effectiveness of the GP approach over the overall DJSS problem [116]. Therefore, it may be possible to separate out the DJSS problems by the underlying properties of the problem domains, e.g., by their machine breakdown scenarios. In addition, it may be possible to evolve multiple "specialist rules" effective for the different problem domains as an alternative to evolving a single generalist rule over the entire problem. To incorporate multitasking to GP for DJSS problems, we will develop two different GP approaches. First, we will develop a *niched* [95, 133] GP approach that evolves a portfolio of rules for the DJSS problem. Niching techniques are used in the literature to promote diversity between the GP individuals in a single GP run [95]. When combined with GP as a multitask optimisation method for the DJSS problem, niching has the potential to aid GP in searching for the effective rules specialised for each type of problem instance (i.e. tasks). If a specific GP individual is effective for a particular task, then removing other individuals that behave similarly but have subpar performances may help improve the search for better individuals in the specific task. Second, we will propose a novel multitask GP approach to the DJSS problem with dynamic job arrivals and machine breakdowns which uses the neighbourhood relations between the different tasks in the DJSS problems to effectively allocate GP individuals to specific tasks in the DJSS problem. By using neighbourhood relations and focusing individuals on specific tasks, we can potentially achieve better efficiency over a standard GP approach without sacrificing the performance of the rules evolved by the multitask GP approach.

1.4 Major Contributions

This thesis makes the following major contributions.

1. This thesis presents ensemble GP approaches for a DJSS problem with dynamic job arrivals by incorporating two cooperative coevolutionary techniques into GP. The first ensemble GP approach uses Potter and De Jong's *cooperative coevolution* [127], which partitions a GP population into multiple subpopulations where the individuals in each subpopulations interact with other subpopulations through representatives. The second ensemble GP approach uses Wu and Banzhaf's *multilevel selection* [153], where individuals in the GP population are combined to form *groups*. The experimental results show that the cooperative coevolutionary GP and multilevel GP approaches can evolve higher quality rules for the DJSS problem than the standard single tree GP approach that evolves a single priority dispatching rule [17]. In addition, this thesis investigated multiple different types of ensemble combination schemes [126]. It showed that using linear combination is the most effective method of combining ensemble outputs. Finally, the distance measures proposed in this thesis compare the behaviours of the different ensembles to obtain insight into the relative performances of the ensemble combination schemes. The analysis shows that the linear combination also provides the best spread in the decisions made by the ensemble member than the other combination schemes for the DJSS problem. These distance measures are also useful not only in this studies but also for future ensemble GP approaches to DJSS problems in the future.

   Part of this contribution has been published in:

   John Park, Yi Mei, Su Nguyen, Aaron Chen, Mark Johnston and Mengjie Zhang. "Genetic Programming based Hyper-heuristics to Dynamic Job Shop Scheduling: Cooperative Coevolutionary Approaches". *Proceedings of the 19th European Conference on Genetic Programming*

(EuroGP 2016). Lecture Notes in Computer Science. **Vol. 9594.** Porto, Portugal. 30 March–1 April 2016. pp. 115–132.

John Park, Yi Mei, Aaron Chen and Mengjie Zhang. "Niching Genetic Programming based Hyper-heuristic Approach to Dynamic Job Shop Scheduling: An Investigation into Distance Metrics". *Proceedings of Genetic and Evolutionary Computation Conference Companion (GECCO 2016 Companion)*. Denver, Colorado, USA. 20–24 July 2016. pp. 109–110.

John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "An Investigation of Ensemble Combination Schemes for Genetic Programming based Hyper-heuristic Approaches to Dynamic Job Shop Scheduling". *Applied Soft Computing*. **Vol. 63.** February 2018, pp. 72–86. DOI: 10.1016/j.asoc.2017.11.020.

2. This thesis provides the first study into investigating the efficacy of GP-HH to DJSS problems with both dynamic job arrivals and machine breakdowns. This investigation showed that the standard GP approach is insufficient to cover all machine breakdown scenarios, and the machine breakdown GP terminals only improve the quality of the GP approach by a small amount. The analysis shows that the rules evolved by the standard GP approach and the GP approach that incorporates machine breakdown specific information behave very similarly to each other. In addition, the preferred choices made by the best rules when machine breakdowns do occur in most decision situations are to select the jobs with the shortest processing times.

   Part of this contribution has been published in:

   John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Investigation the Generality of Genetic Programming based Hyper-heuristic Approach to Dynamic Job Shop Scheduling with Machine Breakdown". *Proceedings of the Third Australasian Conference on Arti-*

ficial Life and Computational Intelligence (ACALCI 2017). Lecture Notes in Artificial Intelligence. Geelong, Australia. 30 January – 2 February 201. pp. 301–313.

John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Investigating Machine Breakdown Genetic Programming Approach for Dynamic Job Shop Scheduling". *Proceedings of the 21st European Conference on Genetic Programming (EuroGP 20188). Lecture Notes in Computer Science.* Parma, Italy. 4–6 April 2018. pp. 253–270.

3. This thesis develops two novel multitask GP approaches for the DJSS problem with both dynamic job arrivals and machine breakdowns. The first multitask GP approach use niching [95] to improve the overall effectiveness of the rules on the DJSS problem. The second multitask GP approach uses a novel neighbourhood-based relation between the different tasks in a DJSS problem to allocate the GP individuals to specific tasks to improve both the efficiency of the evaluation procedure and the effectiveness of the evolved rules. The multitask GP approaches evolve a *portfolio* of "specialist rules" that can handle different machine breakdown scenarios individually. The new approaches outperform the standard GP approach and provide a promising alternative to handling the DJSS problem through evolving single rule that covers all machine breakdown scenarios.

   Part of this contribution has been published in:

   John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Evolutionary Multitask Optimisation for Dynamic Job Shop Scheduling using Niched Genetic Programming". *Proceedings of the 31 Australasian Joint Conference on Artificial Intelligence (AI 2018). Lecture Notes in Computer Science.* Wellington, New Zealand. 11–14 December 2018. pp. 739–751.

   In addition, part of this contribution is currently being adapted to a journal paper:

John Park, Yi Mei, Su Nguyen, Gang Chen and Mengjie Zhang. "Multitask Genetic Programming for Dynamic Job Shop Scheduling Subject to Breakdowns". (in progress).

1.5 Organisation of Thesis

The remainder of this thesis is organised as follows. Chapter 2 provides the problem definition and literature review. Chapters 3, 4, and 5 covers the research that has been carried out to fulfil the first, second and third research objectives respectively. Chapter 6 concludes the thesis.

Chapter 2 provides a detailed description of JSS, which includes the descriptions of the primary DJSS problems that are covered in the thesis. In addition, the chapter covers the relevant research that has been carried out for JSS. This ranges from simple heuristic approaches such as dispatching rules to existing GP approaches that have been applied to JSS problems. In addition, research relevant to the research goals but have mainly been focused on problems outside of JSS are also included in the chapter, such as ensemble learning and multitask learning. Afterwards, the chapter gives a review of the current research into evolutionary scheduling approaches to the DJSS problems.

Chapter 3 covers the three proposed ensemble GP approaches that have been investigated over the course of the research. The cooperative coevolution, niching and multilevel GP approaches are evaluated and the evolved rules behaviours are analysed to show how evolving ensembles can be more effective than evolving single rules. In addition, new analysis measures are proposed, which show the relations between the ensemble members in the evolved rules.

Chapter 4 shows the investigation into DJSS problem with dynamic job arrivals and machine breakdowns which is broken down into two parts. The first part tests the efficacy of GP over the DJSS problem with dynamic job arrivals and machine breakdowns over the different machine break-

down scenarios and shows that the attributes that make up the evolved rules differ between the machine breakdown scenarios. The second part proposes new GP terminals that incorporate machine breakdown information, evaluates the GP rules evolved using the new GP terminals. Analysis of the machine breakdown GP rules shows that the preferred choices made by the GP rules are similar to a simple dispatching rule.

Chapter 5 details the two new GP approaches to DJSS problem that incorporates multitask optimisation. The two multitask approach evolve a portfolio of rules that can handle multiple machine breakdown scenarios simultaneously. The two multitask GP approaches outperform the standard GP approach over the DJSS problem. Extensive analyses of the evolved rules are carried out that identifies the relations between the specialised rules evolved for different machine breakdown scenarios. The analyses also provide connections between the new findings made by the multitask GP approaches to the existing findings made in the literature.

Chapter 6 is the concluding chapter of the thesis by summarising the key findings made in the thesis. The major findings made by the thesis are highlighted. In addition, the opportunities for future work to expand on the research carried out in the thesis are also discussed in the chapter.

Chapter 2

Literature Review

This chapter provides a review of the background literature on the topics related to the research carried out in the thesis. In addition, discussion of the existing research is carried out to support the motivations of the thesis. First, this chapter covers the basic concepts that are relevant to the research carried out in this thesis, job shop scheduling (JSS) problem definitions and an overview of genetic programming (GP). Afterwards, this chapter covers the general related work to scheduling problems (which includes JSS problems), GP based hyper-heuristic (GP-HH) approaches to scheduling problems and auxiliary-related work that are relevant to the specific research goals.

2.1 Basic Concepts

This section covers the basic concepts, which are important fundamentals to the related work to scheduling problems that will be covered in this chapter. First, we describe machine learning, sequencing and scheduling. Afterwards, we cover the differences between heuristics, meta-heuristics and hyper-heuristics. Finally, the definitions for the evolutionary computation (EC), ensemble learning, multi-agent systems, transfer learning and multitask learning are provided.

2.1.1 Machine Learning

Machine learning is an area of artificial intelligence [9], that is described as a field of study that attempts to create a "good and useful approximation" of an observed set of data[9]. Machine learning can also be described as a field of study that gives the computers the ability to learn without being explicitly programmed [9] In other words, machine learning techniques are non-domain specific techniques that attempts to capture the patterns that are present within a set of data. For example, a neural network (NN) is a machine learning technique that can be applied to a wide range of problems, including classification [157] and regression [45]. From the learned model, the researcher can obtain useful insights into the observed data, or use the learned model to make predictions on unseen data.

Machine learning techniques are categorised into three major categories [132]:

1. **Supervised learning**: The observed data, that is handled by the machine learning technique, are *labelled* with explicit values. This means that there are direct input-output pairs, and the goal of machine learning in supervised learning environments is to create a mapping between the inputs and the outputs which provide the best approximation to the data as possible [132]. Examples of problems which are handled by supervised machine learning algorithms are classification and regression [132].

2. **Unsupervised learning**: The observed data handled by the machine learning technique is *unlabelled*. This means that the machine learning algorithms construct a model from only the inputs [132]. An example of problems which are handled by unsupervised machine learning algorithms is clustering [132]. In addition, a machine learning algorithm that carries out both supervised and unsupervised learning for a set of data is used called *semi-supervised learning*.

3. **Reinforcement learning**: The machine learning algorithm learns from a series of rewards and penalties that are provided by the training instances [132]. Based on the series of reinforcements provided by the problem, the reinforcement learning algorithm makes adjustments to the decision-making process, so that it will maximise the rewards and minimise the penalties. The types of problems that reinforcement learning are often applied to are multi-step problems that have an end goal state that needs to be reached by the reinforcement learning agent, e.g., playing a chess game [132].

2.1.2 Sequencing and Scheduling

In a scheduling problem, there are limited resources that carry out specific activities that need to be utilised as effectively as possible [86]. In particular, production scheduling is a type of scheduling problem where the activities that are involved with the scheduling are related to manufacturing (e.g. car or pharmaceutical manufacturing) [125]. JSS is a type of production scheduling problem, where the machines are the limited resources that are used to process arriving jobs. JSS will be discussed in further detail below. Other types of scheduling problems include flow shop scheduling problems, open shop scheduling problems and flexible scheduling problems [125].

To handle a scheduling problem, the algorithm needs to carry out multiple *scheduling decisions*. A scheduling decision requires the algorithm to determine what jobs are considered, what job is selected out of the considered jobs, and the time when the machine begins processing the selected job's operations. The decision for selecting the jobs out of the list of considered jobs is called a *sequencing decision* [125]. Often times the two terms are used interchangeably, particularly when the time set by a scheduling decision is predictable. This will be discussed further in the section below that provides a more detailed description of JSS, along with further

information on other types of decisions that are made in a JSS problem.

2.1.3 Heuristics and Meta-heuristics

There are a number of definitions for heuristics and meta-heuristics. For this thesis, we provide the following definitions for the two terms as follows. On the other hand, the definition of hyper-heuristic is covered in full detail below in Section 2.4.

- **Heuristics** are 'rules-of-thumb' algorithms which can be applied to optimisation problems directly without adaptation [132]. For a particular problem, they often incorporate existing knowledge about the problem to find good solutions for the problem instances [17]. This is useful in cases where attempting to solve computational problems exactly is impractical, such as exact methods taking too much time [98]. This occurs frequently for a number of JSS problems that are encountered in the literature, as the time required to find optimal solutions for JSS problem instances scale poorly with the number of jobs and machines [7, 141]. *Dispatching rules* are examples of heuristics that are used to make scheduling decisions for JSS problems. Although heuristics have no guarantees of optimality in most problems they are applied to, well-designed heuristic approaches can often find "good-enough" or near-optimal solutions for the problem instances they are applied to. Effective dispatching rules approaches have been proposed in the literature for scheduling problems that can generate near-optimal schedules for the problem instances within a fraction of the time required by optimisation techniques that generate optimal solutions [125].

  Although dispatching rules can be effective for various scheduling problems, a major limitation of dispatching rules and other heuristic approaches is that they are problem or domain specific [149]. This means that a dispatching rule designed for one scheduling problem

have no guarantee that they are effective for other scheduling problems. This means that there is often a trade-off in the performances of dispatching rules based on the properties of the scheduling problem. One dispatching rule may be significantly more effective on a specific JSS problem domain but is significantly worse on another JSS problem domain [61, 125].

- **Meta-heuristics** are more complex algorithms than heuristics that are designed to handle difficult optimisation problems. Like heuristic approaches, meta-heuristics have no guarantee they can find optimal solutions. However, meta-heuristics are *problem independent* approaches, unlike heuristic approaches. Meta-heuristics usually make very few assumptions, if any, about the problems they are applied to [114]. In a broad sense, meta-heuristic approaches explore the solution space of the problem instance to iteratively improve on the solution. On the other hand, most heuristic approaches generate solutions for the problem instance based on a fixed assumption and usually do not include mechanisms that can iteratively search for better solutions.

  A meta-heuristic can explore the solution space in two main methods. First, a local search based meta-heuristic moves a single solution point around in the solution space. Examples of a local search based meta-heuristics are tabu search [49] and simulated annealing [79]. Second, a population-based meta-heuristic moves multiple solution points around in the solution space, where the multiple solution points eventually converge towards the best solution found. Examples of population-based meta-heuristics are particle swarm optimisation (PSO) [77] and genetic algorithm (GA) [37, 81]. In addition, a meta-heuristic approach can combine both local search and population-based search together to form hybrids approaches, e.g., combining GA and tabu search together to handle a JSS problem [93].

They can also incorporate low-level heuristics with domain knowledge for the specific problem to improve the overall effectiveness of a meta-heuristic approach [114]. Specific meta-heuristic approaches are discussed further in the related works in regards to JSS problems specifically in Section 2.5.

2.1.4 Evolutionary Computation

Evolutionary computation (EC) is a sub-field of artificial intelligence which focuses on algorithms inspired by various aspects of biology. This includes nature-inspired algorithms or population-based systems to deal with various problems. The two main categories of EC are evolutionary algorithms [37] and swarm intelligence [78].

Evolutionary Algorithms

Evolutionary algorithms (EAs) are a subset of EC that are influenced by the idea of Darwinian evolution and the science of genetics [37]. An EA consists of a population of individuals representing a biological species in nature. The individuals are first evaluated on a set of problem instances to determine their *fitness*, which is a measure of the effectiveness of the individuals. The fitness values of the individuals affect their chances of passing on the genetic material to the individuals that are bred for the next generation. In other words, the population undergoes evolutionary pressure to handle the problem more effectively. The procedure of breeding differs between the evolutionary algorithms. For example, GA [37, 81] and GP [80] are two EAs which use genetic operators such as crossover and mutation to breed the individual for the next generation. Crossover and mutation operators for GP, along with the overview of GP, are discussed in further detail in Section 2.3 below. On the other hand, estimation of distribution algorithms (EDAs) [84] generates a probabilistic model from the individuals that have the best fitness values. Afterwards, the

new individuals for the next generation are generated by sampling from the probabilistic model. Effective EA approaches have been proposed for scheduling problems in the literature as both meta-heuristics [29, 115] and hyper-heuristics [17, 102, 103].

Swarm Intelligence

Swarm intelligence (SI) focuses on the idea of competition and cooperation between the individual members of a swarm. A swarm is an abstract representation of a group of biological organisms that interact with each other to reach a certain goal, e.g., finding food. For example, particle swarm optimisation (PSO) [77] is a swarm intelligence technique where the particles represent a group of organisms such as a flock of bird. The movement of an individual in a PSO algorithm in the search space is influenced in the direction of the best solution found by the individual (called the local optimum) and the direction of the best solution found by the population (called the global optimum). The feedback between the individuals in PSO allows them to better work towards solving the problem. Other examples of SI techniques are ant colony optimisation [41], and artificial bee colony optimisation [76].

2.1.5 Cooperative Coevolution

Coevolutionary algorithms (CEA) are techniques where multi-agent behaviours are incorporated with evolutionary computation techniques [117]. One of the aims of developing CEA approaches is to be able to handle a complex problem more effectively by breaking it down to constituent sub-problems. This allows different individuals of the population to fill in different 'ecological niches' [127], i.e., specifically handle the different sub-problems. The evolved individuals are then combined together to a cohesive solver.

A popular CEA approach is cooperative coevolution [127], where a

population of individuals is partitioned into subpopulations, and individuals from each subpopulation "collaborate" with the representatives of other subpopulations. However, the collaboration between an individual from a subpopulations is limited to the *representatives* from the other subpopulations. A representative is selected randomly when the EA process is initialised but is generally the individual which has the best fitness out of the other individuals in the subpopulation after the first evaluation procedure [127]. One example which uses cooperative coevolution is a GP approach by Nguyen et al. [104], where it is used to evolve multiple scheduling policies to handle dynamic JSS problem with three objectives. Another CEA approach is orthogonal evolution of teams (OET) [131, 136], where the individuals are grouped into separate 'teams', but compete with members of other teams for the selection procedure. In addition, the teams are evaluated, and compete with each other, where unfit teams are removed, and new teams are generated using crossover and mutation between the leftover teams.

2.1.6 Ensemble Learning and Multi-agent Systems

Ensemble learning and *multi-agent systems* consist of a group of machine learning techniques where a group of diverse learners are trained and then combined together to form a system. Ensemble learning is often more associated with problems where a single task needs to be completed for a problem instance, e.g., classification problems [126], whereas multi-agent systems are associated with both single and multi-task problems [117]. The common phrase that "two heads are better than one" is one of the main motivations for studying ensemble learning and multi-agent systems. If there is a method of solving a problem perfectly using a single system, then ensemble learning and multi-agent systems would not be required. However, most problems are often too difficult to be solved perfectly by a single system. By separating out the multiple components to a

system, the risk of making a particularly poor decision is mitigated [126]. The idea of multiple experts that work in tandem to solve a problem has been shown to be very effective in classification [19, 45].

A key component of ensemble learning is diversity between the constituent subcomponents [126]. Without diversity, the subcomponents cannot sufficiently cover for each other's errors, and can potentially result in a bad decision. Therefore, a number of different approaches have been proposed for diversifying the subcomponents of an ensemble [15, 19, 28, 45, 89]. One notable approach of constructing a diverse set of classifiers for classification problems simultaneously is negative correlation learning (NCL) [15, 28, 89]. In NCL, an individual is penalised for misclassifying an instance if the ensemble has also misclassified the instance as well. This means that constituent classifiers do not need to classify all instances as correctly as possible, but only the instances which the other classifiers have also misclassified, allowing the subcomponent to specialise in a particular domain of instances.

2.1.7 Transfer Learning and Multitask Learning

Transfer learning [116] is a field of machine learning where knowledge acquired from one problem domain (called the source domain) is applied to another problem domain (called the target domain). In most standard applications of machine learning, the source domain is representative and the target domain, i.e., they are the same as each other. However, there are often cases where the source domain is not the same as the target domain. Models trained on the source domain do not have any guarantees that they are effective on the target domain, and may need to be rebuilt to be effective for the target domain [116]. However, in many real-world scenarios, rebuilding the model from scratch for the target domain may be impractical due to the lack of training data and the computation time required to rebuild the model [116]. Therefore, a major motivation behind trans-

fer learning is to develop machine learning approaches that are effective on the source domains that they are trained on but can also be effectively *reused* for target domains that are different from the source domains [116].

Multitask learning [27, 116] is a type of transfer learning that have been incorporated with machine learning and applied to different problem domains effectively in the literature. In multitask learning, the machine learning technique is applied to multiple different *tasks* simultaneously for a specific problem using multiple agents. A task can be subsets of problem domains or a separate problem altogether [27]. As an agent learns knowledge from a particular task, it confers *inductive bias* from the extracted knowledge from the task to agents that are being applied to other tasks. Multitask learning is motivated by the assumption that the different related tasks likely share latent features that are useful to each other [116]. Because of this, transferring knowledge from one task to another can likely improve the overall effectiveness of the machine learning algorithm for the problem, i.e., improve the generalisation ability of the machine learning algorithm. However, it is also possible that multitask learning can decrease the generalisation ability of the learning algorithm, as different tasks that are solved simultaneously using multitask learning may not share latent features and the inductive bias from Multitask learning has been applied to a various classification problems [11, 27, 44], but only a limited number of multitask learning approaches use EC to handle optimisation problems. They will be discussed further below in Section 2.7.

2.2 Job Shop Scheduling Problems

This section covers the definitions and mathematical notations for JSS problems and JSS objective functions. Afterwards, active and non-delay scheduling is defined. In addition, various techniques from the literature for JSS, ranging from exact mathematical optimisation to meta-heuristics, are detailed. Finally, we cover the different types of dynamic events that occur

in dynamic JSS (DJSS) problems, which includes dynamic job arrivals and machine breakdowns.

2.2.1 Job Shop Scheduling Definitions

The notation used for JSS problem instances is as follows. In a problem instance, there are M machines on the shop floor, and N jobs arrive on the shop floor. Each job j that arrives at the shop floor has a sequence of N_j operations $o_{1j}, \ldots, o_{N_j j}$ which needs to be processed in order for the job to be completed. Each operation o_{ij} for a job j has a processing time $p(o_{ij})$ (shortened to p_{ij}) for the duration it needs to be processed for at the machine without interruption. On the other hand, if a job can be interrupted, then we denote the problem as allowing *job preemption*. Operation o_{ij} cannot be processed until operations $(o_{1j}, \ldots, o_{(i-1)j})$ have been processed. Each operation o_{ij} needs to be processed on a specific machine $m(o_{ij})$. However, if all jobs follow exactly the same sequence of machines, then the problem is called a *flow shop* scheduling problem instead of a JSS problem. In addition, in a flexible JSS problem, an operation can have a subset of machines which can process it. If a problem has no re-entry, then no two operations for a job enter the same machine. Therefore, such problems have a maximum number of operations for a job j at $N_j = M$. The time when an operation o_{ij} for a job j is ready to be processed on a machine is denoted as operation ready time or job arrival time $r(o_{ij})$, where the ready time of the first operation o_{1j}, $r(o_{1j})$, is called the release time of job j, and is often abbreviated to r_j. In addition, the time when all operations $o_{1j}, \ldots, o_{N_j j}$ for a job j finish processing is denoted as the *completion time* of job j, C_j. With these attributes, we can define *makespan* as $C_{\max} = \max_{j=\{1,\ldots,N\}} C_j$ [125], and define the makespan minimisation objective, which is given by Equation (2.1).

$$\min \qquad C_{\max} \tag{2.1}$$

Other objectives that are studied extensively in the literature are related to flowtimes of jobs. The flowtime of a job j is defined as the time spent by j on the shop floor, i.e., the difference between the completion time C_j and release time r_j [125]. From this, we get the mean flowtime minimisation objective as shown in Equation (2.2).

$$\min \qquad \frac{1}{N}\sum_{j=1}^{N}(C_j - r_j) \tag{2.2}$$

An additional attribute which may be present in JSS problems is the *due date* d_j of a job j. From the due date of a job j, tardiness T_j is given by the positive difference between the completion time C_j and the due date d_j, i.e., $T_j = \max\{C_j - d_j, 0\}$. From the due date of a job j, we can also define slack as the difference between the due date d_j and the remaining processing time $\sum_{i=k}^{N_j} p(o_{ij})$, where operation o_{kj} denotes the operation that job j is currently up to, and operations $o_{1j}, \ldots, o_{k-1j}$ have been processed by machines. Tardiness related objectives in JSS problems are a representation of just-in-time manufacturing environments [128], where the individual orders need to be completed by their scheduled deadline to minimise customer dissatisfaction. Tardiness related objectives that have been investigated for JSS are maximum tardiness minimisation ($\max_{j=\{1,\ldots,N\}} T_j$), mean tardiness ($\frac{1}{N}\sum_{j=1}^{N} T_j$) and percentage of tardy jobs ($\%T$), where $\%T$ denotes the number of tardy jobs out of the N jobs that arrive on the shop floor) [125].

Weighted tardiness related objectives have also been investigated in the literature. In weighted tardiness related objective, a job j has a weight/penalty factor w_j that needs to be taken into account when minimising the tardiness related objective. Examples of weighted tardiness related objectives that have been investigated significantly in the literature are total weighted tardiness (TWT) and mean weighted tardiness (MWT), which are given by Equations (2.3) and (2.4) respectively [125].

$$\min \quad \sum_{j=1}^{N} w_j T_j \tag{2.3}$$

$$\min \quad \frac{1}{N} \sum_{j=1}^{N} w_j T_j \tag{2.4}$$

JSS problems are a subset of scheduling problems, where general scheduling problems involve the processing of jobs at machines. A standard for classifying scheduling problem was initially proposed by Conway et al. [33] and then refined by Lawler et al. [85], which is to use the triplet $\beta|\gamma|\delta$. In this notation, β denotes the machine environments, such as whether it is job shop scheduling (Jm), flow shop scheduling (Fm), flexible job shop scheduling (FJm), etc., and m denotes the number of machines on the shop floor. γ denotes specific job properties. For example, specific job properties can be that all jobs have the same processing time ($p_j = p$), all jobs have same due dates ($d_j = d$), or preemptive jobs are allowed. δ denotes the objective function for the problem, e.g., minimising makespan ($C_{\max}$), minimising TWT ($\sum_j w_j T_j$). Using this notation, a JSS problem with TWT minimisation is denoted as $Jm||\sum_j w_j T_j$.

2.2.2 Active Schedules and Non-delay Schedules

When describing scheduling algorithms for JSS problems, key definitions which need to be covered are *decision situation*, *active* and *non-delay* scheduling [125]. A decision situation is a situation where a scheduling algorithm needs to make a scheduling decision on what job will be processed by a machine m^* after it finishes processing a job and is available [59]. After machine m^* finishes processing a job at time R, it waits for the minimum amount of time possible in a *non-delay* schedule before it starts processing the next job [125, 137]. In other words, if there are jobs waiting at machine m^*, then it will begin processing one of the waiting jobs according to the scheduling algorithm. Otherwise, if there are no jobs waiting at the

machine, it will wait until at least one job arrives at the machine, and select out of the jobs which arrived the earliest to process. This means that as long as there are jobs which have arrived at a machine m^* when it becomes available, any additional jobs which will arrive in the future are not considered for selection by the non-delay scheduling algorithm to be processed next by machine m^* at that decision point.

On the other hand, when a machine m^* finishes processing a job in an *active* schedule, a job which arrives in the future can be processed next by machine m^* instead of a job which is already waiting at the machine [125]. Suppose that a set of jobs $A = \{j_1, \ldots, j_{N_{m^*}}\}$ needs to be processed at machine m^* for the operations $o_{j_1}, \ldots, o_{j_{m^*}}$, where we know the arrival times at machine m^* as $r(o_{j_1}), \ldots, r(o_{j_{m^*}})$. Jobs with uncertain arrival times are not considered by the active scheduling algorithm. Then an active scheduling algorithm first calculates the expected completion times $C'_{j_1}, \ldots, C'_{j_{m^*}}$ for the operations if they were to be processed by the machine, and finds the earliest expected completion time $C'_{\text{earliest}} = \min\{C'_{j_1}, \ldots, C'_{j_{m^*}}\}$, along with the arrival time r_{earliest} of the job with the earliest expected completion time. Then depending on the non-delay factor $\alpha \in [0, 1]$, the set of jobs which are considered by the active scheduling algorithm, A', is shown in Equation (2.5) [105].

$$A' = \{j \in A \mid r(o_j) \leq \alpha(C'_{\text{earliest}} - r_{\text{earliest}}) + r_{\text{earliest}}\} \tag{2.5}$$

The non-delay factor α determines how many future jobs to take into account. If a scheduling algorithm has $\alpha = 0$, then the machine will immediately begin processing some operation as soon as there are jobs at the machine, i.e., the schedule will be non-delay and A' will only consist of the jobs waiting at the machine when a machine becomes available. On the other hand, if $\alpha = 1$, then the scheduling algorithm generates an active schedule, and considers jobs arriving up to the earliest completion time of the jobs which arrive at the machine. If the non-delay factor is strictly between 0 and 1, then the scheduling algorithm is considered a *hybrid* of both

active and non-delay schedules. The non-delay factor does not go beyond 1, as the optimal solution to a JSS problem instance belongs to the set of solutions which are active schedules [48]. The reason why the non-delay factor does not go beyond 1 is that selecting any job that is not considered when $\alpha = 1$ but is considered when $\alpha > 1$ results in a sub-optimal schedule. If a job j is only considered when $\alpha > 1$, then job j arrives on the shop floor after the time C'_{earliest}. This means that if the machine waits for job j, then the job with the earliest completion time could have been processed before job j's arrival. This would mean that there is a schedule that is at least as good, if not better, than the solution generated by selecting job j to be processed. Finally, non-delay scheduling algorithms have no guarantees that they will be able to generate the optimal solution for a JSS problem instance [17].

An example of non-delay and active scheduling for a JSS problem instance with the makespan minimisation objective shown in Figure 2.1 respectively. The properties of jobs in the JSS problem instance are given in Table 2.1. In this problem, there are $N = 3$ jobs and $M = 2$ machines, denoted as m_1 and m_2. The SPT dispatching rule will generate different solutions depending on whether it is active or non-delay. The non-delay schedule (i.e. SPT algorithm with $\alpha = 0$) that is generated is given by Figure 2.1a, and the active schedule with $\alpha = 1$ that is generated is given by Figure 2.1b. In the non-delay schedule, we can see that machine m_1 begins processing j_3 as soon as j_1 has been completed. On the other hand, in the active schedule, machine m_1 processes j_2 before j_3. This is because j_2's second operation has a shorter processing time than j_3's first operation and j_2 arrives at machine m_1 before the expected completion time of j_3. Although non-delay SPT has no guarantee of optimality, for this specific problem instance we can see that the non-delay SPT generates a better solution than active SPT.

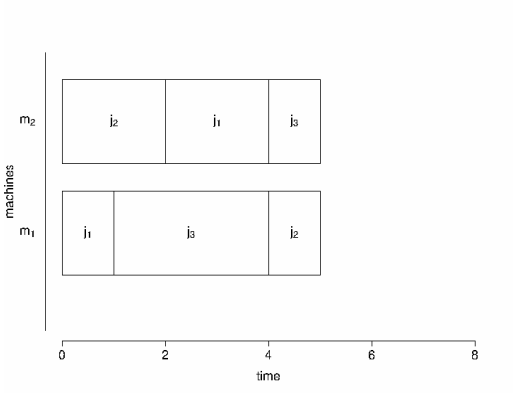
Table 2.1: A static JSS problem instance with $N = 3$ jobs and $M = 2$ machines and makespan minimisation objective.

job	processing order	processing time
j_1	m_1, m_2	1, 2
j_2	m_2, m_1	2, 1
j_3	m_1, m_2	3, 1

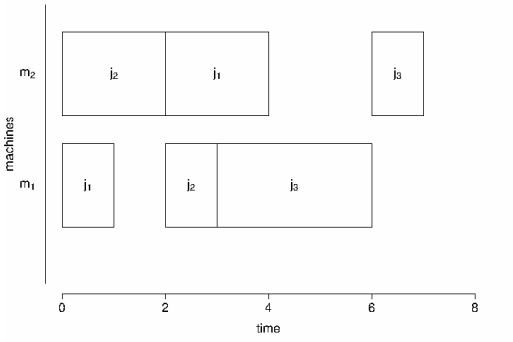
2.2.3 Types of Dynamic Events

As summarised by Ouelhadj and Petrovic [115], the types of dynamic events that can occur in dynamic JSS (DJSS) problems can be categorised into two major parts. The first type of dynamic events is *resource-related*. This means that the dynamic event affects the resources that are present in the dynamic scheduling problems. For DJSS, the resources are the machines that process the jobs. Therefore, examples of resource-related dynamic events are machine breakdowns, machine degradation and maintenance, change in the setup times to prepare for a specific job, and such. In this thesis, we focus on the resource-related dynamic event where machines break down during processing. Machine breakdown events are important dynamic events that can occur in real-world manufacturing scenarios [92]. McKay describes that in real-world scenarios, machines in manufacturing systems are sensitive to various factors, such as temperature and humidity. This affects the productivity of machines and adds in the possibility of machine failure due to extreme conditions. This complex scenario is often simplified in the literature, ranging from machine failures at some fixed point in time [152] to failures occurring stochastically [94].

For this thesis, the machine breakdown events follow Holthaus's [61] model of machine breakdown events used to evaluate man-made dispatching rules. A machine breakdown event immediately results in the machine being unavailable when the breakdown occurs, and the machine undergoes repair to fix the machine. In addition, the DJSS problem handled in



(a) The schedule generated by non-delay SPT



(b) The schedule generated by active SPT ($\alpha = 1$)

Figure 2.1: The schedule generated when SPT is applied to JSS problem instance in Table 2.1

this thesis does not allow preemption [125]. Preemption means that a job cannot leave a machine once the machine started processing the job's operations until the operation has been completed [125]. If a job's operation was being processed at the machine when the machine breakdown occurred in a DJSS problem without preemption, then the job is interrupted and resumed after the machine breakdown [61]. Any work already done for the job's operation is retained after the breakdown, meaning that the

work remaining on the job is the same before and after the machine breakdown. Therefore, the "actual" processing time of the job's operation o_{ij} that is interrupted by the machine breakdown is the expected processing time of the operation p_{ij} plus the time required to repair the machine.

The second type of dynamic events described by Ouelhadj and Petrovic [115] are *job-related*. Examples of dynamic events that affect jobs are either related to the change in the jobs' properties (e.g. the processing times of an operation is uncertain until the operation has been processed or due date of jobs suddenly change), or the number jobs expected on the shop floor change during processing (i.e. new jobs arrive or jobs are cancelled unexpectedly). This thesis focus on dynamic job arrivals [115], where new jobs arrive on the shop floor during processing that needs to be scheduled appropriately. The properties of an arriving job j, such as the processing times of operations, due dates, etc., are unknown until the job arrives on the shop floor at the release time r_j. In addition, the total number of jobs N that arrive on the shop floor is also unknown for DJSS problems with dynamic job arrivals, whereas they are known in advance for static JSS problems. Because of this, certain objectives are not suitable for these type of DJSS problems. For example, makespan minimisation objective requires all operations of all jobs to be completed as soon as possible (Equation (2.1). However, in a DJSS problem instance, there is always the possibility that a new job arrives sometime in the near future after the jobs have been completed. There is no indication whether all jobs have been completed or not in a DJSS problem with dynamic job arrivals. Therefore, many approaches in the literature that handle DJSS problems with dynamic job arrivals minimise flowtime related objectives or tardiness related objectives [59, 60, 67, 105, 124]. Given that the problem is *stable* [125], i.e., the number of waiting jobs do not increase monotonically with the number of job arrivals in the problem instance, there are tardiness and flowtime related objectives that are independent of the number of job arrivals. Examples include mean tardiness, maximum tardiness and percentage tardy, which

are discussed above in the problem definition. For this thesis, we handle tardiness related objectives, and the specific objective will be discussed in each contribution chapter.

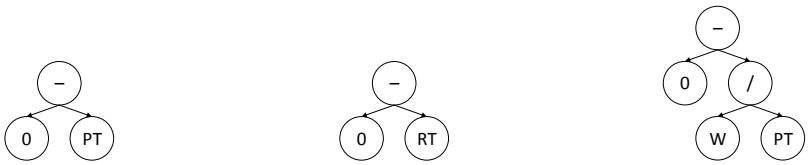
2.3 Genetic Programming

Genetic programming (GP) [80] is an evolutionary computation technique which has been applied to numerous problems, including JSS. In a GP system, there is a population of individuals which compete with each other for survival. The most 'fit' individuals have a chance of surviving to the next generation and have a chance to breed with other individuals to exchange genetic properties. The genetic operations that can be applied to surviving individuals are crossover and mutation, which will be covered in more detail below.

2.3.1 Representation

Individuals in a GP population are automatically generated programs of *variable length* which can be applied to a problem instance. Individuals are automatically generated using a set of pre-defined base components. The base components are categorised into either the terminal set or the non-terminal set. The terminal set consists of symbols which take no arguments, whereas the non-terminal set consists of symbols which take one or more arguments [80]. In JSS, the most prominent representation is to use a tree-based GP, where the individuals represent priority function trees which can be used as priority dispatching rules [17]. This GP system has the terminal set consisting of attributes from the JSS problem (e.g. job and machine attributes), whereas the non-terminals consist of functions and operators. For example, Figure 2.2 shows three possible individuals which could be generated. The terminals are 0, PT (processing time), W (weight) and RT (arrival time), which are the attributes of jobs that arrive on the

shop floor plus the constant 0. The non-terminals are $+$, $-$, $\times$ and $/$, which are basic arithmetic operators. In most cases [17], the division operator $/$ is protected. A protected division operator returns 1 if the denominator is zero. In Figure 2.2a, the individual is interpreted as $-$PT, which is the SPT rule. In Figure 2.2b, the individual is interpreted as $-$RT, which is the FIFO rule. In Figure 2.2c, the individual is interpreted as $-$W/PT, which is the weighted shortest processing time (WSPT) rule. In addition, the three trees have a *depth*, i.e., the maximum distance of any node in the tree from the root node, of 2, 2 and 3 respectively.



(a) SPT rule (b) FIFO rule (c) WSPT rule

Figure 2.2: Dispatching rule heuristics represented by tree-based GP individuals.

2.3.2 Initialisation

In the beginning phase of the GP process, an initialisation procedure is used to randomly generate the individuals in the GP population using the sets of terminals and non-terminals. The two most popular procedures to initialising GP individuals are *full* and *grow* [80]. For both procedures, a maximum depth is determined for each GP individual to restrict the program size. For the full initialisation procedure, all the terminals are located at the bottom of the tree, whereas for the grow method the terminals can be located at any depth of the tree. On the other hand, *ramped half-and-half* [80] is a hybrid of the full and the grow procedures, where the half of the subtrees at the second depth, i.e., from the child nodes of the root node, are generated using full and the other half of the subtrees are generated

using grow. Koza argued that ramped half-and-half is the best generative method that works best over a broad range of problems, and generates a wide variety of tree sizes and shapes.

2.3.3 Evaluation

A key pre-defined component of GP is the *fitness function*. A fitness function assigns fitness values to the individuals so that they can be compared against each other. For example, when dealing with a static JSS problem with makespan minimisation objective ($Jm||C_{\max}$), the fitness function for an individual in the arithmetic tree-based GP population can be defined as follows. First, the individual is applied to training instances as a priority dispatching rule and generates feasible solutions for the training instances as part of the training procedure. Afterwards, the fitness function calculates the mean of the makespans of the solutions. In the case of a dynamic JSS problem, a discrete-event stochastic simulation is often used as a 'training instance' for which the individuals can be applied to [17]. The individual's performance over the simulation, based on the objective of the problem, is used for the individual's fitness. If an individual ω has a fitness value lower than an individual ψ, then individual ω is considered better than individual ψ. Researchers have proposed various methods of evaluating individuals effectively. Surrogate modelling [59, 108] is one such approach, which will be discussed in more detail in Section 2.1.5.

2.3.4 Selection and Breeding

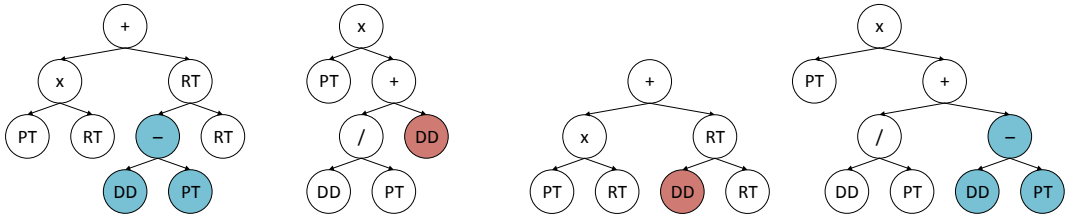
After the fitness values have been assigned to the individuals in the GP population, the next generation of individuals are generated by carrying out the breeding procedure [80]. This thesis uses the breeding procedure that was recommended by Koza [80], which is a very commonly used breeding procedure. After duplicating the individuals with the best fitnesses through elitism, the remainder of the population are filled by gen-

erating offsprings via genetic operators that are applied to select parent individuals. The two most common selection procedures to select parent individuals from the population are roulette wheel selection (or fitness-proportionate selection) [13] and tournament selection [13]. Roulette wheel selection normalises the fitnesses of the individuals into probabilities, and then randomly selects individuals based on the associated probability values. In tournament selection, a small group of individuals are randomly sampled from the population. The individual with the best fitness out of the group is used as a parent for a genetic operator, whereas all other individuals that partook in the tournament are placed back into the population for further selection. Tournament selection is a very commonly used selection procedure in the literature [80] and will be used for this research. The process of generating offspring individuals by applying genetic oeprators to parent individuals continues until enough individuals have been generated for the population.

2.3.5 Genetic Operators

After the individuals have been selected, an individual can either undergo crossover with another individual, undergo mutation, or remain the same. Crossover exchanges subcomponents between two individuals. For example, suppose that we have a tree-based GP system with the terminal set $\{\mathrm{PT}, \mathrm{RT}, \mathrm{DD}\}$ and non-terminal set $\{+, -, \times, /\}$. Figure 2.3 shows an example of the crossover operator being applied to two GP individuals. In the figure, individuals representing functions $\mathrm{PT} \times \mathrm{RT} + (\mathrm{DD} - \mathrm{PT})/\mathrm{RT}$ (parent A) and $\mathrm{PT}(\mathrm{DD}/\mathrm{PT} + \mathrm{DD})$ (parent B) are crossed with each other by exchanging the components $\mathrm{DD} - \mathrm{PT}$ and DD respectively, which are randomly selected. This results in two offsprings: individuals which represent functions $\mathrm{PT} \times \mathrm{RT} + \mathrm{DD}/\mathrm{RT}$ (offspring A) and $\mathrm{PT}(\mathrm{DD}/\mathrm{PT} - \mathrm{DD} + \mathrm{PT})$ (offspring B).

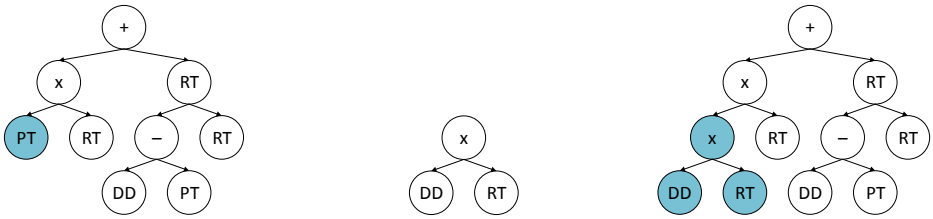
On the other hand, using the same set of terminals and non-terminals,



(a) Parent A (b) Parent B (c) Offspring A (d) Offspring B

Figure 2.3: Example of the crossover operator being applied to GP individuals.

Figure 2.4 shows an example of the mutation operator being applied to a GP individual. In the figure, the DD leaf node of the individual representing a function $\text{PT} \times \text{RT} + (\text{DD} - \text{PT})/\text{RT}$ is removed, and is replaced by the randomly generated replacement shown in Figure 2.4b, resulting in an offspring representing a function $\text{DD} \times \text{RT}^2 + (\text{DD} - \text{PT})/\text{RT}$.



(a) Parent (b) Replacement (c) Offspring

Figure 2.4: Example of mutation operator being applied to a GP individual.

2.4 Hyper-heuristics

Described by Cowling et al. [34] as "heuristics to choose heuristics", hyper-heuristics have gained the attention of researchers whose goals were to design generic but effective methods to problems [21]. Burke et al. [21]

describe one of the main motivations for developing hyper-heuristic approaches is to handle the "challenge of automating the design and tuning of heuristic methods to solve hard computational search problems". Therefore, a hyper-heuristic is a high-level approach that is independent of the problem domains that it is applied to.

As proposed by Burke et al. [22], hyper-heuristics can be categorised into two major categories: *heuristic selection* and *heuristic generation*. The first category of hyper-heuristics is heuristic selection based approaches [22], where the hyper-heuristic approach selects from existing low-level heuristics to determine the best heuristic to apply to a specific problem instance. Heuristic selection methods build a solution to a problem instance incrementally. Starting from an empty solution, the hyper-heuristic approach applies a low-level heuristic to the problem instance that is the most suitable based on the current problem state [22]. GA approaches have been applied to various problems as a hyper-heuristic approach that uses heuristic selection, where the indices in the vectors correspond to the low-level heuristic that can be applied to a problem instance [22]. For a JSS problem, GA can be combined with man-made dispatching rules to determine which job is selected during a decision situation [124].

The second category is heuristic generation based approaches that generate new heuristics from low-level heuristic components [22]. GP based hyper-heuristics (GP-HHs) are examples of heuristic generation approaches where base heuristics are combined to form new heuristics. A high-level example of a GP-HH being applied to a JSS problem this is shown in Figure 2.5. In the figure, a tree-based GP takes in the base heuristics $\{\text{PT}, \text{RT}, \text{DD}\}$ and the operators $\{+, -, \times\}$ to initialise the individuals in the population. Individuals in the GP population represent dispatching rules which can solve problem instances in the problem domain the GP-HH system is designed for. The GP system then passes the dispatching rule onto the JSS problem domain, and get a goodness measure back in the form of the individual's fitness. The final output is a priority dispatching rule

RT+PT×DD, which can be reapplied the JSS problem. GP-HH approaches are popular hyper-heuristic approaches in the literature [21] and are also popular for JSS problems [17, 102, 103].

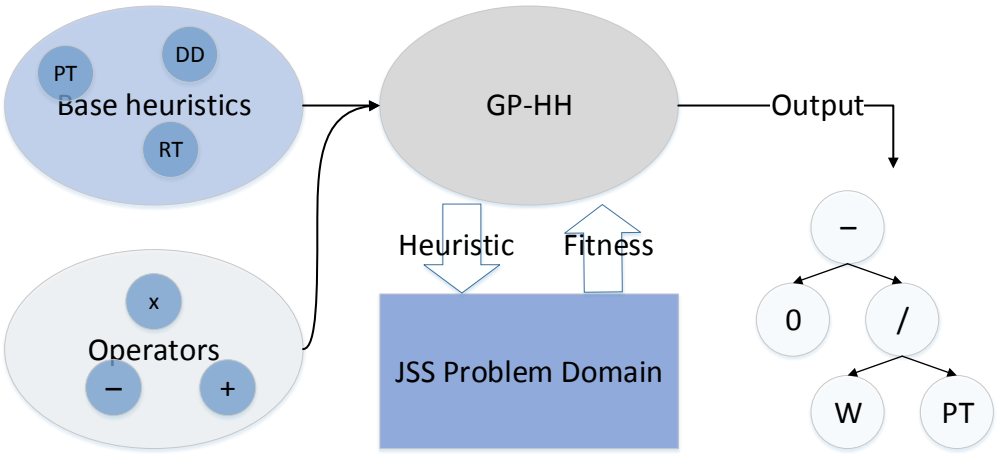


Figure 2.5: Overview of a tree-based GP-HH applied to JSS

Both heuristic selection and heuristic generation approaches have their advantages and disadvantages. In a situation where there are a number of effective low-level heuristics that have already been designed for a problem, then heuristic selection technique can *reuse* the low-level heuristic and can boost their performances further [124]. On the other hand, in an unseen problem where effective low-level heuristics have not yet been designed yet, heuristic generation techniques can effectively generate new low-level heuristics from the base heuristic component. Designing a new heuristic can be difficult and requires an extensive trial-and-error process and base heuristic components are intuitive for a human expert to design in comparison [21]. Certain approaches also combine both heuristic selection and heuristic generation to form a hybrid approach that is more effective than either approach applied to the problem separately [124].

In comparison to heuristics and meta-heuristics described above, hyper-heuristics are techniques that are independent of the problem like meta-

heuristics. However, hyper-heuristics do not solve the problem directly like heuristics or meta-heuristics but generate a generated heuristic that can be used to solve the problem. The main advantage of hyper-heuristic is that the generated heuristic is *reusable* and can be applied to different problem instances in the problem without having to rerun the hyper-heuristic approach. Specific GP-HH approaches to JSS will be discussed in further detail below.

2.5 Static and Dynamic JSS Techniques

This section covers the exact optimisation, heuristic and meta-heuristic approaches to JSS techniques which have been proposed in the literature.

2.5.1 Exact Optimisation Techniques

In general, the initial techniques for scheduling problems were to build models for specific scheduling problems. This resulted in the algorithms which could solve specific problems directly, such as Jackson's algorithm [69], which could solve the two-machine JSS problem with makespan minimisation optimally. Jackson's algorithm is an extension of Johnson's algorithm [74] for two-machine flow shop problems with makespan minimisation. However, Garey et al. [46] showed that static makespan minimisation JSS problems with the number of machines $M > 2$ machines are NP-hard. This means that, unless P $=$ NP, there is no algorithm that runs in polynomial time in the worst case scenario for JSS problems with more than two-machines and makespan minimisation objective. In addition, other objectives for JSS problems, such as TWT, are NP-hard [125]. Researchers have suggested using search techniques to handle more difficult JSS problems.

Branch-and-bound [83] is one of the search techniques which have been used significantly in the literature for static JSS problems [7, 10, 20, 24, 25].

In a branch-and-bound, the algorithm carries out a depth-first search to find feasible solutions to a JSS problem instance [125]. After a feasible solution is found, the algorithm eliminates any branches where the lower bound on the problem is worse than the best solution found so far. The algorithm then exhaustively searches for new solutions until all branches have been explored to ignored, and the best solution found is returned. A branch-and-bound algorithm's search is augmented by using a heuristic to guide the search direction of the algorithm, and researchers have proposed various effective heuristics to guide the branch-and-bound algorithm [7, 10, 20, 24, 25]. A notable branch-and-bound approach has been proposed by Carlier and Pinson [25] in 1989, where they were able to find an optimal solution for a JSS problem instance proposed by Muth and Thompson [100] with $N = 10$ jobs, $M = 10$ machines and 10 operations per job. Branch-and-bound techniques are covered in detail in a survey paper by Potts and Strusevich [128].

Dynamic programming is an exact optimisation technique which has been applied to static JSS problems [50, 87, 125]. Dynamic programming approaches divide a JSS problem instance into constituent sub-problems and attempt to solve over the sub-problems to solve the problem instance. Lawler and Moore [87] proposed a method of applying dynamic programming to a single-machine scheduling problem with TWT minimisation and suggested methods of extending it to parallel and 2 machine flow shop problems. A more recent dynamic programming approach to JSS with makespan minimisation has been proposed by Gromicho et al. [50] in 2012, where they adapt an approach proposed by Held and Karp [58] for the travelling salesman problem (TSP). In their empirical analysis, they show that the dynamic programming approach can generate optimal solutions for moderate benchmark instances, where problem instances have up to $N = 10$ jobs and $M = 5$ machines.

Although exact mathematical optimisation techniques guarantee an optimal solution for a JSS problem instance, they generally take too long

to generate optimal solutions for problem instances that have more than $10 - 15$ jobs [7, 50]. In addition, they are not suitable for DJSS problems as the dynamic events that occur during processing can results in predicted schedule no longer being optimal.

2.5.2 Heuristic Techniques

As heuristic approaches often rely on a static assumption about the problem, in most cases they do not generate optimal solutions to a JSS problem instance, unlike an exact optimisation technique. However, exact optimisation techniques do not scale well with the size of the problem instances, i.e., the number of arriving jobs and the number of operations per job. Because of this, heuristic approaches can be applied to large static JSS problem instances which are would be too computationally intensive to generate optimal solutions using exact optimisation techniques. For example, effective heuristic approaches have been applied to problem instances with up to 8700 jobs and 9 machines [118]. In addition, heuristic approaches can be applied to DJSS problems, whereas it is impractical to apply exact optimisation techniques to DJSS problem. An exact optimisation technique generates a schedule to a JSS problem instance before processing and needs an accurate prediction of the quality of the schedule in advance. However, it is not possible to get an accurate prediction of the quality of the schedule in a DJSS problem instance as the properties of the problem instance changes during processing, which means that the predicted schedule may not be optimal after the jobs have been processed [140].

Dispatching rules are one of the most prominent heuristic approaches for JSS problems. Examples of small dispatching rules are SPT, FIFO and earliest due date (EDD) [125]. The EDD rule prioritises jobs with the earliest due date time. More complex examples of dispatching rules, called composite dispatching rules (CDRs) [72], combines multiple shop floor at-

tributes together. Examples of CDRs are the "cover over time" (COVERT) [26] and the "apparent tardiness cost" (ATC) rules [148]. In COVERT, the priority of a job is calculated from the expected waiting times of the jobs for the remaining operations, and the job's slack. The expected waiting times of the operations are calculated based on the previous waiting times on the respective machines for the jobs. COVERT is a look-ahead rule designed to be effective for tardiness related objectives [148]. COVERT was extended further by Vepsalainen and Morton [148] to be effective for weighted tardiness related objectives as well. On the other hand, ATC extends a previously proposed look-ahead rule for the weighted tardiness problem [99] by using the average processing time of waiting jobs and an adjustable factor value k. Vepsalainen and Morton [148] showed that the extended COVERT and the ATC rule can obtain better performance than the prior rules to the JSS problem.

Jayamohan and Rajendran [72] compare various dispatching rules which have been proposed in the literature, such as rules proposed by Holthaus and Rajendran [62], and also present new dispatching rules. The new dispatching rules incorporate flow due date, expected waiting time of operation, and operation due date (denoted as FDD, PW and ODD respectively) as part of existing dispatching rules. Operational due date is the time when an operation is expected to complete based on the job's arrival times and the processing times from the first operation to the specific operation [14]. Flow due date is an expected time when the job is expected to complete based on its arrival time and total processing times of operations, i.e., the operational due date of the final operation of the job. Jayamohan and Rajendran [72] compared the rules are compared on both flowtime and tardiness minimisation objectives for the DJSS problem. The results showed that the new rules generally show the best performance on the DJSS problem instances, but there is a trade-off in the performances of the rules based on the properties of the problem instances. Jayamohan and Rajendran extended the comparative study further [73] to incorporate

weighted COVERT and weighted ATC rules [148] and to propose new dispatching rules.

Holthaus and Rajendran [63] proposed the 2PT$+$WINQ$+$NPT priority rule and variations of the rule for the dynamic JSS problems with flowtime and tardiness minimisation objectives. PT denotes the processing time for the job's operation. WINQ denotes the work-in-next-queue, which is the sum of processing times of operations of other jobs waiting at the *next* machine that the job will be processed on, which is 0 if the job will be completed after being processed at the machine it is currently arrived at. NPT denotes the processing time of the job's operation on the next machine it needs to be processed and is equal to 0 if the job is completed after being processed at the current machine. They show that 2PT$+$WINQ$+$NPT consistently outperformed benchmark dispatching rules such as the RR rule (Raghu and Rajendran's rule) [129], which was previously the best rule for minimising mean tardiness for dynamic JSS problem instances.

Outside of dispatching rules, a more complex example of a heuristic approach is the shifting bottleneck heuristic, which was proposed by Adams et al. [3] in 1988 for static JSS problem with makespan minimisation objective. The shifting bottleneck heuristic defines how "critical" an unscheduled machine is by solving "to optimality a one-machine scheduling problem that is a relaxation of the original problem", then by calculating the differences in the expected completion times to the actual completion times of a jobs' operation on the machine. It then finds the bottleneck machine, i.e., the machine with the greatest difference in the expected and actual operation completion times, and sequences it optimally to generate a new schedule. It then finds the next bottleneck machine, sequences it optimally, and then looks back at the machines it has previously sequenced and re-optimise each machine using an algorithm developed by Carlier [24], which is a branch-and-bound method designed for optimising single-machine problem instances. All other sequences are kept fixed while a machine is being re-optimised. Adams et al. showed that the shifting bot-

tleneck procedure can find high-quality, sometimes optimal, solutions for problem instances with up to $N = 15$ jobs and $M = 15$ machines quickly (in 1988).

2.5.3 Meta-heuristic Techniques

Meta-heuristics for JSS problems have also been extensively studied in the literature, where both local search based techniques and population-based techniques have been effectively applied to different JSS problems. In addition, hybrid approaches that combine both local search with a population have also been applied to JSS problems. Both local search and population based techniques have mainly been applied to DJSS problems with small number of dynamic events, e.g., problem instances with up to 100 jobs, and very few meta-heuristic approaches have been applied to DJSS problems with large number of dynamic job arrivals [5, 4].

Local Search based Techniques

Taillard [142] proposed a Tabu Search (TS) [49] approach to a static JSS problems. Taillard showed that the proposed TS algorithm provided better solutions than the previously found best solutions for the benchmark JSS problems [10, 141]. Additional TS approaches include an approach by Dell'Amico and Trubian that apply the TS algorithm to various benchmark JSS problems [3, 10, 100]. Dell'Amico and Trubian showed that their TS approach outperforms various benchmark approaches from the literature [147], and also found optimal solutions to the static JSS problem instances. In addition, De Bontridder [36] proposed a TS approach that uses a disjunctive graph representation for the neighbourhoods. De Bontridder showed that the proposed TS approach is effective against other benchmark algorithms in the literature both in terms of computation time and solution quality over a static JSS problem.

Simulated annealing (SA) [79] approaches have also been applied ef-

fectively to JSS problems in the literature. A notable SA approach is the large step random walk (LSRW) method proposed by Kreipl [82]. LSRW is considered to be one of the best local search methods for handling static JSS problems with TWT minimisation objectives according to Nguyen et al. [105], outperforming other highly effective algorithms such as the hybrid GA approach proposed by Zhou et al. [158]. Other local search based techniques also include variable neighbourhood search (VNS) [54]. Adibi et al. [5] proposed a VNS approach for a multi-objective DJSS problem with dynamic job arrivals and machine breakdowns. In their approach, VNS is hybridised with an artificial NN (ANN), where ANN is used to optimise the parameters of the VNS approach. They showed that VNS can outperform common man-made dispatching rules such as SPT and FIFO. Adibi and Shahrabi [4] extended the VNS approach to cluster the neighbourhood solutions and showed that the modified VNS approach produced better results than the benchmark VNS approach without clustering. However, they do not provide a comparison to GP-HH approach, which have shown evolve more effective rules than the man-made dispatching rules in the literature [17].

Population-based Techniques

Cheng et al. [29] provide a survey of genetic algorithm (GA) approaches to JSS problems. GA is an evolutionary computation (EC) technique, which consists of a population of individuals represented by a fixed length chromosome. Genetic operators such as crossover and mutation are applied to the individuals to generate the next generation of individuals, and individuals which perform poorly are eliminated from the population. A notable example of a GA approach to JSS problems with TWT minimisation objective was proposed by Zhou et al. [158]. The individuals in the GA approach are encodings representing a permutation of the jobs on the machines on the shop floor. The GA algorithm is hybridised with existing heuristics for JSS, such as the weighted COVERT rule [148]. The GA

individual determines the first job that is scheduled by the jobs on the shop floor, and the embedded heuristic schedules the remaining jobs after the first jobs have been completed. Zhou et al. [158] showed that the hybrid GA outperforms pure GA approaches and significantly reduce the computation time required to generate a solution. Other meta-heuristic approaches which use EC techniques include artificial bee colony (ABC) optimisation [31], ant colony optimisation (ACO) [32] and particle swarm optimisation (PSO) [135, 151].

2.6 GP-HH for Scheduling Problems

GP-HH approaches for production scheduling problems have grown exponentially over the past 20 years [103]. To cover the various GP-HH approaches that have been applied to scheduling problems in the literature, this section categorises the relevant GP-HH approaches according to the framework proposed by Nguyen et al. [103]. In the framework, the GP procedure to evolve scheduling heuristics for the problem is broken down into four major steps. The first step is the definition of the scheduling heuristic determines how the scheduling decisions are made using the GP individuals. The second step is determining the GP representation, function sets and terminal sets. The third step is determining the qualities of the scheduling heuristics from the GP evaluation procedure and the fitness functions. The fourth step is the GP search mechanism, which is the underlying GP process that is independent of the scheduling problem. Afterwards, we provide a summary of the GP approaches that are discussed in this section.

2.6.1 Scheduling Heuristic Definitions

Nguyen et al. [103] provide a generalised schedule construction algorithm that carries out job sequencing decision and show that there are two com-

ponents to job sequencing decisions that can be modified. The first component is the non-delay factor that determines which jobs are considered for selection. In most GP-HH approaches, the sequencing decision only considers the jobs already waiting at the machine during a decision situation (i.e. the non-delay factor is zero) [17, 102]. Nguyen et al. [105] investigated evolving dispatching rules where the non-delay factor could be automatically tuned by the GP approach during the evolutionary process for two out of the three representations they investigated. They found that best rules evolved by GP had non-zero non-delay factors for the static JSS problem handled by the GP approach.

The second component to a scheduling heuristic described by Nguyen et al. [103] is the decision-making process for determining which job is selected to be processed. Prominently, GP-HH approaches that evolve priority dispatching rules that carry out the job selection [17, 102]. Other job selection methods include a GP approach by Nguyen et al. [106] that evolve iterative dispatching rules (IDRs). IDRs are rules that are applied multiple times to a JSS problem instance. After an IDR is applied to a JSS problem instance and a schedule is generated, it uses the information from the previously generated schedule, e.g., the recorded finish time. On the next run, the IDR uses the information from the previous schedule to adjust how the priorities of the individuals are calculated, and can potentially obtain generate better schedules. Nguyen et al. showed that the IDRs evolved by GP significantly outperformed the standard GP approach. Park et al. [119] proposed a GP-HH approach that evolves stochastic dispatching rules that are also applied multiple times over a scheduling problem instance to potentially obtain better solutions. They showed that the stochastic dispatching rules evolved by the GP approach can outperform dispatching rules evolved by a standard GP approach. However, dispatching rules that are applied multiple times over a JSS problem instance can only optimise to a certain point in DJSS problem instances due to the constrained information horizon [61].

Other job selection methods include GP-HH approaches that use multiple rules to help with the decision-making process. GP approaches that use multiple rules include Jakobović and Budin's [70] GP-3 approach. They first develop a benchmark GP approach that evolves single dispatching rules for a single-machine problem with the TWT minimisation objective. Afterwards, GP-3 is applied to a JSS problem with TWT minimisation objective. GP-3 approach evolves a decision tree and two dispatching rules. The decision tree determines which dispatching rule is applied to a decision situation during processing. They showed that the GP-3 performs better than the benchmark GP approach on the JSS problem. Jakobović et al. [71] then use the same GP system for a parallel machine problem, where jobs can be processed on any of the machines on the shop floor. In addition, Yin et al. [155] were the first to use GP for a single-machine DJSS problem with machine breakdowns. In their approach, a GP individual is represented by two trees. The first tree is used to calculate the priorities of jobs waiting at the machine. The second tree is used to calculate the amount of idle time to add in between the processing of jobs to avoid having jobs be disrupted by the breakdowns. The two tree approach outperformed other heuristic approaches in the literature for handling machine breakdowns.

Along with dispatching rules that handle job sequencing decisions, researchers have also proposed GP approaches to handle scheduling problems with *routing decisions* for flexible job shop scheduling problems [125], and *due date assignment decisions* for scheduling problems where the due dates of the jobs have to be assigned endogenously [30, 130]. Nguyen et al. [107] proposed a cooperative coevolutionary GP approach to evolving both a sequencing rule (i.e. a dispatching rule) and a due date assignment (DDA) rule simultaneously for a multi-objective DJSS problem. The scheduling policies, which are the sequencing and DDA rule pairs, evolved by the cooperative coevolutionary GP outperforms existing GP approaches to evolving scheduling policies in the literature, and signifi-

cantly outperform man-made scheduling policies used in the literature. In addition, Yska et al. [156] proposed a cooperative coevolutionary GP approach for a dynamic flexible JSS problem to evolve both a sequencing rule and a routing rule simultaneously. They showed that evolving the routing rule simultaneously with the sequencing rule can significantly boost the performance of the GP than using a fixed man-made routing rule with the sequencing rule.

2.6.2 GP Representation, Terminal Sets and Function Sets

A number of GP representation have been investigated in the literature. The most prominent representation is a tree-based GP where the individuals represent arithmetic function trees [17, 102, 103]. The individuals that represent arithmetic function trees are usually applied to the scheduling problem instances as priority dispatching rules. One of the earliest examples of a GP approach that uses arithmetic representation is Dimopoulos and Zalzala [40], where they use GP to evolve priority dispatching rules for a static single-machine problem with total tardiness minimisation. The rules evolved by Dimopoulos and Zalzala's GP approach performs better than the man-made benchmark dispatching rules. On the other hand, Nguyen et al. [105] investigated three different tree-based GP representation. The first GP representation evolves decision trees, the second GP representation evolves priority functions which can be used in a priority dispatching rule, and the third GP representation combines the first two GP representations together. They show that the third GP representation outperforms the first two, and is competitive with effective meta-heuristics such as GA [158] for static JSS problems.

The selection of terminal sets and function sets are also an important factor in the effectiveness of the evolved rules. For example, Hildebrandt et al. [60] incorporate terminals such as ODD [14] to evolve dispatching rules for a DJSS problem with flowtime minimisation objective. They

show that the pseudo due date information, even on the DJSS problem that has no tardiness related objectives, can result in effective rules. In addition, Hunt et al. [66] incorporated "look-ahead" terminals into an existing GP terminal set. The look-ahead terminals incorporate information from the decisions that have been made earlier in the schedule such as waiting jobs at machines. They show that the effectiveness of the rules evolved by GP improves with "look-ahead" terminals.

Mei et al. [95] carried out feature selection of the GP terminals on the DJSS problem with MWT minimisation objective. First, the individuals in the GP population are diversified using the *clearing procedure* [133], which is a *niching technique* commonly used in the literature [122, 133]. Afterwards, the best diverse set of individuals are analysed based on their structures. They identified a list of features that were included in the best rules, which includes terminals such as PT, W and WINQ. Furthermore, Mei et al. [96] also investigated the effectiveness of GP terminals, depending on whether the GP terminals are *time-variant* or *time-invariant*, for a DJSS problem with tardiness related objectives. Time-variant terminals are terminals that are dependent on the duration of processing. An example of a time-variant terminal is the jobs' due dates. The jobs that arrived early during processing are likely to have lower due dates than the jobs that arrived late during processing. This affects how the priorities are calculated by the GP evolved dispatching rules, as the relative values of terminals such as due date become a bigger factor compared to other terminals such as the job's processing times, which is a time-invariant terminal. A time-invariant counterpart to a job's due date is the relative due date, which is the difference between a job j's due date d_j and the current time t ($d_j - t$). They showed that the GP that uses only time-invariant terminals, where the time-variant terminals are replaced with time-invariant counterparts, generally perform better than the benchmark GP approach that uses both time-variant and time-invariant terminals.

2.6.3 Estimating the Quality of Scheduling Heuristics

Researchers have investigated various evaluation procedures and developed fitness functions to improve the effectiveness of GP-HH for scheduling problems. The standard procedure is to apply the GP individual to a set of scheduling problem instances, and use the average objective of the generated schedules as the fitness of the individual [17, 102, 103]. Example of this is shown by Geiger et al. [47], where GP is applied to various static JSS problems that are polynomial time solvable and static JSS problems that are NP-hard. They show that GP can evolve optimal rules for P problems and perform better than man-made dispatching rule benchmarks for the NP-hard problems. However, the average objective can be biased towards specific scheduling problem instances, as certain scheduling problem instances may have higher lower bound values than others. Therefore, *normalising* the objectives can reduce the bias towards scheduling problem instances. Hildebrandt et al. [60] and Mei et al. [97] both incorporate objective normalisation to the fitness calculation. Mei et al. [97] discuss three different procedures to normalise the objectives and a practical method of using a reference rule to normalise the objectives of the schedules.

Surrogate modelling has also been incorporated into GP-HH approaches to improve the computation cost required to evolve the rules, reduce the number of evaluations and to improve the training convergence of GP [103]. Hildebrandt and Branke [59] investigated two different surrogate models that are incorporated to a GP-HH approach for a DJSS problem with flowtime minimisation objective. The first surrogate GP approach compares the structure of the best rule found to the other GP individuals to calculate the surrogate fitness of the GP individuals. The surrogate fitness is then used to determine whether the individual show enough promise to warrant full evaluation on the training set. The second surrogate GP approach compares the phenotypic behaviour of the best rule against the GP individuals to calculate the surrogate fitness of the GP indi-

viduals. To calculate the phenotypic characteristics of a GP individual, the Holthaus's rule ($2\text{PT} + \text{WINQ} + \text{NPT}$) [63] is used as a reference rule after the GP individual makes decisions on sample decision situations. Hildebrandt and Branke showed that the phenotypic surrogate GP approach performed better than the genotypic surrogate GP approach. They also showed that the surrogate model results in a significantly improved computation time required to evolve the rules over the benchmark standard GP approach. In addition, their analysis showed that the phenotypic surrogate GP performed very close to a "perfect" surrogate.

In addition to Hildebrandt and Branke's [59]'s surrogate GP, Nguyen et al. [110] also proposed surrogate GP approach for a DJSS problem. In Nguyen et al.'s surrogate GP approach, simplified simulation models are used to calculate the surrogate fitness of the GP individuals. The simplified simulation models are significantly smaller in size to the standard DJSS simulation models used as training instances, but provide a reasonable estimation for the fitness of the GP individuals. They investigate two different simplified simulation models to be used as surrogates and showed that one of the proposed surrogate GP approaches was tied for the fastest evolution time with Hildebrandt and Branke's [59] phenotypic surrogate GP approach. In addition, the Nguyen et al.'s two surrogate GP approaches also performed the best on the test set out of the GP approaches evaluated.

Local search procedures have also been used for a GP's evaluation procedure to improve the effectiveness of the output dispatching rules. Hunt et al. [68] proposed two methods to assist in the decision-making process of the GP individual using local search. First, they first calculate an expected completion time of a job waiting at a machine and calculate an estimate TWT from the expected completion times. This is used to compare the decisions made by a GP individual and the decision that would be made by a local search algorithm, and the GP individual is penalised if its decision is suboptimal compared to local search. Local search is also

used as a tie-breaker if the GP individual assigns the highest priority to at least two jobs. They showed that incorporating local search improves the quality of the output rules, but certain parameters had to be adjusted due to the incorporation of the local search (e.g. GP population size is set to 50).

2.6.4 GP Search Mechanism

Although the standard GP process is effective at evolving rules, more complex and specialised GP approaches are required for more complex scheduling problems. For example, various GP-HH approaches have been proposed for multi-objective scheduling problems. An early example of a multi-objective GP-HH approach was proposed by Tay and Ho [143], where they combine the three objectives in a flexible JSS problem into a single objective and solve the problem using a standard GP approach. However, Hildebrandt et al. [60] showed that Tay and Ho's approach performs poorly on dynamic environments.

Other approaches have incorporated multi-objective evolutionary algorithms to GP to handle multi-objective scheduling problems. Nguyen et al. [107] proposed several multi-objective GP-HH approaches for a DJSS problem with both sequencing and DDA decision. Four multi-objective GP-HH approaches incorporate NSGA-II [39], SPEA2 [159], HaD-MOEA [65], and a proposed cooperative coevolution modified from Potter and De Jong's [127] cooperative coevolution. In the proposed cooperative coevolution, the GP population is partitioned into two subpopulations. The first subpopulation consists of GP individuals that are applied to the DJSS problem instances as sequencing rules, and the second subpopulation consists of GP individuals that are applied to the DJSS problem instances as DDA rules. Nguyen et al. showed that the proposed cooperative coevolutionary GP approach generally performed better than the other multi-objective GP approaches investigated.

Masood et al. [90] proposed a many-objective GP approach to handle a DJSS problem with up to four objectives. A many-objective optimisation problem is a subset of multi-objective optimisation problems that have more than three objectives [38]. Masood et al. first identify four conflicting objectives in the DJSS problem handled by the GP approach. Afterwards, they incorporate NSGA-III [38], a modified NSGA-II designed to handle many-objective optimisation problems, to the GP approach, and show that GP with NSGA-III show significant improvement over GP with NSGA-II.

2.6.5 GP-HH for Scheduling Problems Summary

GP-HH approaches have been extensively applied to scheduling problems in the literature to automatically evolve effective dispatching rules. In general, the dispatching rules evolved by the GP approaches generally outperform the man-made dispatching rules and are quite suited to DJSS problem with dynamic job arrivals due to the short reaction times of the dispatching rules and their ability to cope with the dynamic environment [105]. However, the existing GP approaches to DJSS problems focus on dynamic job arrivals. Only a limited number of research has handled other types of dynamic events [155], and no GP-HH approach has been proposed for a DJSS problem with both dynamic job arrivals and machine breakdowns. Having multiple types of dynamic events adds an additional layer of complexity to the problem, and the effectiveness of the rules evolved by GP is unknown for these DJSS problems.

For further reading, there are multiple survey papers on evolutionary scheduling approaches, which includes GP-HH approaches. Hart et al. [55] provide a survey of evolutionary scheduling approaches to both various scheduling problems, including JSS problems. They discuss both meta-heuristic and hyper-heuristic approaches to scheduling problems in detail. They also cover GP-HH approaches that have been proposed in the literature, although only a few GP-HH approaches have been devel-

oped at the time of writing. Branke et al. [17] provide a more modern survey of hyper-heuristics and automated heuristic design for scheduling problems. Although they focus primarily on GP and GP specific design considerations, they also discuss other hyper-heuristic approaches to scheduling problems such as gene expression programming approaches by Nie et al. [111, 112, 113]. Nguyen et al. [102], in addition to providing a comprehensive survey of evolutionary scheduling approaches, discuss challenges and future directions to EC approaches to scheduling problems. One of the challenges discussed by Nguyen et al. are scheduling problems that have different sources of disturbances, as most existing approaches handle scheduling problem with one source of disturbance, e.g., dynamic job arrivals or machine breakdowns. They also explain that scheduling problems with multiple decisions and multiple objectives are key research directions. Finally, Nguyen et al. [103] provide the unified framework for evolving scheduling heuristics using GP and categorises the existing GP-HH approaches under the framework.

2.7 Related Work for Research Goals

This section covers the related work that has been carried out in various fields of research that are relevant to the research goals. First, we cover the ensemble learning outside of scheduling and the limited number of ensemble GP-HH approaches that have been proposed in scheduling problems during the period of the research that contributed towards the thesis. Afterwards, we cover the machine breakdown specific approaches to dynamic scheduling problems. Finally, we specifically cover multitask learning related to optimisation, where multitask learning have been applied to various discrete and continuous optimisation problems.

2.7.1 Ensemble Learning

Examples of a few key ensemble approaches that have been applied to problems outside of classification are as follows. Early examples of ensemble learning approaches are bagging by Breiman [19] and boosting by Freund and Schapire [45]. In bagging, subsets of problem instances are sampled from the training set uniformly, and the training subsets are used to train the individual ensemble members. In boosting, the ensemble members are usually trained sequentially, where the weights of the training instances are adjusted based on the performance of the trained ensemble members. This provides a pressure during the training of the next ensemble member to perform more effectively on the training instances that the trained ensemble members performed poorly on. Another example of an ensemble approach applied to binary classification with unbalanced data was proposed by Bhowan et al. [15]. Bhowan et al. decomposed the binary classification problem to a multi-objective problem and applied GP based NSGA-II to build a diverse range of individuals that are combined to form an ensemble. After testing various diversity measures for the ensembles, they showed that the ensemble evolved by GP produce significantly less false positives than other popular machine learning techniques such as support vector machines [57]. Polikar [126] provides a comprehensive survey of ensemble learning approaches and in-depth discussions on applying ensemble of rules over single constituent rule.

A limited number of ensemble GP-HH approaches have also been proposed in the literature. Hart and Sim [56] proposed the NELLI-GP approach, which is a hybrid of GP and artificial immune system (AIS). The GP component evolves the dispatching rules that are combined into ensembles by the AIS, where the GP individuals are assigned to specific JSS problem instances as specialists. They showed that NELLI-GP can significantly outperform previously proposed ensemble GP approaches for the static JSS problem. In addition, Durasevic and Jakobovic [42] compared various ensemble GP approaches that evolve ensembles of dispatching

rules for a DJSS problem with up to 100 jobs in the problem instances. The ensemble algorithms incorporated into the GP approach are bagging [19], boosting [45], a simple method of combining the best evolved dispatching rules called simple ensemble combination (SEC), and cooperative coevolution. They showed that SEC performed the best after evolving $20,000$ dispatching rules over the training instances. This is followed by the GP approaches that use bagging and boosting.

2.7.2 Techniques to DJSS subject to Machine Breakdowns

As described by Ouelhadj and Petrovic [115], scheduling techniques that handle dynamic scheduling problems can be categorised into three major groups. Examples of specific approaches to DJSS problems with machine breakdowns in the three major groups are discussed in detail below.

Completely Reactive Scheduling

The first category of approaches that handle DJSS problems with machine breakdowns is *completely reactive* scheduling approaches [115]. In completely reactive scheduling, the scheduling algorithm makes the scheduling decision during processing. This means that the scheduling decisions are made at each decision situation as soon as the machine becomes available. The solution to the DJSS problem instance is available after all jobs have been processed on the shop floor. Because the decisions are made in real-time, completely reactive scheduling approaches need to have short reaction times and be able to cope with dynamic environments effectively [105, 115]. Therefore, dispatching rules are completely reactive scheduling algorithms [115].

To handle large DJSS problem instances with stochastic machine breakdown events, Holthaus [61] compares the application of various dispatching rules in dynamic JSS problems under various flowtime and tardiness minimisation objectives with machine breakdowns. Holthaus shows that

for minimising the mean flowtime the PT + WINQ rule performs the best out of the selected rules. However, when it comes to handling objectives dealing with due dates, such as mean tardiness, the performances of the rules depend on the properties of machine breakdowns, such as the failure rate of a machine. Subramaniam et al. [138] covers machine breakdowns in a flexible JSS problem, and uses a two selection procedure for selecting both a machine and a job to process. As jobs can be processed at different machines for a single operation in a flexible job shop, they compare several different methods of selecting machines for the dispatching decision and dispatching rules to each other and show that incorporating machine selection significantly improves the performance over using the dispatching rules individually.

Predictive-reactive Scheduling

The second category of approaches that handle DJSS problems with machine breakdowns is *predictive-reactive* scheduling approaches [115]. A predictive-reactive scheduling algorithm generates a schedule before processing. When a machine breakdown event occurs during processing, the generated schedule is adjusted to accommodate for the machine breakdown event. The goal of many predictive-reactive scheduling algorithms is to minimise the impact of machine breakdown on the original schedule [115]. Adjusting the initial schedule to accommodate for the unknown breakdown event is called *rescheduling*. In the literature [2, 152, 155], an additional criterion of minimising disruption is added with machine breakdown, where the goal is to minimise the difference between the initial schedule before processing and the final schedule after all jobs have been processed.

There are several examples of predictive-reactive scheduling algorithms in the scheduling literature. Wu et al. [152] apply predictive-reactive scheduling to a single-machine scheduling problem with makespan minimisation. They define the disruption criterion as the difference in the

start times of jobs between the initial and final schedules, which is denoted as *starting time deviation*. They propose two local search heuristics, along with a GA approach to the problem, and compare the approaches to an optimal schedule generated using Carlier's branch-and-bound algorithm [24]. They show that the schedules generated by GA are more 'stable' than the benchmark optimal schedule. Abumaizar and Svestka [2] proposed the Affected Operation (AO) algorithm to handle rescheduling for a JSS problem with machine breakdown. The problem has the objectives of minimising makespan, minimising starting time deviation and minimising sequence deviation. For every job j, sequence deviation is the number of jobs which were expected to be processed before job j in the initial schedule that ended up being processed after job j. They compare the algorithm to two common benchmark rescheduling techniques, which are right-shift rescheduling and total rescheduling [115], and show that the AO algorithm provides a final schedule which has significantly lower deviation. In addition, the VNS approaches proposed by Adibi et al. [5] and Adibi and Shahrabi [4] are considered as predictive-reactive scheduling, as they carry out scheduling decisions once a "planning horizon" is reached to schedule newly arrived jobs onto the machines.

Robust Pro-active Scheduling

The third category of approaches that handle DJSS problems with machine breakdowns is *robust pro-active* scheduling approaches [115]. Robust pro-active scheduling algorithm predicts when the machine breakdowns occur and generates the schedule so that the impact of machine breakdown is minimised. This is done by inserting idle times between the job processing so that jobs are less likely to be interrupted by machine breakdowns.

Mehta and Uzsoy [94] proposed a robust pro-active scheduling approach where they calculate the amount of idle time to allocate before a machine processes the next job given the probability of a machine breaking down and the time required to repair the machine. They show that

predictive scheduling can significantly improve the predictability, while only suffering very slightly in terms of the primary objective, which is to minimise the maximum difference in the completion times of jobs and their due dates, i.e., the maximum *lateness* of jobs. Predictability is defined as the sum of deviations of the completion times of jobs in the initial schedule from the completion times of jobs in the final schedule.

Al-Hinai and ElMekkawy [8] proposed a two-stage GA approach that first optimises the primary objective, then optimises both the primary objective and a stability measure. Three stability measures are proposed, and the statistical tests showed that the stability measure that minimises the difference between the expected and the actual completion times for the operations produced the best results. Ahmadi et al. [6] combines one of the stability measures proposed by Al-Hinai and ElMekkawy [8] in conjunction with two multi-objective evolutionary algorithms (MOEAs), NSGA-II and non-dominated ranking genetic algorithm (NRGA), to handle a flexible JSS problem with machine breakdowns. The two MOEAs are used to generate Pareto fronts that tradeoff the stability of the schedule with the makespan minimisation objective. Due to the lack of benchmarks, the two MOEAs are compared against each other in terms of a number of different criterion, e.g., diversity, spacing, number of solutions on the Pareto front, computation time. They showed that there neither of the two MOEAs perform better on all criterion than the other, where NRGA showed better diversity than NSGA-II, but NSGA-II had better spacing than NRGA.

Machine Breakdown Summary

The three different types of dynamic scheduling approaches to scheduling problems with machine breakdowns have their merits and limitations. Completely reactive scheduling approaches such as dispatching rules have low computation cost, are usually intuitive and interpretable, and easy to implement [115]. However, completely reactive scheduling approaches do

not generate a global schedule in advance, which means that they often tend to be quite myopic [66, 115].

On the other hand, predictive-reactive scheduling approaches do generate a global schedule, which means that they often perform very well on DJSS problems with a small number of dynamic events. This often includes DJSS problems where the jobs are presented in a *batch* [125], i.e., a bulk arrival of jobs occur on the shop floor. However, depending on the rescheduling strategies used by the predictive-reactive scheduling approaches, handling large DJSS problem instances may be prohibitively expensive in terms of computation time [115]. This is often circumvented by making local adjustments instead of generating a new schedule from scratch whenever a dynamic event occurs [115].

Finally, robust pro-active scheduling approaches show significant promise, but predicting machine breakdown events in advance require significant knowledge of the problem domain in advance. In addition, a robust pro-active scheduling approach is specific to the properties of the DJSS problem, meaning changes in the properties of the problem will require the approach to be redesigned.

For further reading, Ouelhadj and Petrovic [115] provide a comprehensive survey of JSS problems with machine breakdowns. In addition, Suresh and Chaudhuri [140] cover machine breakdown in brief detail along with JSS problems where unknown events occur.

2.7.3 Multitask Learning for Optimisation

Although the concept of multitask learning has been applied to various machine learning problems earlier in the literature [139], the term "multitask learning" was first defined by Caruana [27]. Caruana incorporated multitask learning to an ANN to improve their performances on a set of classification problems. Caruana also carried out an extensive analysis of how multitask learning improved the performance of ANN. They found

that the benefits of multitask learning "are due to the extra information contained in the training signals for the extra tasks", i.e., the extra tasks provide inductive biases that generally improve the effectiveness of ANN. For further reading on multitask learning, Pan and Yang [116] carry out a comprehensive survey of transfer learning approaches, including multitask learning.

On the other hand, multitask learning approaches to optimisation problems that use evolutionary computation techniques (called evolutionary multitasking [35]) have been limited. Gupta et al. [51] proposed a framework for incorporating multitask learning into optimisation problems called *multifactorial optimisation* (MFO). Afterwards, they use a multifactorial evolutionary algorithm (MFEA) to handle up to three optimisation problems simultaneously. Optimisation problems handled by the MFEA include continuous optimisation problems and combinatorial optimisation problems. Gupta et al. also provide a decoding scheme so that an individual in the population can be decoded as a solution to the problem instances in three optimisation problems handled simultaneously. They showed that the MFEA show promise in improving the convergence of the EA approach in complex optimisation problem and show better performance over solving the optimisation problems independently.

Gupta et al. [53] proposed a MFEA approach that handles both MFO and multi-objective optimisation problem. The proposed multi-objective MFEA (MO-MFEA) is modified from the NSGA-II [39] approach to handle up to two multi-objective continuous optimisation problems simultaneously. They showed that the MO-MFEA approach applied to two optimisation problems simultaneously show a better convergence than NSGA-II applied independently to the two optimisation problems.

Da et al. [35] apply MFEA to additional continuous single objective optimisation problems in the literature. They investigate seven different problems and the MFEA is designed so that they can solve two problems simultaneously. Da et al. showed that the MFEA generally outperforms

the single task EA counterpart. However, there is one case where the single task EA performs better than the MFEA, where the convergence curves show that the inductive bias introduced from the two tasks actually negatively contributes towards the performance of MFEA.

The existing evolutionary multitasking approaches show that there are significant promises to applying multitask learning to optimisation problems. On the other hand, evolutionary multitasking based on GP for automatic design of dispatching rules for a JSS problem has not yet been proposed, and warrant further investigation. For further reading, Gupta et al. [52] provide a comprehensive survey of transfer learning that includes evolutionary multitasking.

2.8 Summary

There are a number of different approaches which have been proposed in the literature for a number of JSS problems. In particular, papers which have proposed dispatching rules heuristics have been very prominent in dynamic JSS due to their simplicity and their ability to cope with dynamic environments. Because the manual design of effective dispatching rule is a difficult task, the idea of automatically designing new dispatching rules to different scheduling problems have been proposed in the literature. GP is one of the more popular methods of automatically generating dispatching rules, as a GP individual can intuitively be interpreted as a simple heuristic, e.g., as a priority dispatching rule. However, the idea of using GP-HH for scheduling problems is still a relatively new concept, and there are many more investigations that can be carried out to enhance GP-HH approaches. The following are a few remarks on the existing GP-HH approaches to scheduling problems.

- Most of the existing GP methods to JSS problems evolve single priority dispatching rules, which need to make complex decisions when

scheduling the jobs onto the machines. Some of these complex decisions made early in the process can drastically impact the final outcome, meaning that they will need to be handled carefully. However, due to the fact that dispatching rules are myopic in nature [18], a single rule is more likely to make a particularly poor decision than a group of rules working together [126].

- Although there are many meta-heuristic approaches to dynamic scheduling problems with breakdowns, there are only a very limited number of hyper-heuristic approaches to handling dynamic scheduling problems with machine breakdowns. In addition, the meta-heuristic approaches primarily focus on dynamic scheduling problems with fixed job arrivals and usually do not consider problems with multiple types of dynamic events. Yin et al. [155] have proposed a GP-HH approach to a single-machine JSS problem with machine breakdown, but the problem instances solved by the GP approach are relatively small and do not include dynamic job arrivals. Holthaus [61] has studied multiple man-made dispatching rules for dynamic JSS problems with machine breakdowns, but no work has proposed a GP-HH approach for dynamic JSS problems with machine breakdowns.

- Finally, although GP-HH approaches that handle multiple scheduling decisions have been proposed in the literature [107, 156], they have not considered separating out problems into explicit tasks to handle simultaneously. In addition, evolutionary multitasking approaches have been effectively applied to various continuous and combinatorial optimisation problems, but they have been applied to the problems as *meta-heuristics* and have not been applied to DJSS problems. In other words, a hyper-heuristic approach to DJSS problem that can incorporate multitask learning has not yet been proposed in the literature.

This thesis will propose new approaches that mainly aim to address

these limitations.

Chapter 3

Ensemble-based GP Approaches to DJSS

3.1 Introduction

GP-HH approaches can evolve effective dispatching rules for the DJSS problem with dynamic job arrivals. However, a GP evolved dispatching rule needs to be able to handle complex decisions when it handles a DJSS problem instance, and it is possible that such decisions can be better handled using multiple rules. Ensemble learning has been shown to improve the generalisation ability of algorithms in problems outside of JSS [45, 126, 157], and is likely to improve the effectiveness of rules evolved by GP in terms of the performance and may likely make the evolved rules less sensitive to the changes in the properties of the DJSS problem.

3.1.1 Chapter Goals

The goal of this chapter is to develop GP approaches that evolve effective ensemble of dispatching rules for a DJSS problem for the purposes of minimising mean tardiness. The ensemble of dispatching rule is expected to be more effective on the DJSS problem than the standard GP approach.

In other words, the ensemble GP approach must be more effective than the standard GP approach on the problem domain that the rule is evolved on, and on the unseen problem domains. To achieve this goal, we develop novel ensemble GP approaches that can evolve diverse ensembles of dispatching rules. This objective is broken down into three major sub-objectives:

- First, this chapter develops ensemble GP approaches for the DJSS problem by incorporating an ensemble algorithm from the literature. Various ensemble algorithms have been proposed in the literature for problems outside of DJSS (e.g. such as difficult classification [126]), but no ensemble GP have been proposed in the literature for DJSS problems with dynamic job arrivals.

  A major design consideration that needs to be made when developing an ensemble GP approach is that applying a GP individual to each training instance is very expensive in a DJSS problem with dynamic job arrivals. The DJSS problem often requires simulations with thousands of jobs [105] compared to static problems which typically have at most 100 jobs [141]). This means that the number of training instances tends to be limited for DJSS problems. Therefore, this thesis will focus on ensemble approaches that can evolve the ensemble components simultaneously instead of using ensemble frameworks that train ensemble components separately on different training sets (e.g. bagging [19] and boosting [134]).

- Second, this chapter investigates several combination schemes used by the ensemble GP approaches to improve the effectiveness of ensemble rules evolved by the GP approaches. The method for combining the outputs of the ensemble components is an important contributing factor to the overall effectiveness of the ensemble when they are applied to classification problems [126]. Because of this, its likely that the ensemble combination scheme is also a key com-

ponent for the ensemble GP process when evolving ensemble rules for DJSS problems, and can likely further improve the overall quality of the ensembles evolved by the GP approach. We can likely obtain more insight into the properties of the DJSS problem from the relative performances of the combination schemes and the behaviours of the ensembles evolved by the different combination schemes.

- Finally, this chapter proposes new analysis measures to compare specific behaviours between the evolved ensembles from the different combination schemes. Analysis of evolved rule behaviours is important, as this helps provide insight into how GP evolves the ensemble of dispatching rules and the DJSS problem itself. Identifying the strengths and weaknesses of the different ensemble GP approaches and the combination schemes used can help develop more effective ensemble GP approaches in the future.

3.1.2 Chapter Organisation

This chapter is divided into two major parts. First, we provide an overview of the two ensemble GP approaches investigated, followed by the experimental design and the results that compare the two ensemble GP approaches. Afterwards, the remaining sections cover the ensemble combination schemes incorporated into the ensemble GP approach that performs the better out of the two approaches, followed by the diversity measures used to analyse the behaviours of the ensembles evolved by GP. This is followed by experimental design, the results and the discussion for the ensemble combination schemes. Finally, a chapter summary that wraps up this chapter is provided.

3.2 Ensemble GP Algorithms

This section describes the two ensemble GP algorithms that have been developed for the DJSS problem in this thesis. The first ensemble GP algorithm is called "ensemble genetic programming for job shop scheduling" (EGP-JSS), which incorporates Potter and De Jong's *cooperative coevolution* algorithm [127] into the GP process to evolve ensembles of dispatching rules for the DJSS problem. The second ensemble GP algorithm is called "multilevel genetic programming for job shop scheduling" (MLGP-JSS), which incorporates Wu and Banzhaf's multilevel genetic programming to the GP process to evolve ensembles for the DJSS problem [153]. First, we will cover the overall GP process for the two ensemble GP approaches, describe the evaluation procedure for the GP individuals and detail the selection and breeding procedures.

3.2.1 EGP-JSS Overview

EGP-JSS approach evolves ensembles of dispatching rules by *partitioning* the GP population into S evenly sized subpopulations of size K that interact via *representatives* of each subpopulation. A representative of a subpopulation is the individual with the best fitness out of the subpopulation found so far. If the individuals have not been evaluated yet, i.e., straight after the GP population has been initialised, then the representative of a subpopulation is a random individual out of the subpopulation. The interactions between the subpopulations are limited to these representatives, and selection and breeding are carried out independently for each subpopulation. The advantage of Potter and De Jong's cooperative coevolution [127] is that it does not require the ensemble members to be trained on different problem instances (unlike other ensemble approaches such as bagging [19] or boosting [134]). This is ideal for DJSS problems with dynamic job arrivals due to the relatively expensive computation cost required to apply a GP individual to a single training instance. For example,

in Hildebrandt and Branke's [59] DJSS simulation model, a GP individual needs to complete 2500 jobs in a training instance before the simulation is completed, and only 10 training simulations are carried out to evaluate a GP individual. On the other hand, Taillard's dataset [141], a popular static JSS dataset, has at most 100 jobs in a problem instance.

A key concept in EGP-JSS and Potter and De Jong's cooperative coevolution [127] is the "collaboration" between the individuals in a GP subpopulation with the representatives of the other subpopulations to solve the DJSS problem. To evaluate an individual x in a subpopulation s_i in EGP-JSS, the individual is paired up with the representatives $r_1, \ldots, r_S$ from subpopulations $s_1, \ldots, s_S$ *minus* the representative r_i from subpopulation s_i. The individual and the representatives form an *ensemble* E that is then applied to the DJSS problem instances $I_1, \ldots, I_{|\mathcal{T}|}$ in the training set $\mathcal{T}$. Applying the ensemble to a training instance as described in Section 3.2.3 to generate a schedule for the DJSS problem. Potter and De Jong's cooperative coevolutionary approach allows for individuals to collaborate with the members of the other subpopulation by forming the ensemble during the evaluation procedure. For example, in an island models [88, 145], the individuals are transferred between the different subpopulations (i.e. "islands") during the selection and breeding procedure, but evaluation of an individual is carried out independent of individuals in other subpopulations. The individual being evaluated need to collaborate effectively with the representatives and contribute towards ensuring that the ensemble makes good decisions so that the ensemble has a good performance on the training set.

After the schedule is generated for all DJSS problem instances in the training set using the ensemble, the fitness $f(x)$ of individual x is calculated from the ensemble's performance over the training set $\mathcal{T}$ as given in Equation (3.2) in Section 3.2.3. This is done for all individuals in the subpopulations, and for all subpopulations of the GP population. Afterwards, the representatives of the subpopulations are updated to the best individ-

uals in the subpopulations, and the selection and breeding procedure is carried out. The output of EGP-JSS process is the ensemble E_{best} with the best performance over the training set $\mathcal{T}$ out of the ensembles evaluated in the last generation. The EGP-JSS process described above is given in Algorithm 1.

Unlike Potter and De Jong's cooperative coevolutionary approach, a subpopulation is not destroyed if they are unproductive (i.e. the fitness of the representative do not improve after a certain number of generations). This is because destroying and regenerating a new subpopulation of individuals will require a large number of generations for it to be effective, which would make the EGP-JSS process either prohibitively expensive in terms of computation cost, or evolve ineffective rules when the time budget is limited.

3.2.2 MLGP-JSS Overview

Compared to the EGP-JSS approach that partitions the GP population into multiple subpopulations, the MLGP-JSS approach has a single GP population and combines the individual into *groups* that cooperate with each other. In other words, the subpopulations are predefined in EGP-JSS, whereas the groups are generated during the evolutionary process for MLGP-JSS. The MLGP-JSS process is broken down into three major steps, which consists of two "levels" of evolution. After the population of GP individuals have been initialised and evaluated, the first major step of MLGP-JSS carries out evolution on the *group level*, where groups are bred, evaluated and added to the GP population. The second step is to carry out evolution on the *individual level*, where GP individuals are bred, evaluated and added to the population. The final step is the selection procedure, where only the elite groups and individuals are retained in the population for the next generation. After the termination criterion is reached, i.e., the maximum number of generations, the final output is the best group of in-

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

Input : Training set $\mathcal{T}$
Output: Best ensemble E_{best}
initialise GP subpopulations $\mathcal{P}_1, \ldots, \mathcal{P}_S$ of sizes K;
for $i \leftarrow 1$ **to** S **do**
    choose random individual from subpopulation $\mathcal{P}_i$ as representative r_i;
end
for $gen \leftarrow 1$ **to** G **do**
    for $i \leftarrow 1$ **to** S **do**
        for *each individual* x *in* $\mathcal{P}_i$ **do**
            form ensemble $E = \{x, r_1, \ldots, r_S\} - \{r_i\}$;
            evaluate ensemble E on training set $\mathcal{T}$ and calculate individual fitness $f(x)$ (Equation (3.2)) ;
        end
    end
    update representatives $r_1, \ldots, r_S$;
    update best ensemble E_{best};
    for $i \leftarrow 1$ **to** S **do**
        carry out selection and breeding for subpopulation $\mathcal{P}_i$;
    end
end

dividuals found so far (g_{best}). The overall MLGP-JSS process is shown in Algorithm 2, where M_{breed} is the number of groups bred at each generation, and N_{breed} is the number of individuals bred at each generation.

Evolution on the Group Level:

There are three evolutionary operators which breed new groups from existing individuals and groups in the population. The first operator is *cooperation*. Cooperation combines two *entities* together to form a new group containing all individuals from both entities without duplicates. An entity can be either an individual or a group. This means that two individuals, an individual and a group, or two groups can be merged to form another

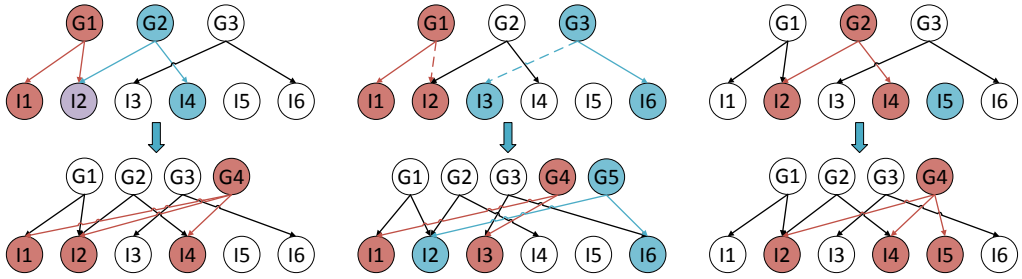
Algorithm 2: The pseudocode for the MLGP-JSS approach.

Input : Training set $\mathcal{T}$
Output: Best group g_{best}
initialise population $\mathcal{P}$ with $N_{retained}$ individuals;
for *each individual* $x \in \mathcal{P}$ **do**
    evaluate individual x on training set $\mathcal{T}$ and calculate fitness $f(x)$;
end
for $gen \leftarrow 1$ **to** G **do**
    for $i \leftarrow 1$ **to** M_{breed} **do**
        breed group g from population $\mathcal{P}$;
        evaluate group g on training set $\mathcal{T}$ and calculate fitness $f(g)$;
        add group g to population $\mathcal{P}$;
    end
    for $i \leftarrow 1$ **to** N_{breed} **do**
        breed individual x from population $\mathcal{P}$;
        evaluate individual x on training set $\mathcal{T}$ and calculate fitness $f(x)$;
        add individual x to population $\mathcal{P}$;
    end
    update the best group g_{best};
    retain $M_{retained}$ best groups in population $\mathcal{P}$;
    retain $N_{retained}$ best individuals from population $\mathcal{P}$;
end

group. The entities for cooperation are selected using roulette wheel selection over all entities [153]. An example is shown in Figure 3.1a, where group G1 is combined with group G2 to form the new group G4, which contains individuals I1, I2 and I4.

The second evolutionary operator on the group level is the group crossover operator. In group crossover, roulette wheel selection selects two groups from the GP population as parents [153]. The two parents randomly exchange one individual to produce the child groups. Individuals in a parent group have equal probabilities of being exchanged. In Figure 3.1b, crossover occurs between parents G1 and G3, exchanging individuals I2 and I3 respectively. This generates groups G4 and G5.

The final evolutionary operator on the group level is the group mutation operator. In the group mutation, a group is selected through roulette wheel selection, and either an individual is added to or removed from the group [153]. If an individual is being added, then an individual is selected through roulette wheel selection over the individuals in the GP population. If an individual is being removed, then an individual in the group is randomly selected with uniform probability. An example of a mutation operator adding an individual is shown in Figure 3.1c, where individual I5 is added to G2 to produce offspring G4.



(a) Cooperation Operator (b) Crossover Operator (c) Mutation Operator

Figure 3.1: Examples of group operators used to breed new groups.

Evolution on the Individual Level:

Crossover and mutation operators used to breed the individuals are the standard operators for tree-based GP [80]. To select the parent individuals to breed new offspring, a group g is selected based on a probability proportional to the group's fitness. It is expected that individuals in a group contribute to achieving cooperation despite their fitnesses [153]. Therefore, an appropriate number of parent individuals are selected from group g with a uniform probability. After crossover or mutation is carried out, the newly bred children do not automatically become part of the group from which the parent individuals were selected from, but are inserted back into the pool of individuals after evaluating the children's fitness.

3.2.3 Evaluation Procedures

The evaluation procedure for GP individuals differs between the EGP-JSS and the MLGP-JSS approaches. In an EGP-JSS approach, an ensemble generated from a GP individual collaborating the representatives is not assigned a fitness value, whereas a group in the MLGP-JSS approach has a separate fitness value from the GP individuals that make up the group. To evaluate a GP individual x for the EGP-JSS approach, the ensemble E that combines individual x with the representatives from the other subpopulations is applied to the training set $\mathcal{T}$ as a *non-delay* ensemble of dispatching rules. To apply an ensemble E of dispatching rule to a DJSS problem instance I, the decision situation that occurs during processing is handled by ensemble E by calculating the priorities of the jobs using the ensemble members and then the priority results are combined using a combination scheme. For the EGP-JSS and the MLGP-JSS approaches, we use the *majority voting* combination scheme. Figure 3.2 shows majority voting with three ensemble members being applied jointly to a decision situation with five jobs. In ensembles with majority voting, the job with the highest number of votes is selected to be processed. In the figure, rule 1 votes on job 4 as job 4 is assigned the highest priority by rule 1 out of the five jobs. On the other hand, rules 2 and 3 vote on job 1. This results in job 1 being selected by the ensemble to be processed by the machine at the decision situation. In the decision situations where two jobs are tied in the number of votes, then we use the apparent tardiness cost (ATC) [148] rule as a tie-breaker. ATC is a man-made dispatching rule effective for tardiness related objectives. This process continues until all arriving jobs have been processed by the machines and a schedule is generated.

After the ensemble E is applied to the DJSS problem instance I and a schedule is generated, the mean tardiness $Obj(E, I)$ of the schedule is *normalised* using a *reference rule* R. This procedure adopts Hildebrandt et al. [60] and Mei et al.'s [97] GP approaches to reduce bias towards specific DJSS problem instances. If a particular DJSS problem instance has a

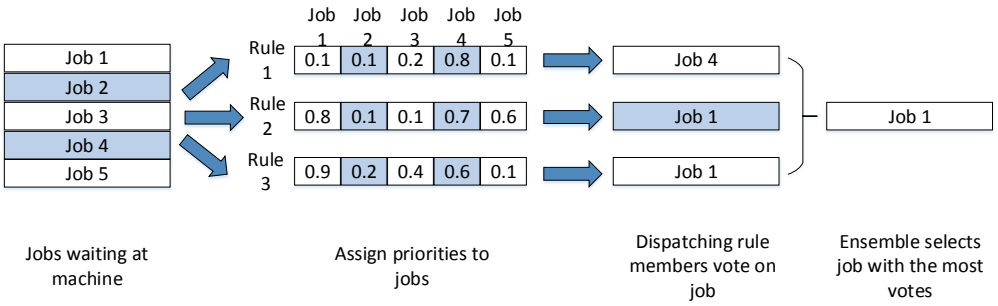


Figure 3.2: Example of majority voting for ensembles being applied to a decision situation.

greater optimal mean tardiness value than another DJSS 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. The reference rule R is applied to the same problem instance I as the ensemble E, and the mean tardiness $Obj(R, I)$ of the schedule generated by the reference rule is used to normalise the objective value of ensemble's schedule as shown in Equation (3.1). The reference rule used for the ensemble GP approaches is the ATC rule [148]. This normalisation procedure also normalises the mean tardiness values of the schedules generated by the GP individuals in MLGP-JSS approach that are applied to the training instances as single priority dispatching rules.

$$Obj'(E, I) = \frac{Obj(E, I)}{Obj(R, I)} \tag{3.1}$$

For the EGP-JSS approach, the individual x being evaluated is combined into an ensemble E and the ensemble is aplied to all problem instances. Afterwards, the fitness $f(x)$ of the individual x is calculated by averaging out the normalised mean tardiness values over the training instances as shown in Equation (3.2).

$$f(x) = \frac{1}{|\mathcal{T}|} \sum_{I \in \mathcal{T}} Obj'(E, I) \tag{3.2}$$

On the other hand, the fitness of a group g is separate from the fitness of GP individuals in the MLGP-JSS approach. The fitness of a group g is calculated by applying the group to the training set $\mathcal{T}$ as a non-delay ensemble of dispatching rules. The performance of the group g on the training set $\mathcal{T}$ is the average of normalised mean tardiness values, and the fitness $f(g)$ is calculated as shown in Equation (3.3). The equation is modified from Wu and Banzhaf's [153]'s group fitness calculation. In Wu and Banzhaf's group fitness calculation, the performance of group g is multiplied by a penalty factor that incorporates the group size GS_g. This prevents the group from increasing in size with minimal improvement. After some preliminary testing, the penalty factor was adjusted for the MLGP-JSS approach as shown in the equation.

$$f(g) = \frac{1}{|\mathcal{T}|} \sum_{I \in \mathcal{T}} Obj'(g, I) \times \left[1 + \frac{(\text{GS}_g - 3)^2}{(\text{GS}_g \times 3)}\right] \tag{3.3}$$

On the other hand, the fitness of an individual x for MLGP-JSS is calculated by applying the individual to the training set $\mathcal{T}$ as a single non-delay priority dispatching rule (described in Section 2.3.1 (Page 37)). The normalised mean tardiness values from individual x's schedules are used directly as the fitness of the individual, i.e., $f(x) = \frac{1}{|\mathcal{T}|} \sum_{I \in \mathcal{T}} Obj'(x, I)$.

3.3 Ensemble GP Experimental Design

This section covers the design of the experiments carried out to evaluate the EGP-JSS and the MLGP-JSS approaches. First, we discuss the DJSS model used to evaluate the GP approaches. This is then followed by the GP benchmark used for comparison. We use GP as the benchmark for the two ensemble GP approaches, as they are the state-of-the-art for the

DJSS problem with dynamic job arrivals [17]. Other methods of handling DJSS problems with a large number of dynamic job arrivals includes dispatching rules [63, 72, 73] and variable neighbourhood search [5, 4], both of which are outperformed by GP in the literature [109]. Afterwards, we cover the GP representation, terminal and function sets used by all GP approaches evaluated in the experiments and the parameter settings for the GP approaches.

3.3.1 DJSS Simulation Model

Discrete-event simulations are used in the literature to simulate DJSS problem instances with large number of dynamic job arrivals [66, 67, 60, 104, 105, 124]. In a discrete-event simulation, the job arrivals and the jobs' properties (e.g. the job's operations and their processing times) are generated stochastically from probability distributions with a given seed. For this thesis, we denote the configuration of distribution models used to simulate the DJSS problem instance as a *simulation configuration*.

To evaluate the EGP-JSS and the MLGP-JSS approaches, we use an existing simulation model proposed by Hunt et al. [66]. Hunt et al.'s model has a diverse range of simulations with different properties and provides a reliable starting point to evaluate the ensemble GP approaches. In Hunt et al.'s simulation model [66], two training sets $4op$ and $8op$ are used to evolve the rules. The job arrival times follow a Poisson process with a mean of λ, which is called the *arrival rate*. The arrival rate is calculated from the *utilisation rate* ρ, the mean processing time of operations of the arriving jobs μ, and the mean proportion of the machines required by the job to process all operations p_M. The arrival rate given the three parameters is shown in Equation (3.4). The utilisation rate is the proportion of time machine spends processing job operations on the shop floor. The processing times of job operations follow a discrete uniform distribution that has a lower bound of 1 and an upper bound of $2\mu - 1$, i.e., has an interval

$[1, 2\mu - 1]$. The number of operations per job N_j depends on the simulation configuration, and there are 10 machines on the shop floor, which means that $p_M = N_j/10$.

$$\lambda = \mu \times \rho \times p_M \tag{3.4}$$

After a job j's arrival time r_j and processing times $p_{1j}, \ldots, p_{N_j j}$ of the operations have been stochastically generated, the due date d_j of job j is calculated as shown in Equation (3.5). In the equation, h denotes the *tightness* of the due date and controls the urgency of the arriving jobs. In Hunt et al.'s simulation model, the tightness used to calculate the due date of the job is random from 3, 5, and 7 with equal probabilities for the training sets, and random from 2, 4, and 6 with equal probabilities for the test set.

$$d_j = r_j + h \times \sum_{i=1}^{N_j} p_{ij} \tag{3.5}$$

The remainder of the parameter values for Hunt et al's simulation model [66] are given in Table 3.1. In the table, the parameter with multiple values (excluding the tightness parameter h) indicates that the specific dataset has different simulation configurations that use different values of the parameter. For example, $4op$ and $8op$ have two different configurations with utilisation rates of 85% and 95%, meaning that each GP individual is applied two simulation runs during evaluation. On the other hand, the test set consists of $2 \times 2 \times 5 = 20$ different configurations with different combinations of mean processing time, utilisation rate and the number of operations per job parameter values. $\mathrm{Unif}\,[2, 10]$ value in the number of operations per job parameter for the test set denotes that the number of operations per job follows a discrete uniform distribution with the interval $[2, 10]$.

Table 3.1: Simulation configurations used for the generating arriving jobs in DJSS problem instances.

Parameter	$4op$	$8op$	Test
Warm-up period	500		
Max jobs completed	2500		
Mean processing time (μ)	25		$25, 50$
Utilisation rate (ρ)	$85\%, 95\%$		$90\%, 97\%$
# of operations per job (N_j)	4	8	$4, 6, 8, 10$, Unif $[2, 10]$
# of configurations	2	2	20

3.3.2 GP Benchmark

As GP has been the most popular automatic heuristic design approach in the recent years [103], we use the "standard" GP-HH approach as the benchmark to evaluate the ensemble GP approaches investigated [17]. In this case, a standard GP-HH approach evolves a single priority dispatching rule via the GP process shown in Section 2.3 (Page 37), which is one of the most prominent methods of evolving dispatching rules using GP [103]. To ensure consistency between the ensemble GP approaches and the benchmark GP, we use the same normalisation procedure (Equation (3.1)) and the same fitness function for the GP individual (Equation (3.2)) for the benchmark GP process. In addition, the GP representation, terminal set, function set and parameter settings that are described in Sections 3.3.3 and 3.3.4 below are kept consistent between all the GP approaches.

3.3.3 GP Representation, Terminals and Function Sets

The terminal set used by GP approaches consist of a mixture of terminal sets used by existing GP-HH approaches in the literature [60, 66, 105]. 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 [66]. The function set consists of the arithmetic operators $+$, $-$, $\times$, protected $/$, `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. After carrying out some manual sensitivity analysis, we determined that an error margin of $\epsilon = 10^{-6}$ was sufficient to account for rounding errors that occurred for the protected division operator, and is used as the error margin for the protected division operator used in this thesis. `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 3.2.

Table 3.2: The terminal set and the function set used for the GP approaches, where job j is one of the jobs waiting at the machine m to process operation o_{ij}.

Terminal	Description
RJ	The operation ready time of job j
RO	Remaining number of operations of job j
RT	Remaining total processing times of job j
PT	The 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	$+$, $-$, $\times$, $/$, `if`, $\max$, $\min$

3.3.4 GP Parameter Settings

The GP parameters required for the EGP-JSS, the MLGP-JSS and the benchmark GP approaches are shown in Table 3.3. In the table, the "Common Parameters" category are parameter values shared by all GP approaches, the "EGP-JSS Parameters" and "MLGP-JSS Parameters" are the parameters required by the EGP-JSS and the MLGP-JSS approaches respectively.

Table 3.3: GP parameters used by the EGP-JSS, the MLGP-JSS and the benchmark GP approaches for evolving rules.

Approach	Parameters	Value
Common Parameters	Total population size	1024
	No. of generations	51
	GP crossover / mutation / reproduction rate	80% / 10% / 10%
	GP maximum initial depth	8
	GP maximum depth	17
	Selection method	Tournament selection of size 7
	No. of elites per subpop	1
EGP-JSS Parameters	No. of GP subpopulations	4
	Subpopulation size	256
MLGP-JSS Parameters	No. of groups breed	200
	No. of groups retained	100
	Group cooperation / crossover / mutation rate	31.25% / 50% / 18.75%

The common parameters listed in Table 3.3 are modified from parameter settings used by Hunt et al. [66] for their GP approach to a DJSS problem with dynamic job arrivals. The major differences of the common parameter settings from Hunt et al.'s setting are the crossover, mutation, reproduction rates and the maximum depth of the trees. The crossover, mutation and reproduction rates are set to 80%, 10% and 10% respectively, where we increased the reproduction rate slightly so that more individuals from the current generation could be retained in the next generation

without their behaviour being significantly modified from the crossover and the mutation operators. In addition, the maximum depth of the trees is adapted from Koza's GP parameters [80]. This may be beneficial for the EGP-JSS and MLGP-JSS approaches to evolve good individuals as representatives. In addition, total population sizes across different GP approaches are set to be identical. The EGP-JSS approach has four subpopulations of size 256 that equals $4 \times 256 = 1024$ in total. Four subpopulation size was arbitrary selected after we found that the number of subpopulations ranging from three to ten did not significantly affect the performance in a preliminary test. On the other hand, MLGP-JSS approach breeds the same number of GP individuals as the benchmark GP during the breeding procedure and retains 1024 individuals. The number of groups bred and retained was chosen after carrying out preliminary parameter tuning. The group cooperation, crossover, and mutation rates are adapted from Wu and Banzhaf's original parameter settings [153].

3.4 Ensemble GP Approach Results

This section covers the evaluation of EGP-JSS and MLGP-JSS approaches and the combination schemes integrated with one of the ensemble GP approaches. To compare the benchmark GP, the EGP-JSS and the MLGP-JSS approaches, each GP approach is applied to the training set to evolve a dispatching rule. This is repeated until a set of thirty rules are evolved by each GP approach on the training set, which is equivalent to the number of runs carried out by existing GP-HH approaches in the literature [105, 43, 42]. 30 independent runs allows us to have enough random sample to carry out the Wilcoxon's signed rank test to compare two different GP approaches against each other [105, 43, 42]. Afterwards, the rules are evaluated on the test set as described below to determine which ensemble GP approach is more promising so that it can be investigated further in terms of the combination schemes. In addition, the best rules from the

GP approaches are sampled from the GP rule sets based on their average performance over the entire test set.

3.4.1 Rule Set Comparison

After the 30 rules are evolved by the GP approaches, the evolved rules are applied to the simulations in the test set. The test set consists of 20 different simulation configurations, each with different processing times (μ), utilisation rates (ρ) and the number of operations per job (N_j). A rule is applied to a simulation with a specific simulation configuration 30 times with different seeds to obtain 30 independent results. Obtaining 30 results for each simulation configuration for a rule allows us to get a good representatation of the rule's performance on the simulation configuration and allows us to carry out statistical testing between the two GP rules using Wilcoxon's signed-rank test. This procedure is repeated for all subsequent GP comparisons that are carried out in this thesis.

After we obtain 30 mean tardiness results for a GP rule from the DJSS simulations for a simulation configuration, the "performance" of the GP rule on the simulation configuration is the average mean tardiness value of the 30 results. The performances of the GP rules are then used to compare the GP approaches themselves, i.e., we compare the performances of the rules evolved by EGP-JSS and the benchmark GP to carry out an overall comparison of EGP-JSS and the benchmark GP (and likewise for MLGP-JSS and the benchmark GP). A set of rule performs *significantly* better on the simulation configuration if the difference in the performances between the two rule sets satisfies the two-sided statistical test at $p = 0.05$. The performances of the GP rules are shown in Table 3.4. The test set simulation configurations shown in the table is partitioned into four subsets based on expected processing time (μ) and the utilisation rate (ρ). $\langle x, y, z \rangle$ category denote that the configuration has $\mu = x$, $\rho = y$ and $N_j = z$. In addition, $x \pm y$ denotes that the rule set has x mean and y standard devi-

ation performance on the particular simulation configuration. The corresponding EGP-JSS or MLGP-JSS result is highlighted blue if the EGP-JSS or the MLGP-JSS rules significantly outperforms the benchmark GP rules for the simulation configuration. In addition, the evolution times in the table denotes the mean times taken to evolve the thirty rules for the three GP approaches.

As we can see from the results in Table 3.4, both EGP-JSS and MLGP-JSS generally perform better than the benchmark GP approach. For the rules evolved on $4op$, EGP-JSS and MLGP-JSS perform significantly better than the benchmark GP rules in certain simulation configurations (e.g. $\langle\ 50,\ 90\%,\ [2, 10]\ \rangle$ and $\langle\ 25,\ 97\%,\ 8\ \rangle$ for both EGP-JSS and MLGP-JSS). They also do not have worse performance in simulation configurations where they do not perform significantly better than the benchmark GP. For the rules evolved on $8op$, EGP-JSS performs significantly better than the benchmark GP for all configurations and MLGP-JSS performs significantly better on the simulation configurations $\langle\ 25,\ 97\%,\ 6\ \rangle$ and $\langle\ 50,\ 97\%,\ 4\ \rangle$. In general, EGP-JSS performs better against the benchmark GP than MLGP-JSS over a greater number of simulation configurations.

A possible reason why the EGP-JSS approach performs better than the MLGP-JSS approach is that MLGP-JSS uses roulette wheel selection for choosing different the parents for the group cooperation, crossover and mutation operator like the MLGP approach by Wu and Banzhaf [153]. During the training procedure, the groups generally have similar fitness to each other. This means that the roulette wheel may not be biased enough towards a potentially good group, resulting in a lack of exploitation. Therefore, as EGP-JSS generally show better performance than MLGP-JSS, we will investigate the combination schemes for the EGP-JSS approach.

Table 3.4: Comparison of the performances of the GP approaches over the test simulation model. Rules are evolved from $4op$ and $8op$.

Data Subset		$4op$			$8op$		
		EGP-JSS	MLGP-JSS	GP	EGP-JSS	MLGP-JSS	GP
Evolution Time (s)		16555	7079	6701	21128	6202	5418
MT ($\times 10^2$)	$\langle 25, 90\%, 4 \rangle$	1.49 ± 0.06	1.53 ± 0.58	1.55 ± 0.18	1.73 ± 0.16	1.83 ± 0.75	1.92 ± 0.32
	$\langle 25, 90\%, 6 \rangle$	2.18 ± 0.10	2.22 ± 0.99	2.30 ± 0.27	2.45 ± 0.21	2.61 ± 1.25	2.77 ± 0.48
	$\langle 25, 90\%, 8 \rangle$	3.64 ± 0.24	3.90 ± 1.21	3.85 ± 0.80	4.71 ± 0.73	5.24 ± 1.62	5.29 ± 0.94
	$\langle 25, 90\%, 10 \rangle$	6.53 ± 0.43	6.83 ± 2.38	6.92 ± 1.39	8.16 ± 1.17	9.11 ± 3.32	9.37 ± 1.66
	$\langle 25, 90\%, [2, 10] \rangle$	1.30 ± 0.07	1.34 ± 0.50	1.42 ± 0.20	1.41 ± 0.12	1.48 ± 0.59	1.58 ± 0.31
	$\langle 50, 90\%, 4 \rangle$	2.74 ± 0.15	2.83 ± 1.24	3.00 ± 0.43	3.02 ± 0.24	3.25 ± 1.56	3.56 ± 0.77
	$\langle 50, 90\%, 6 \rangle$	5.46 ± 0.30	5.83 ± 2.25	5.82 ± 1.01	6.60 ± 0.73	7.21 ± 2.83	7.40 ± 1.00
	$\langle 50, 90\%, 8 \rangle$	8.28 ± 0.51	8.62 ± 3.20	8.74 ± 1.41	10.04 ± 1.12	10.95 ± 4.28	11.44 ± 1.83
	$\langle 50, 90\%, 10 \rangle$	1.53 ± 0.10	1.56 ± 0.66	1.71 ± 0.27	1.58 ± 0.15	1.65 ± 0.74	1.79 ± 0.37
	$\langle 50, 90\%, [2, 10] \rangle$	3.19 ± 0.19	3.29 ± 1.57	3.57 ± 0.56	3.32 ± 0.30	3.51 ± 1.85	3.83 ± 0.85
	$\langle 25, 97\%, 4 \rangle$	8.53 ± 0.51	9.07 ± 3.92	9.12 ± 1.65	9.92 ± 0.84	10.58 ± 4.51	10.93 ± 1.24
	$\langle 25, 97\%, 6 \rangle$	13.72 ± 0.76	14.43 ± 6.61	14.96 ± 2.63	15.83 ± 1.36	17.02 ± 7.96	17.90 ± 2.22
	$\langle 25, 97\%, 8 \rangle$	1.35 ± 0.12	1.40 ± 0.81	1.60 ± 0.31	1.31 ± 0.17	1.40 ± 0.87	1.50 ± 0.35
	$\langle 25, 97\%, 10 \rangle$	2.18 ± 0.24	2.27 ± 1.47	2.65 ± 0.60	2.07 ± 0.30	2.22 ± 1.64	2.46 ± 0.81
	$\langle 25, 97\%, [2, 10] \rangle$	9.63 ± 0.54	10.19 ± 4.65	10.42 ± 1.84	10.77 ± 0.70	11.32 ± 5.15	11.80 ± 1.34
	$\langle 50, 97\%, 4 \rangle$	13.99 ± 0.73	14.56 ± 6.75	15.37 ± 2.60	15.54 ± 0.92	16.33 ± 7.48	17.41 ± 2.28
	$\langle 50, 97\%, 6 \rangle$	1.49 ± 0.07	1.54 ± 1.28	1.65 ± 0.21	1.59 ± 0.14	1.68 ± 1.48	1.79 ± 0.29
	$\langle 50, 97\%, 8 \rangle$	2.48 ± 0.15	2.57 ± 1.29	2.80 ± 0.43	2.65 ± 0.24	2.83 ± 1.55	3.08 ± 0.62
	$\langle 50, 97\%, 10 \rangle$	5.55 ± 0.35	5.87 ± 2.64	5.90 ± 1.09	6.60 ± 0.74	7.19 ± 3.33	7.38 ± 0.98
	$\langle 50, 97\%, [2, 10] \rangle$	9.86 ± 0.66	10.29 ± 3.92	10.63 ± 2.06	11.71 ± 1.27	12.81 ± 5.21	13.33 ± 1.92

3.4.2 Best Rule Comparison

After comparing the rule sets, the best rules from the GP approaches are compared against each other and benchmark man-made dispatching rules. The benchmark man-made dispatching rules are earliest due date (EDD), first-in-first-out (FIFO), shortest processing time (SPT), ATC and cost over time (COVERT) [125, 148]. EDD, FIFO and SPT are simple but frequently used rules in the literature [125]. On the other hand, ATC and COVERT are effective rules for tardiness related objectives [148]. Comparing the best GP rules allows us to determine whether it is possible for an evolved single GP rule to handle the sequencing decisions as effectively as an evolved ensemble. If best single rules can perform competitively as the best ensembles, this implies that single rules can exhibit the similar behaviours as ensembles. The results for the best GP rules evolved on training set $4op$ are shown in Figure 3.3, and the best GP rules evolved on training set $8op$ are shown in Figure 3.4.

From the results shown in the figures, the best GP rules evolved by EGP-JSS, MLGP-JSS and the benchmark GP show comparable performances to each other. The best rules evolved on training set $4op$ outperform the benchmark man-made rules in all simulation configurations, whereas the best rules evolved on training set $8op$ outperform the man-made rules on simulations with a utilisation rate 90%, and do not have worse performance than the man-made rules on simulations with utilisation rate 97%.

The GP rules having comparable performance shows that GP can evolve single rules that can handle the complex sequencing decisions as effectively as ensemble rules. However, EGP-JSS and MLGP-JSS being better overall than the benchmark GP approach show that the GP can more consistently evolve good rules by handling the sequencing decisions using ensembles.

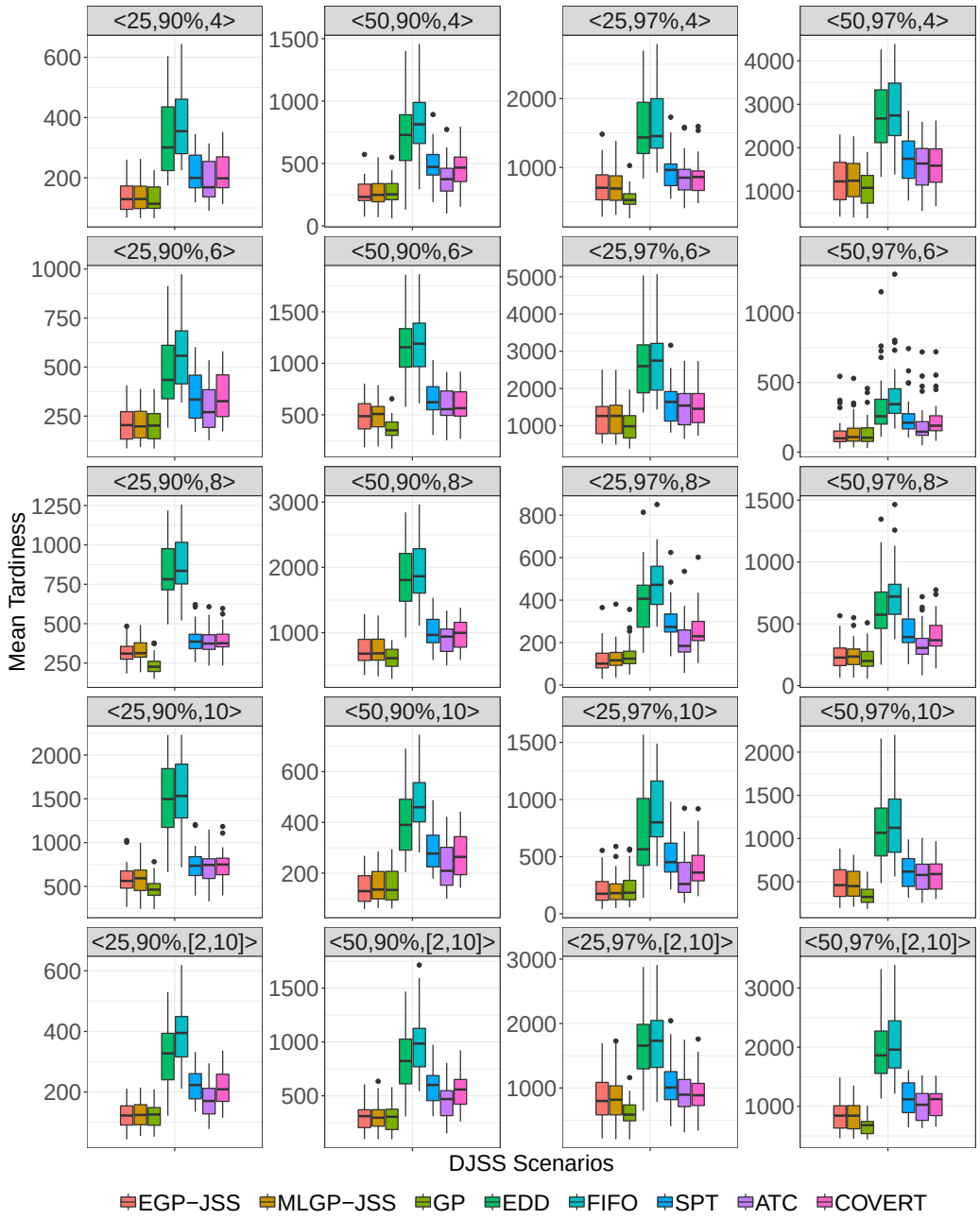


Figure 3.3: Comparison of the performances of the best GP rules and benchmark man-made rules over the test simulation model. Rules are evolved from $4op$.

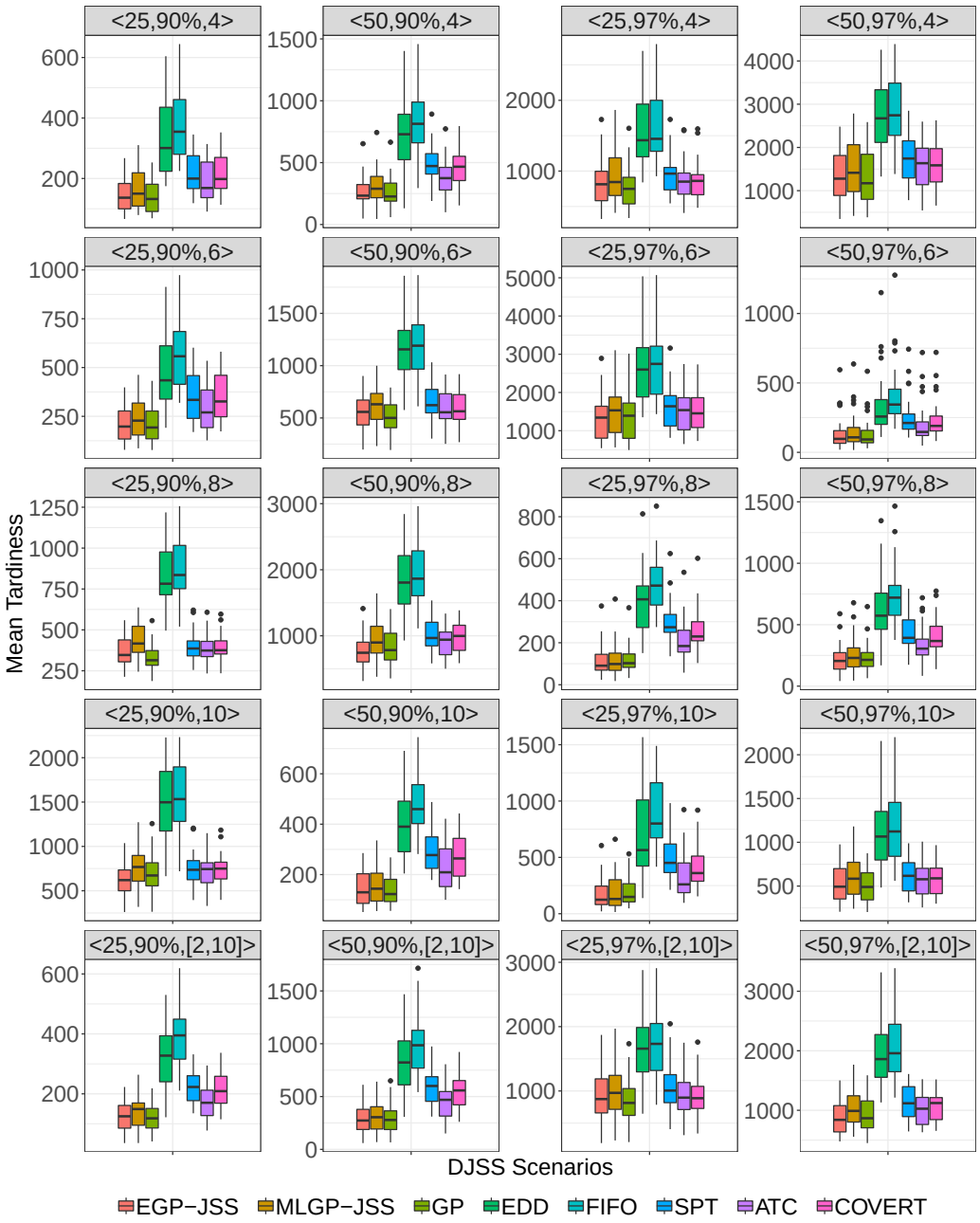


Figure 3.4: Comparison of the performances of the best GP rules and benchmark man-made rules over the test simulation model. Rules are evolved from $8op$.

3.5 Ensemble Combination Schemes

This section describes the combination schemes investigated for the EGP-JSS approach and the modifications to the EGP-JSS approach required to incorporate some of the combination schemes. The combination schemes investigated are majority voting, linear combination, weighted majority voting and weighted linear combination [126]. To evolve ensembles that use weighted majority voting and weighted linear combination scheme EGP-JSS, a genetic algorithm (GA) [81] subpopulation is added to the EGP-JSS approach where the GA individuals represent the weights of the ensemble members from the GP subpopulations.

3.5.1 Combination Schemes Investigated

To combine the ensemble member outputs during the job sequencing decisions at decision situations, the priority values $\delta_{j,x}$ assigned to a job j by an ensemble member x is converted to a score value $Score(j, x)$. After summing up the score values assigned to each job, the job with the highest sum score is selected by the ensemble to be processed by the machine at the decision situation. In addition, the ATC rule [148] is used as a tiebreaker in situations where two jobs with the highest scores have equal scores as each other. The priority to score conversion for each combination scheme is given in detail below.

Majority Voting:

As described above in Section 3.2.3, majority voting selects the job with the most number of votes given by the ensemble members. In other words, an ensemble member x contributes a score $Score(j, x) = 1$ if job j is assigned the highest priority by the ensemble member. Otherwise, ensemble member x contributes a score of zero to job j.

Linear Combination:

The score of a job assigned by a member is normalised from the assigned priorities using min-max normalisation. Given that a member x assigns priorities $\delta_{1,x}, \ldots, \delta_{K,x}$ to the K jobs waiting at the machine, the priority assigned to job j is converted into a score $Score(j, x)$ as shown in Equation (3.6). In the Equation, $\delta_{\min}$ and $\delta_{\max}$ are the minimum priority and maximum priority assigned to all jobs waiting at the machine by member x.

$$Score(j, x) = \frac{\delta_{j,x} - \delta_{\min}}{\delta_{\max} - \delta_{\min}} \tag{3.6}$$

Weighted Majority Voting:

Each member x has a weight W_x in weighted majority voting. The ensemble member x contributes score $Score(j, x) = W_x$ to a single job j with the highest assigned priority by ensemble member x, and zero scores to other jobs waiting at the machine at the decision situation.

Weighted Linear Combination:

Each member x has a weight W_x in weighted linear combination. An intermediate value is first calculated for a job j by ensemble member x by normalising the priorities (Equation (3.6)). Afterwards, the intermediate value is multiplied by the member's weight W_x to get the final score contributed towards job j by ensemble member x.

Combination Scheme Example:

When applied to a decision situation, different combination schemes can potentially select different jobs for processing. For example, Figure 3.5 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 EGP-JSS.

3.5.2 Incorporating Weighted Combination Schemes to EGP-JSS

To evolve ensembles that use weighted combination schemes, EGP-JSS's GP process is modified to evolve weights for 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 extra computation after the GP process is completed. Given that there are S GP subpopulations of size K, an additional $(S + 1)$th GA subpopulation of size K_W are also initialised along with the GP population. The GA individuals are real-valued vectors of length S, where the gene value at index i corresponds to the weight W_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

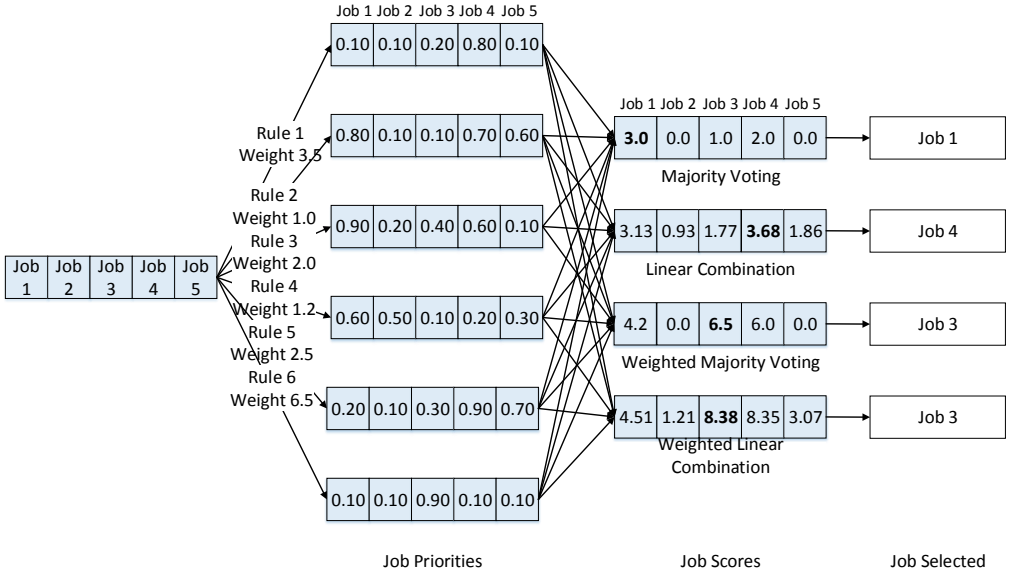


Figure 3.5: Examples of the combination schemes for an ensemble with six members being applied to a decision situation with five jobs waiting at the machine. The weights of the members are $W = 3.5, 1.0, 2.0, 1.2, 2.5, 6.5$ respectively.

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.

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 Figure 3.6, where the ensemble consists of GP individuals I1, I2 and I3 from the three GP subpopulations. Afterwards, I1, I2 and I3 are weighted using the values from GA 1. The weighted ensemble is then applied to the training instances, and the fitness of the GA individual is calculated as described in Section 3.2.3. 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 op-

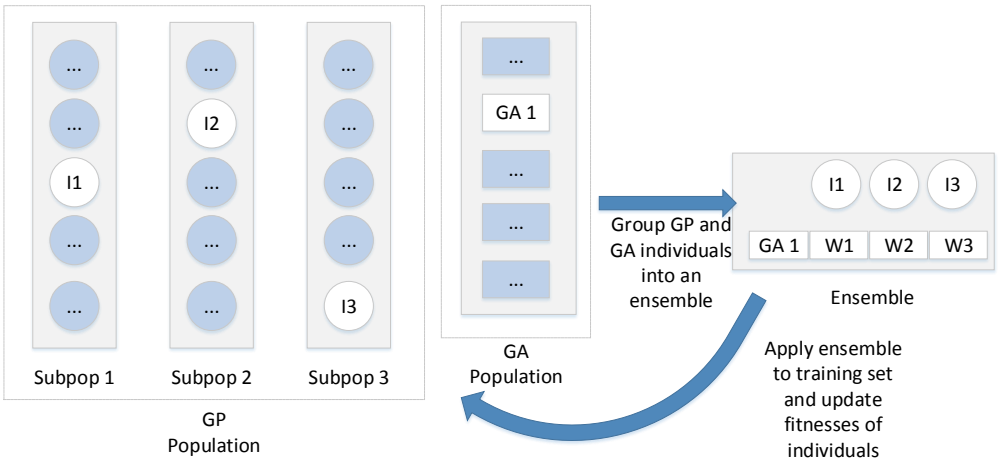


Figure 3.6: Example of modified EGP-JSS generating an ensemble with weighted members.

erator and a Gaussian mutation operator [81] 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.

3.6 Diversity Measures

Based on the EGP-JSS algorithm, we carry out several behavioural analyses of the evolved ensembles that are evolved with different combination schemes. Many researchers that have proposed GP-HH approaches in the literature to DJSS have also analysed the structures of the rules evolved by GP-HH (i.e. genotype [80]) and the behaviours (i.e. phenotype [80]) of evolved rules [17]. However, due to the limited ensemble evolutionary scheduling approaches to DJSS in the literature [42, 56], the amount of analysis on the behaviours of evolved ensembles is limited. For exam-

ple, Hart and Sim [56] have carried out both structural and behavioural analysis of ensembles evolved by their hyper-heuristic approach. They performed an extensive analysis of ensembles' performances and analyse the relation between the structural make-up of the ensembles (using the terminal distributions) and how well it solves 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 make diverse but effective decisions for the ensemble to be effective [126]. For example, one ensemble member may prioritise jobs with short processing times, and another ensemble member may prioritise jobs with urgent due dates. Both processing times and due dates are important attributes in a scheduling decision-making process [97], and each decision situation will likely require rules that are comprised of complex combinations of the shop floor attributes to make an effective decision. In classification, complex decisions occur at the class boundaries, where it is ambiguous whether an instance belongs to one class or another [126]. 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 [126]. 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 [126].

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 [17], 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 [59] and diversity measures [101]. 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.

3.6.1 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 DJSS problem instances. These decision situations are generated by applying a *sampling dispatching rule* over a number of 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, Ω decisions are uniformly 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 waiting jobs that may potentially lead to "bad" decisions if they are selected. 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, as they are sampled directly from the simulations, 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.

3.6.2 New Measure 1 – Decision Conflict (DC)

Decision conflict (DC) measure 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 j out of Ω sampled decisions, suppose there are K_j jobs waiting at the machine and the member x of ensemble E assigns the highest priority to a job $j_{j,x}$. Then the number V_{j,j^*} of members that assign the highest priority to job j^* at decision j is given by Equation (3.7). From this, we get DC(E) in Equation (3.8), where it is the proportion of decisions out of Ω decisions where there are at least two jobs are tied for the highest number of top priority assignment.

$$V_{j,j^*} = |\{x \in E | j^* = j_{j,x}\}| \tag{3.7}$$

$$\text{DC}(E) = \frac{1}{\Omega}|\{j \in [1, \ldots, \Omega], \exists i, j[i \neq j \wedge \max_k\{V_{j,k}\} = V_{j,i} = V_{j,j}]\}| \tag{3.8}$$

For the majority voting combination scheme, DC 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, the tiebreaker is unlikely to be used, as the jobs are selected based on the total scores that are real-number values. However, DC 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.

3.6.3 New Measure 2 – High Contribution Members (HC)

High contribution member (HC) measure 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 j out of Ω decisions, suppose that a member x of an ensemble E assigns the highest priority to job $j_{j,x}$. If a job is selected by ensemble E, then the decision between member x and ensemble E itself can be considered to be "overlapping" for decision j. In other words, given that ensemble E selects job $j_{j,E}$ at decision j, the overlap $q^{\text{HC}}_{x,E}$ between member x and ensemble E over Ω decisions is given by Equation (3.9). Afterwards, we get $\text{HC}(E)$ in Equation (3.10), where it is equal to the member with the most overlap with the ensemble over Ω decisions.

$$q^{\text{HC}}_{x,E} = \frac{1}{\Omega}|\{j \in [1, \ldots, \Omega] | j_{j,x} = j_{j,E}\}| \tag{3.9}$$

$$\text{HC}(E) = \max_{x}\{q^{\text{HC}}_{x,E}\} \tag{3.10}$$

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 HC 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 DJSS problem instance by itself. On the other hand, the converse (i.e. high HC and good test performance) will show that having a single highly biased member in the evolved ensembles may be more effective for scheduling.

3.6.4 New Measure 3 – Low Job Ranks Members (LJR)

Low job ranks (LJR) measure calculates the worst average ranks of the ensemble members using a rule ranking system adopted from Hildebrandt and Branke [59], i.e., the "spread" of the decisions made by the ensemble members. First, at decision j 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 x of ensemble E, it is assigned a "decision-rank", denoted as $r_{j,x}$ based on the rank of the job that it assigned the highest priority to. Member rank $r_{j,x}$ is then normalised based on the number of jobs waiting at the machine (K_j) at decision j, i.e., $r'_{j,x} = r_{j,x}/K_j$. An example of how a member's normalised ranks for the decision situations are calculated is shown in Table 3.5.

Table 3.5: Normalised rank calculation for a member x of an ensemble E over Ω decision situations.

Decision	Jobs	Priorities by Member x	Rank by ensemble E	$r'_{j,x}$
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	
...	...	...	...	...
Ω	1	0.2	1	$\frac{2}{3}$
	2	0.7	2	
	3	0.1	3	

After the normalised member ranks are calculated, LJR 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.

$$\text{LJR}(E) = \max_{x}\{\frac{1}{\Omega}\sum_{j=1}^{\Omega} r'_{j,x}\} \tag{3.11}$$

LJR is proposed to measure the diversity in decisions made by the ensemble members. High LJR 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 correlated with the performances of the ensembles over the DJSS problem instances.

3.7 Ensemble Combination Scheme Experimental Design

This section covers the experimental design to evaluate and analyse the ensemble combination schemes incorporated into the EGP-JSS approach. The experimental setup and parameter settings are mostly kept consistent as the ones used to evaluate the EGP-JSS and the MLGP-JSS approaches that use majority voting (Section 3.4). Therefore, this section discusses the parameters that have been modified to further improve the benchmark GP and the EGP-JSS approaches. First, we describe the adjustments to the experimental designs discussed in above (Section 3.4) and then describe the GP parameters used by the unweighted EGP-JSS (majority voting and linear combination) and the weighted EGP-JSS (weighted majority voting and weighted linear combination) approaches.

3.7.1 Adjustments to Ensemble GP Experimental Design

The DJSS dataset used to evaluate the EGP-JSS approach with the different combination schemes is the simulation model used by Hunt et al. [66] as described in Section 3.4.1. However, during training for the benchmark GP and the EGP-JSS, a different seed is used in conjunction with the training set's simulation configuration at every generation. This results in different DJSS training instances being used to evaluate the GP individuals. This has been shown to improve the generalisation ability of the rule output from the GP process compared to fixing the seed [60].

3.7.2 EGP-JSS Parameter Settings

The GP parameters required for the EGP-JSS approaches are shown in Table 3.6. In the table, the "Common Parameters" category are the parameter values shared by all EGP-JSS approaches with the different combination schemes. "Unweighted EGP-JSS Parameters" category are the parameters specific to the EGP-JSS approach that uses majority voting and linear combination schemes. "Weighted EGP-JSS Parameters" category are the parameters specific to EGP-JSS approach that use weighted majority voting and weighted linear combination schemes. The EGP-JSS with the weighted combination schemes has a reduced GP subpopulation size to account for the GA subpopulation. The parameters used by the benchmark GP approach used for comparison against the EGP-JSS approaches are kept consistent as the GP parameters described in Table 3.3.

3.8 Ensemble Combination Scheme Results and Discussions

This section covers the evaluation and the analysis of the EGP-JSS approach with the different combination schemes. First, we provide the plot

Table 3.6: GP parameters used by the EGP-JSS for the different combination schemes.

Approach	Parameters	Value
Unweighted EGP-JSS	No. of GP subpopulations	4
	Subpopulation size	256
Parameters Weighted EGP-JSS Parameters	No. of GP subpopulations	4
	GP subpopulation size	231
	GA subpopulation size	100
	GA crossover /mutation /reproduction rate	90% / 10% / 0%
	GA genome value range	$[0, 10]$
	Crossover type	One-point
	Mutation distribution	Gaussian dist. with std. of 0.5

of the fitnesses of the individuals in the ensemble GP approach with the different combination schemes as they are applied to the two training sets $4op$ and $8op$ to evolve the 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 above (Section 3.3.1). Afterwards, the analysis procedure described above (Section 3.6) is carried out on the ensemble GP approach with the different combination schemes to measure the values of DC, HC and LJR from the evolved rules. The discussion of the results will be provided after all results have been provided, i.e., after the results from the analysis procedure applied to the ensemble GP approach with the different combination schemes.

3.8.1 Combination Scheme Training Fitness Convergence Curves

The training performances of the EGP-JSS evolved using the different combination schemes are evaluated against the training performance of the benchmark GP as follows. For a GP run, the training performance at a

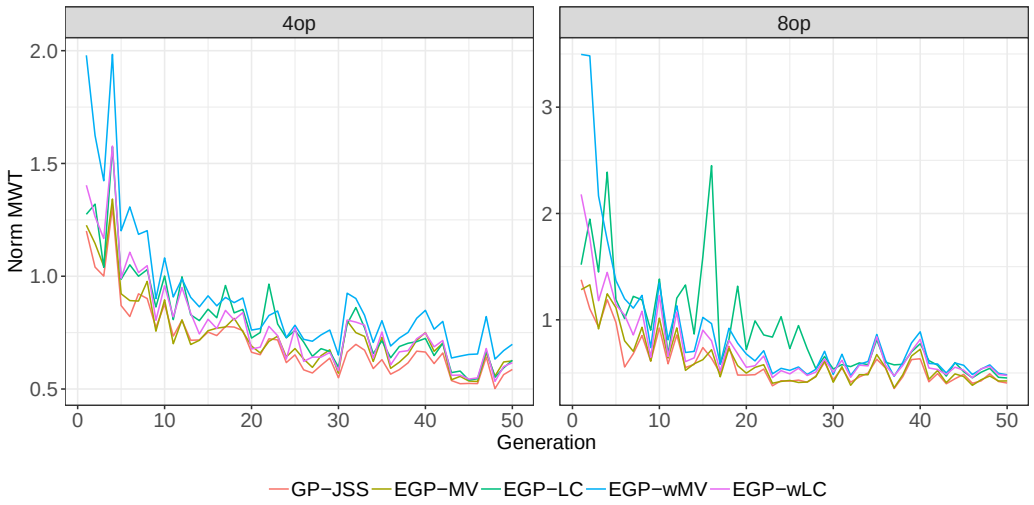


Figure 3.7: The average fitnesses of the GP approaches over the training sets.

specific generation is the individual with the best fitness in the population. For an EGP-JSS run, the training performance at a specific generation is the average performance of the individuals with the best fitness in each subpopulation. The average performance best individuals are used instead of the ensembles, as the best individuals will be updated as the representative and directly affect how the ensembles are generated in the next generation. The training performances over the generations are then averaged out over the multiple independent runs. The average training performances of the GP approaches are shown in Figure 3.7. EGP-JSS evolved using majority voting is called EGP-MV, EGP-JSS evolved using linear combination is called EGP-LC, weighted majority voting is called EGP-wMV, and weighted linear combination is called EGP-wLC.

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

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

3.8.2 Combination Scheme Test Performance

The rules evolved by EGP-JSS on the different combination schemes and the benchmark GP rules are applied to the training and test sets to compare the performances. As described in the comparison of EGP-JSS and MLGP-JSS above, a rule is applied to each simulation configuration 30 times to calculate the rule's performance from the mean tardiness objective values of the generated schedules. In addition, the EGP-JSS rules and the benchmark GP rules are compared in terms of their performance using two-sided Wilcoxon's signed-rank test at $p = 0.05$. The performances of the rules over the entire simulation model are shown in Table 3.7 for the rules evolved from $4op$ and in Table 3.8 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 sections highlighted with *red* mean that the EGP-JSS with the particular combination scheme is significantly worse than the benchmark GP approach for the simulation configuration, and the sections highlighted with *blue* mean that EGP-JSS rules performed significantly better than the GP rules.

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

Table 3.7: Comparison of the performances of EGP-JSS (with the different combination schemes) and the benchmark GP over the simulation model. Rules are evolved from $4op$.

Data Subset		EGP-JSS				GP
		EGP-MV	EGP-LC	EGP-wMV	EGP-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
	$\langle 25, 90\%, 4 \rangle$	1.49 ± 0.06	1.39 ± 0.09	1.60 ± 0.11	1.44 ± 0.10	1.49 ± 0.20
	$\langle 25, 90\%, 6 \rangle$	2.22 ± 0.11	2.06 ± 0.12	2.45 ± 0.23	2.15 ± 0.19	2.38 ± 0.74
	$\langle 25, 90\%, 8 \rangle$	3.30 ± 0.22	2.96 ± 0.38	3.24 ± 0.61	3.08 ± 0.46	2.99 ± 0.56
	$\langle 25, 90\%, 10 \rangle$	6.03 ± 0.37	5.56 ± 0.66	6.17 ± 0.91	5.73 ± 0.65	5.99 ± 2.14
	$\langle 25, 90\%, [2, 10] \rangle$	1.32 ± 0.07	1.23 ± 0.09	1.46 ± 0.15	1.30 ± 0.11	1.37 ± 0.29
	$\langle 50, 90\%, 4 \rangle$	2.86 ± 0.19	2.64 ± 0.20	3.31 ± 0.39	2.87 ± 0.29	3.25 ± 1.27
	$\langle 50, 90\%, 6 \rangle$	5.11 ± 0.31	4.69 ± 0.62	5.12 ± 0.72	4.84 ± 0.60	4.82 ± 0.84
	$\langle 50, 90\%, 8 \rangle$	7.95 ± 0.50	7.30 ± 0.84	8.27 ± 0.90	7.61 ± 0.65	8.11 ± 3.01
	$\langle 50, 90\%, 10 \rangle$	1.60 ± 0.11	1.49 ± 0.14	1.85 ± 0.24	1.62 ± 0.20	1.75 ± 0.47
	$\langle 50, 90\%, [2, 10] \rangle$	3.30 ± 0.23	3.05 ± 0.29	3.79 ± 0.43	3.32 ± 0.42	3.84 ± 1.72
	$\langle 25, 97\%, 4 \rangle$	7.96 ± 0.55	7.35 ± 1.10	7.88 ± 1.22	7.52 ± 1.01	7.47 ± 1.50
	$\langle 25, 97\%, 6 \rangle$	13.18 ± 0.80	12.54 ± 1.66	13.90 ± 1.55	12.92 ± 1.20	13.90 ± 4.82
	$\langle 25, 97\%, 8 \rangle$	1.40 ± 0.12	1.33 ± 0.18	1.66 ± 0.26	1.45 ± 0.21	1.58 ± 0.49
	$\langle 25, 97\%, 10 \rangle$	2.27 ± 0.23	2.11 ± 0.28	2.73 ± 0.46	2.38 ± 0.45	2.89 ± 1.82
	$\langle 25, 97\%, [2, 10] \rangle$	9.10 ± 0.67	8.64 ± 1.41	9.19 ± 1.25	8.76 ± 1.11	8.91 ± 1.77
	$\langle 50, 97\%, 4 \rangle$	13.95 ± 1.00	13.19 ± 1.80	14.96 ± 1.35	13.75 ± 1.36	15.16 ± 5.48
	$\langle 50, 97\%, 6 \rangle$	1.52 ± 0.09	1.44 ± 0.12	1.71 ± 0.17	1.54 ± 0.16	1.64 ± 0.33
	$\langle 50, 97\%, 8 \rangle$	2.54 ± 0.17	2.38 ± 0.22	2.95 ± 0.35	2.59 ± 0.28	2.95 ± 1.16
	$\langle 50, 97\%, 10 \rangle$	5.07 ± 0.37	4.69 ± 0.67	4.97 ± 0.85	4.79 ± 0.68	4.72 ± 0.91
	$\langle 50, 97\%, [2, 10] \rangle$	9.20 ± 0.65	8.63 ± 1.12	9.40 ± 1.32	8.83 ± 0.99	9.28 ± 3.37

3.8.3 Behavioural Analysis and Further Discussion

For each test simulation configuration in the simulation model, a problem instance is generated and DC, HC and LJR are calculated for EGP-JSS approaches using the procedure described in Section 3.6. 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 (Ω). After the param-

Table 3.8: Comparison of the performances of EGP-JSS (with the different combination schemes) and the benchmark GP over the simulation model. Rules are evolved from $8op$.

Data Subset		EGP-JSS				GP
		EGP-MV	EGP-LC	EGP-wMV	EGP-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, 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, 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, 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, 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, 90%, [2, 10] $\rangle$	1.31 ± 0.06	1.25 ± 0.12	1.48 ± 0.17	1.33 ± 0.11	1.29 ± 0.10
	$\langle$ 50, 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, 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, 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, 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, 90%, [2, 10] $\rangle$	3.15 ± 0.14	3.02 ± 0.30	3.55 ± 0.37	3.21 ± 0.22	3.25 ± 0.36
	$\langle$ 25, 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, 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, 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, 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, 97%, [2, 10] $\rangle$	9.79 ± 0.62	9.28 ± 1.33	10.70 ± 1.34	9.93 ± 1.43	9.29 ± 1.42
	$\langle$ 50, 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, 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, 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, 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, 97%, [2, 10] $\rangle$	10.16 ± 0.67	9.76 ± 1.68	12.09 ± 2.26	10.98 ± 1.77	10.11 ± 2.00

eter tuning, $\epsilon = 10$ and $\Omega = 50$, i.e., 50 decision situations sampled from a problem instance are used to calculate DC, HC and LJR and the decision situations have at least 10 jobs. The results of applying the analysis measures are shown in Figure 3.8.

From the figure, the DC 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 EGP-MV rules

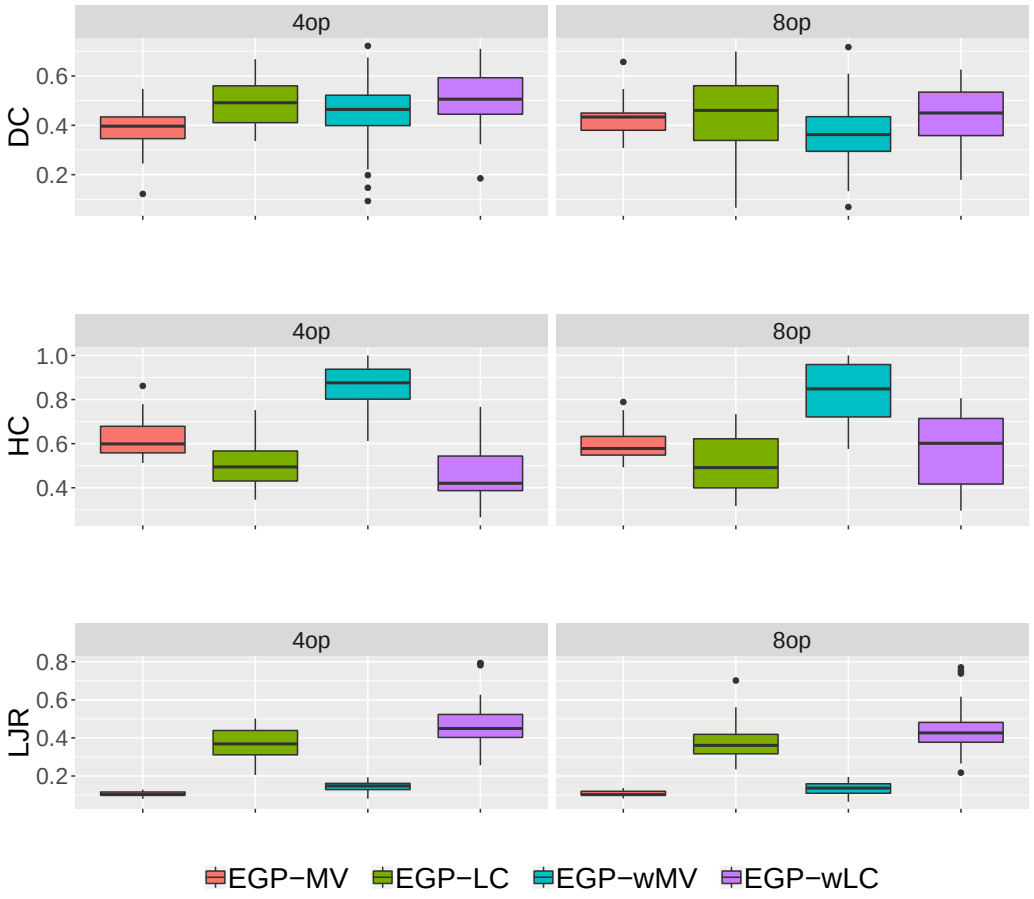


Figure 3.8: The analysis measures for the EGP-JSS approaches plotted against the performance over the problem instances. $4op$ and $8op$ denote that the GP rules are evolved from the respective training sets.

evolved over $4op$ and the EGP-wMV rules evolved over $8op$, the GP rules evolved over both training sets have DC 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, EGP-MV rules evolved from $4op$ have an average DC value of 0.39 and EGP-MV evolved from $8op$ have an average DC 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 tiebreaker rule being used. The relatively high DC value is significant for the EGP-MV rules compared to the other EGP-JSS rules. EGP-LC, EGP-wMV and EGP-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 EGP-LC, EGP-wMV and EGP-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 EGP-MV rules more than the other EGP-JSS rules. This may explain why the linear combination schemes generally perform better than the respective majority voting schemes for the DJSS problem instances.

On the other hand, EGP-wMV rules have high HC compared to other GP rules evolved over the training sets, whereas the other GP rules have similar HC values. HC is used to measure the bias of an ensemble towards a specific ensemble member. Therefore, for EGP-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 earlier EGP-JSS results (Tables 3.7 and 3.8), where the EGP-wMV rules generally perform worse than other EGP-JSS combination schemes, it is clear that having high HC value negatively affects the quality of the GP evolved ensembles. As expected, we found that the EGP-wMV rules behave similarly to single rules given their high HC values. It increases the difficulty for other members of an EGP-wMV ensemble to cover for the "high-contribution" rules' errors. This is reflected in the performance results, where EGP-wMV generally performs 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 ef-

fectively 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 EGP-wMV and the test performance of EGP-wMV rules. In the results for the training performances of the GP individuals, EGP-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, EGP-LC and EGP-wLC rules have high LJR 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 (EGP-wMV and EGP-wLC) have higher LJR values than the unweighted counterpart (EGP-MV and EGP-LC respectively). The differences in the LJR 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 EGP-MV and EGP-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 EGP-LC and EGP-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 more diverse ensemble members. From the results, EGP-LC rules perform well against

EGP-MV rules for $4op$ and comparably for $8op$. In addition, EGP-wLC rules perform well against EGP-wMV rules.

3.9 Chapter Summary

The goal of this chapter was to develop an effective ensemble GP approach for the DJSS problem with dynamic job arrival with a focus on improving the generality over standard GP approach. To achieve this goal, we first investigated the ensemble algorithm that can be incorporated into the GP approach. The two ensemble algorithms investigated are Potter and De Jong's cooperative coevolution [127] and Wu and Banzhaf's multilevel genetic programming [153], where these algorithms are modified to form the ensemble genetic programming for job shop scheduling (EGP-JSS) approach and the multilevel genetic programming for job shop scheduling (MLGP-JSS) approaches. After investigating the ensemble algorithms that are incorporated into the GP approach, we investigate the combination schemes that are effective for the ensemble GP approach. The combination schemes investigated are majority voting, linear combination, weighted majority voting and weighted linear combination.

Out of the ensemble algorithm investigated, the EGP-JSS approach performs significantly better than the MLGP-JSS approach over the DJSS dataset used in this chapter. Because of this, we incorporated the combination scheme to the EGP-JSS approach to compare against the benchmark GP approach, which is a standard GP that evolves a single priority dispatching rule [103]. The results from the combination scheme show that the EGP-LC rules generally perform better than EGP-MV, EGP-wMV and EGP-wLC rules, and also outperform the benchmark GP approach for most of the DJSS problem. The analysis of the rules shows that the average DC values for all EGP-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 mem-

bers by reducing information loss are more desirable. This may be likely the reason why EGP-LC performs better at handling complex decision situations than other combination schemes.

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

Finally, the analysis results show that EGP-LC and EGP-wLC have higher LJR values than EGP-MV and EGP-wMV. Therefore, it is possible that EGP-LC and EGP-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 LJR can also be partially correlated to better performances, as EGP-LC rules have better performance than EGP-MV rules for $4op$ and EGP-wLC rules have better performance than EGP-wMV rules.

The DJSS problem handled by the ensemble GP approach has dynamic job arrivals, but do not consider other types of dynamic events. In a real-world event, it is likely that multiple types of unforeseen events occur during processing. A popular type of dynamic events that are investigated in the literature is machine breakdown events [115]. Therefore, the next chapter wil carry out an investigation into the DJSS problem with dynamic job arrivals and random machine breakdowns. The chapter afterwards will then develop multitask GP approaches for the DJSS problem that, like the ensemble GP approaches investigated in this chapter, evolve multiple

rules simultaneously (i.e. a rule portfolio).

Chapter 4

GP to DJSS Problems Subject to Machine Breakdowns

4.1 Introduction

Dynamic job arrivals and machine breakdowns have been extensively researched in the literature [115], but there has been very little research in the literature that has investigated large DJSS problems with both dynamic job arrivals and machine breakdowns (i.e. DJSS problems with job arrivals that range in the thousands [61]). In particular, no research has applied GP to evolving dispatching rules to handle dynamic job arrivals and machine breakdowns. Real-world scenarios are likely to have a number of unforeseen events that need to be accounted for and are unlikely to only have single types of dynamic events that occur during processing. DJSS problems with a large number of dynamic job arrivals require an approach that has short reaction times, and machine breakdowns add a significant temporary bottleneck due to the relatively long duration required to repair the machines compared to the processing times of the jobs. DJSS problems with multiple types of dynamic events is an important direction of research that needs to be considered [102]. Therefore, combining dynamic job arrivals with machine breakdowns provide a realistic representation of

practical DJSS problems.

4.1.1 Chapter Goals

The goal of this chapter is to develop an effective GP approach that can evolve machine breakdown specific dispatching rules for the DJSS problem with dynamic job arrivals and machine breakdowns. Afterwards, we can obtain insight into the properties of the rules effective for specific DJSS problem domains for future GP approaches. The DJSS problem will use mean weighted tardiness (MWT) minimisation objective, which is a popular objective used in the literature [125]. MWT minimisation objective is a significant objective in the field of DJSS research [102], and incorporating weights into the mean tardiness objective used in the previous chapter adds further complexity to the job sequencing decision that needs to be handled by the GP evolved rules. To achieve this goal, we carry out the following two sub-objectives:

- First, this chapter carries out an investigation into the efficacy of GP for the DJSS problem. The efficacy investigation will evolve GP rules for different machine breakdown scenarios and evaluate them over the all machine breakdown scenarios while also coping with dynamic job arrivals. If the rules evolved by the standard GP approach is effective for multiple machine breakdown scenarios outside of the scenario that they are evolved on, then we can verify the generality of GP over the DJSS problem. The efficacy investigation will carry out a structural analysis of the GP evolved rules to obtain insights into the differences in the properties of the rules evolved by GP for different machine breakdown scenarios.

- Second, after the efficacy investigation, this chapter further studies the usefulness of machine breakdown GP terminals in the GP approach by designing machine breakdown specific GP terminals. By

doing this, GP can evolve rules that explicitly utilise machine breakdown information in the job dispatching decision-making process. This may result in the evolved rules being able to make better decisions for both machine breakdown and non-machine breakdown JSS problem instances than rules evolved without machine breakdown information. Afterwards, by analysing specific machine breakdown GP evolved rules, we are expecting to understand further how the rules behave in DJSS with dynamic job arrivals and machine breakdowns, allowing us to potentially develop more effective machine breakdown GP approaches in the future.

4.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. First, we provide a description of the DJSS simulation model used to investigate the efficacy of GP, a brief overview of the baseline GP. After the GP overview, the experimental design, the results, the analysis and the discussion for the efficacy investigation are covered. Second, we provide the description of the machine breakdown GP terminals incorporated into the GP approach, the design of the experiments, the results and the discussion. Finally, we provide a chapter summary that summarises the findings of this chapter.

4.2 Framework for Investigating the Generality of GP

This section describes the framework that is used to investigate the efficacy of the baseline GP described above to DJSS problems with dynamic job arrivals and machine breakdowns. This covers the DJSS simulation model with different levels of machine breakdowns. We will also explain how the GP is applied to the DJSS simulation models to evolve specialist rules and generalist rules for the different machine breakdown scenarios.

4.2.1 DJSS Simulation subject to Machine Breakdowns

The DJSS simulation model Δ used to evaluate the efficacy of GP is modified from the simulation model proposed by Holthaus [61]. Holthaus's simulation model consists of discrete-event simulations that are commonly used to evaluate GP approaches as they are applied to DJSS problems with dynamic job arrivals [59, 60, 61, 66]. Various man-made dispatching rules have been applied to Holthaus's [61] simulation model to show the trade-off in the effectiveness of the rules for the different machine breakdown scenarios. Holthaus's simulation model provides a reliable starting point to begin our investigation. After adapting the simulation model, we can tune the parameters used by the simulations to refine the experiments and to better evaluate the GP approach.

In Holthaus's [61] simulation model, the parameters related to generating job processing times and job arrival times are the same as Hunt et al.'s [66] simulation model described in Section 3.5.1 (page 3.5.1). The simulations have $M = 10$ machines on the shop floor. For each simulation, there is a "warm-up" period of 500 jobs which do not contribute towards the objective value, and the simulation is terminated once the 2500th job has been completed. The parameters for generating job arrivals are consistent with Holthaus's simulation model. The mean processing time of the operations is $\mu = 25$, i.e., the processing times of the jobs are uniformly distributed between $[1, 49]$. The arrival times of the jobs follow a Poisson process, with utilisation rate $\rho = 90\%$, which is consistent with the utilisation rate used by Holthaus [61]. In Holthaus's simulation model, the number of operations is uniformly distributed between 2 to 14 operations. The weight of a job is either 1, 2, or 4 with probabilities 0.2, 0.6 and 0.2 respectively, which is a standard method of generating weights for jobs in due-date related DJSS problems [66, 105].

The following parameters for generating job arrivals are modified from Holthaus's [61] simulation model. This thesis focuses on DJSS problems with no re-entry, i.e., no two operations of a job are processed on the same

machine. In other words, a job can have at most 10 operations as there are 10 machines on the shop floor. Therefore, the number of operations per job is modified to be uniformly distributed between 2 to 10 operations. In other words, the expected proportion of machines required by a job to complete the operations is $p_M = (2 + 10)/2 = 6$. Arrival rate of jobs is calculated from the utilisation rate, the mean processing times of operations and the expected proportion of machines required by a job as given in Equation (3.4) on page 3.4. In addition, the parameter tightness h is different. A job j's due date d_j is calculated from the job's arrival time r_j, tightness h, and the total processing time of operations as given in Equation (3.5) on page 3.5. Tightness of $h = 3$ and $h = 5$ are used, where $h = 3$ represents tight due dates and $h = 5$ loose due dates. This is adjusted from the original tightness values of $h = 4, 8$ used by Holthaus [61]. During our preliminary experiments, we found that due date tightness $h = 8$ was too "loose" and resulted in GP evolved rules generating schedules for simulations where the MWT values are zero, i.e., all jobs were being completed before their due date.

The method for generating the machine breakdown events are consistent with Holthaus's [61] simulation model. The machine repair times and the times between machine breakdowns (excluding the repair times) are exponentially distributed. The mean repair time (r) and the mean time to breakdowns (η) are the same for all machines on the floor. In addition, for the configuration used for the simulation, r depends on the mean processing times of the operations μ and the machine breakdown level parameter (π). The machine breakdown level can be considered as the proportion of time the machine is being repaired during the duration of the simulation. For example, if the breakdown level $\pi = 2.5\%$ and all jobs took 2500 time units to process, then the total repair time for all machines is approximately $2.5\% \times 2500 = 62.5$ time units. In other words, the machine breakdown level is given by $\pi = r/(\eta + r)$, which means that $\eta = r/\pi - r$ [61]. The simulation model has variable configurations for the following

parameters: due date tightness (h), mean repair times of machines (r) and breakdown level (π). The optional configurations are $r \in \{\mu, 5\mu, 10\mu\}$ and $\pi \in \{0, 2.5\%, 5\%\}$. As the simulation configurations with zero breakdown level do not have machine breakdown, simulations with $\pi = 0$ do not use the mean repair time parameter. Therefore, there are seven different machine breakdown scenarios considering the different combinations of breakdown level and mean repair times. The two due date tightness configurations and the configurations for the machine breakdowns result in a total of 14 different simulation configurations.

4.2.2 GP-HH Training Procedure

To evolve and evaluate the GP rules, different subsets of DJSS simulations in the simulation model are used to evolve different sets of GP rules. Figure 4.1 shows an overview of how the simulation mode lΔ is used to evolve different sets of GP rules that are either "generalists" or "specialist" over the machine breakdown level (π). The generalist rules are designed to be effective for the different machine breakdown scenarios, whereas the specialist rules [23] are designed to be effective for specific machine breakdown scenarios. The specialist rules are GP rules evolved on a specific machine breakdown level, whereas the generalist rules are GP rules evolved on all machine breakdown level.

First, the simulation model Δ in Figure 4.1 is partitioned into three subsets based on the machine breakdown level. In the subsets, the simulations with machine breakdown level $\pi = 0$ do not have machine breakdowns, and the simulations with $\pi = 2.5\%$ and $\pi = 5\%$ have "medium" and "high" levels of machine breakdowns respectively. The subsets are denoted as Δ_N, Δ_M and Δ_H respectively. Overall, the three subsets Δ_N, Δ_M and Δ_H contain six configurations after considering the different possible combinations of due date tightness values. The specialist rules are evolved from Δ_N, Δ_M and Δ_H and are designed to cope with the specific

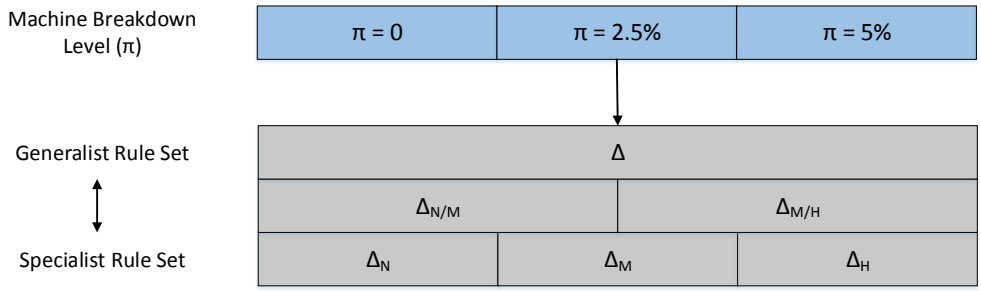


Figure 4.1: Overview of how the simulation model used for the DJSS problem is partitioned to train GP rules specialised for different machine breakdown configurations.

level of machine breakdown. Additionally, $\Delta_{N/M}$ and $\Delta_{M/H}$ combine two smaller subsets together (e.g. Δ_N and Δ_M for $\Delta_{N/M}$, and are used to evolve "intermediate" sets of rules. If the intermediate rules are competitive by the specialist rules, e.g., rule evolved from Δ_H does not perform significantly worse than $\Delta_{M/H}$ for simulations with $\pi = 5\%$, then it is likely that GP can generalise well over different machine breakdown scenarios even without incorporating information about machine breakdowns. Finally, all possible configurations in the simulation model Δ, i.e., configurations from Δ_N, Δ_M and Δ_H combined, are used to evolve the final set of general rules. Overall, this results in a total of 6 sets of GP rules that range from generalists to specialist over the DJSS problem. This procedure was first covered by Burke et al. [23] for improving the generality of the GP-HH approach for a bin packing problem.

The set of rules evolved from a specific training set is denoted as 'DR-x' with the suffix as the training set. For example, DR-N denotes the set of GP evolved rules which have been evolved from subset Δ_N, i.e., simulations with no machine breakdowns. In addition, at each generation, the seeds used to stochastically generate the jobs and the machine breakdowns

are rotated during the training procedure of the GP-HH. This means that the simulations used in one generation will be different to the simulations used for the next generation. This has consistently shown to improve the generalisation ability of the evolved rules to DJSS problems [60].

4.2.3 GP Evaluation Procedure

The evaluation procedure for the baseline GP is modified from the evaluation procedure used by the benchmark GP approach in Section 3.5.2 (97). First, a GP individual is evaluated over a set of DJSS simulations to calculate its fitness as follows. GP individual x is applied to a simulation I as a *non-delay* priority dispatching rule [125]. After a schedule is generated for the simulation I and the MWT objective value $Obj(x, I)$ is calculated, the objective value is normalised using a *reference rule* (as shown in Equation (3.1) in the previous chapter). The reference rule used for the DJSS problem with dynamic job arrivals and machine breakdowns is the *weighted apparent tardiness rule* (wATC) [148], which is a modified version of the man-made ATC rule that accounts for weighted tardiness related objectives. The wATC is an effective man-made dispatching rule for handling DJSS problems with weighted tardiness related objectives, and are likely to be able to cope with the machine breakdowns effectively. The fitness calculation for GP individual x is the average of the normalised MWT objective values as shown in Equation (4.1), where $Obj'(x, I)$ denotes the normalised objective calculated by the individual x for simulation I.

$$f_x = \frac{1}{|\mathcal{T}|} \sum_{I \in \mathcal{T}} Obj'(x, I) \tag{4.1}$$

4.3 GP Generality Investigation Experimental Design

This section covers the experimental setups required to carry out the investigation of the efficacy of GP and the analysis of the baseline GP approach. First, we cover the GP representation, the terminal set and the function set for the baseline GP, followed by the GP parameter settings.

4.3.1 GP Representation, Terminals and Function Sets

For the baseline GP approach that is used to investigate the efficacy of GP for the DJSS problem, we use a tree-based GP representation [80]. The GP individuals represent arithmetic function trees that calculate the priorities of jobs during decision situations. The GP terminals used by the baseline GP approach is given in Table 4.1. The GP terminals shown in the table consist of the GP terminals used in the earlier chapter to evolve dispatching rules for the DJSS problem with dynamic job arrival (Table 3.2). This makes the baseline GP approach more consistent with the existing GP approaches. However, as the objective of the DJSS problem in this chapter is MWT (whereas the DJSS problem in the previous chapter did not have job weights), an additional GP terminal W is added to the list of existing GP terminals. In addition, the function set used by the GP approaches to evolve the rules in the earlier chapter is used, i.e., the function set consists of the operators $+$, $-$, $\times$, protected $/$, $\max$, $\min$, and `if`.

4.3.2 GP Parameter Settings

The GP parameter used for the efficacy investigation are given in Table 4.2. The GP parameters described in the table are modified from the GP parameters used for the benchmark GP approach that evolves single dispatching rule in Table 3.3 in Section 3.3.4 (Page 87). The parameters modified are the population size and the maximum depth of the GP trees. The

Table 4.1: Terminal set for the baseline GP used for efficacy investigation, where job j is one of the jobs waiting at the machine m^* to process operation o_{ij}.

Terminal	Description
RJ	The operation ready time of job j
RO	Remaining number of operations of job j
RT	Remaining total processing times of job j
PT	The operation processing time of job j
RM	Machine m^* ready time
NJ	Number of non-delay jobs waiting at machine m^*
DD	Due date of job j
W	Job's weight w_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	$+, -, \times, /, \texttt{if}, \max, \min$

population size of 256 is used for the efficacy investigation compared to the population size of 1024 used in the previous chapter. This is done to save on the computation cost, as the baseline GP is applied to multiple different training sets to evolve rule sets ranging from generalist rules to specialist rules. The maximum initial depth of the GP trees and the maximum possible depth of the GP trees, which are set to 4 and 8 respectively. This allows us to observe specific evolved rule's structures during analysis compared to the previous maximum tree depth of 17.

Table 4.2: GP parameters used to evolve rules for the GP efficacy investigation.

Parameters	Value
Population size	256
No. of generations	51
GP crossover rate	80%
GP mutation rate	10%
GP reproduction rate	10%
GP maximum initial depth	4
GP maximum depth	8
Selection method	Tournament selection of size 7
No. of elites	1
wATC k value	3.0

4.4 GP Generality Investigation Results and Discussion

This section covers the results for the GP efficacy investigation for the DJSS problem with dynamic job arrivals and machine breakdowns. To carry out the efficacy investigation, 30 independent runs of the baseline GP process is applied to the various subsets of the simulation model Δ as described in Section 4.2.2, evolving a set of 30 dispatching rules. The rule sets are denoted as DR-N, DR-M, DR-H, DR-N/M, DR-M/H and DR-All based on the training subset that the GP process is applied to. For example, DR-N denotes the set of dispatching rules evolved by GP on the subset $\langle$ 0%, 0, 5 $\rangle$, i.e., the subset of the simulation model Δ that consists of simulations with no machine breakdowns. Afterwards, the rule sets are applied to the entire simulation model Δ, and the performances of the rules are compared against each other. Finally, the evolved rule sets are analysed based on their frequencies of the GP terminals.

4.4.1 Evolved Rule Performance Evaluation

Each configuration in the DJSS simulation model Δ is used to generate 30 different simulations as part of the test set. In total, this results in a total of $18 \times 30 = 540$ test instances using the 18 different configurations. The sets of GP evolved rules are then applied to the test instances to generate schedules for the simulations, and the MWT of the schedules are compared against each other as part of the general evaluation procedure. A set of GP evolved rules is *significantly* better than another rule set if the difference in the MWT values satisfies the two-sided Wilcoxon's signed-rank test at $p = 0.05$. The performances of the rule sets over the different simulations are shown in Figure 4.2. Each box plot shows the results over simulations in a configuration, and the configurations are categorised by the due date tightness and breakdown level, where the first value above the box plot is the due date tightness and the second value above the box plot is the machine breakdown level. For example, 3 then 0% shown in the top left box plot indicates that due date tightness $h = 3$ and machine breakdown level $\pi = 0\%$.

From the results, we can see that for the simulations with $\pi = 0$ and $\pi = 5\%$ DR-N and DR-H generally perform well over the respective problem domain they are trained on but perform poorly on simulations with high levels of machine breakdowns (for DR-N) and simulations with no machine breakdowns (for DR-H). Under the statistical test, the difference in the performance is significant between DR-N and DR-H. When the best-fit rules DR-N and DR-H are compared to the intermediate rules DR-N/M and DR-M/H, the two best-fit rules perform slightly better than the intermediate rules over their respective machine breakdown levels the specialise rules are evolved on. The difference in the performances are significant between DR-H and DR-M/H, but not between DR-N and DR-N/M. However, on machine breakdown level $\pi = 0.025$, it is observed that most sets of rules with the exception of DR-H have a similar performance to each other, where the slight differences in the performances are not sig-

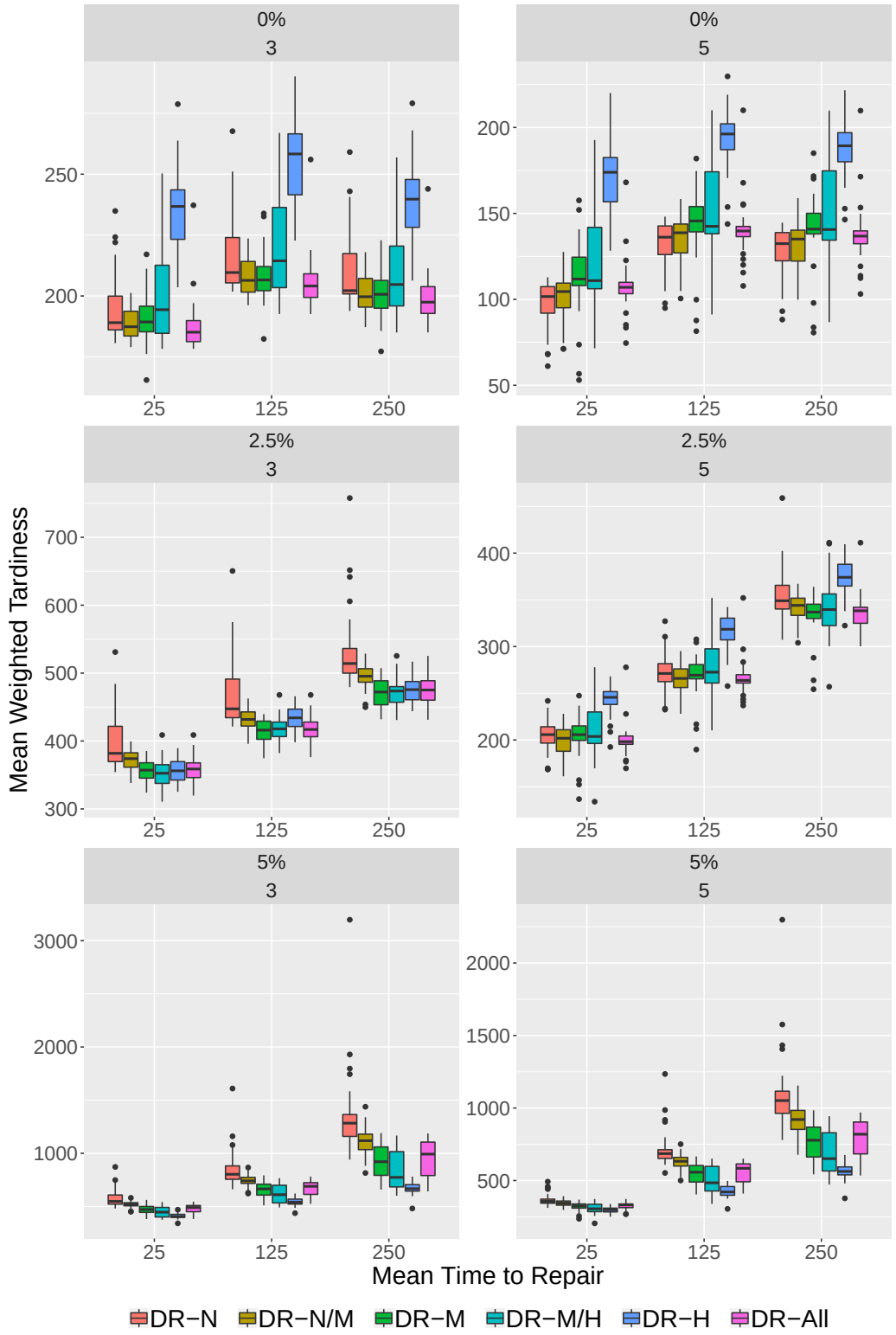


Figure 4.2: The comparisons of the MWT performances of the GP rules evolved over different training sets over the simulations in the test set.

nificant. Finally, the generalist rule DR-All perform well over simulations with no machine breakdowns and simulations with machine breakdown level $\pi = 0.025$, but performs significantly worse than DR-H and DR-M/H for simulations with $\pi = 0.05$. Overall, it may be likely that standard GP-HH approach may not be able to generalise well when it comes to DJSS problems with dynamic job arrivals with machine breakdowns, and the quality of the rules evolved by a standard GP-HH approach is likely too sensitive to the proportion of time that the machine is broken down during the simulation.

4.4.2 GP Terminal Distribution Analysis

After evaluating the performances of the GP evolved rules, the terminals that are used by the GP rules are compared against each other to analyse the internal structures of the rules. For the sets of GP rules, the proportion of various terminals utilised in these rules are shown in Figure 4.3.

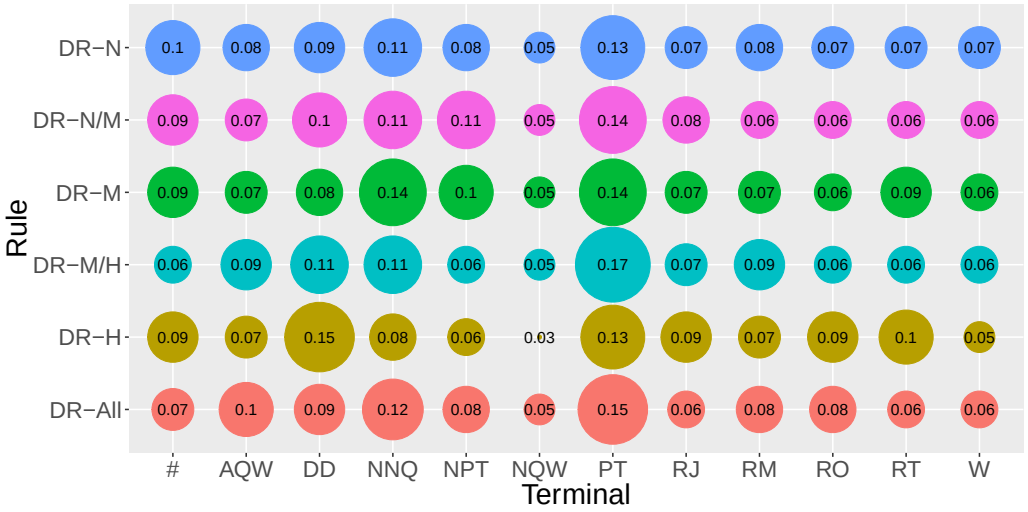


Figure 4.3: The proportion of terminals used by the sets of rules evolved by the GP-HH approaches.

From Figure 4.3, the most prominent terminals that are used by all sets

of rules is the processing time of current operation (PT), followed by the due date of the job (DD) and the number of jobs waiting at the next machine (NNQ). On the other hand, DR-H rules have a higher proportion of due date terminal compared to the other sets of evolved rules (e.g. the proportion of due dates terminals in DR-H is 15% compared to 9% in DR-N). This is likely due to the fact that in simulations with high machine breakdown level (e.g. $\pi = 0.05$) processing time becomes unreliable in determining the expected duration of time that a job requires on any machine for the corresponding operation to complete. Compared to simulations with lower machine breakdown levels (e.g. $\pi = 0$ and $\pi = 0.025$), it is more likely in the simulations with high machine breakdown levels that the machine breaks down during processing of a job. This results in the job getting stuck on the machine while it is being repaired and taking longer than expected to finish processing. Instead, processing urgent jobs can be a more reliable method of generating good schedules for simulations with high levels of machine breakdowns. In other words, it is likely that the rules that prioritise shorter jobs than urgent jobs generate inferior solutions in high machine breakdown scenarios. More frequent machine breakdowns that occur in high machine breakdown scenarios likely make the processing times used by the dispatching rules less reliable when compared to low machine breakdown scenarios. In addition, DR-H has a lower proportion of terminals that take the attributes for when the job reaches the next machine (NNQ, NPT and NQW), as the additional uncertainty introduced by the high level of machine breakdown may make the terminals less effective at reducing the myopic nature of dispatching rules [66].

4.5 Machine Breakdown GP terminals

This section describes the machine breakdown terminals which are incorporated to the baseline GP approach's terminal set. This allows GP

to evolve dispatching rules that may make better decisions during decision situations, potentially leading to better performance than GP evolved rules which do not incorporate machine breakdowns. First, we describe the update made to the baseline GP terminal set that is shown earlier (Table 4.1) and provide a classification tree showing the updated terminal set and the machine breakdown GP terminals. This is followed by the descriptions and justifications for the machine breakdown GP terminals. The first approach replaces existing terminals related to operation processing times and add repair time of machines if necessary. This approach is denoted as "augmented" approach, as it attempts to *improve* certain benchmark terminals by incorporating machine breakdown information. The second approach adds *new* machine breakdown terminals, which "captures" the machine breakdowns happening on the shop floor, to the existing set of GP terminals.

4.5.1 Update to the Baseline GP Terminals

In addition to the set of terminals described in Section 4.3.1, two GP terminals frequently used in the literature [60, 95, 107] are added to the set of terminals used by the baseline GP when being compared against GP approach that incorporates the machine breakdown GP terminals. The two GP terminals added to the baseline GP are slack (SL) sl_j of a job j and work in next queue (WINQ) $winq_j$ for a job j [125]. Slack is the amount of time remaining between job j's due date d_j and the minimum completion time of the job, i.e., $sl_j = d_j - \sum_{i=k}^{N_j} p_{ij} - t$ [125]. In the equation, k denotes the kth operation that job j is up to at the current time t of processing. On the other hand, WINQ is the total processing time required by all jobs currently waiting at the next machine m'_j that job j visits plus the remaining time for the machine to complete the operation being processed, if there is any (denoted as $wr_{m'_j,j'}$). Given that the processing times of the jobs waiting at the next machine m'_j are $p_1, \ldots, p_{N_{m'_j}}$, then

$winq_j = wr_{m'_j,j'} + \sum_{i=1}^{N_{m'_j}} p_i$. Slack will likely result in rules that will prioritise jobs that are urgent and also still has large amount of processing remaining so that the jobs can be completed before their due date. WINQ provides information about the workload on the next machine that the job visits, and likely will result in the rules assigning lower priorities to jobs that will be visiting potentially bottlenecked machines.

The updated GP terminals and the proposed machine breakdown terminals are shown in Figure 4.4. In the figure, we group the different terminals (shown in red boxes) based on what shop floor attribute they are classified. The GP terminals shown in the red boxes and not under the "machine breakdown" category are the updated list of GP terminals for the baseline GP approach. This consists of terminals described in Table 4.1, slack and WINQ. The GP terminals shown in the highlighted red boxes that are under the "machine breakdown" category are the new machine breakdown terminals that will be incorporated into the GP terminals. They are discussed in further detail below (Chapters 4.5.2 and 4.5.3).

4.5.2 Augmented GP Terminals

The following terminals in the original GP terminal set (as described in Figure 4.4) are replaced by terminals that add repair times of the machines: job's *operation processing time* (PT), job's *next operation processing time* (NPT) and *work in next queue* (WINQ). We start by assuming that the machine breakdown that occurs after the decision situation is known in advance and that we know the repair time of the machine breakdown, and use that information for the augmented GP terminals. This is done to simplify the problem and to explore whether adding the explicit knowledge furhter improves the effectiveness of GP. The replaced GP terminals return the original value if the job's operation is not interrupted by a machine breakdown, and adds the repair time of the machine otherwise. The terminals that incorporate the machine breakdown information is denoted with the prefix

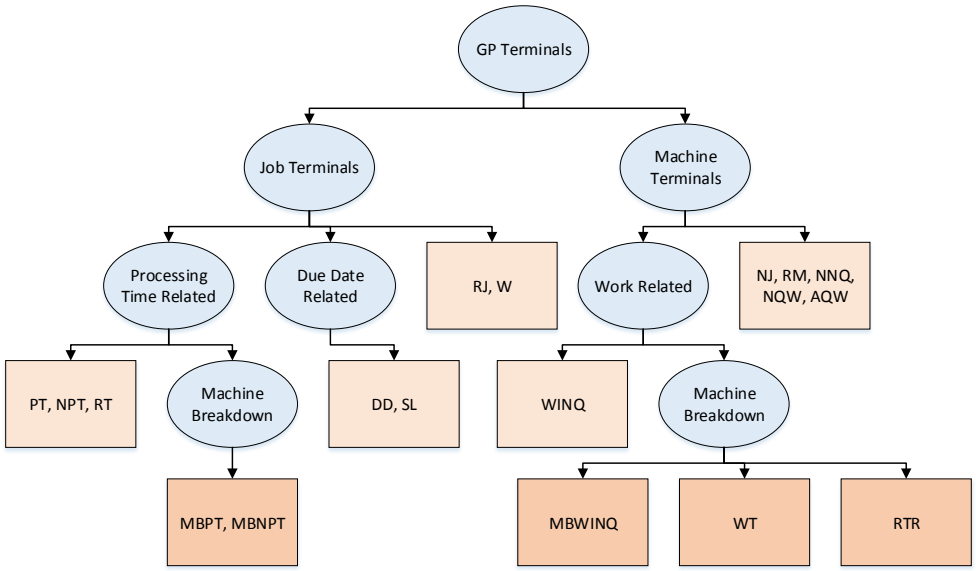


Figure 4.4: The classification of the updated list of GP terminals used by the baseline GP and the machine breakdown terminals that are incorporated into the machine breakdown GP approaches.

'MB-' (e.g. MBPT for machine breakdown adjusted processing time). The machine breakdown GP approach that incorporates the MBPT, MBNPT and MBWINQ terminals is denoted as GP-Aug.

Machine breakdown adjusted processing time (MBPT):

The machine breakdown adjusted processing time terminal (MBPT) replaces the processing time terminal (PT). Given that the current time during the decision situation is t and the processing time of job's current operation j is p_{ij}, MBPT terminal returns the actual duration of time required to process the job's operation after counting the additional time for repairing the machine breakdown interruption into account. In other words, if the job is *not* interrupted by a machine breakdown, i.e., if the operation completes earlier than the breakdown time b_t^m of the current machine m,

then the job's actual processing time p'_{ij} is equal to the expected processing time p_{ij}. Otherwise, the actual processing time is the sum of the processing time and the machine repair time r_t^m required to get the machine back up and running before the operation is resumed. The value returned by MBPT$(j) = p'_{ij}$, where the calculation for p'_{ij} is shown in Equation (4.2).

$$p'_{ij} = \begin{cases} p_{ij} & \text{if } t + p_{ij} < b_t^m \\ p_{ij} + r_t^m & \text{otherwise} \end{cases} \tag{4.2}$$

Machine breakdown adjusted next processing time (MBNPT):

The machine breakdown adjusted next processing time terminal (MBNPT) replaces the next processing time terminal (NPT). MBNPT terminal returns zero if the job j's current operation o_{ij} is the last operation before job j's completion. Otherwise, given that the next operation $o_{(i+1)j}$ is processed on machine m'_j, the repair time of m'_j is added to the next processing time $p_{(i+1)j}$ if it will be interrupted by a breakdown at machine m'_j at operation $o_{(i+1)j}$ earliest possible completion at machine m'_j. The earliest possible time that job j can be completed is determined with the assumption that operation o_{ij} is selected immediately by machine m, and the successive operation $o_{(i+1)j}$ is then processed by machine m'_j as soon as operation o_{ij} is completed. The time when operation o_{ij} completes is given by the current time t and the actual processing time p'_{ij}, which depends on whether the operation is interrupted by machine breakdown (Equation (4.2)). In other words, the earliest time operation $o_{(i+1)j}$ can be processed at machine m'_j is at $t + p'_{ij}$ immediately after the completion of operation o_{ij}. Therefore, if machine m'_j breaks down before time $t + p'_{ij} + p_{(i+1)j}$, then repair time $r_t^{m'_j}$ of machine m'_j is added to the operation $o_{(i+1)j}$'s processing time $p_{(i+1)j}$ as shown in Equation (4.3).

$$\text{MBNPT}(j) = \begin{cases} p_{(i+1)j} & \text{if } t + p'_{ij} + p_{(i+1)j} < b_t^{m'_j} \\ p_{(i+1)j} + r_t^{m'_j} & \text{otherwise} \end{cases} \tag{4.3}$$

Machine breakdown adjusted work in next queue (MBWINQ):

The machine breakdown adjusted work in next queue terminal (MBWINQ) replaces the work in next queue terminal (WINQ). Both WINQ and MBWINQ terminals return zero if the job j's current operation o_{ij} is the last operation before job j's completion. Otherwise, given that machine ${m'}_j$ is required by operation $o_{(i+1)j}$, the standard WINQ terminal returns the total processing times of the jobs that are currently waiting at machine ${m'}_j$ plus the remaining time required to process the operation currently being processed by machine ${m'}_j$, i.e., the work remaining. MBWINQ modifies the work remaining time calculated by WINQ by adding the machine ${m'}_j$'s repair time if the work is interrupted by machine breakdown at time $b_t^{{m'}_j}$. In other words, MBWINQ$(j) = {wr'}_{{m'}_j,j'} + \sum_{i=1}^{N_{{m'}_j}} p_{j_i}$, where $p_{j_1}, \ldots, p_{j_{N_{{m'}_j}}}$ are the processing times of jobs waiting at machine ${m'}_j$, ${wr'}_{{m'}_j j'}$ is the actual work remaining required on j' being processed on machine ${m'}_j$ before it becomes available. The calculation for actual work remaining ${wr'}_{{m'}_j j'}$ is given in Equation (4.4), where $s_{j'}$ denotes the time when j' started, $p_{j'}$ is the processing time required by j' at machine ${m'}_j$ and t is the current time.

$$ {wr'}_{{m'}_j j'} = \begin{cases} s_{j'} + p_{j'} - t & \text{if } s_{j'} + p_{j'} < b_t^{{m'}_j} \\ s_{j'} + p_{j'} - t + r_t^{{m'}_j} & \text{otherwise} \end{cases} \tag{4.4} $$

In summary, the augmented GP approach replaces three existing terminals (PT, NPT and WINQ) with equivalent terminals that incorporate information about machine breakdowns that interrupt job processing. The existing terminals are related to the processing times of the jobs waiting on the shop floor, where repair times need to be added onto the processing times if we expect the jobs to be interrupted by machine breakdowns. By doing this, we expect the GP rule to be able to use the "actual" processing times of the jobs to make better decisions on what job should be processed next by the machines during decision situations.

4.5.3 Reactive GP Terminals

Reactive machine breakdown terminals are added to the GP terminal set described in Figure 4.1 and incorporate information about current machine status. As the two terminals incorporate information about the potential wait time of a job waiting at a machine for the next machine it visits, they are investigated separately. The two terminals being investigated are the *repair time remaining next machine terminal* (RTR) and the *minimum wait time next machine terminal* (WT). The two reactive GP terminals may allow rules to make better decisions by prioritising jobs with low expected wait times compared to jobs with high expected wait times. This may lead to jobs spending less time waiting at busy machines, and the evolved rules may generate higher quality schedules. The machine breakdown GP approach that incorporates the RTR terminal is denoted as GP-RTR, and the machine breakdown GP approach that incorporates the WT terminal is denoted as GP-WT.

Repair Time Remaining Next Machine (RTR):

The repair time remaining next machine RTR returns zero if a job j waiting at a machine at time t is currently on its last operation or the next machine m'_j visited by j is currently not broken down. Otherwise, given that machine m'_j broke down at time $b_t^{m'_j}$ and the repair time is $r_t^{m'_j}$, the value given by RTR $= b_t^{m'_j} + r_t^{m'_j} - t$.

Minimum Wait Time Next Machine (WT):

The minimum wait time next machine WT returns the earliest time that the machine to be visited by job j next becomes available. If the current operation of j is the last operation before completion, then WT returns zero. In addition, if the next machine m'_j that job j visits is currently not busy and is not broken down, i.e., is completely available, then WT returns zero. Otherwise, the WT returns the duration of time required for machine m'_j

to be available. If machine m'_j is currently processing a job j' or is broken down with an interrupted job, then it returns the actual work remaining $wr'_{m'_j j'}$ which is given in Equation (4.4). Otherwise, if the m'_j is broken down and a job was not interrupted by the machine breakdown, WT returns the remaining repair time of machine m'_j as given by the terminal RTR.

4.6 Design of Experiment

This section covers the experimental setups required to evaluate the machine breakdown GP terminals. First, we carry out further tuning of the parameters used for the DJSS simulation model (described in Section 4.2.1) to investigate a wider range of machine breakdown scenarios. Afterwards, we provide the GP parameter settings and how they are modified from Table 4.2.

4.6.1 Modified DJSS Dataset for Machine Breakdown GP Terminals

To evaluate the GP that incorporates the machine breakdown GP terminals, we modify the DJSS simulation model Δ described in Section 4.2.1. The modified simulation model is denoted as Δ' and use the simulation parameter settings described in Table 4.3. The DJSS simulation model Δ' adjusts the machine breakdown events by modifying the machine breakdown levels and the repair times of the machine breakdowns. The machine breakdown levels include more "severe" cases of machine breakdown scenarios. From the investigation into the efficacy of GP, we observed that there are significant differences in the effectiveness of the specialist rules on the different machine breakdown scenarios. Therefore, it may be possible that the machine breakdown GP terminals effectiveness may be more pronounced for high machine breakdowns levels than low

machine breakdown levels, and provides an extreme case of the problem to be handled by the GP approaches that use machine breakdown GP terminals. The simulation parameters used to evaluate the baseline GP approach and the GP approaches with the machine breakdown GP terminals are shown in Table 4.3. In the table, the high machine breakdown levels $\pi = 10\%$ and $\pi = 15\%$ are added to the list of possible machine breakdown levels for the simulation configurations in DJSS simulation model described in Section 4.2.1. In addition, to accommodate for the new machine breakdown levels, the utilisation rate is lowered from $\rho = 90\%$ to $\rho = 80\%$ for all simulation configurations. Otherwise, the joint effect of job utilisation rate at $\rho = 90\%$ with the highest machine breakdown level at $\pi = 15\%$ will exceed 100% total utilisation rate. This will likely result in an unstable simulation, where the duration of the simulation affects the overall number of jobs waiting at the machines. By lowering the job utilisation rate to 80%, the total utilisation rate will be below 100% when combined with the machine breakdown level. The preliminary tests using man-made dispatching rules showed that this resulted in a stable simulation.

On the other hand, the machine repair times for the breakdowns are fixed to the values $r_t = 37.5$, 137.5, 262.5. The machine repair times are set to values that are not multiplicative of the mean processing time of job operations so that processing time terminal cannot be used as a substitute for the machine repair time. This investigation into the machine breakdown GP terminals incorporate repair time information to the evolved rule's decision-making process. Because of this, fixing the repair times simplifies the DJSS problem to concretely determine whether adding explicit knowledge further improves the effectiveness of GP. In addition, the repair time values were selected after running the baseline GP approach on different breakdown levels and durations of repair times. The preliminary experiments showed that the baseline GP rules could effectively handle the fixed repair time durations by using processing time terminals.

Finally, the simulation configuration that is used to evolve the rules from the baseline GP, GP-Aug, GP-WT and GP-RTR has 15% breakdown level, repair time of 262.5 and due date tightness of 3. In addition, the different seeds are used to stochastically generate the job arrivals and the machine breakdowns at every generation.

Table 4.3: Updated simulation configurations used for the generating arriving jobs and machine breakdowns in DJSS simulation model Δ'.

Simulation Model Parameter	Value
Number of machines (M)	10
Utilisation rate (ρ)	80%
Mean processing time (μ)	25
Weight/probability ((w, p))	$\{(1, 20\%), (2, 60\%), (4, 20\%)\}$
Due date tightness (h)	3 or 5
Machine breakdown level (π)	0%, 2.5%, 5%, 10% or 15%
Repair time (r_t)	37.5, 137.5 or 262.5
No. of configurations	30
Training configuration ($\langle \pi, r_t, h \rangle$)	$\langle$ 15%, 262.5, 3 $\rangle$

4.6.2 GP Parameter Settings

The GP parameter is modified from the GP parameters used in the efficacy investigation above (Table 4.2). The population size is increased from 256 to 1024 as we apply the GP approach to a single subset ($\langle$ 15%, 262.5, 3 $\rangle$) of the simulation model Δ'. The subset $\langle$ 15%, 262.5, 3 $\rangle$ allows us to determine if the machine breakdown terminals negatively contribute towards the performance of the rules for zero or low machine breakdown scenarios. In other words, there may be a potential trade-off where the machine breakdown GP approaches perform better than the baseline GP on the high machine breakdown scenarios but perform worse on the low machine breakdown scenarios. The larger population size is consistent with the population size used in the previous chapter and in the literature

[66]. On the other hand, the remaining GP parameters are kept consistent. The number of generations for the GP process is 51. The GP crossover, mutation and reproduction rates are 80%, 10% and 10% respectively. The GP's maximum depth during initialisation is 4 and the maximum depth for all generations is 8. Tournament selection of size 7 is used during the selection and breeding procedure.

4.7 Machine Breakdown GP Results and Discussion

This section covers the evaluation of machine breakdown GP approach against the DJSS problem. To compare the GP approaches in their abilities to consistently evolve effective rules, the baseline GP and the machine breakdown GP approaches evolve 30 rules on the simulation configuration $\langle$ 15%, 262.5, 3 $\rangle$ of the simulation model Δ'. The 30 rules are then applied to the rest of the simulation model, and the overall performances of the rule sets and the best rules from the GP approaches are compared. Afterwards, we carry out structural and behavioural analysis of the rules evolved by the GP approaches.

4.7.1 Machine Breakdown GP Terminal Evaluation

The performance evaluation of the evolved rules for the machine breakdown GP approaches is broken down into two steps. First, we compare the performances of all rules evolved by the GP approaches against each other. An evolved rule is applied to each simulation configuration in DJSS simulation model Δ' 30 times with different seeds to obtain 30 MWT objective values. The 30 objective value are averaged out to get the performance of the evolved rules, and the performances of the rule evolved by the machine breakdown GP approaches are compared against the baseline GP approach. Second, the best rules for all simulation configurations

used for testing are also compared against each other. For the best rules, the 30 MWT objective values obtained after the best rule is applied to each simulation configuration are compared against each other. In the performance evaluation, the rules evolved by the baseline GP approach is denoted as "GP", the rules evolved by the machine breakdown GP that use Augmented GP Terminals (Section 4.5.2) is denoted as "GP-Aug", rules evolved using WT and RTR (Section 4.5.3) are denoted "GP-WT" and "GP-RTR" respectively.

Rule Set Results

The results of the performance evaluation are shown in Table 4.4. In the table, $\langle \pi, r_t, h \rangle$ denotes that the simulation configuration has the respective breakdown level π, repair time r_t, and due date tightness h. In addition, each entry $x \pm y$ is the mean (x) and standard deviation (y) of the performance $Perf$ of the rules for the simulation configuration respectively. A set of GP rule is *significantly* better than another set of rules if they satisfy the two-sided Wilcoxon's signed-rank test at $p = 0.05$. If a set of GP evolved rules that use the machine breakdown GP terminals is significantly better than the set of benchmark GP rules in terms of their performance, then the particular entry is highlighted blue.

Although the differences are not significant, the results show that the three machine breakdown approaches (GP-Aug, GP-WT and GP-RTR) have observable improvement in the performances than the benchmark GP with respect to some simulation configurations. In particular, the GP-WT rules have slightly better performances for all simulation configurations than the benchmark GP rules. In addition, the GP-RTR rules have slightly better performance than the benchmark GP rules for most simulation configurations except configurations $\langle\ 0\%,\ 37.5,\ 5\ \rangle$ and $\langle\ 15\%,\ 37.5,\ 5\ \rangle$. Finally, the results of the comparison between the GP-Aug rules and the benchmark GP rules are mostly mixed, where GP-Aug rules are slightly better or worse than the benchmark rules on roughly an equal number of simula-

Table 4.4: Comparison of the performances of the baseline GP approach and the machine breakdown GP approaches over the simulation configurations. Rules are evolved from $\langle\, 15\%, 262.5, 3 \,\rangle$.

Model Subset		MB			GP
		GP-Aug	GP-WT	GP-RTR	
MWT ($\times 10^2$)	$\langle\, 0\%, 37.5, 5 \,\rangle$	0.74 ± 0.17	0.66 ± 0.07	0.67 ± 0.13	0.67 ± 0.16
	$\langle\, 0\%, 37.5, 3 \,\rangle$	1.12 ± 0.16	1.05 ± 0.07	1.06 ± 0.12	1.07 ± 0.15
	$\langle\, 0\%, 137.5, 5 \,\rangle$	0.60 ± 0.16	0.53 ± 0.06	0.53 ± 0.11	0.53 ± 0.14
	$\langle\, 0\%, 137.5, 3 \,\rangle$	1.31 ± 0.18	1.24 ± 0.07	1.24 ± 0.12	1.26 ± 0.16
	$\langle\, 0\%, 262.5, 5 \,\rangle$	0.74 ± 0.17	0.66 ± 0.07	0.66 ± 0.13	0.66 ± 0.16
	$\langle\, 0\%, 262.5, 3 \,\rangle$	1.36 ± 0.19	1.28 ± 0.08	1.29 ± 0.13	1.30 ± 0.16
	$\langle\, 2.5\%, 37.5, 5 \,\rangle$	1.60 ± 0.89	1.54 ± 0.52	1.55 ± 0.75	1.59 ± 0.69
	$\langle\, 2.5\%, 37.5, 3 \,\rangle$	2.92 ± 0.98	2.78 ± 0.52	2.86 ± 0.76	2.95 ± 0.88
	$\langle\, 2.5\%, 137.5, 5 \,\rangle$	38.39 ± 11.90	36.07 ± 11.10	38.91 ± 10.38	39.20 ± 12.65
	$\langle\, 2.5\%, 137.5, 3 \,\rangle$	42.53 ± 12.23	40.49 ± 11.10	42.96 ± 10.87	43.23 ± 12.89
	$\langle\, 2.5\%, 262.5, 5 \,\rangle$	88.51 ± 25.15	89.94 ± 23.00	92.91 ± 23.63	94.43 ± 27.18
	$\langle\, 2.5\%, 262.5, 3 \,\rangle$	92.11 ± 24.34	93.81 ± 21.81	96.36 ± 22.91	98.17 ± 26.54
	$\langle\, 5\%, 37.5, 5 \,\rangle$	1.64 ± 0.44	1.53 ± 0.20	1.57 ± 0.35	1.58 ± 0.36
	$\langle\, 5\%, 37.5, 3 \,\rangle$	2.76 ± 0.57	2.68 ± 0.30	2.74 ± 0.52	2.78 ± 0.50
	$\langle\, 5\%, 137.5, 5 \,\rangle$	9.04 ± 2.17	8.29 ± 2.03	8.91 ± 1.89	9.05 ± 2.42
	$\langle\, 5\%, 137.5, 3 \,\rangle$	11.14 ± 2.18	10.65 ± 1.87	11.04 ± 1.97	11.28 ± 2.29
	$\langle\, 5\%, 262.5, 5 \,\rangle$	36.43 ± 11.35	34.32 ± 10.75	37.00 ± 10.13	37.38 ± 12.40
	$\langle\, 5\%, 262.5, 3 \,\rangle$	36.56 ± 11.29	34.33 ± 10.20	36.74 ± 9.61	37.27 ± 12.09
	$\langle\, 10\%, 37.5, 5 \,\rangle$	3.69 ± 0.61	3.53 ± 0.27	3.60 ± 0.50	3.63 ± 0.60
	$\langle\, 10\%, 37.5, 3 \,\rangle$	4.60 ± 0.54	4.51 ± 0.26	4.57 ± 0.44	4.63 ± 0.51
	$\langle\, 10\%, 137.5, 5 \,\rangle$	6.11 ± 1.21	5.79 ± 0.35	5.89 ± 0.81	6.07 ± 1.26
	$\langle\, 10\%, 137.5, 3 \,\rangle$	8.15 ± 1.28	7.91 ± 0.44	8.02 ± 0.86	8.29 ± 1.39
	$\langle\, 10\%, 262.5, 5 \,\rangle$	11.72 ± 1.95	11.07 ± 1.56	11.43 ± 1.56	11.74 ± 2.22
	$\langle\, 10\%, 262.5, 3 \,\rangle$	13.14 ± 1.50	12.61 ± 1.52	13.08 ± 1.21	13.29 ± 1.76
	$\langle\, 15\%, 37.5, 5 \,\rangle$	6.20 ± 0.67	5.89 ± 0.21	6.07 ± 0.50	6.06 ± 0.80
	$\langle\, 15\%, 37.5, 3 \,\rangle$	7.73 ± 0.63	7.52 ± 0.18	7.66 ± 0.50	7.74 ± 0.82
	$\langle\, 15\%, 137.5, 5 \,\rangle$	9.02 ± 0.94	8.53 ± 0.59	8.69 ± 0.64	8.81 ± 1.17
	$\langle\, 15\%, 137.5, 3 \,\rangle$	11.73 ± 0.81	11.38 ± 0.42	11.55 ± 0.58	11.71 ± 1.10
	$\langle\, 15\%, 262.5, 5 \,\rangle$	12.55 ± 1.23	12.01 ± 1.29	12.31 ± 0.95	12.48 ± 1.44
	$\langle\, 15\%, 262.5, 3 \,\rangle$	16.60 ± 1.27	16.08 ± 1.33	16.37 ± 0.97	16.59 ± 1.50

tion configurations. However, the observed differences are not statistically significant. Nonetheless, by analysing the rules further, we have obtained an important understanding of how GP can be applied effectively to the machine breakdown problem.

Best Rule Results

After comparing the rule sets, the best rules from the GP approaches are compared against each other. The best rules are the rules that have the best average MWT performances over all simulations out of the rule sets. The results of the best rules being applied to each simulation configuration are shown in Table 4.5, where each entry $x \pm y$ is the mean (x) and standard deviation (y) of the MWT values generated by the best rule after being applied to 30 independent runs over the simulation configuration. Student's t-test at $p = 0.05$ is used to determine if the MWT values of the schedules generated by the best rule are *significantly* better than another best rule. If a machine breakdown GP best rule is significantly better than the best baseline GP rule, then it is highlighted blue.

The best rules from the machine breakdown GP approaches show a greater difference in the performance to the best rule from the benchmark GP approach. The best machine breakdown GP rules are significantly better than the best benchmark GP rule for certain simulation configurations, e.g., all three machine breakdown GP rules perform better than the GP rule for the $\langle\ 15\%, 262.5, 3\ \rangle$ simulation configuration. Therefore, although the overall differences in the GP approaches are not significant, machine breakdown GP approaches show promise in that they have the potential to evolve higher quality individual rules than the baseline GP approach.

After comparing the best machine breakdown GP rules against the best baseline GP rule, we compared the best rule evolved by the baseline GP approach against the following man-made dispatching rules: weighted SPT (wSPT), wATC and weighted COVERT (wCOVERT). The comparison in the previous Chapter (Figures 3.3 and 3.4 on pages 93 and 94 respec-

Table 4.5: Comparison of the best rules over the simulation configurations.

Data Subset		MB			GP
		GP-Aug	GP-WT	GP-RTR	
MWT ($\times 10^2$)	$\langle$ 0%, 37.5, 5 $\rangle$	0.69 ± 0.31	0.63 ± 0.36	0.63 ± 0.31	0.63 ± 0.29
	$\langle$ 0%, 37.5, 3 $\rangle$	1.06 ± 0.27	1.01 ± 0.28	1.01 ± 0.27	1.01 ± 0.25
	$\langle$ 0%, 137.5, 5 $\rangle$	0.55 ± 0.17	0.47 ± 0.16	0.49 ± 0.14	0.50 ± 0.15
	$\langle$ 0%, 137.5, 3 $\rangle$	1.24 ± 0.34	1.21 ± 0.39	1.23 ± 0.37	1.18 ± 0.33
	$\langle$ 0%, 262.5, 5 $\rangle$	0.67 ± 0.32	0.60 ± 0.30	0.61 ± 0.27	0.63 ± 0.29
	$\langle$ 0%, 262.5, 3 $\rangle$	1.30 ± 0.46	1.25 ± 0.46	1.27 ± 0.43	1.24 ± 0.41
	$\langle$ 2.5%, 37.5, 5 $\rangle$	1.42 ± 0.96	1.72 ± 1.49	1.13 ± 0.70	1.19 ± 0.56
	$\langle$ 2.5%, 37.5, 3 $\rangle$	2.47 ± 2.17	2.54 ± 2.14	2.15 ± 2.04	2.33 ± 2.48
	$\langle$ 2.5%, 137.5, 5 $\rangle$	15.84 ± 9.08	16.73 ± 10.11	15.46 ± 8.12	17.85 ± 7.83
	$\langle$ 2.5%, 137.5, 3 $\rangle$	19.36 ± 11.74	21.39 ± 12.86	18.30 ± 12.22	21.28 ± 11.67
	$\langle$ 2.5%, 262.5, 5 $\rangle$	44.44 ± 24.05	51.07 ± 23.93	48.60 ± 26.00	46.87 ± 24.62
	$\langle$ 2.5%, 262.5, 3 $\rangle$	50.49 ± 35.66	56.92 ± 37.17	54.08 ± 37.06	52.61 ± 35.43
	$\langle$ 5%, 37.5, 5 $\rangle$	1.59 ± 0.63	1.63 ± 0.79	1.41 ± 0.53	1.42 ± 0.49
	$\langle$ 5%, 37.5, 3 $\rangle$	2.92 ± 1.16	2.89 ± 1.25	2.54 ± 0.94	2.55 ± 0.88
	$\langle$ 5%, 137.5, 5 $\rangle$	4.59 ± 2.45	4.44 ± 2.91	4.61 ± 2.41	5.62 ± 2.84
	$\langle$ 5%, 137.5, 3 $\rangle$	7.00 ± 3.67	7.06 ± 4.24	7.01 ± 3.47	8.03 ± 3.90
	$\langle$ 5%, 262.5, 5 $\rangle$	14.29 ± 7.32	15.76 ± 8.02	14.78 ± 7.12	16.50 ± 6.88
	$\langle$ 5%, 262.5, 3 $\rangle$	14.30 ± 5.66	14.56 ± 6.10	14.97 ± 4.69	16.87 ± 4.78
	$\langle$ 10%, 37.5, 5 $\rangle$	3.83 ± 1.34	3.83 ± 1.26	3.29 ± 1.10	3.40 ± 1.24
	$\langle$ 10%, 37.5, 3 $\rangle$	4.96 ± 1.64	4.84 ± 1.42	4.45 ± 1.23	4.38 ± 1.18
	$\langle$ 10%, 137.5, 5 $\rangle$	5.65 ± 1.13	5.41 ± 1.45	5.23 ± 1.12	5.58 ± 1.28
	$\langle$ 10%, 137.5, 3 $\rangle$	7.71 ± 1.41	7.62 ± 1.60	7.16 ± 1.50	7.64 ± 1.69
	$\langle$ 10%, 262.5, 5 $\rangle$	8.39 ± 2.58	7.56 ± 3.37	8.18 ± 2.77	9.28 ± 2.95
	$\langle$ 10%, 262.5, 3 $\rangle$	10.12 ± 1.38	9.27 ± 1.59	10.44 ± 1.52	11.12 ± 1.48
	$\langle$ 15%, 37.5, 5 $\rangle$	5.89 ± 1.48	5.52 ± 1.59	5.60 ± 1.31	5.56 ± 1.16
	$\langle$ 15%, 37.5, 3 $\rangle$	7.70 ± 1.33	7.52 ± 1.27	7.37 ± 1.26	7.07 ± 0.99
	$\langle$ 15%, 137.5, 5 $\rangle$	7.66 ± 1.27	6.79 ± 1.30	7.55 ± 1.15	7.90 ± 1.16
	$\langle$ 15%, 137.5, 3 $\rangle$	10.97 ± 1.56	10.62 ± 1.88	10.51 ± 1.34	10.74 ± 1.11
	$\langle$ 15%, 262.5, 5 $\rangle$	10.27 ± 1.25	8.85 ± 1.04	9.89 ± 1.11	10.91 ± 1.03
	$\langle$ 15%, 262.5, 3 $\rangle$	13.76 ± 1.99	13.10 ± 2.15	13.90 ± 1.75	14.81 ± 1.67

tively) showed that EDD and FIFO rules perform poorly in comparison to the best GP rules for the DJSS problem, and therefore are omitted for this comparison. The man-made dispatching rules are applied to the DJSS simulation model in the same way that the best GP rules are applied to the simulation model as described above. Comparing the baseline GP rule against the man-made rules us to verify whether GP rules maintain effectiveness over man-made dispatching rules to DJSS problems with both dynamic job arrivals and machine breakdowns. These results are presented in Table 4.6. If a man-made rule that performs significantly *better* than the best baseline GP rule for a simulation configuration, then the result is highlighted blue. Otherwise, if a man-made rule performs significantly *worse* than the best baseline GP rule, then the result is highlighted red.

As we can see from the results, the baseline GP approach generally outperforms the man-made dispatching rules in almost all scenarios with non zero machine breakdowns except for the configurations $\langle\ 5\%, 37.5, 5\ \rangle$, $\langle\ 10\%, 37.5, 5\ \rangle$ and $\langle\ 10\%, 37.5, 3\ \rangle$ for the wATC and the wCOVERT rules. On the other hand, the baseline GP rule generally performs worse than the man-made rules for zero machine breakdown scenarios, where the wATC and the wCOVERT rules perform significantly better on all zero machine breakdown simulation configurations except for $\langle\ 0\%, 137.5, 3\ \rangle$ and $\langle\ 0\%, 262.5, 5\ \rangle$. This is likely attributed to the fact that the baseline GP rule is evolved from the simulations from the configuration $\langle\ 15\%, 262.5, 3\ \rangle$ which has the highest breakdown level. It is likely that the training simulations have significantly different properties to simulations in zero machine breakdown scenarios. Therefore, the GP rule evolved on $\langle\ 15\%, 262.5, 3\ \rangle$ likely do not have the necessary behaviour required to handle zero machine breakdown scenarios as effectively as the man-made dispatching rules but handle the non zero machine breakdown scenarios more effectively.

Table 4.6: Comparison of the best GP rule against man-made dispatching rules.

Data Subset		GP	Man-made		
			wSPT	wATC	wCOVERT
MWT ($\times 10^2$)	$\langle$ 0%, 37.5, 5 $\rangle$	0.63 ± 0.29	0.58 ± 0.26	0.10 ± 0.14	0.26 ± 0.18
	$\langle$ 0%, 37.5, 3 $\rangle$	1.01 ± 0.25	0.99 ± 0.23	0.61 ± 0.19	0.73 ± 0.21
	$\langle$ 0%, 137.5, 5 $\rangle$	0.50 ± 0.15	0.47 ± 0.16	0.05 ± 0.04	0.18 ± 0.08
	$\langle$ 0%, 137.5, 3 $\rangle$	1.18 ± 0.33	1.16 ± 0.32	0.75 ± 0.30	0.89 ± 0.28
	$\langle$ 0%, 262.5, 5 $\rangle$	0.63 ± 0.29	0.59 ± 0.26	0.10 ± 0.09	0.26 ± 0.16
	$\langle$ 0%, 262.5, 3 $\rangle$	1.24 ± 0.41	1.21 ± 0.38	0.79 ± 0.33	0.93 ± 0.35
	$\langle$ 2.5%, 37.5, 5 $\rangle$	1.19 ± 0.56	3.64 ± 2.80	2.81 ± 2.66	3.12 ± 2.68
	$\langle$ 2.5%, 37.5, 3 $\rangle$	2.33 ± 2.48	5.44 ± 4.49	5.04 ± 4.48	5.16 ± 4.47
	$\langle$ 2.5%, 137.5, 5 $\rangle$	17.85 ± 7.83	82.04 ± 42.73	81.74 ± 42.99	81.59 ± 42.87
	$\langle$ 2.5%, 137.5, 3 $\rangle$	21.28 ± 11.67	85.11 ± 40.12	85.16 ± 40.37	85.15 ± 40.33
	$\langle$ 2.5%, 262.5, 5 $\rangle$	46.87 ± 24.62	231.06 ± 96.06	231.60 ± 96.25	231.35 ± 96.35
	$\langle$ 2.5%, 262.5, 3 $\rangle$	52.61 ± 35.43	230.64 ± 89.45	230.79 ± 89.65	230.66 ± 89.50
	$\langle$ 5%, 37.5, 5 $\rangle$	1.42 ± 0.49	1.55 ± 0.58	0.66 ± 0.47	1.02 ± 0.52
	$\langle$ 5%, 37.5, 3 $\rangle$	2.55 ± 0.88	2.95 ± 1.15	2.49 ± 1.15	2.66 ± 1.15
	$\langle$ 5%, 137.5, 5 $\rangle$	5.62 ± 2.84	20.18 ± 12.75	19.42 ± 12.82	19.54 ± 12.77
	$\langle$ 5%, 137.5, 3 $\rangle$	8.03 ± 3.90	23.07 ± 12.06	22.78 ± 12.11	22.80 ± 12.03
	$\langle$ 5%, 262.5, 5 $\rangle$	16.50 ± 6.88	85.01 ± 43.42	84.95 ± 43.81	84.62 ± 43.72
	$\langle$ 5%, 262.5, 3 $\rangle$	16.87 ± 4.78	78.52 ± 34.70	78.40 ± 34.78	78.36 ± 34.73
	$\langle$ 10%, 37.5, 5 $\rangle$	3.40 ± 1.24	3.47 ± 1.28	2.35 ± 1.31	2.67 ± 1.20
	$\langle$ 10%, 37.5, 3 $\rangle$	4.38 ± 1.18	4.39 ± 1.32	4.02 ± 1.29	4.11 ± 1.27
	$\langle$ 10%, 137.5, 5 $\rangle$	5.58 ± 1.28	9.62 ± 3.63	8.70 ± 3.74	8.78 ± 3.61
	$\langle$ 10%, 137.5, 3 $\rangle$	7.64 ± 1.69	12.78 ± 5.01	12.62 ± 4.95	12.59 ± 4.96
	$\langle$ 10%, 262.5, 5 $\rangle$	9.28 ± 2.95	24.72 ± 15.23	24.25 ± 15.49	24.05 ± 15.41
	$\langle$ 10%, 262.5, 3 $\rangle$	11.12 ± 1.48	24.88 ± 9.21	24.71 ± 9.19	24.75 ± 9.15
	$\langle$ 15%, 37.5, 5 $\rangle$	5.56 ± 1.16	6.82 ± 1.93	5.89 ± 2.20	5.92 ± 1.92
	$\langle$ 15%, 37.5, 3 $\rangle$	7.07 ± 0.99	8.58 ± 2.14	8.30 ± 2.15	8.41 ± 2.22
	$\langle$ 15%, 137.5, 5 $\rangle$	7.90 ± 1.16	11.81 ± 2.27	11.30 ± 2.35	11.23 ± 2.29
	$\langle$ 15%, 137.5, 3 $\rangle$	10.74 ± 1.11	14.24 ± 2.59	14.15 ± 2.69	14.03 ± 2.64
	$\langle$ 15%, 262.5, 5 $\rangle$	10.91 ± 1.03	20.00 ± 4.89	20.08 ± 4.97	19.60 ± 4.95
	$\langle$ 15%, 262.5, 3 $\rangle$	14.81 ± 1.67	24.58 ± 5.51	24.60 ± 5.47	24.53 ± 5.50

4.7.2 Rule Analysis

Rule analyses for the machine breakdown GP approaches are separated into structural analysis and behavioural analysis. The structural analysis calculates the distribution of the GP terminals in the evolved rules, and analyse the layout of the best GP rules to observe how the GP terminals are utilised. The behavioural analysis applies the best rules to a sample simulation to investigate decisions that are made by the rules during the decision situations collected from simulation runs over Δ'. Details on decision situations are sampled is given below.

Structural Analysis

The GP terminal distribution of the rules evolved by the GP approaches are shown in Figure 4.5. The terminal distributions are calculated using the procedure described in Section 4.4.2, where the proportion of GP terminals for each individual rules are first calculated before being plotted as shown in the figure. We used the box plot for visualisation in this analysis since certain certain terminal proportions were not clearly visible in a bubble plot (which is used for Figure 4.3 in Section 4.4.2). In addition, we show the proportion of the WT and RTR in each of the rule in the 30 rules evolved by GP-WT and GP-RTR approaches in Figure 4.6.

In addition to calculating the terminal proportions, we analyse the best rules from the GP approaches. The best rules are simplified to remove any redundant branches (e.g. if an `if` will only return the "if" sub-branch, then the `if` operator is replaced with the "if" branch) before analysing the structures of the rules. The simplified best rules for GP-Aug, GP-WT, and GP-RTR are shown in Figure 4.7a, 4.7b, and 4.7c respectively.

From the Figure 4.5, the most frequent terminal that occurs for the GP approaches are due date (DD), then weight (W) and then processing time (PT). The high frequency of due date terminal is consistent with the trend shown in the GP terminal analysis carried out in Section 4.4.2 above (in

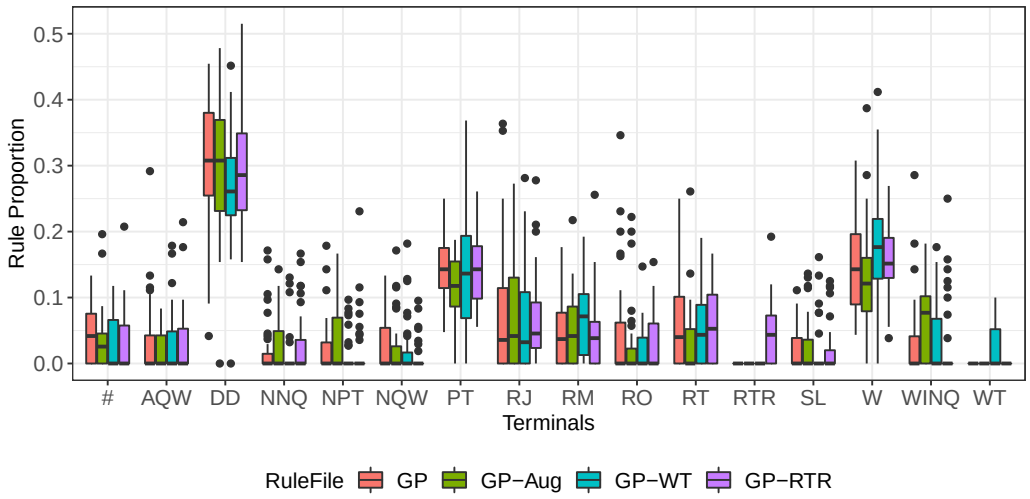


Figure 4.5: The proportion of terminals used by the sets of rules evolved by the GP-HH approaches.

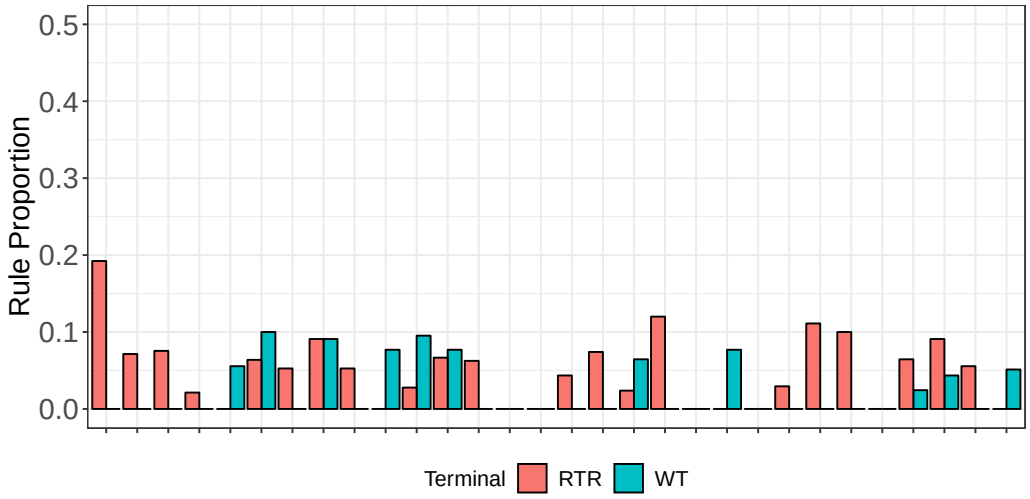
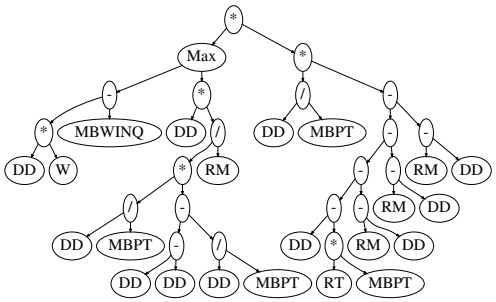
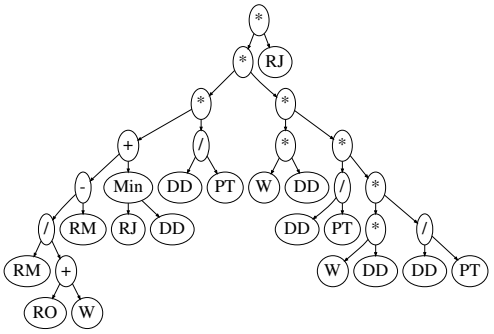


Figure 4.6: The proportion of the machine breakdown terminals for the individual rules in the GP-RTR and GP-WT approaches.

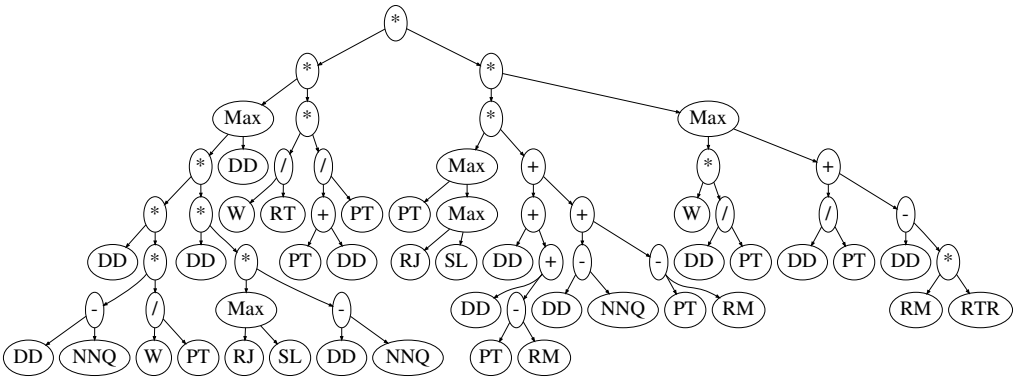
Figure 4.3), where the proportion of due date terminals used by GP increased as the machine breakdown level of the DJSS simulations used during training increased. As the rules are trained on $\langle$ 15%, 262.5, 3 $\rangle$,



(a) Best GP-Aug rule



(b) Best GP-WT rule



(c) Best GP-RTR rule

Figure 4.7: The structures of the best rules found by the GP approaches.

i.e., on 15% breakdown level, it is expected that the evolved rules have a high proportion of due date terminals. In addition, the proportion of

processing time terminal in Figure 4.5 is also consistent with the results shown in Figure 4.3. These results reinforce the hypothesis from Section 4.4.2 that processing urgent jobs based on their due dates may be a more reliable method of generating good schedules for the high machine breakdown scenarios than prioritising shorter jobs, as the rules have a higher proportion of due date terminal than processing time terminal.

An important observation from the best rules evolved by GP-WT and GP-RTR is the lack of machine breakdown terminals that make up the best rules. The best rule from GP-WT has *no* occurrence of the WT terminal that is incorporated into the terminal set, and the best rule from GP-RTR has *one* occurrence of the RTR terminal. Further analysis of the proportion of WT and RTR that make up the other rules that are evolved by GP-WT and GP-RTR as shown in Figure 4.6 shows that the two terminals are used quite infrequently. Therefore, it may be possible that the standard GP approach is not strong enough to search the heurstic space to generate an effective combination of the machine breakdown GP terminals with the standard GP terminals to evolve machine breakdown specific rules.

For the best rules from the GP approaches, the method in which the non-machine breakdown related terminals are combined may also be a factor in the effectiveness of the rules. These include the frequent occurrence of important terminals such as the job's weights and processing time in the best GP rules. Intuitively, important jobs with short processing time should be prioritised. However, in all three machine breakdown GP rules (and the best benchmark GP rule), there are many segments of the tree that form DD/PT, which indicates that the best rules prioritise jobs with high due date and low processing time. This is contrary to the expectation that jobs with low due dates (i.e. jobs that are more urgent) should be prioritised first. A possible explanation is that the due date terminal is *time-variant*, i.e., expected due dates of jobs steadily increases with the duration of the simulation. On the other hand, the processing time terminal is *time-invariant*, i.e., the expected processing times of jobs do not increase

with the length of the simulation for the DJSS problem. Therefore, the relative differences in the due dates of jobs waiting at a machine become smaller compared to the absolute values of the due dates during the duration of the simulation, whereas the relative differences of processing times of jobs stay the same compared to the absolute values of the processing times. This may result in the due date of a job for long simulations being used by the best rules as an arbitrarily large value that can be combined with the processing time terminal using the protected / operator to form a composite that prioritises short processing times.

Behavioural Analysis

To analyse the behaviours of the best rules when machine breakdowns occur, decision situations are sampled from the simulations with machine breakdowns in the DJSS simulation model Δ' and the best rules are applied to the decision situations. The wATC rule is used as the *sampling rule* (as described in Section 3.4.1 (Page 89)), which is applied to a simulation to sample decision situations as follows. The decision situations that are considered for selection are decision situations where at least one job will be interrupted by a future machine breakdown if it was selected to be processed (i.e. the machine's breakdown time is less than the current time plus the job's processing time), and the breakdown time is less than the mean processing time of operations (μ in Table 4.3). Afterwards, 50 decision situations are selected randomly among all decision situations and the best rules from the baseline GP approach and the machine breakdown GP approaches are applied to the 50 decision situations sampled from the simulation.

After the best rules are applied to the decision situations, we compare the processing time of the job selected by the best rules against the shortest processing times of the jobs waiting at the machine, i.e., jobs that would be selected by the SPT dispatching rule. Processing time related terminals are a significant component of the rules evolved by GP according to the results

from Figure 4.3, and likely has a significant impact on the decisions made by the evolved rules. By comparing the jobs selected by the best rules against the jobs selected with the shortest processing times, we can gauge the complexity of the decision made by the best rules. The results for each simulation configuration is shown in Figure 4.8, where the "Similarity" denotes the similarity of the best rule's decision to the SPT rule for the particular simulation configuration.

As we can see from the figure, the best rules have a very high similarity to the SPT dispatching rule in the decision situations where the machine breakdowns occur shortly after the decision situation. The similarities are prominent for configurations with low repair times ($r_t = 37.5$), which indicates that the best rules in those scenarios will likely make "simple" decisions. On the other hand, the best rules show greater differences in the behaviours to the SPT rules for higher levels of machine breakdowns, even though the decision situations in the different simulations are sampled so that they are interrupted by machine breakdowns. This indicates that the properties of the simulations before the machine breakdown are significantly different from each other for the different machine breakdown scenarios. In addition, this result supports the earlier finding in Figure 4.3 that the urgencies of the jobs based on the due dates are utilised more often in simulations with high levels of machine breakdowns.

4.8 Chapter Summary

The goal of this chapter was to develop an effective machine breakdown GP approach for the DJSS problem with dynamic job arrivals and machine breakdowns that evolves machine breakdown GP rules. To achieve this goal, we first carried out an investigation into the efficacy of a standard GP approach. This chapter first develops a DJSS simulation model for evaluating the standard GP-HH approach by modifying an existing DJSS simulation model proposed in the literature [61] and modify the training

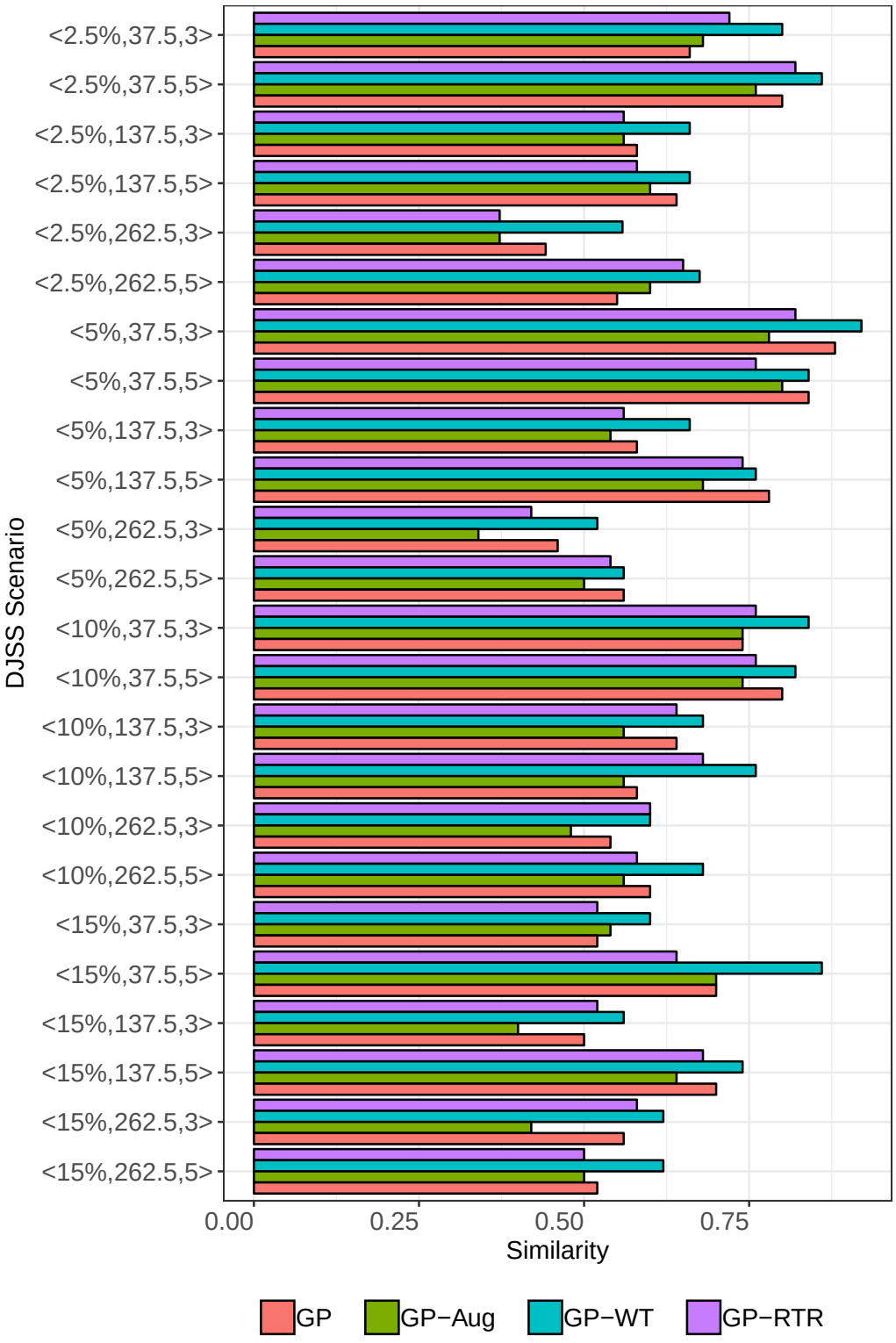


Figure 4.8: Similarity of the behaviours of the best rules against the SPT rule.

procedure proposed by Burke et al. [23] to evaluate the GP over the simulation model. In addition, GP evolved rules are analysed by comparing the distribution of the GP terminals between the rules evolved on different machine breakdown scenarios. Afterwards, the chapter develops a series of machine breakdown GP terminals that are incorporated into the GP approaches. The first set of GP terminals (called "augmented terminals") replace existing processing time related terminals (PT, NPT and WINQ) with equivalent terminals that take potential machine breakdown into account. The second set of GP approaches (called "reactive terminals") add new terminals (RTR and WT) that give information on the current state of the shop floor.

The efficacy investigation shows that GP evolved rules performances are specific to the machine breakdown scenario they are evolved on. In other words, the rules evolved on zero machine breakdown scenarios perform significantly worse than the rules evolved on high machine breakdown scenarios on the high machine breakdown simulations, and vice versa. This is contrary to the results in the literature when GP is evolved on simulations with different levels of utilisation rate [105], where rules evolved on DJSS simulations with high utilisation rate also generally perform well on DJSS simulations with low utilisation rate. In addition, the generalist rules are heavily biased towards the simulations with no machine breakdowns and perform poorly on high machine breakdown level.

The analysis of the terminal distribution shows that the evolved rules are mostly made up of processing time terminal (PT), followed by the due date terminal (DD). The rules evolved on a high machine breakdown level have a lower proportion of processing time terminal compared to the rules evolved on a lower machine breakdown level, but have a higher proportion of due date terminals. This is potentially due to the added uncertainty associated with the duration of time required to process a job in the high machine breakdown level.

Finally, although the machine breakdown GP terminals do not signifi-

cantly improve on the baseline GP approach overall, the best rules evolved by the machine breakdown GP approaches show promise and outperform the baseline GP's best rule in certain simulation configurations. The analysis of the evolved rules shows that the rules in evolved using the reactive machine breakdown GP terminals (GP-WT and GP-RTR) have very few reactive machine breakdown GP terminals in the structures of the evolved best rules. Therefore, it is possible that the standard GP's heuristic search ability is not strong enough to utilise the machine breakdown GP terminals to form structures more effective for the DJSS problem over non-machine breakdown GP terminals. In addition, the behavioural analysis shows that in the decisions that are shortly followed by machine breakdowns, the best rules share significant similarities to the SPT rule for simulations with low machine breakdown level. More investigation will need to be made in the future.

Although this chapter investigates GP for the DJSS problem with dynamic job arrivals and machine breakdowns, it does not consider a GP approach that evolves multiple rules. The previous chapter showed that the ensemble GP approach can outperform the standard GP approach for the DJSS problem with dynamic job arrivals, but with no machine breakdowns. It may be possible that handling the DJSS problem with dynamic job arrivals and machine breakdowns using multiple rules may be more effective than evolving a single rule that handles the different machine breakdown scenarios simultaneously. Therefore, the next chapter will develop GP approaches that are an alternative method of handling the DJSS problem with dynamic job arrivals and machine breakdowns by evolving a *portfolio* of rules that can handle different machine breakdown scenarios.

Chapter 5

Developing Multitask GP-HH Approaches

5.1 Introduction

As shown in Chapter 4, the standard GP approach for the DJSS problem with dynamic job arrivals and machine breakdowns is sensitive to the different machine breakdown levels. In the DJSS problem with a wide range of machine breakdown scenarios, a single "generalist" rule has a difficult time covering for the different machine breakdown scenarios effectively. An alternate approach is to evolve "specialist" rules separately for each machine breakdown scenario, but this procedure is time-consuming. Instead, it may be possible to handle the DJSS problem by decomposing the problem into multiple tasks based on the machine breakdown scenarios to be handled by a multitask GP approach. Multitask learning has the advantage over learning rules separately as the transfer of knowledge between problem domains has been shown to boost the effectiveness of rules [27]. Therefore, treating the different machine breakdown scenarios as the multiple tasks and applying a multitask GP approach to the problem may improve the efficiency of solving the DJSS problem. In addition, multitasking may also reduce the number of evaluations required by the GP

individuals to save on computation cost while being competitive in terms of the performance to the standard GP approach.

5.1.1 Chapter Goals

The goal of this chapter is to develop *multitask* GP approaches that evolve an effective *portfolio* of rules over the DJSS problem. Each rule in the portfolio aims to be effective on a specific machine breakdown scenario. This is contrary to the ensemble approaches proposed in Chapter 3, where the ensemble aims to be effective over the entire DJSS problem that they are applied to. Multitask learning ideas and approaches have been effectively and successfully applied to problems outside of DJSS [27, 116]. However, multitask learning has not yet been incorporated into GP to evolve rules for the DJSS problem. This objective develops two multitask GP approaches for the DJSS problem with different focuses as follows.

- First, this chapter develops a novel multitask GP approach by incorporating a *niching* technique [133, 95]. Niching techniques are used in the literature to promote diversity between the GP individuals in a single GP run [95]. When combined with GP as a multitask optimisation method for the DJSS problem, niching can effectively aid GP in searching for the effective rules specialised for each type of problem instance (i.e. tasks). If a specific GP individual is effective for a particular task, then filtering out other individuals that behave similarly but have poor performances may help improve the search for better individuals in the specific task. The aim of the niched GP (NGP) approach is to improve on the standard GP approach over the DJSS problem by incorporating multitasking while being competitive in terms of the computation cost. In other words, the niched GP approach aims to evolve effective rules that can perform well over the entire problem domain (called the "generalist" rule) and evolve rules that are effective for specific machine breakdown scenarios (called

the "specialist" rules) simultaneously. The generalist rules are likely to be effective on the DJSS problem instances where the properties of the problem instances are unknown. The specialist rules can be applied to DJSS problem instances where the properties of the problem instances are known in advance, as the specialist rule will likely have a better performance over the problem instances with the specific machine breakdown scenario than the generalist rules.

- Second, this chapter develops a novel multitask GP approach to the DJSS problem which uses the neighbourhood relations between the different tasks in the DJSS problems to best determine the tasks that the GP individuals in the population should be assigned to. For simplicity, this multitask GP approach is called neighbourhood-based GP (NBGP). The neighbourhood relations between the different tasks are formed based on the properties of the problem instances. Afterwards, the individuals are assigned to specific tasks, and can potentially branch out to other tasks if they perform well on the assigned task. By using neighbourhood relations and focusing individuals to specific tasks, GP can minimise wasting computation time evaluating the GP individuals in tasks where they perform poorly on, and prioritise the evaluation of good individuals. In other words, the aim of the NBGP approach is to improve the computation cost required for GP to evolve the rules over the standard GP approach while maintaining competitive performance.

- In summary, the goal of NGP is to use niching to evolve more effective rules than the standard GP approach. On the other hand, the goal of NBGP is to use neighbourhood relations to minimise redundant evaluations and to improve the *both* the effectiveness of the evolved rules and the efficiency of evolving rules over the standard GP approach. For both multitask GP approaches, we carry out behavioural analysis to obtain further insight into the properties of the

evolved rules on the different machine breakdown scenarios.

5.1.2 Chapter Organisation

The remainder of this chapter is organised as follows. First, we provide a general description of how the multitask GP is applied to the DJSS problem and the representation of tasks. The subsequent section describes the niched GP approach, which is then followed by the section that describes the NBGP approach. Afterwards, the design of the experiments is discussed in the third subsequent section, followed by the results, discussion and analysis section and the chapter summary section.

5.2 Applying Multitasking to a DJSS Problem

This section provides a description of how the DJSS problem is separated into a multitasking problem. First, we provide a description of the DJSS simulation model, followed by a detail of how the DJSS simulation model is handled by the multitask GP approach.

5.2.1 DJSS Simulation Model

The DJSS simulation model is the modified Holthaus's simulation model described in Section 4.3.1 (Page 127) that generates the job arrivals and machine breakdowns stochastically. For reference, the parameters used for the simulation model is given in Table 5.1.

The DJSS simulation model results in a total of seven different machine breakdown scenarios, where each machine breakdown scenario consists of two different configurations with the due date tightness values. This results in a total of 14 different simulation configurations. A simulation configuration is abbreviated as $\langle x, y, z \rangle$, where x denotes the machine breakdown level, y is the mean repair time, and z denotes the due date tightness.

Table 5.1: The parameters used for simulating a DJSS problem instance.

Parameters		Value
Shop floor parameters	Number of machines	10
	Warm up jobs	500
	# completed jobs before simulation termination	2500
	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

In addition, a machine breakdown scenario is abbreviated as $\langle x, y \rangle$, where x denotes the machine breakdown level and y the mean repair time.

5.2.2 Multitasking on the DJSS Simulation Model

To evolve rules for each machine breakdown scenario of the DJSS simulation model, a standard GP approach would need to apply seven independent runs of the GP process. This process is shown in Figure 5.1, where each GP process that is applied to a machine breakdown scenario evolves a specialist rule specific to the scenario. In other words, applying a standard GP approach to scenario $\langle$ 2.5%, 25, 5 $\rangle$ evolves a dispatching rule specialised for the machine breakdown scenario with breakdown level 2.5% and mean repair time 25. On the other hand, a generalist rule takes the entire DJSS simulation model and outputs a single rule for all seven machine breakdown scenarios.

In this thesis, a multitask GP approach is applied to all machine break-

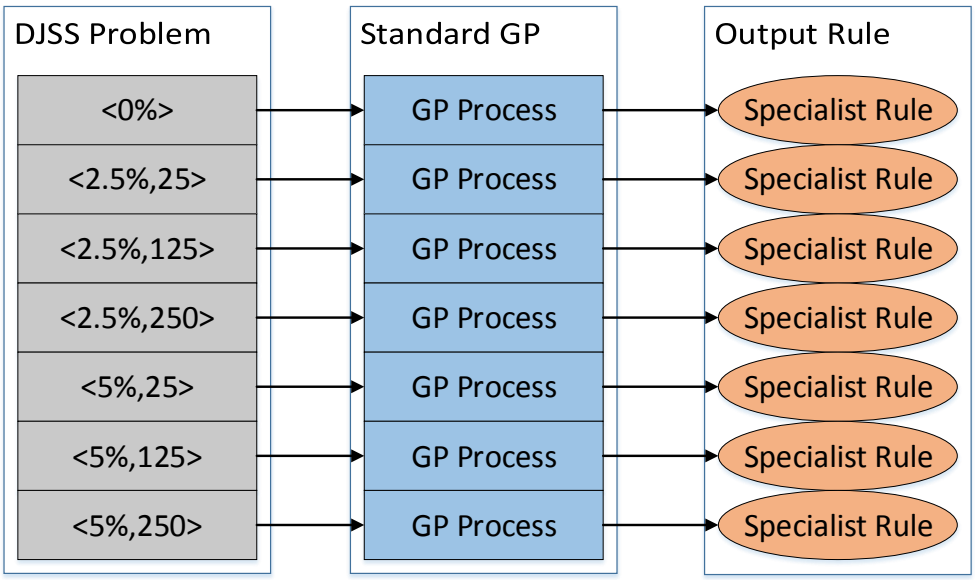


Figure 5.1: Evolving specialist rules from the DJSS simulation model using a standard GP approach.

down scenarios simultaneously. Each machine breakdown scenarios is treated as a "task" in the multitask optimisation procedure. An example of this process is shown in Figure 5.2, where the entire DJSS simulation model is used by the multitask GP process. The multitask GP approach then outputs multiple specialist rules simultaneously that are specific to the machine breakdown scenarios. The NBGP approach follows the process provided in the figure and outputs specialist rules. On the other hand, the NGP approach outputs both the specialist rules and a generalist rule when it is applied to the DJSS simulation model. The differences are due to the design of the NGP and NBGP algorithms, which are discussed further below.

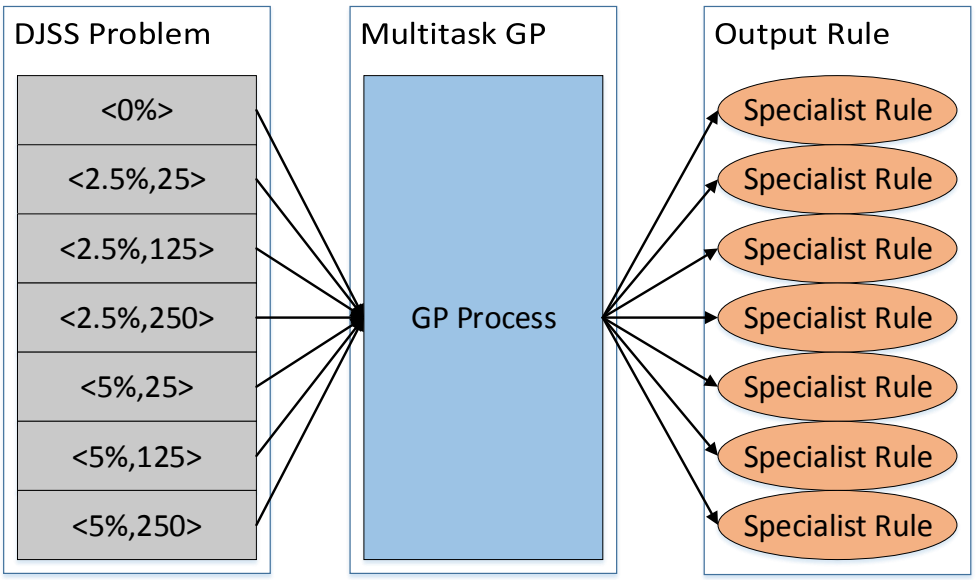


Figure 5.2: Evolving specialist rules from the DJSS simulation model using a multitask GP approach.

5.3 Niched GP Approach for Multitasking

This section covers the NGP approach that is used to evolve a generalist rule and specialist rules for the different machine breakdown scenarios simultaneously. This allows us to share the latent features discovered for the different machine breakdown scenarios during the NGP process to improve the effectiveness of the generalist and the specialist rules. First, we give a general overview of the niched GP process. This is then followed by further discussion on specific details on the GP selection and evaluation procedure, which includes a description of the niching technique used to adjust the fitness of the GP individuals.

5.3.1 Niched GP Overview

The NGP approach in this paper is extended from a niched GP approach proposed by Mei et al. [95] used to evolve a diverse set of rules. For the NGP approach, each machine breakdown scenario in the DJSS problem represent a "niche" that can be occupied by a GP individual that has good performance for the particular scenario. In other words, the NGP keeps track of the GP individuals that are the best for the different machine breakdown scenarios during training. By doing this, GP is demonstrated 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 NGP process, a single population of the GP individuals are first randomly initialised and the set of specialist rule $\mathcal{S}$ is empty. For each generation when a GP individual x is being evaluated, the individual is first evaluated on the "general" training set $\mathcal{T}$ that contains machine breakdown scenarios $S_1, \ldots, S_N$ to calculate its fitness $f(x)$. The individual x's performance over the different machine breakdown scenario in training set $\mathcal{T}$ is also used to update the current generation niched individuals $t_1, \ldots, t_N$. After all individuals have been evaluated, the current generation niched individuals $t_1, \ldots, t_N$ are further evaluated on niched training sets with problem instances specific to single-machine breakdown scenarios. If the current generation niched individuals perform better on the specific niched training sets than the overall niched individuals R_{s1}, $\ldots$, R_{sN}, then the overall niched individuals are updated to the current generation niched individuals. The evaluation procedure for calculating GP individuals' fitness and for updating the overall niched individuals is described in full in the subsection below (Section 5.3.2). Afterwards, the clearing algorithm is used to adjust the GP individuals' fitness before the selection procedure, which is described in further detail below (Section 5.3.3). This process continues until the maximum number of generations has been reached, at which point the algorithm reports the best overall rule as the generalist rule R_g and the set of specialist rules $R_{s1}, \ldots, R_{sN}$.

The pseudocode that summarises the niched GP process is shown in Algorithm 3.

Algorithm 3: The pseudocode for the NGP approach that evolves a generalist rule and specialist rules.

Input : Training set $\mathcal{T}$ consisting of machine breakdown scenarios $S_1, \ldots, S_N$ and niched training sets $\mathcal{V}_1, \ldots, \mathcal{V}_N$.
Output: The set of specialist rules $R_{s1}, \ldots, R_{sN}$ and the generalist rule R_g.

initialise GP population $\mathcal{P}$;
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* x *in GP population* $\mathcal{P}$ **do**
        evaluate individual x on training set $\mathcal{T}$ and calculate performances $Perf(x, \mathcal{T}_1), \ldots, Perf(x, \mathcal{T}_N)$ over the different scenarios;
        update $f(x)$ from the performances over $Perf(x, \mathcal{T}_1), \ldots, Perf(x, \mathcal{T}_N)$ the different niches;
    end
    update current generation niche individuals $t_1, \ldots, t_N$;
    update generalist rule R_g;
    for t_n *in* $t_1, \ldots, t_N$ **do**
        evaluate t_n on niched training set $\mathcal{V}_n$ and calculate the performance $Perf(t_n, \mathcal{V}_n)$;
        update $R_{sn} \leftarrow t_n$;
    end
    apply the `Clearing`$(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ procedure and adjust GP individual fitness values;
    carry out selection and breeding for population $\mathcal{P}$;
end
output the set of specialist GP rules $\mathcal{S}$ and the best overall GP rule R_g;

5.3.2 GP Evaluation Procedure

A GP individual x is applied to DJSS problem instances as a non-delay priority dispatching rule. After generating the schedule for a problem instance I using individual x, the MWT objective value is normalised using

the wATC reference rule [148]. The performance $Perf(x, \mathcal{T}_n)$ for a subset of the training set $\mathcal{T}$ that belongs to niche n is calculated as the average normalised objective values of the schedules generated as shown in Equation (5.1).

$$Perf(x, \mathcal{T}_n) = \frac{1}{|\mathcal{T}_n|} \sum_{I \in \mathcal{T}_n} Obj'(x, I) \tag{5.1}$$

The performance is then compared against the current generation niched individual t_n's performance $Perf(t_n, \mathcal{T}_n)$ over $\mathcal{T}_n$, and the current generation niched individual is updated to individual x if individual x's performance is better than current generation niched individual's performance.

The fitness $f(x)$ of the individual is calculated as the average of the performances over the subsets of the training set $\mathcal{T}$ as shown in Equation (5.2). The fitness of the individual is adjusted further based on its proximity to the overall niched individual as described below in Section 5.3.3. If individual x is the best fitness found so far, then the best overall individual R_g is updated to individual x.

$$f(x) = \frac{1}{N} \sum_{n=1}^{N} Perf(x, \mathcal{T}_n) \tag{5.2}$$

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 $R_{s1}, \ldots, R_{sN}$. To compare the current generation niched individuals t_n to overall niched individual R_{sn}, 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). The calculation for the performance $Perf(x, \mathcal{V}_n)$ of a GP individual x on the niched training set $\mathcal{V}_n$ is the same as calculation of niche specific performance on the generalist training set $\mathcal{T}$ (as shown in Equation (5.1)). If the performance $Perf(t_n, \mathcal{V}_n)$ of the current generation niched individual t_n is better than

the performance $Perf(R_{sn}, \mathcal{V}_n)$ of the overall niched individual R_{sn} over the niched training set $\mathcal{V}_n$, then R_{sn} is updated to the current generation's niche individual t_n. Otherwise, the individual R_{sn} is kept the same.

Given that the same training set $\mathcal{T}$ is used, the NGP approach will likely have a greater computation time than a standard GP approach with the aim of evolving one rule from a single population because it further evaluates the current generation niched individuals on the niched training sets $\mathcal{V}_1, \ldots, \mathcal{V}_N$ on top of the standard evaluation procedure. When evolving dispatching rules to 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 [103]. As the NGP approach will have additional # *of niches* $\times$ *niched training sets sizes* simulation runs, the NGP approach requires a total of $|\mathcal{P}| \times |\mathcal{T}| + |\mathcal{V}_1| + \cdots + |\mathcal{V}_N|$ simulation runs. If the niched training sets have the same number of DJSS problem instances, then the total number of simulation runs is $|\mathcal{P}| \times |\mathcal{T}| + N \times |\mathcal{V}|$, where $|\mathcal{V}|$ the size of the niched training sets. However, the NGP approach will still have significantly shorter computation time compared to evolving generalist and specialist rules using a standard GP separately, which requires $2 \times |\mathcal{P}| \times |\mathcal{T}|$ simulation runs to evaluate all the GP individuals per generation.

5.3.3 GP Clearing Procedure

After the overall niche individuals $R_{s1}, \ldots, R_{sN}$ are updated and the best individual for the current generation has been found, the clearing algorithm `Clearing`$(\mathcal{P}, R_{s1}, \ldots, R_{sN}, R_g, \sigma, \kappa)$ is applied as the selection procedure before the individuals undergo the standard tournament selection procedure. Clearing is a niching technique that has been used by a number of existing multimodal approaches in the literature [123, 121, 95] and is generally used to find multiple solutions in the solution space. The clearing algorithm used by NGP is modified from the algorithm used by Mei

et al. [95], and is the NGP approach's method of sharing the behaviours of the niched individuals to the rest of the GP population for multitasking. Mei et al.'s [95] approach defines the niches automatically based on the distances of the individuals and does not distinguish between the different machine breakdown scenarios in the DJSS problem. For the NGP approach, the niched individuals $R_{s1}, \ldots, R_{sN}$ are set before the clearing procedure (as described in the overview in Section 5.3.1) and then are used to adjust the fitness of the other GP individuals in the population. This is done by removing individuals with poor performances that behave too similar to the niched individuals from the population. Therefore, we use a distance measure $\Delta(x, x')$ that calculates the differences in the behaviours of individuals x and x' in the clearing algorithm in Algorithm 4. In the algorithm, the individuals with poor performances within the *distance* σ from either the specialist rules R_{s1}, ..., R_{sN} or the generalist rule R_g are removed from the GP population if the niche has reached its capacity κ.

The distance $\Delta(x, x')$ between two individuals x and x' are calculated based on how the two individuals rank the jobs at sample decision situations. The distance measure is adapted from the phenotypic measure used by Hildebrandt and Branke [59] and has been used effectively by Mei et al. [95] to evolve a diverse set of dispatching rules. Individuals x and x' assigns priorities to the jobs waiting at the machine for a decision situation. The priorities are then used to determine the ranks of the jobs. The job j with the highest priority has a rank $r_j = 1$, the job j' with the second highest priority has a rank $r_{j'} = 2$, and so on. Afterwards, the wATC *reference rule* applied to the same decision situation and ranks $r_j(x)$ and $r_j(x')$ assigned by individuals x and x' respectively to the job j that is assigned the highest priority by the reference rule are compared. This is done over Ω decision situations, which gives us the distance $\Delta(x, x')$ as shown in Equation (5.3).

Algorithm 4: The pseudocode for the niched clearing procedure `Clearing`$(\mathcal{P}, R_{s1}, \ldots, R_{sN}, R_g, \sigma, \kappa)$.

Input : A GP population $\mathcal{P}$, overall niched GP individuals $R_{s1}, \ldots, R_{sN}$, a generalist rule R_g, a niche radius σ, and a niche capacity κ.
Output: Updated population $\mathcal{P}$ with adjusted fitness values.

```
sort fitness of GP population P from best to worst;
for each individual R from R_s1, ..., R_sN, R_g do
    set current niche size size ← 1;
    for each individual x in GP population P do
        if x is not a niched individual or the best individual then
            if fitness f(x) < ∞ and Δ(R, x) < σ then
                if size < κ then
                    size ← size + 1;
                else
                    f(x_j) ← ∞;
                end
            end
        end
    end
end
```

$$\Delta(x, x') = \sqrt{\sum_{i=1}^{\Omega} (r_j(x) - r_j(x'))^2} \tag{5.3}$$

The Ω decision situations are obtained by applying the weighted shortest-processing-time (WSPT) rule to the problem instances in the first DJSS problem instance I in the training set $\mathcal{T}$. During processing as the WSPT rule is applied to a problem instance I, any decision situation with at least ϵ jobs is considered. Therefore, the range of values for the ranks of jobs, which depends on the number of waiting jobs at the machine, is at least ϵ at the minimum. Out of the remaining decision situations, Ω is selected randomly sampled from the problem instance.

5.4 Neighbourhood-based GP Approach for Multitasking

The aim of NBGP is to improve on the efficiency of evolving rules to DJSS problems with different machine breakdown scenarios while maintaining competitive performance to a standard GP approach. To do this, NBGP first defines the neighbourhood relations between the different types of DJSS problem instances based on the machine breakdown properties of the problem instances. The neighbourhood relation is then used to determine which individual should be evaluated on which machine breakdown scenario so that the GP only needs to evaluate the individuals on the training instances they are likely to perform well on. In terms of computation time, NBGP will likely have a significantly lower computation cost compared to a standard GP approach that applies the individuals to all machine breakdown scenarios during training. Below, we provide a general overview of NBGP, describe how the individuals are evaluated on the training instances, and how the selection procedure is carried out when the individuals have not been evaluated on all problem instances in the training set.

5.4.1 NBGP Approach Overview

The general overview of the NBGP approach is shown in Algorithm 5. After carrying out the standard initialisation procedure where the GP individuals are randomly generated, the individuals undergo evaluation on the training set $\mathcal{T}$ that consists of DJSS problem instances with different machine breakdown scenarios. The subsets of the training set $\mathcal{T}$ that consists of the DJSS problem instances under a specific machine breakdown scenario S are denoted $\mathcal{T}_S$.

For the first generation, the individuals are applied to the entire training set $\mathcal{T}$ as a non-delay dispatching rule. After the first generation, the in-

dividuals will have a specific task (which corresponds to a machine breakdown scenario) they are assigned to, and will first be applied to the DJSS problem instances corresponding to the specific machine breakdown scenario before further evaluation is potentially performed on neighbour machine breakdown scenarios (described in Section 5.4.2). If the individual performed well in a specific machine breakdown scenario S, then they are assigned as the "specialist" rule R_{sS} for machine breakdown scenario S.

To generate the next generation of individuals after all current generation GP individuals have been evaluated, the breeding procedure `Breed`($\mathcal{P}$) is carried out for the GP population $\mathcal{P}$. The breeding procedure is a modification of the standard GP breeding procedure, and breeds specific numbers of individuals assigned to each machine breakdown scenario. Further details on the breeding procedures are given below in Section 5.4.3. The individuals bred for a specific machine breakdown scenario are generated by selecting the parents from the GP individuals that have high performances on the machine breakdown scenario or its neighbours. The best specialist individuals for every machine breakdown scenarios are also carried over via elitism. This procedure is described in full in Section 5.4.3. The final output of the NBGP approach is the last generation's best individuals for each scenario, which are outputted as the set of specialist rules $R_{s1}, \ldots, R_{sN}$. As GP individuals in the NBGP process are not always evaluated over the entire training set $\mathcal{T}$ except for the first generation, the NBGP approach does not generate a generalist rule R_g like the NGP approach.

5.4.2 Neighbourhood Scenario Evaluation Procedure

A key component of the NBGP is that the GP individuals are assigned to specific machine breakdown scenario S that they are specialised for. The individual is assigned to a specific machine breakdown scenario S after they are generated from the GP selection and breeding procedure, which

Algorithm 5: The pseudocode for the NBGP approach that evolves specialist rules.

Input : Training set $\mathcal{T}$ with neighbourhood relations between the machine breakdown scenarios $S_1, \ldots, S_N$.
Output: The specialist rules $R_{s1}, \ldots, R_{sN}$.

```
Initialise GP population P of GP individuals that do not have assigned scenarios;
for gen ← 1 to G do
    for each individual x in GP population P do
        if x has an assigned scenario S then
            evaluate individual x to training subset T_S and calculate fitness f(x, S);
        else
            evaluate individual x to training set T and calculate scenario fitness f(x, S) for all scenarios S = 1, ..., N;
        end
    end
    for each individual x with an assigned scenario S in GP population P do
        evaluate x on the neighbour scenarios (Algorithm 6).
    end
    update the specialist rules R_{s1}, ..., R_{sN};
    retain E best individuals for each scenario for the next generation;
    breed next generation of individuals from P with assigned scenarios (Section 5.4.3);
end
```

is described below (in Section 5.4.3). The GP individuals are applied to the problem instances in the training set that belongs to the specific machine breakdown scenarios, and their performances are calculated for the problem instances. The evaluation procedure on the neighbour scenarios is shown in Algorithm 6. First, the immediate neighbouring scenarios S_1, $\ldots$, S_N to the scenario S_i assigned to GP individual x are always utilised for evaluation. In other words, individual x are applied to DJSS problem instances in scenarios S_1, $\ldots$, S_N to calculate the scenario fitness $f(x, S_1)$, $\ldots$, $f(x, S_N)$. The neighbour machine breakdown scenarios are the scenar-

ios that are adjacent to scenario S in terms of the machine breakdown level and mean repair time. In general, two scenarios are neighbours to each other if they share the same machine breakdown level and have different mean repair time, or vice versa. The neighbourhood relations between the machine breakdown scenarios is discussed further below in Section 5.5.4 as we discuss the DJSS simulation model used to evaluate the NBGP approach. The machine breakdown level, mean repair times and the neighbourhood relations between the different machine breakdown scenarios are given in the experimental design section below (Section 5.5).

Algorithm 6: The pseudocode for the evaluation procedure for a GP individual on the neighbour scenarios.

```
Input : A GP individual x, a training set T, and an assigned starting machine
        breakdown scenario S.
Output: Updated GP individual x that has been evaluated on the neighbour
        scenarios.
evaluate individual x on the starting scenario S;
initialise a scenario queue q;
for each scenario S_i that neighbours scenario S do
    queue up neighbour scenario S_i on the queue q for evaluation;
end
while scenario queue q is not empty do
    poll scenario S' from queue q;
    if individual x has not been applied to scenario S' then
        evaluate individuals on training subset T_S';
        for each scenario S'_i that neighbours scenario S' do
            calculate probability Prob(x, S'_i) (Equation (5.4));
            randomly offer scenario S'_i based on the probability Prob(x, S'_i);
        end
    end
end
```

After the GP individual x has been evaluated on the immediate neighbour scenarios, individual x has the additional opportunity to be evaluated on indirect neighbour scenarios. For example, if an individual has

been evaluated on scenario S' and there a scenario S'_i that neighbours scenario S', then given that the individual has not already been evaluated on scenario S'_i the probability of the individual being evaluated on scenario S is given by Equation (5.4), which is a min-max normalisation that converts the individual's scenario fitness to a probability value. In the equation, $f_{best}(S)$ is the fitness of the best individual for scenario S, and $f_{worst}(S)$ is the fitness of the worst individual for scenario S. The checks to jump to a neighbour scenario occur after the individual has been evaluated on the machine breakdown scenarios and continues until either the individual fails the checks (which is random but based on the probability calculated in Equation (5.4)), or the individual has been evaluated on all machine breakdown scenarios.

$$Prob(x, S'_i) = 1 - \left| \frac{f(x, S') - f_{best}(S')}{f_{best}(S') - f_{worst}(S')} \right| \tag{5.4}$$

5.4.3 Selection and Breeding Procedures

The next generation GP individuals for the population are generated as follows. First, the top E individuals from each scenario are transferred from the current generation's GP population to the next generation's GP population. Afterwards, the NBGP approach cycles through the machine breakdown scenarios one-by-one and generates equal numbers of individuals for each machine breakdown scenario. Starting from machine breakdown scenario S_1 out of the scenarios $S_1, \ldots, S_N$, the parent GP individuals to be used in crossover, mutation and reproduction is selected as follows. The order of machine breakdown scenarios is not significant for the selection and the breeding procedures.

Tournament Selection

The parents are selected for breeding the next generation individual for machine breakdown scenario S_i using tournament selection. To select

members for the tournament selection, the individuals that have been applied to problem instances in machine breakdown scenario S_i are considered. Individuals that have not been evaluated on the scenario S_i will be ignored during the selection process. Our assumption is that individuals which were not evaluated on scenario S_i are unlikely to have a good fitness in scenario S_i. If an individual was not evaluated on scenario S_i, then it indicates that it did not manage to jump from one of the scenarios that are neighbours to scenario S_i. In other words, the probability of the individual jumping to the scenario is likely to have been quite low, and it is likely that the individual's fitness on the neighbour scenario is quite poor. Therefore, although it is possible that the individual could have good fitness for scenario S_i (if it was evaluated on scenario S_i) but have poor fitness on the neighbour scenarios, it is more likely that the individual has poor fitness on both scenario S_i and the neighbour scenarios.

Member Ranking in Tournament Selection

After the members are selected for the tournament selection, the fitness of the GP individuals on the machine breakdown scenario S_i and the other machine breakdown scenarios are aggregated to form a score value that can be compared against each other. First, a GP individual x is ranked based on individual x's relative scenario fitness $f(x, S_i)$ compared to other individuals evaluated on the same scenario S_i^1. This is done for all machine breakdown scenarios in the DJSS training set, and the rank of individual x on scenario S_i is denoted as $r(x, S_i)$. If an individual has not been evaluated on a particular scenario and therefore do not have a rank for the scenario, then the rank is imputed from the other individuals that are part of the tournament selection. The imputation methods investigated gives the individual with the average rank over all individuals involved in the tournament selection with respect to a specific scenario. The imputation methods allow individuals that may have not have the chance to be evaluated on the machine breakdown scenarios to participate in the tournament

selection.

After the rank values have either been calculated or filled out for the individuals in the tournament, we calculate the *score* of an individual x (denoted as $Score(x, S_i)$) as shown in Equation (5.5). In the equation, S_i^1, ..., S_i^N are the neighbour machine breakdown scenarios of scenario S_i, and w is the weight of the neighbourhood scenarios. $S_i^{1'}$, ..., $S_i^{N'}$ are the neighbours of the scenarios S_i^1, ..., S_i^N, and so forth. In other words, further away from the scenario S_i under consideration, less the rank of the individual on a different scenario contributes towards the individual's score. The weight w can be used to tune the exploration and exploitation of GP, as we can potentially have good individuals in the neighbour machine breakdown scenarios that can also perform well (or poorly) on the current scenario. In addition, the weight can also tune the rules outputted by the NBGP approach to be more general to every machine breakdown scenarios or specific to the machine breakdown scenarios. The $r_{worst}(S)$ value is the worst rank out of all the GP individuals that have been applied to the problem instances in scenario S. Finally, the individual with the highest score in the tournament selection is selected to be the parent individual.

$$Score(x, S_i) = \varphi(x, S_i) + w \sum_{n=1}^{N} \varphi(x, S_i^n) + w^2 \sum_{n'=1}^{N'} \varphi(x, S_i^{n'}) + \dots \tag{5.5}$$

$$\varphi(x, S) = \frac{r_{worst}(S) - r(x, S)}{r_{worst}(S)} \tag{5.6}$$

An example of the member ranking for the tournament selection is shown in Table 5.2. The example in the table use tournament selection of size 7, the machine breakdown scenario S_i that the offspring individual is being generated for has neighbours S_i^1 and S_i^2, and $w = 0.25$. In addition, the examples have 100 GP individuals that have been applied to problem instances in each machine breakdown scenarios, which results in the average rank of 50 for each scenario. After calculating the scores for

the individuals in Equation (5.5), individual x_5 has the highest score and is the winner of the tournament selection.

Table 5.2: Member ranking for tournament selection using the average rank for missing rank values. Ranks in brackets indicate missing ranks that have been filled out. Each scenario has 100 individuals evaluated on the scenario.

Individual	Ranks			Score
	S_i^1	S_i	S_i^2	
x_1	2	20	30	1.22
x_2	40	4	34	1.28
x_3	(50)	(50)	40	0.78
x_4	35	(50)	(50)	0.79
x_5	20	10	5	**1.34**
x_6	(50)	30	(50)	0.95
x_7	1	35	(50)	1.03
Average	50	50	50	–

5.5 Design of Experiment

This section covers the design of the experiments carried out to evaluate the NGP and the NBGP approaches. First, we discuss the GP benchmark used for comparison. Afterwards, we cover the GP terminal and function sets used by all GP approaches evaluated in the experiments and the parameter settings for the GP approaches.

5.5.1 Benchmark GP Approach

To evaluate the NGP and NBGP evolved rules, we use a standard single-tree GP representation [17, 103] with the same terminal and function set (Table 5.3). This standard GP approach used as the benchmark follows the

same major steps as the benchmark GP approaches described in the previous chapters. For the evaluation procedure, a GP individual is applied to the training set as a non-delay priority dispatching rule, and the individual's fitness is calculated from the mean normalised objectives generated by the individual over the DJSS training instances (as shown in Equation (3.1)). The best individual obtained in the last generation is the output dispatching rule for the benchmark GP process.

5.5.2 GP Terminals and Function Set

The GP terminals and function sets are kept consistent as the representation, terminals and function sets used in Section 4.2.1 (Page 122). The terminals used for the benchmark GP approach, the NGP and the NBGP approaches are given in Table 5.3. The function set consists of the operators $+$, $-$, $\times$, protected $/$, binary operators $\max$, $\min$ and a ternary operator `if`.

5.5.3 GP Parameter Settings

The GP parameters for the benchmark GP, the NGP and the NBGP approaches are kept consistent as the parameters used in the previous chapters and are shown in Table 5.4 for reference. The additional parameters required for the NGP approach are the niche radius σ and the niche capacity κ. These parameters are kept consistent as the niching parameters used by Mei et al. [95]. They used niche capacity of $\kappa = 1$, i.e., any individual too close to the overall niched individuals are removed. They tested three different parameter values for the niche radius ($\sigma = 0$, $\sigma = 1$ and $\sigma = 5$), and found that niche radius $\sigma = 1$ produced the best results for evolving effective rules while niche radius $\sigma = 5$ had the worst results overall. In addition, the NBGP approach has the additional parameters neighbour weight w and the number of elites E kept per generation for the selection procedure. After carrying out preliminary tests for $w = 0.25$ and $w = 0.5$,

Table 5.3: Terminal set for GP, where a job j is waiting at the available machine m at a decision situation.

Terminal	Description
RJ	The operation ready time of job j
PT	The 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
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

we found that $w = 0.5$ frequently results in situations where the individual with the best fitness for the specific machine breakdown scenario S not being selected. Therefore, the lower neighbour weight of $w = 0.25$ is used to evolve the GP rules for the NBGP approach. In addition, we tested three different values for the GP elitism for the NBGP approach, where we retain the top E individuals for each of the different machine breakdown scenarios plus the top E individuals with the best overall fitness. The three values tested during the evaluation procedure are $E = 1, 5, 10$.

5.5.4 GP Training Procedures

To evaluate the NGP and the NBGP approaches, the GP approaches are applied to the DJSS simulation model described in Section 5.2.1 as follows to evolve the portfolio of rules for the DJSS problem. For the DJSS simulations, the seeds used to stochastically generate the job arrivals and

Table 5.4: Parameters for the benchmark GP, the NGP and the NBGP approaches.

Parameters	Approach		
	GP	NGP	NBGP
Population size	1024		
No. of generations	51		
GP mutation rate	10%		
GP reproduction rate	10%		
GP maximum initial depth	4		
GP maximum depth	8		
Selection method	Tournament selection of size 7		
wATC k value	3.0		
Niche radius σ	-	1	-
Niche capacity κ	-	1	-
Neighbour scenario weight w	-	-	0.25
No. of elites E	-	-	1, 5, 10

machine breakdowns are rotated every generation, similar to the training procedure used in the previous chapters.

Benchmark GP Training Procedure

To evolve the generalist rules from the benchmark GP approach, all simulation configurations described in the DJSS simulation model are used as the training set $\mathcal{T}$ to evaluate the GP individuals during the GP evaluation procedure. This means that each GP individual in the GP that evolves the generalist rules will be applied to 14 different DJSS simulations. To evolve the specialist rules, the simulation configurations that correspond to the different machine breakdown scenarios are used during the GP evaluation procedure, e.g., the specialist rules for machine breakdown scenario breakdown level 2.5% and mean repair time 25 are applied to the simu-

lation configurations $\langle$ 2.5%, 25, 5 $\rangle$ and $\langle$ 2.5%, 25, 3 $\rangle$. This means that while evolving the specialist rules, the GP individuals are applied to two different DJSS simulations during the evaluation procedure.

NGP Training Procedure

To evolve the portfolio of rules simultaneously from the NGP approach, all simulation configurations described in the DJSS simulation model are used as the "general" training set $\mathcal{T}$. The niches for the NGP are represented by the seven different machine breakdown scenarios in the DJSS simulation model. The current generation niched individual are the individuals that performed the best on the simulations under the different machine breakdown scenarios in training set $\mathcal{T}$. To evaluate the current generation niched individuals further, the niched training sets $\mathcal{V}_1, \ldots, \mathcal{V}_7$ correspond to the simulation configurations in the DJSS simulation model under the different machine breakdown scenarios. For example, $\mathcal{V}_1$ consists of DJSS simulations with the configurations $\langle$ 0%, 0, 5 $\rangle$ and $\langle$ 0%, 0, 3 $\rangle$. However, unlike the general training set $\mathcal{T}$, the seeds used to generate the job arrivals and machine breakdowns are not rotated for the niched training set, i.e., the simulations are fixed for the duration of the GP process.

NBGP Training Procedure

To evolve the portfolio of rules simultaneously from the NBGP approach, we first define the neighbourhood relations for the DJSS simulation model that is used as the training set $\mathcal{T}$. For the NBGP approach, two scenarios are neighbours to each other if they have the same π and different repair time distribution r, or if they have the same repair time distribution r and different machine breakdown level π. It is likely that scenarios that have both different machine breakdown level π and repair time distribution r have too much difference in their properties to be neighbours to each other,

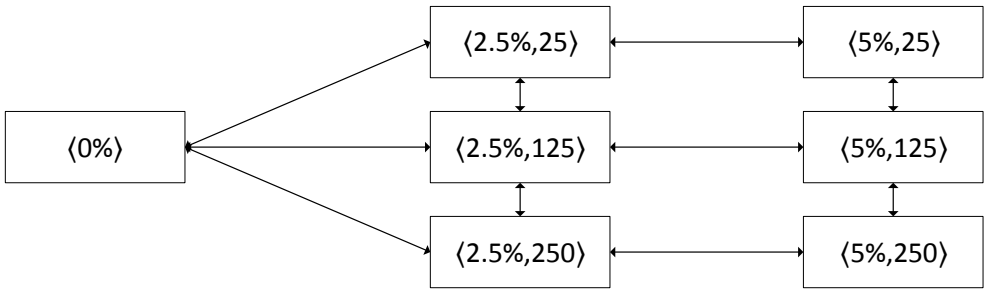


Figure 5.3: The neighbourhood relation between the different machine breakdown scenarios for the DJSS dataset.

and are not considered to be neighbours in the NBGP approach. However, since the repair time distribution is not defined for the scenario with machine breakdown level $\pi = 0$ (i.e. scenarios with no machine breakdowns), we considered zero machine breakdown scenarios to be neighbours to the three scenarios with machine breakdown level $\pi = 2.5\%$ (machine breakdown scenarios $\langle\ 2.5\%, 25, 5\ \rangle$, $\langle\ 2.5\%, 125, 5\ \rangle$, and $\langle\ 2.5\%, 250, 5\ \rangle$).

On the other hand, to evolve the rules for the benchmark GP approach to compare against the NBGP approach, different training sets are used to evaluate the GP individuals based on whether the benchmark GP is used to evolve generalist rules or specialist rules. This means that at each evaluation procedure, the GP individuals are applied to all 14 different problem instances. To evolve the specialist rules, the specific machine breakdown scenarios in the DJSS dataset is used to evaluate the GP individuals, i.e., the GP individuals are applied to two different problem instances every generation.

5.6 Results and Discussion

This section covers the evaluation of the NGP and the NBGP approach. First, we compare the number of simulation runs carried out by the dif-

ferent GP approaches per generation, and compare the computation times required to evolve the GP rules. Afterwards, the evolved rules are compared by their test performance. Finally, we carry out the analysis procedure on the structure and the behaviours of the rules. The behaviours of the rules evolved by the NGP and the NBGP approaches are first analysed, and then any particular good rules evolved by the GP approaches are also analysed further.

5.6.1 Simulation Usage and Computation Time Evaluation

Figure 5.4 shows the number of simulation runs to evaluate the GP individuals in each generation for the NBGP approach. In the figure, the hard cutoffs lines show the number of simulation runs required to evaluate the GP individuals in the benchmark GP, and the number of simulation runs required by the NGP approach. As each individual in each benchmark GP approach is applied to *two* simulations in the specific machine breakdown scenario and there are *seven* machine breakdown scenarios, the total number of simulation runs for the benchmark GP approach is $1024 \times 2 \times 7 = 14336$. On the other hand, the NGP approach requires additional $7 \times 2 = 14$ simulation runs to evaluate the current generation niched individuals to compare against the overall niched individuals, resulting in a total of $14336 + 14 = 14350$ simulation runs per generation. In addition, "NBGP Elite-x" in the figure denotes that the NBGP approach that retains $E = x$ elite GP individuals for each machine breakdown scenario. Measuring the average number of simulation runs per generation allows us to calculate the probability of a GP individual transitioning from one machine breakdown scenario to another. The number of simulation runs is expected to directly correlate with the overall computation time required to evolve the dispatching rules from the GP approaches, i.e., lower number of simulation runs is expected to result in lower computation time.

From the results, after the first round of selection and breeding have

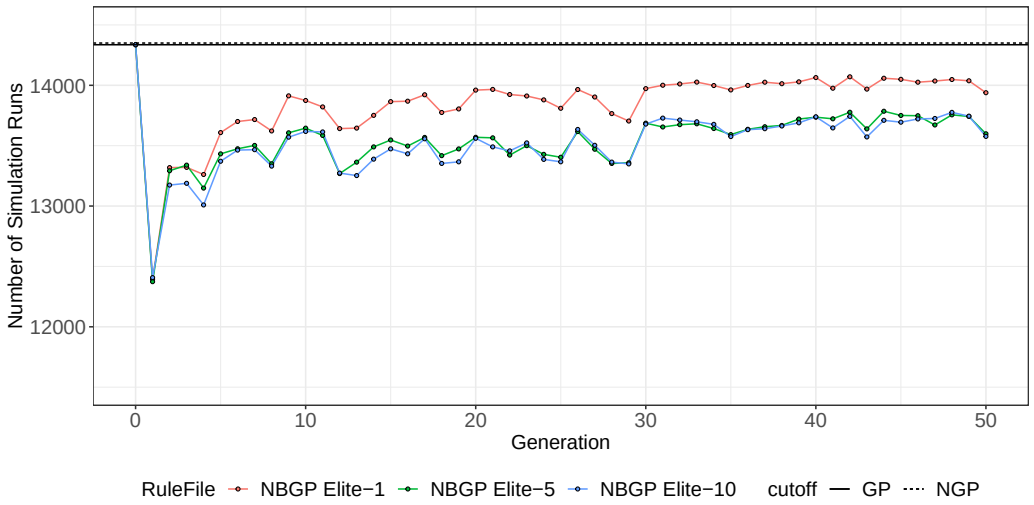


Figure 5.4: Number of simulation runs required by the NBGP approach per generation. The error bars show the standard deviation for the number of simulations over the 30 runs of the NBGP approach.

been carried out, the number of simulation runs at generation 1 is around 12400 for the NBGP approaches with the different number of elites. However, the number of simulation runs per generation increases with the later generations. On generation 50, the number of simulation runs for the NBGP Elite-1/Elite-5/Elite-10 approaches are 13940/13600/13580 respectively when averaged over the 30 independent runs. In other words, this indicates that the individuals are jumping to more neighbours in the later generations than the earlier generations. This is likely when there are multiple individuals with the same relatively high scenario fitnesses. In other words, this result indicates that the GP individuals in the machine breakdown scenarios are converging to specific rule behaviours. In addition, the number of simulation runs over the generations is the highest for the NBGP Elite-1 approach out of the NBGP approaches, and NBGP Elite-5 and Elite-10 approaches have relatively a similar number of simulation runs over the generations. In other words, the individuals in NBGP Elite-1 are more likely to transition from the neighbour scenarios to the neigh-

bour's neighbour scenario than the individuals in NBGP Elite-5 or Elite-10, i.e., NBGP Elite-1 individuals likely have good fitnesses in the neighbour scenarios. Finally, the average number of simulation runs per generation for the NBGP Elite-1/Elite-5/Elite-10 approaches are $13848/13547/13522$ respectively. In comparison to the number of simulation runs per generation for the benchmark GP approach (14336), the NBGP approaches requires $(14336 - 13848)/14336 = 3.4\%/5.5\%/5.6\%$ less simulation runs. Therefore, it is expected that NBGP Elite-10 will take the shortest amount of time to evolve the specialist rules, followed by NBGP Elite-5 then NBGP Elite-1.

After comparing the number of simulation runs per generation, the computation times required by the different GP approaches to evolve the rules are measured and shown in Table 5.5. In the table, $x \pm y$ denotes that the mean computation time is x and the standard deviation of the computation times is y. The benchmark and the benchmark GP, the NGP and the NBGP approaches are implemented in a Java program ran on Intel(R) Core(TM) i7 CPU 3.60GHz. In the table, the independent computation times required by the benchmark GP approaches to evolve rules for the different machine breakdown scenarios are summed up before the comparison is carried out. As the NGP approach evolves the generalist and the specialist rules simultaneously over single runs, the computation times on the individual machine breakdown scenarios are omitted and only shown in the "Total" category. On the other hand, the NBGP approach's computation time is shown in the "Specialist Combined" category, as it evolves the specialist rules simultaneously. The "Generalist" category is left blank for the NBGP approach as it does not evolve generalist rules. To compare the computation times, we use a two-sided Wilcoxon's signed-rank test at 95% confidence, i.e., $p = 0.05$. The computation time of the NGP or the NBGP approach is highlighted *blue* runs significantly *faster* than the benchmark GP approach for the corresponding category.

From the tables, compared to the combined amount of time required

Table 5.5: Comparison of the computation time required to evolve the rules for the GP approaches (in 10^4 seconds).

Approach		Computation Time ($\times 10^4$ s)				
		GP	NGP	NBGP Elite-1	NBGP Elite-5	NBGP Elite-10
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	–	2.10 ± 0.30	2.05 ± 0.26	1.99 ± 0.25
Total		4.27 ± 0.39	2.32 ± 0.34	–	–	–

to evolve the specialist rules or the generalist rules individually using the benchmark GP approach, the NGP approach takes a significantly longer amount of time. This is due to the additional evaluation required to further evaluate the niched individuals in the NGP 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 NGP approach is significantly faster than the benchmark GP approach. In addition, out of the computation times for the NBGP approach, NBGP Elite-1 has the highest computation time, followed by NBGP Elite-5 then NBGP Elite-10. However, the NBGP approaches do not have a significantly different computation time to the benchmark GP.

The difference in the number of simulation runs required per generation to evaluate the GP individuals in the benchmark GP and the NGP approaches do not exactly match the differences in the computation times. For the NGP approach, 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 experi-

ments 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 for evolving the specialist rules. Instead, the additional computation time is likely due to the fitness adjustments made to GP individuals that are close to the niched individuals in the clearing algorithm (Algorithm 4). It may also be likely that the evolved GP rules for the niched GP approach are also bigger, which results in longer computation time required to calculate the priorities of jobs during the simulation.

The difference in the computation times also do not exactly match the differences in the number of simulation runs for the benchmark GP and the NBGP approaches as well. For example, the relative differences in the number of simulation runs are smaller between NBGP Elite-5 and Elite-10 than between NBGP Elite-1 and Elite-5, but the relative differences in the computation times are smaller between NBGP Elite-1 and Elite-5 than between NBGP Elite-5 and Elite-10. In addition, compared to the relative differences in the number of simulation runs between NBGP and the benchmark GP discussed above, the computation times for NBGP Elite-1 on average are very similar to those of the benchmark GP approach. This is likely because of the modified breeding procedure, selection procedure and the individual fitness calculation in the NBGP approach increases the computation cost of NBGP.

5.6.2 Performance Results

After the rules are evolved by the benchmark GP, the NGP and the NBGP approaches, the rules are applied to the DJSS simulation model to obtain their performances. For the specialist rules, the evolved rules are applied to the corresponding DJSS simulation configurations with the same machine breakdown scenario the rules are specialised for. In other words, the specialist rules evolved for machine breakdown scenario $\langle\, 2.5\%, 25, 5 \,\rangle$ are applied to DJSS simulations with machine breakdown scenario $\langle\, 2.5\%,$

$25, 5 \rangle$ that use different seeds to generate the dynamic job arrivals and machine breakdowns. On the other hand, the generalist rules are applied to all 14 DJSS simulation configurations. The rules are applied to simulations with specific DJSS simulation configuration 30 independent times to get 30 MWT values. Afterwards, the averages of the MWT values are used as the overall MWT performance measures of the rules. The overall MWT performances of the specialist rules evolved by the NGP and the NBGP approaches with the different levels of elitism are then compared against the overall MWT performances of the specialist rules evolved by the benchmark GP approach to evaluate the effectiveness of the NBGP approach for each machine breakdown scenarios. As the NBGP do not evolve generalist rules, a pairwise comparison is carried out between the benchmark GP and the NGP generalist rules using the overall MWT performances. To compare the performances, we use a two-sided Wilcoxon's signed-rank test at $p = 0.05$. The performance results for the GP approaches are shown in Table 5.6 for the specialist rules, and Table 5.7 for the generalist rules. In the tables, if the rules evolved by the NGP or the NBGP approaches perform significantly *better* than the benchmark GP approach for a machine breakdown scenario, then the corresponding entry in the table that provides the performance of the NBGP rules for the machine breakdown scenario is highlighted *blue*. Otherwise, if the NGP or the NBGP rules performs significantly *worse* than the GP rules for a machine breakdown scenario, then the corresponding entry is highlighted *red*.

From the tables, we can see that the NGP approach generally outperforms the benchmark GP approach in terms of both the specialist rules performances and the generalist rules performances. In addition, the NGBP Elite-1 and Elite-5 rules significantly outperform the benchmark specialist rules for three different simulation configurations, and NGBP Elite-10 rules outperform the benchmark rules for two simulation configuration. This is negatively correlated to the computation time required to evolve the rules, as the computation times from the fastest to the slowest went

Table 5.6: Comparison showing the mean and the standard deviation of the MWT performances for the specialist rules evolved by the benchmark GP, the NGP and the NBGP approaches over the test simulation runs.

Approach		GP	NGP	NBGP		
				Elite-1	Elite-5	Elite-10
MWT ($\times 10^2$)	$\langle$ 0%, 0, 5 $\rangle$	1.94 ± 0.22	2.26 ± 0.44	2.22 ± 0.16	2.22 ± 0.16	2.30 ± 0.23
	$\langle$ 0%, 0, 3 $\rangle$	3.43 ± 0.17	4.68 ± 1.35	3.64 ± 0.54	3.64 ± 0.54	3.84 ± 0.78
	$\langle$ 2.5%, 25, 5 $\rangle$	3.55 ± 0.23	3.43 ± 0.21	3.61 ± 0.22	3.61 ± 0.22	3.66 ± 0.30
	$\langle$ 2.5%, 25, 3 $\rangle$	4.72 ± 0.22	4.60 ± 0.21	4.76 ± 0.40	4.76 ± 0.40	4.80 ± 0.35
	$\langle$ 2.5%, 125, 5 $\rangle$	4.69 ± 0.17	4.50 ± 0.10	4.56 ± 0.20	4.56 ± 0.20	4.61 ± 0.21
	$\langle$ 2.5%, 125, 3 $\rangle$	6.10 ± 0.16	5.81 ± 0.08	5.95 ± 0.15	5.95 ± 0.15	6.00 ± 0.19
	$\langle$ 2.5%, 250, 5 $\rangle$	6.28 ± 0.15	6.12 ± 0.17	6.20 ± 0.16	6.20 ± 0.16	6.21 ± 0.18
	$\langle$ 2.5%, 250, 3 $\rangle$	7.71 ± 0.15	7.55 ± 0.11	7.66 ± 0.15	7.66 ± 0.15	7.70 ± 0.14
	$\langle$ 5%, 25, 5 $\rangle$	4.58 ± 0.19	4.50 ± 0.13	4.47 ± 0.18	4.47 ± 0.18	4.52 ± 0.17
	$\langle$ 5%, 25, 3 $\rangle$	6.42 ± 0.18	6.50 ± 0.26	6.36 ± 0.15	6.36 ± 0.15	6.42 ± 0.16
	$\langle$ 5%, 125, 5 $\rangle$	6.51 ± 0.20	6.40 ± 0.15	6.44 ± 0.19	6.44 ± 0.19	6.49 ± 0.22
	$\langle$ 5%, 125, 3 $\rangle$	8.45 ± 0.26	8.42 ± 0.19	8.49 ± 0.12	8.49 ± 0.12	8.54 ± 0.19
	$\langle$ 5%, 250, 5 $\rangle$	8.74 ± 0.35	8.77 ± 0.33	8.82 ± 0.29	8.82 ± 0.29	8.85 ± 0.32
	$\langle$ 5%, 250, 3 $\rangle$	11.15 ± 0.30	11.20 ± 0.29	11.23 ± 0.27	11.23 ± 0.27	11.27 ± 0.30

from NGBP Elite-10, NGBP Elite-5 to NGBP Elite-1. Therefore, the NGP approach is effective in terms of raw performance in comparison to the benchmark GP approach and the NBGP approach is effectiveness compared to the benchmark GP approach either in terms of computation time or in terms of performance. NBGP Elite-1 and Elite-5 have comparative computation times to the benchmark GP approach but outperform the benchmark GP in multiple simulation configurations. We can also infer from the results that the inductive biases [27] that are introduced to the individuals specialised for different machine breakdown scenarios positively contribute towards the overall effectiveness of the rules evolved for the machine breakdown scenarios.

The only configuration scenarios where the benchmark specialist rules significantly outperform the NGP and the NGBP specialist rules are on the

Table 5.7: Comparison showing the mean and the standard deviation of the MWT performances for the generalist rules evolved by benchmark GP and the NGP approaches over the test simulation runs.

Approach		GP	NGP
MWT ($\times 10^2$)	$\langle$ 0%, 0, 5 $\rangle$	2.21 ± 0.10	2.16 ± 0.13
	$\langle$ 0%, 0, 3 $\rangle$	3.29 ± 0.11	3.25 ± 0.06
	$\langle$ 2.5%, 25, 5 $\rangle$	3.46 ± 0.13	3.37 ± 0.12
	$\langle$ 2.5%, 25, 3 $\rangle$	4.52 ± 0.12	4.45 ± 0.06
	$\langle$ 2.5%, 125, 5 $\rangle$	4.54 ± 0.14	4.45 ± 0.13
	$\langle$ 2.5%, 125, 3 $\rangle$	5.93 ± 0.16	5.82 ± 0.08
	$\langle$ 2.5%, 250, 5 $\rangle$	6.18 ± 0.20	6.06 ± 0.14
	$\langle$ 2.5%, 250, 3 $\rangle$	7.62 ± 0.18	7.51 ± 0.11
	$\langle$ 5%, 25, 5 $\rangle$	4.52 ± 0.21	4.40 ± 0.15
	$\langle$ 5%, 25, 3 $\rangle$	6.47 ± 0.27	6.33 ± 0.17
	$\langle$ 5%, 125, 5 $\rangle$	6.56 ± 0.26	6.42 ± 0.16
	$\langle$ 5%, 125, 3 $\rangle$	8.74 ± 0.35	8.55 ± 0.21
	$\langle$ 5%, 250, 5 $\rangle$	9.20 ± 0.46	8.95 ± 0.27
	$\langle$ 5%, 250, 3 $\rangle$	11.65 ± 0.46	11.43 ± 0.32

simulations with zero machine breakdowns, where the benchmark specialist rules perform significantly better than NGP and the NGBP Elite-10 specialist rules for $\langle$ 0%, 0, 5 $\rangle$ and $\langle$ 0%, 0, 3 $\rangle$, and better than NGBP Elite-1 and NBGP Elite-5 specialist rules for $\langle$ 0%, 0, 5 $\rangle$. To determine why the benchmark GP is particularly good at scenarios with no machine breakdowns, we carry out an analysis of the benchmark specialist rules in further detail below.

The table also shows that the generalist rules for the benchmark GP approach perform 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 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.

5.6.3 Analysis Procedure

This section carries out the analysis of the evolved rules. First, we analyse the behaviours of the rules evolved by the NGP and the NBGP approaches in terms of their behaviours. Afterwards, we investigate why the performances of the benchmark GP specialist rules are significantly better than the NGP and the NBGP approaches for the machine breakdown scenario $\langle\, 0\%, 0, 5 \,\rangle$ by analysing the benchmark GP approach's evolved rules.

NGP and NBGP Rule Behavioural Analyses

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 NGP and the NBGP approaches using the job rank distance measure proposed by Hildebrandt and Branke [59] and used by the clearing algorithm $\texttt{Clearing}(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ (Equation (5.3)). 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 distances in between the generalist rule and the specialist rule sets evolved by the NGP

approach are shown in Figure 5.5. On the other hand, the mean and the standard deviations of the distances of the specialist rule sets evolved by NBGP Elite-1, NBGP Elite-5, and NBGP Elite-10 are shown in Figures 5.6, 5.7, and 5.8 respectively. In the figures, we provide a heat map of the average distances of two sets of rules as visual aids.

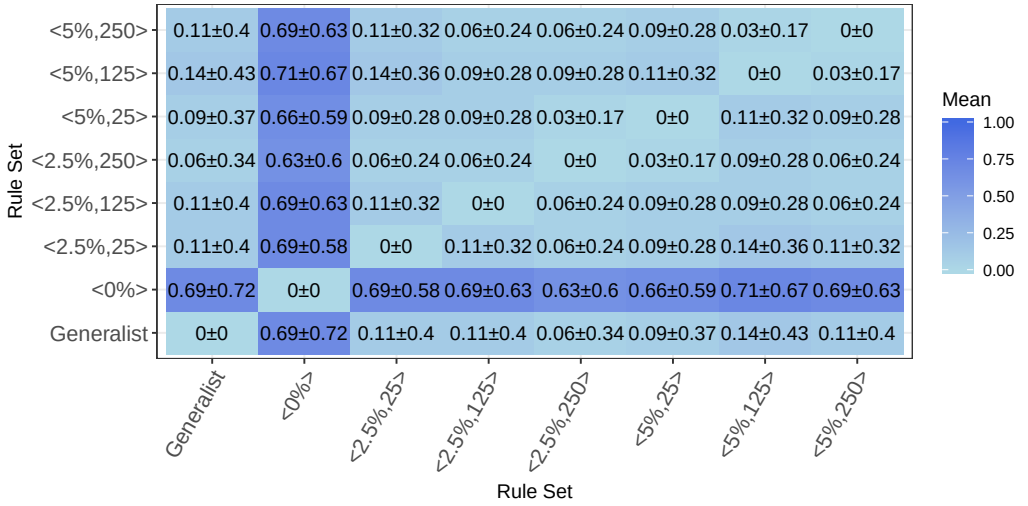


Figure 5.5: Heat map showing the pairwise mean and standard deviations of the average distances between the rules evolved by the NGP approach.

Compared to the other scenarios, the specialist rules that specialise in the scenario with zero machine breakdown have a higher average distance from the specialist rules evolved on other machine breakdown scenarios and the generalist rules for the NGP and the NBGP approaches. 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. On the other hand, the mean distances of the $\langle\ 0\%,\ 0,\ 5\ \rangle$ specialist rules for the NBGP approach gets smaller with the number of elites parameters. This could potentially be due to the elite individuals discouraging further exploration of the specific machine breakdown scenarios, as new individuals may not be generated for the specific machine breakdown scenario

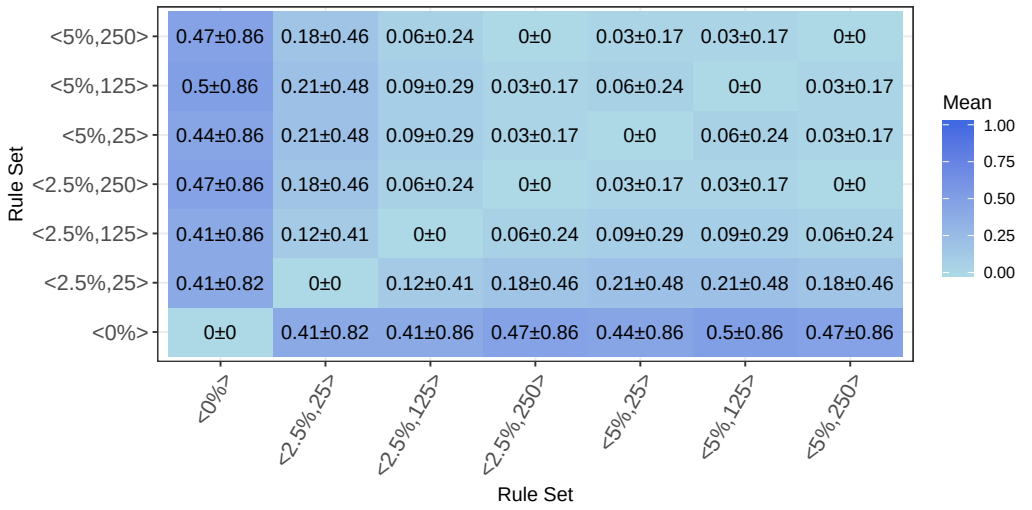


Figure 5.6: Heat map showing the pairwise mean and standard deviations of the average distances between the NBGP Elite-1 rules.

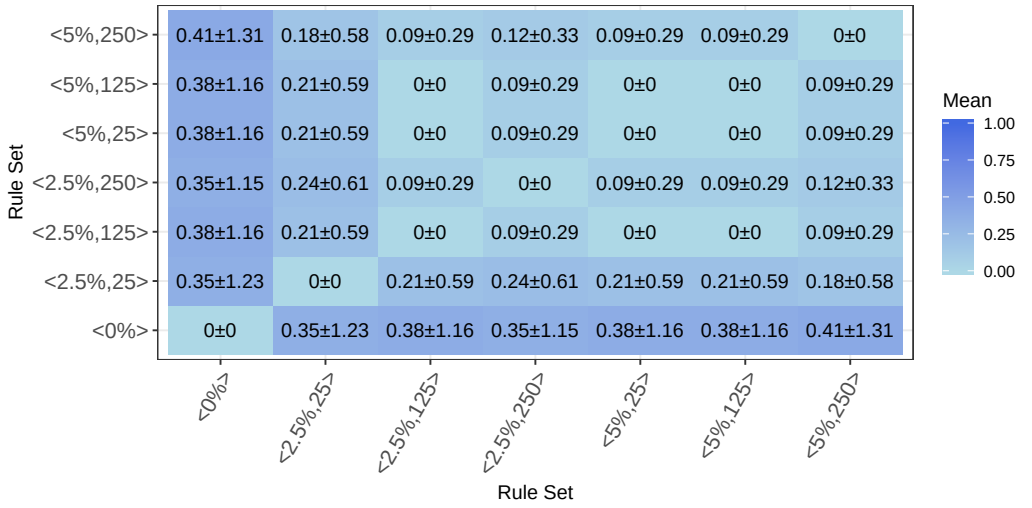


Figure 5.7: Heat map showing the pairwise mean and standard deviations of the average distances between the NBGP Elite-5 rules.

when the best individuals from the previous generation are copied over. This may result in a loss of diversity and knowledge sharing between the GP individuals for the machine breakdown scenarios. This may explain

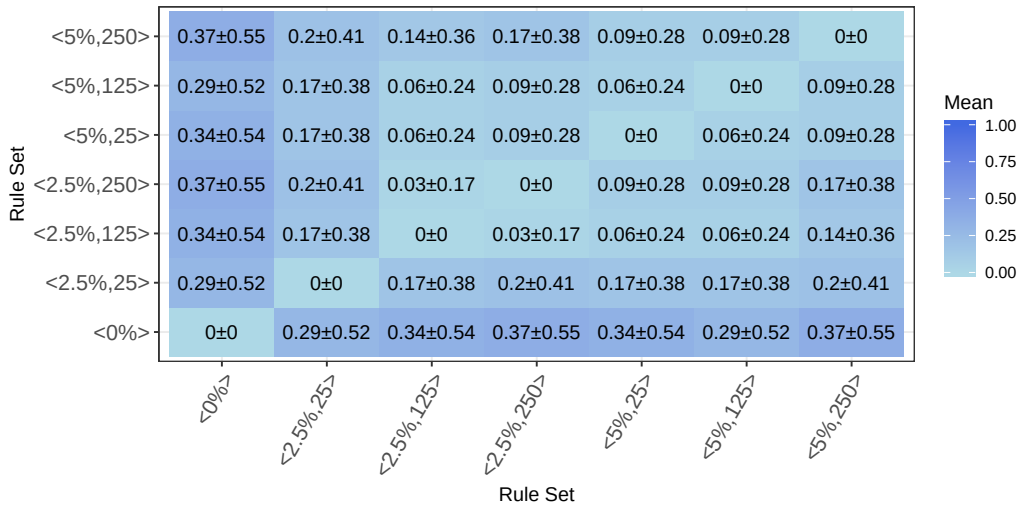


Figure 5.8: Heat map showing the pairwise mean and standard deviations of the average distances between the NBGP Elite-10 rules.

why NBGP Elite-1 has the best performance out of the three E parameters tested.

Benchmark GP Rule Structural Analysis

To determine why the benchmark GP specialist rules performed much better than the NGP and the NBGP specialist rules on the machine breakdown scenario $\langle\ 0\%,\ 0,\ 5\ \rangle$, we first analysed the performance of the specialist rules evolved by the benchmark GP approach. We found that one specialist rule had a particularly good overall MWT performance value compared to the other benchmark GP rules as shown in Figure 5.9. The structure of best benchmark GP specialist rule for scenario $\langle\ 0\%,\ 0,\ 5\ \rangle$ is shown in Figure 5.10 after carrying out basic trimming to remove redundant branches.

One significant observation that can be made from the best benchmark GP rule for scenario $\langle 0\%, 0\rangle$ is that the rule has almost no due date terminals (represented as DD in the tree). Due date is an important terminal

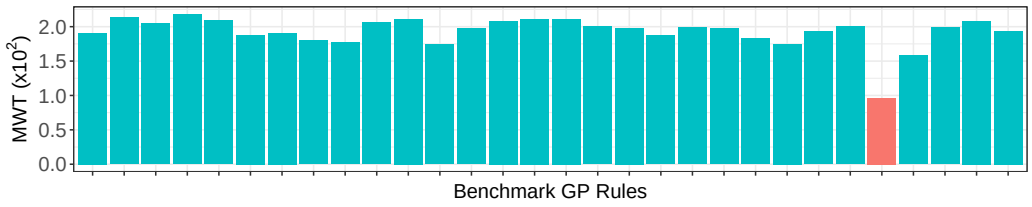


Figure 5.9: The overall MWT performance values of the individual benchmark GP rules evolved for scenario $\langle\, 0\%, 0, 5 \,\rangle$ on simulation configuration $\langle\, 0\%, 0, 5 \,\rangle$. The rule with the best MWT is highlighted red.

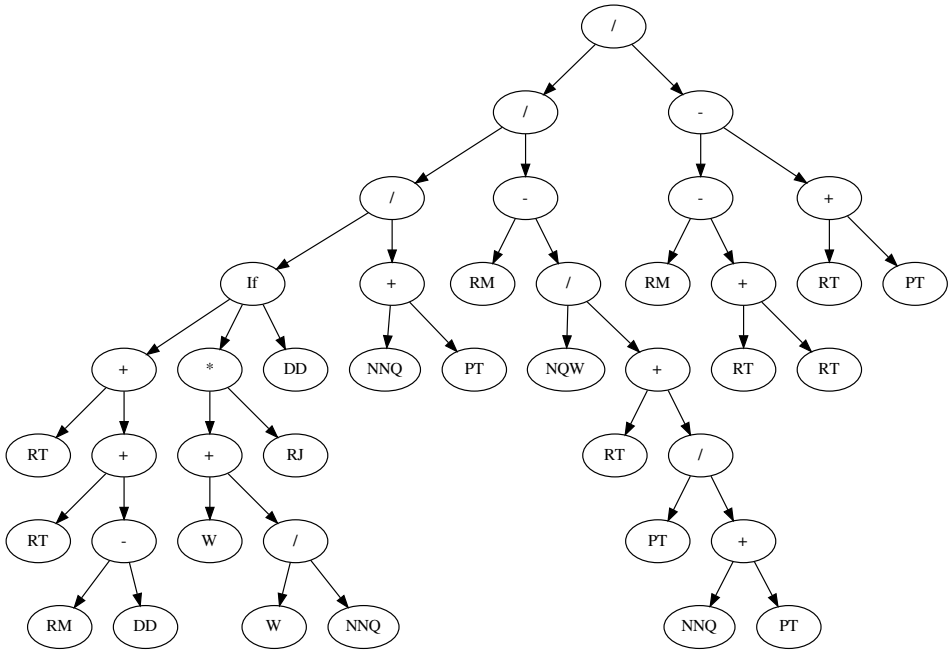


Figure 5.10: The best benchmark GP rule evolved for the machine breakdown scenario $\langle\, 0\%, 0, 5 \,\rangle$.

that occurs frequently in a tardiness related objective [97], and also occurs frequently in the other rules that are evolved by both the benchmark GP approach and the NBGP approach. The effectiveness of the rule in Figure

5.10 without using due date terminal may be because due date is one of the few *time-variant* terminals [96] that generally increases along the duration of the simulation. The only other time-variant terminal is the machine ready time terminal RM. Mei et al. [96] showed that using *time-invariant* terminals (i.e. terminals that are unaffected by the times when the decision situations are carried) generally results in improved performance of the GP rules compared to a GP rule that uses equivalent time-variant terminals. For example, they replaced the due date with a time-invariant counterpart called "relative due date", which is the due date minus the time of the decision situation [96]. This means the range of values for relative due dates will not increase with the duration of the simulation. In the multitask approach, the duration of the simulation may significantly differ between simulations that have no machine breakdown and the simulations which have machine breakdown. This means that the time-variant due date works on a different scale to the other GP terminals for the different machine breakdown scenarios. This can then result in rules evolved on certain machine breakdown scenario on that use due dates performing poorly on other machine breakdown scenarios. On the other hand, since the rules are evolved separately for the benchmark GP approach, the time-variant terminals like due date are less likely to affect the effectiveness of the rules evolved for the no machine breakdown scenario.

5.7 Chapter Summary

The goal of this chapter is to develop multitask GP approaches that evolve an effective portfolio of rules over the DJSS problem with dynamic job arrivals and machine breakdowns. To achieve this goal, we developed two new multitask GP approaches. The first is called the niched GP (NGP) approach that incorporates the clearing procedure to discourage poor GP individuals with similar behaviours to the good individual for specific machine breakdown scenarios from being kept for the next generation. The

second is called the neighbourhood-based GP (NBGP) approach that defines a neighbourhood relation between the machine breakdown scenarios and evaluates individuals only on the relevant machine breakdown scenarios to improve the efficiency and the effectiveness of the output GP rules. This chapter also carried out an extensive analysis of the multitask GP approaches both in terms of the behaviour and the structure.

The NGP and the NBGP approaches show that multitask GP shows improved effectiveness over the benchmark GP for the DJSS problem. The multitask GP approaches also gave much more promising results than the method of directly incorporating the machine breakdown information directly into the GP's evolutionary process described in the previous chapter. GP has a difficult time exploring the search space to evolve rules that can effectively utilise the machine breakdown information. In addition, the analysis of specific rules during the evaluation procedure showed that machine breakdown scenario $\langle\ 0\%,\ 0,\ 5\ \rangle$ where the NGP and the NBGP specialist rules performed significantly worse than the benchmark specialist rules shows that a specific benchmark rule contributed heavily towards the overall effectiveness of the rule set. The specific rule has a very infrequent number of due date terminals, which is important in due date related DJSS problems. A hypothesis is that the terminals do not occur in the specialist rule as it is time-variant.

The analysis of the behaviours of the NGP and the NBGP rules show that there is a large difference between the rules evolved on zero machine breakdown scenarios to the rules evolved on machine breakdown scenarios. For example, the difference between the $\langle\ 0\%,\ 0,\ 5\ \rangle$ and the $\langle\ 2.5\%,\ 25,\ 5\ \rangle$ rules are greater than the difference between the $\langle\ 2.5\%,\ 25,\ 5\ \rangle$ and the $\langle\ 5\%,\ 25,\ 5\ \rangle$ rules. In addition, the analysis also shows that NBGP with a greater number of elite individuals results in fewer distances between the rule sets for the different machine breakdown scenarios. Combined with the worse performance of the NBGP Elite-10 rules compared to the NBGP Elite-1 rules, we hypotheses that having a greater number of elites

result in less exploration in the specific machine breakdown scenarios as less new GP individuals are being generated from crossover and mutation. This then results in less discovery of useful knowledge from the machine breakdown scenarios that can later be shared with other scenarios, resulting in worse performance overall.

Chapter 6

Conclusions

This thesis focuses on genetic programming based hyper-heuristic (GP-HH) to automatically evolve effective dispatching rules for dynamic job shop scheduling (DJSS) problems. The overall goal of this thesis was to carry out an extensive investigation of GP-HH approaches to improve the overall effectiveness of evolved dispatching rules for complex DJSS problems which have not been investigated in the literature. This goal was successfully achieved by developing new types of GP approaches that can evolve multiple dispatching rules simultaneously. The effective GP approaches developed are the ensemble GP approaches that evolve ensembles of dispatching rules, and the multitask GP approaches that evolve portfolio of rules to handle different subsets of a DJSS problem. The proposed GP approaches that evolve multiple rules were examined and compared with a standard GP representation on various DJSS problems with multiple dynamic events including dynamic job arrivals. GP-HH is state-of-the-art for the DJSS problems with a large number of dynamic job arrivals and generally produces the rules that outperform existing man-made dispatching rules [109]. The results show that the new GP approaches proposed in this thesis can clearly outperform the benchmark GP approaches, and show that new GP approaches better handle DJSS problems with multiple types of dynamic events than the standard

GP approaches.

The rest of this chapter provides the research goals that are achieved, followed by the main conclusions found from the major contributions made by this thesis. Afterwards, we provide a discussion of potential research areas that could be carried out in the future related to the research carried out in this thesis. Finally, we provide a discussion of more general future works that are not within the scope of the thesis but are important fields of research in DJSS.

6.1 Achieved Objectives

This thesis has achieved the following research objectives:

- This thesis proposes the first ensemble GP approach that can evolve effective ensembles of dispatching rules for the DJSS problem with dynamic job arrivals. No hyper-heuristic approach that evolves ensembles of rules has been proposed for a JSS problem. The first approach, "ensemble genetic programming for job shop scheduling" (EGP-JSS) [120], which incorporates Potter and DeJong's cooperative coevolution [127], showed that ensembles can generally outperform the single rules evolved by a standard GP approach. This thesis also showed that another major cooperative coevolutionary techniques such as multilevel genetic programming (MLGP) can also be incorporated with GP to produce an ensemble GP approach (called MLGP for JSS, or MLGP-JSS) which can outperform the standard GP approach for a DJSS problem. In addition, when an appropriate combination scheme is used with the ensemble GP approach, the ensemble GP approach's performance can be improved further on the DJSS problem. We investigated four combination schemes with the EGP-JSS approach: majority voting, linear combination, weighted majority voting and weighted linear combination. Out of the combina-

tion schemes investigated, the results show that the linear combination scheme produces the best GP rules when incorporated into the EGP-JSS approach. Finally, three diversity measures are proposed to analyse the behaviours of the rules evolved on the four combination schemes. The diversity measures provide useful insights into rules evolved by the EGP-JSS and can be used for future ensemble GP approaches to DJSS problems.

- This thesis provides the first investigation of the DJSS problem with both dynamic job arrival and machine breakdown using GP. Although GP has been applied to DJSS problems with dynamic job arrivals and to DJSS problems with machine breakdowns, it has not been applied to a DJSS problem that has both types of dynamic events simultaneously. The results showed that the standard GP approach that evolves a single rule is not general enough to cover for a DJSS problem over a range of machine breakdown scenarios. The investigation also carries out an evaluation of preliminary GP terminals that are specific to machine breakdown scenarios. The results show that a GP approach that evolves single dispatching rules have a difficult time incorporating the machine breakdown information from the machine breakdown GP terminals to evolve rules that can significantly outperform the standard GP approach. The analysis of the evolved rules in this investigation shows that processing time related terminals are more effective on scenarios with no machine breakdowns or with a low machine breakdown level than on scenarios with a high machine breakdown level. On the other hand, terminals that return the values that are not affected due to machine breakdowns (e.g. due date terminal) are more effective on scenarios with a high breakdown level than on scenarios with low or zero breakdown level.

- This thesis proposes the first multitask GP approach for the DJSS

problem with dynamic job arrivals and machine breakdowns. Multitask optimisation has been applied to other combinatorial optimisation problems, but have not yet been applied to DJSS. The two multitask GP approaches, niched GP (NGP) and neighbourhood-based GP (NBGP), evolve *portfolio* of rules specialised for the different machine breakdown scenarios in the DJSS problem. The NGP approach also evolves generalist rule over the entire DJSS problem. The rules evolved by NGP show significant improvement over the standard GP approach in terms of performance. On the other hand, the rules evolved by NBGP either are generally better than the standard GP approach or can be evolved at a significantly lower computation time. The multitask GP approaches indicate that using the behaviours of the rules specialised on different machine breakdown scenarios improves the generalisation ability of evolved rules. Finally, the analysis of the multitask GP approach show that there are greater differences in the behaviours between the rules evolved for the scenario with no machine breakdowns and the rules evolved for the scenarios with machine breakdowns than the differences between any two rule sets evolved on scenarios with machine breakdowns.

6.2 Main Conclusions

This thesis develops new GP approaches that evolve multiple dispatching rules simultaneously for the DJSS problems that improve upon existing GP approaches and are better able to cope with the unforeseeable events in the DJSS problems that can significantly affect the effectiveness of the evolved dispatching rules. This section discusses the main conclusions for the three research objectives drawn from the three contribution chapters (Chapter 3 to Chapter 5).

6.2.1 Investigation of Ensemble GP Approaches

Chapter 3 proposes new ensemble GP approaches that evolve ensemble of dispatching rules for the DJSS problem with dynamic job arrivals. The ensemble GP approaches incorporate cooperative coevolutionary algorithm from the literature. We also investigated different schemes for combining the outputs of the ensemble members together.

Incorporated Ensemble Algorithms

From the two cooperative coevolutionary algorithms investigated to evolve ensembles of dispatching rules, we found that incorporating Potter and De Jong's [127] to develop the EGP-JSS approach gave the best results and significantly outperformed the benchmark GP approach that evolves single priority dispatching rules.

An ensemble of dispatching rules evolved by the proposed ensemble GP approaches have significantly better performances than single dispatching rules evolved by the benchmark GP approach, indicating that complex decisions can be better handled by multiple rules than a single constituent rule [126]. Out of the two ensemble GP approaches developed that incorporate cooperative coevolutionary algorithms [127, 153], the EGP-JSS approach evolves more effective ensembles than the MLGP-JSS approach when both approaches use the majority voting combination scheme. Therefore, the combination schemes are incorporated into the EGP-JSS approach instead of the MLGP-JSS approach for their investigation.

Combination Schemes

The results showed found that the linear combination scheme gave the best performance out of the four combination schemes that are investigated and incorporated into the EGP-JSS approach. In addition, the results also showed that the proposed method of incorporating weighted combi-

nation schemes (weighted majority voting and weighted linear combination) did not work as effectively as the incorporated non-weighted combination schemes (majority voting and linear combination).

The analysis of the evolved ensembles in Section 3.8.3 (Page 110) using the three diversity measures showed that rules evolved by the linear combination scheme assign the highest priorities to different jobs more often than the other rules (based on analysis measure DC from Section 3.6.2 (Page 3.6.2)). This indicates that the linear combination scheme allows ensemble members to make more diverse decisions, but is also able to exploit the diversity in the decisions made by the ensemble members due to how the ensemble member outputs are converted to scores.

The analysis in Section 3.8.3 (Page 110) also showed that the rules evolved using the weighted majority voting showed that one ensemble member was dominant in the decision-making process based on the analysis measure HC in Section 3.6.3 (Page 103), i.e., the job that they assign the highest priority to have the highest chance of being selected by the ensemble as a whole. In conjunction with the result that the weighted majority voting ensembles had the worst performance overall, it is possible that weighted majority voting ensembles are behaving similarly to single dispatching rules. An ensemble that behaves too similar to a single dispatching rule will likely have ensemble members that are unable to cover for each other's errors, which will result in ensembles that perform no better than single dispatching rules.

Finally, the analysis in Section 3.8.3 (Page 110) shows that the linear combination ensembles and the weighted linear combination ensembles have high proportions of individuals that have assigned the highest priority to a job that is assigned a low rank by the ensemble as a whole (analysis measure LJR in Section 3.6.4 (Page 104)). This further supports the observation that the GP with linear combination can produce more diverse ensemble members because less information is lost when the priority values are converted to scores. Furthermore, it may also be the case that higher

LJRs can also be partially correlated to better performances, as the evolved linear combination rules have better performance than the evolved majority voting rules, and the weighted linear combination rules have better performance than the weighted majority voting rules.

6.2.2 Investigation of GP for a DJSS Problem subject to Machine Breakdown

Chapter 4 carried out an investigation of the effectiveness of GP approach into the DJSS problem with both dynamic job arrivals and machine breakdowns. Although effectively GP approaches have been proposed to DJSS problems with dynamic job arrivals or machine breakdowns independently, GP has not been applied to a DJSS problem with both types of dynamic events.

The Generality of GP for the DJSS Problem

The results in the chapter showed that rules evolved by the standard GP approach are not general enough to cover for the entire DJSS problem.

The performances of the rules are specific to the machine breakdown scenarios they are evolved on. For example, rules evolved on no machine breakdown scenario perform well on DJSS simulations, but perform poorly on simulations with high levels of machine breakdown, and vice versa. The trade-off means that applying GP to machine breakdown scenarios with high breakdown level does not result in rules that can cover for DJSS simulations with lower machine breakdown levels. This is contrary to the results in the literature where rules evolved on simulations with a specific utilisation rate can handle simulations with lower utilisation rate values [105], but not the other way around. Finally, the generalist rules are heavily biased towards the no machine breakdown scenarios and perform poorly on the scenarios with high breakdown level.

The analysis in Section 4.2.3 (Page 126) of the terminal distribution

shows that rules evolved on different machine breakdown scenarios have differences in the proportions of GP terminals that make up the rules. The rules evolved on higher machine breakdown levels have lower proportions of processing time terminal (PT) compared to the rules evolved on lower machine breakdown levels, and have a higher proportion of due date terminals (DD). This is likely due to the DJSS simulations with high machine breakdown level have a greater frequency of machine breakdowns than the simulations with low machine breakdown level (given that the mean repair times are the same). This results in more jobs being interrupted and may result in greater uncertainty associated with the job's actual processing times.

Machine Breakdown GP Terminals

Adding the machine breakdown GP terminals to the standard GP approach that evolves single priority dispatching rules showed slight improvements over the standard GP approach without the machine breakdown GP terminals. Although the differences in the overall performances of the evolved rules are not significant, the best rules evolved by the machine breakdown GP are significantly better than the best rule evolved by the benchmark GP on certain simulation configurations.

The analysis in Section 4.7.2 (Page 150) of the best rules evolved by the machine breakdown GP approach shows that the machine breakdown GP terminals occur very infrequently in the structures of the evolved rules. It may be possible that the machine breakdown GP terminals may be facilitating the evolution of good GP rules but appear infrequently in the final structures of the evolved rules. In addition, the behavioural analysis shows that the decisions that are made by the best rules in decision situations that are "close" to machine breakdowns share significant behavioural similarities to the SPT dispatching rule. The similarities to the SPT rule is the highest for decision situations in scenarios with low machine breakdown level and decreases as machine breakdown level increases.

This correlates with the GP terminal distributions from the efficacy investigation, as the proportion of PT terminal decreases as the breakdown level that the rules are evolved on increases.

6.2.3 Developing Multitask GP Approaches to DJSS Problem subject to Machine Breakdown

Chapter 5 proposes new multitask GP approaches that evolve *portfolios* of dispatching rules specialised for each machine breakdown scenario in a DJSS problem with dynamic job arrivals and machine breakdowns. The two new multitask GP approaches developed in the chapter are the NGP approach and the NBGP approach.

Evolving Rule Portfolios using Multitask GP

The results show that the proposed NGP approach generally performs better than the benchmark GP approach for the DJSS problem. In other words, except for the rules specialised for simulations with zero machine breakdowns, the specialist rule sets evolved by the NGP approach are generally better than the specialist rule sets evolved by the benchmark GP approach. In addition, the generalist rules evolved by the NGP approach outperforms the generalist rules evolved by the benchmark GP approach in all but one simulation configuration. This shows that the discovery of latent features that are shared to other "tasks", that occurs in multitask learning and optimisation, also improves the generalisation ability of GP evolved rules for the DJSS problem.

The results also show that the proposed NBGP approach can have a better computation time required to evolve the rules than the benchmark GP approach or generally have a better performance than the benchmark GP approach with comparable evolution times. The trade-off is based on the number of elite individuals in each machine breakdown scenarios in the current generation that are retained for the next generation of GP indi-

viduals. The NBGP specialist rules only perform significantly worse than the benchmark GP specialist rules on the simulation configurations that are under zero machine breakdown scenarios. The overall results from the NBGP also support the results from NGP, which shows that incorporating multitasking is an effective method of handling DJSS problems with dynamic job arrivals and machine breakdowns.

The simulation configurations that the NGP and the NBGP specialist rules performed worse than the benchmark GP specialist rules are configurations with no machine breakdowns. The analysis of the benchmark GP specialist rules to determine why they outperformed the NGP and the NBGP specialist rules showed that the best specialist rule evolved for the no machine breakdown scenario mostly consisted of time-invariant terminals, which have been shown to be more effective than time variant terminal counterparts [96]. Even though terminals such as due date (DD) are important in tardiness related objective, it is likely that the benchmark, the NGP and the NBGP approaches could not explore the heuristic space effectively during the evolution of the rules to evolve a rule that can effectively incorporate the time-variant DD terminal with the time-invariant terminals (e.g. PT terminal). This likely resulted in the DD terminal occurring very infrequently in the best specialist rule evolved by the benchmark GP approach.

Rule Portfolio Diversity

The analysis of the rules evolved by the NGP and the NBGP approach in Section 5.6.3 (Page 193) shows that the behaviours of the evolved rules differ significantly between the rules evolved on the scenario with no machine breakdowns than the rules evolved on scenarios with any level of machine breakdowns. In other words, the differences between the rules evolved on breakdown level 0% and the rules evolved on breakdown level 2.5% are greater than the differences between the rules evolved on breakdown level 2.5% and the rules evolved on breakdown level 5.0%. As the

difference in the behaviours of the evolved rules is different from the incremental difference in the machine breakdown level, the rules that are effective on DJSS problem with dynamic job arrivals are likely to be very different from the rules that are effective on DJSS problem with dynamic job arrivals and machine breakdowns.

The behavioural analysis in Section 5.6.3 (Page 193) also shows that the difference in the NBGP rules evolved on breakdown level 0% to the other rules decreases as the number of elites increases. This could potentially be due to the lack of exploration being carried out. As more elites are retained from the previous generation, less new individuals are generated by the NBGP from crossover and mutation. This then may result in less discovery of useful knowledge from the specific machine breakdown scenarios, resulting in more similarity in the behaviours of the rules. In addition, the reduced discovery of useful knowledge is supported by NBGP with 10 elite individuals retained having worse overall performance than the NBGP with one elite individual retained, as the discovered knowledge from one machine breakdown scenario could be used to boost the effectiveness of GP individuals in other machine breakdown scenarios.

6.3 Discussions

This section highlights key areas of future work related to the research that has been carried out in this thesis.

6.3.1 Evolving Ensembles on DJSS Problems Subject to Machine Breakdowns

This thesis develops ensemble GP approaches that can evolve effective ensembles of dispatching rules that can outperform the standard GP approach over the DJSS problem. Other effective ensemble GP approaches have also been proposed by researchers [42] after EGP-JSS, which shows

that ensemble learning is a promising field that warrants further investigation in the future. In particular, ensemble GP approach that can evolve ensembles from a limited number of simulation runs would be promising, as DJSS problems handled in this thesis have a large number of job arrivals, which results in each simulation run being very expensive in terms of the computation time. Furthermore, this thesis does not apply ensemble GP approach to DJSS problems with both dynamic job arrivals and machine breakdowns. It is likely that an ensemble GP approach can perform better than the standard GP approach for the DJSS problem, as it may be able to better cope with the added uncertainty introduced by machine breakdown.

6.3.2 Incorporating Diversity to Ensemble GP Approaches to DJSS Problems

Diversity between the ensemble members is an important component for the effectiveness of an ensemble. The results showed that ensembles evolved with the linear combination schemes show greater diversity between the ensemble members compared to the other combination schemes investigated in Chapter 3, which also is correlated with better performances by the evolved rules. However, although we carried out some preliminary investigation into incorporating some diversity measures into the ensemble GP approach, they did significantly improve the effectiveness of the evolved rules. Further investigation into directly incorporating an effective diversity measure may be promising.

6.3.3 Incorporating Machine Breakdown Information into GP

Although this thesis investigated using machine breakdown GP terminals, the results show that the machine breakdown GP approach does not

significantly improve over the standard GP approach overall. In addition, the multitask GP approaches proposed in this thesis do not evolve rules that directly incorporate machine breakdown information into the decision-making process. Although GP has difficulties evolving rules that can incorporate the machine breakdown terminals effectively, there may be other feasible methods of incorporating machine breakdown information that results in more effective evolved rules. For example, a GP approach that evolves a rule for regular decision situations and a rule specifically for decision situations that occur shortly before machine breakdowns may be more effective than the standard GP approach for the DJSS problem.

6.3.4 Identifying Machine Breakdown Scenarios Automatically

The thesis proposes an effective multitask GP approach for the DJSS problem with dynamic job arrivals and machine breakdowns that can outperform the standard GP approach. However, the multitask GP approaches require domain knowledge about the DJSS problem to evolve the rules. The NGP approach uses niched training sets that only consist of simulations that belong to machine breakdown scenarios to further evaluate the niched GP individuals, and the NBGP approach uses neighbourhood relation between the different machine breakdown scenarios. In addition, the evolved specialist rules are evaluated on the specific machine breakdown scenarios. In a real-world scenario, it is likely that the machine breakdown scenarios are not known in advance for both the training and the testing procedure. In addition, the differences between the different machine breakdown scenarios may not be as distinct as the machine breakdown scenarios investigated in this thesis. Therefore, a multitask GP approach that can distinguish between the different machine breakdown scenarios without directly incorporating the domain knowledge into the training

process may be required. In addition, a mechanism that automatically determines the appropriate specialist rule to be used for the decision situations for a specific DJSS problem instance that belongs to an unknown machine breakdown scenario would be promising.

6.4 Future Work

This section provides a brief discussion of general research areas that are outside the scope of this thesis but are relevant to the topics covered in this thesis and are important fields of research.

6.4.1 Surrogate Modelling

Hildebrandt and Branke [59] and Nguyen et al. [108] have proposed surrogate GP approach to DJSS to improve the training convergence of GP and to improve the efficiency of evolving rules. Surrogate modelling is an important field of research to DJSS [102], as it alleviates the expensive computation cost required to run a DJSS simulation by carrying out a partial evaluation of the GP population. Surrogate GP approaches proposed in the literature share a similar goal with the proposed neighbourhood based genetic programming (NBGP) approach in this thesis (Page 172). In both surrogate GP and the NBGP, the goal is to improve the efficiency of evolving rules over the standard GP approach while maintaining competitive performance. Therefore, a multitask GP approach that can incorporate concepts from surrogate modelling may be able to further improve the efficiency of evolving rules, and vice versa.

6.4.2 Transfer Learning and Concept Drift

Transfer learning is a prominent field of research in the literature [116] outside of JSS, but have not been significantly investigated for GP-HH in

DJSS problems. In particular, concept drift [116] may be a promising direction of research to DJSS with dynamic job arrivals. In a real-world scenario, it is possible that the properties of the manufacturing environment change over time, i.e., there is a "drift" in the properties of the problem. In a real-world scenario, the properties of the manufacturing environments can change over time. For example, a change of season from summer to winter may mean that certain machines on the shop floor take longer to heat up, i.e., take longer to process operations. A GP evolved rule needs to be able to cope with the drift effectively or need to be able to detect quickly when a drift occurs. An approach that can effectively detect when a drift occurs may allow the manufacturer to make an informed decision on whether to evolve a new rule if the old rule can no longer maintain competitive performance.

6.4.3 Multi-objective Optimisation and Multiple Types of Dynamic Events

Multi-objective optimisation is an important field of research to DJSS problems and has been investigated extensively in the literature [91, 107, 143]. Real-world manufacturing environments often have multiple conflicting objectives that need to be optimised simultaneously [154]. However, many multi-objective DJSS problems handled by the GP-HH approaches in the literature often have only one type of dynamic event that occurs during processing, e.g., dynamic job arrivals [91, 107]. Therefore, research into multi-objective DJSS problems with multiple types of dynamic events (e.g. dynamic job arrivals and machine breakdowns) is a potentially important research direction that will require new and specialised GP approaches to handle.

Bibliography

[1] ABDUL-RAZAQ, T. S., POTTS, C. N., AND VAN WASSENHOVE, L. N. A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics 26*, 2–3 (1990), 235–253.

[2] ABUMAIZAR, R. J., AND SVESTKA, J. A. Rescheduling job shops under random disruptions. *International Journal of Production Research 35*, 7 (1997), 2065–2082.

[3] ADAMS, J., BALAS, E., AND ZAWACK, D. The shifting bottleneck procedure for job shop scheduling. *Management Science 34*, 3 (1988), 391–401.

[4] ADIBI, M. A., AND SHAHRABI, J. A clustering-based modified variable neighborhood search algorithm for a dynamic job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology 70*, 9-12 (2014), 1955–1961.

[5] ADIBI, M. A., ZANDIEH, M., AND AMIRI, M. Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems with Applications 37*, 1 (2010), 282–287.

[6] AHMADI, E., ZANDIEH, M., FARROKH, M., AND EMAMI, S. M. A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. *Computers & Operations Research 73* (2016), 56–66.

[7] AITZAI, A., BENMEDJDOUB, B., AND BOUDHAR, M. Branch-and-bound and PSO algorithms for no-wait job shop scheduling. http://dx.doi.org/10.1007/s10845-014-0906-7, 2014.

[8] AL-HINAI, N., AND ELMEKKAWY, T. Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. *International Journal of Production Economics 132*, 2 (2011), 279–291.

[9] ALPAYDIN, E. *Introduction to Machine Learning*, 2 ed. MIT press, 2010.

[10] APPLEGATE, D., AND COOK, W. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing 3*, 2 (1991), 149–156.

[11] ARGYRIOU, A., EVGENIOU, T., AND PONTIL, M. Multi-task feature learning. In *Proceedings of the 19th International Conference on Neural Information Processing Systems (NIPS 2006)* (Cambridge, MA, USA, 2006), MIT Press, pp. 41–48.

[12] ARMENTANO, V. A., AND SCRICH, C. R. Tabu search for minimizing total tardiness in a job shop. *International Journal of Production Economics 63*, 2 (2000), 131–140.

[13] BÄCK, T. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[14] BAKER, K. R., AND BERTRAND, J. A dynamic priority rule for scheduling against due-dates. *Journal of Operations Management 3*, 1 (1982), 37–42.

[15] BHOWAN, U., JOHNSTON, M., ZHANG, M., AND YAO, X. Evolving diverse ensembles using genetic programming for classification

with unbalanced data. *IEEE Transactions on Evolutionary Computation 17*, 3 (2013), 368–386.

[16] BLAZEWICZ, J., DOMSCHKE, W., AND PESCH, E. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research 93*, 1 (1996), 1–33.

[17] BRANKE, J., NGUYEN, S., PICKARDT, C. W., AND ZHANG, M. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation 20*, 1 (2016), 110–124.

[18] BRANKE, J., AND PICKARDT, C. W. Evolutionary search for difficult problem instances to support the design of job shop dispatching rules. *European Journal of Operational Research 212*, 1 (2011), 22–32.

[19] BREIMAN, L. Bagging predictors. *Machine Learning 24*, 2 (1996), 123–140.

[20] BRUCKER, P., JURISCH, B., AND SIEVERS, B. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics 49*, 1–3 (1994), 107–127.

[21] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., ÕZCAN, E., AND QU, R. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[22] BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÕZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*, Lecture Notes in Computer Science. Springer, 2010, pp. 449–468.

[23] BURKE, E. K., HYDE, M., KENDALL, G., AND 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.

[24] CARLIER, J. The one-machine sequencing problem. *European Journal of Operational Research 11*, 1 (1982), 42–47.

[25] CARLIER, J., AND PINSON, E. An algorithm for solving the job-shop problem. *Management Science 35*, 2 (1989), 164–176.

[26] CARROLL, D. *Heuristic sequencing of jobs with single and multiple components*. PhD thesis, Massachusetts Institute of Technology, 1965.

[27] CARUANA, R. Multitask learning. *Machine Learning 28*, 1 (Jul 1997), 41–75.

[28] CHEN, H., AND YAO, X. Multiobjective neural network ensembles based on regularized negative correlation learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 12 (2010), 1738–1751.

[29] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering 36*, 2 (1999), 343–364.

[30] CHENG, T. C. E., AND GUPTA, M. C. Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research 38*, 2 (1989), 156–166.

[31] CHONG, C. S., SIVAKUMAR, A. I., LOW, M. Y. H., AND GAY, K. L. A bee colony optimization algorithm to job shop scheduling. In *WSC '06: Proceedings of the 38th Winter Simulation Conference* (2006), pp. 1954–1961.

[32] COLORNI, A., DORIGO, M., MANIEZZO, V., AND TRUBIAN, M. Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science 34*, 1 (1994), 39–53.

[33] CONWAY, R. W., MAXWELL, W. L., AND MILLER, L. W. *Theory of Scheduling*. Addison-Wesley, 1967.

[34] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III*, vol. 2079 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2001, pp. 176–190.

[35] DA, B., ONG, Y., FENG, L., QIN, A. K., GUPTA, A., ZHU, Z., TING, C., TANG, K., AND YAO, X. Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results. *CoRR* (2017).

[36] DE BONTRIDDER, K. Minimizing total weighted tardiness in a generalized job shop. *Journal of Scheduling 8*, 6 (2005), 479–496.

[37] DE JONG, K. A. *Evolutionary Computation: A Unified Approach*. MIT press, 2006.

[38] DEB, K., AND JAIN, H. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation 18*, 4 (2014), 577–601.

[39] DEB, K., PRATAP, A., AGARWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*, 2 (2002), 182–197.

[40] DIMOPOULOS, C., AND 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.

[41] DORIGO, M., MANIEZZO, V., AND COLORNI, A. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics 26*, 1 (1996), 29–41.

[42] DURASEVIĆ, M., AND JAKOBOVIĆ, D. Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. *Genetic Programming and Evolvable Machines 19*, 1 (2018), 53–92.

[43] DURASEVIĆ, M., JAKOBOVIĆ, D., AND KNEŽEVIĆ, K. Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing 48* (2016), 419–430.

[44] EVGENIOU, T., AND PONTIL, M. Regularized multi-task learning. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004)* (New York, NY, USA, 2004), ACM, pp. 109–117.

[45] FREUND, Y., AND SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, vol. 904 of *Lecture Notes in Computer Science*. 1995, pp. 23–37.

[46] GAREY, M. R., JOHNSON, D. S., AND SETHI, R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*, 2 (1976), 117–129.

[47] GEIGER, C. D., UZSOY, R., AND AYTU, H. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling 9*, 1 (2006), 7–34.

[48] GIFFLER, B., AND THOMPSON, G. L. Algorithms for solving production-scheduling problems. *Operations Research 8*, 4 (1960), 487–503.

[49] GLOVER, F., AND LAGUNA, M. *Tabu Search*. Springer, 1999.

[50] GROMICHO, J. A., VAN HOORN, J. J., DA GAMA, F. S., AND TIMMER, G. T. Solving the job-shop scheduling problem optimally

by dynamic programming. *Computers & Operations Research 39*, 12 (2012), 2968–2977.

[51] GUPTA, A., ONG, Y. S., AND FENG, L. Multifactorial evolution: Toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation 20*, 3 (2016), 343–357.

[52] GUPTA, A., ONG, Y. S., AND FENG, L. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence 2*, 1 (2018), 51–64.

[53] GUPTA, A., ONG, Y. S., FENG, L., AND TAN, K. C. Multiobjective multifactorial optimization in evolutionary multitasking. *IEEE Transactions on Cybernetics 47*, 7 (2017), 1652–1665.

[54] HANSEN, P., MLADENOVIĆ, N., AND PÉREZ, J. A. M. Variable neighbourhood search: methods and applications. *Annals of Operations Research 175*, 1 (2010), 367–407.

[55] HART, E., ROSS, P., AND CORNE, D. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines 6*, 2 (2005), 191–220.

[56] HART, E., AND SIM, K. A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation 24*, 4 (2016), 609–635.

[57] HEARST, M. A., DUMAIS, S. T., OSUNA, E., PLATT, J., AND SCHOLKOPF, B. Support vector machines. *IEEE Intelligent Systems and their Applications 13*, 4 (1998), 18–28.

[58] HELD, M., AND KARP, R. M. A dynamic programming approach to sequencing problems. In *Proceedings of the 16th ACM National Meeting (ACM 1961)* (1961), pp. 71.201–71.204.

[59] HILDEBRANDT, T., AND BRANKE, J. On using surrogates with genetic programming. *Evolutionary Computation 23*, 3 (2015), 343–367.

[60] HILDEBRANDT, T., HEGER, J., AND 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, July 2010), ACM, pp. 257–264.

[61] HOLTHAUS, O. Scheduling in job shops with machine breakdowns: an experimental study. *Computers & Industrial Engineering 36*, 1 (1999), 137–162.

[62] HOLTHAUS, O., AND RAJENDRAN, C. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics 48*, 1 (1997), 87–105.

[63] HOLTHAUS, O., AND RAJENDRAN, C. Efficient jobshop dispatching rules: Further developments. *Production Planning & Control 11*, 2 (2000), 171–178.

[64] HORST, R., AND ROMEIJN, H. E. *Handbook of Global Optimization*, vol. 2. Springer Science & Business Media, 2002.

[65] HUANG, V. L., SUGANTHAN, P. N., QIN, A. K., AND BASKAR, S. Multiobjective differential evolution with external archive and harmonic distance-based diversity measure. Tech. rep., Nanyang Technological University, 2005.

[66] HUNT, R., JOHNSTON, M., AND 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, July 2014), ACM, pp. 927–934.

[67] HUNT, R., JOHNSTON, M., AND 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)* (July 2014), pp. 618–625.

[68] HUNT, R., JOHNSTON, M., AND ZHANG, M. Using local search to evaluate dispatching rules in dynamic job shop scheduling. In *Proceedings of Evolutionary Computation in Combinatorial Optimization (EvoCOP 2015)* (Cham, April 2015), vol. 9026 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 222–233.

[69] JACKSON, J. An extension of Johnson's result on job-lot scheduling. *Naval Research Logistics Quarterly 3*, 3 (1956), 201–204.

[70] JAKOBOVIĆ, D., AND BUDIN, L. Dynamic scheduling with genetic programming. In *EuroGP '06: Proceedings of the 9th European Conference on Genetic Programming* (2006), vol. 3905 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 73–84.

[71] JAKOBOVIĆ, D., JELENKOVIĆ, L., AND BUDIN, L. Genetic programming heuristics for multiple machine scheduling. In *EuroGP '07: Proceedings of the 10th European Conference on Genetic Programming* (2007), vol. 4445 of *Lecture Notes in Computer Science*, Springer Berlin Heidelburg, pp. 321–330.

[72] JAYAMOHAN, M. S., AND RAJENDRAN, C. New dispatching rules for shop scheduling: A step forward. *International Journal of Production Research 38*, 3 (2000), 563–586.

[73] JAYAMOHAN, M. S., AND RAJENDRAN, C. Development and analysis of cost-based dispatching rules for job shop scheduling. *European Journal of Operational Research 157*, 2 (2004), 307–321.

[74] JOHNSON, S. M. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics 3* (1954), 61–68.

[75] JONES, A., AND RABELO, L. C. Survey of job shop scheduling techniques. *Wiley Encyclopedia of Electrical and Electronics Engineering* (1999), 1–14.

[76] KARABOGA, D. An idea based on honey bee swarm for numerical optimization. Tech. rep., Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[77] KENNEDY, J., AND EBERHART, R. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks* (1995), vol. 4, pp. 1942–1948.

[78] KENNEDY, J. F., EBERHART, R. C., AND SHI, Y. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[79] KIRKPATRICK, S., GELATT, C. D., AND VECCHI, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[80] KOZA, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[81] KRAMER, O. *Genetic Algorithm Essentials*, vol. 679. Springer, 2017.

[82] KREIPL, S. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling 3*, 3 (2000), 125–138.

[83] LAND, A. H., AND DOIG, A. G. An automatic method for solving discrete programming problems. *Econometrica* (1960), 497–520.

[84] LARRAÑAGA, P. *A Review on Estimation Distribution Algorithms*. Springer, 2002.

[85] LAWLER, E. L., LENSTRA, J. K., AND RINNOOY KAN, A. H. G. Recent developments in deterministic sequencing and scheduling: A survey. In *Deterministic and Stochastic Scheduling*, vol. 84 of *NATO Advanced Study Institutes Series*. Springer Netherlands, 1982, pp. 35–73.

[86] LAWLER, E. L., LENSTRA, J. K., RINNOOY KAN, A. H. G., AND SHMOYS, D. B. Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, vol. 4 of *Handbooks in Operations Research and Management Science*. Elsevier, 1993, pp. 445–522.

[87] LAWLER, E. L., AND MOORE, J. M. A functional equation and its application to resource allocation and sequencing problems. *Management Science 16*, 1 (1969), 77–84.

[88] LIN, S.-C., GOODMAN, E. D., AND PUNCH, W. F. Investigating parallel genetic algorithms on job shop scheduling problems. In *International Conference on Evolutionary Programming* (1997), Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 383–393.

[89] LIU, Y., AND YAO, X. Ensemble learning via negative correlation. *Neural Networks 12*, 10 (1999), 1399–1404.

[90] MASOOD, A., MEI, Y., CHEN, G., AND ZHANG, M. Many-objective genetic programming for job-shop scheduling. In *Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC 2016)* (July 2016), pp. 209–216.

[91] MASOOD, A., MEI, Y., CHEN, G., AND ZHANG, M. A pso-based reference point adaption method for genetic programming hyper-heuristic in many-objective job shop scheduling. In *Proceedings of the 2017 Australasian Conference on Artificial Life and Computational Intel-*

ligence (2017), Lecture Notes in Artificial Intelligence, Springer International Publishing, pp. 326–338.

[92] MCKAY, K. N., SAFAYENI, F. R., AND BUZACOTT, J. A. Job-shop scheduling theory: What is relevant? *Interfaces 18*, 4 (August 1988), 84–90.

[93] MEERAN, S., AND MORSHED, M. S. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of Intelligent Manufacturing 23*, 4 (2012), 1063–1078.

[94] MEHTA, S. V., AND UZSOY, R. M. Predictable scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation 14*, 3 (1998), 365–378.

[95] MEI, Y., NGUYEN, S., XUE, B., AND 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.

[96] MEI, Y., NGUYEN, S., AND ZHANG, M. Evolving time-invariant dispatching rules in job shop scheduling with genetic programming". proceedings of the 20th european conference on genetic programming (eurogp 2017). Lecture Notes in Computer Science, pp. 147–163.

[97] MEI, Y., ZHANG, M., AND 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), pp. 365–372.

[98] MICHALEWICZ, Z., AND FOGEL, D. B. *How to Solve It: Modern Heuristics*, 2 ed. Springer Science & Business Media, 2013.

[99] MORTON, T. E., AND RACHAMADUGU, R. M. V. Myopic heuristics for the single machine weighted tardiness problem. Tech. rep., Carnegie-Mellon University, 1982.

[100] MUTH, J. F., AND THOMPSON, G. L. *Industrial Scheduling*. Prentice-Hall, 1963.

[101] NGUYEN, Q. U., NGUYEN, X. H., O'NEILL, M., AND AGAPITOS, A. An investigation of fitness sharing with semantic and syntactic distance metrics. In *Genetic Programming*, Lecture Notes in Computer Science. 2012, pp. 109–120.

[102] NGUYEN, S., MEI, Y., MA, H., CHEN, A., AND ZHANG, M. Evolutionary scheduling and combinatorial optimisation: Applications, challenges, and future directions. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2016)* (July 2016), pp. 3053–3060.

[103] NGUYEN, S., MEI, Y., AND ZHANG, M. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems 3*, 1 (2017), 41–66.

[104] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND 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)* (June 2012), pp. 1–8.

[105] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND 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.

[106] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Learning iterative dispatching rules for job shop scheduling with genetic

programming. *The International Journal of Advanced Manufacturing Technology 67*, 1-4 (2013), 85–100.

[107] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation 18*, 2 (2014), 193–208.

[108] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In *Proceedings of the 10th International Conference on Simulated Evolution and Learning* (2014), vol. 8886 of *Lecture Notes in Computer Science*, Springer, pp. 656–667.

[109] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE Transactions on Cybernetics 45*, 1 (2015), 1–14.

[110] NGUYEN, S., ZHANG, M., AND TAN, K. C. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics 47*, 9 (2017), 2951–2965.

[111] NIE, L., BAI, Y., WANG, X., AND LIU, K. Discover scheduling strategies with gene expression programming for dynamic flexible job shop scheduling problem. In *Proceedings of the Third International Conference on Advances in Swarm Intelligence, Part II (ICSI 2012)* (Berlin, Heidelberg, June 2012), Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 383–390.

[112] NIE, L., GAO, L., LI, P., AND LI, X. A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing 24*, 4 (2013), 763–774.

[113] NIE, L., SHAO, X., GAO, L., AND LI, W. Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. *The International Journal of Advanced Manufacturing Technology 50*, 5–8 (2010), 729–747.

[114] OSMAN, I. H., AND KELLY, J. P. *Meta-heuristics: An Overview*. Springer, 1996.

[115] OUELHADJ, D., AND PETROVIC, S. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling 12*, 4 (August 2009), 417–431.

[116] PAN, S. J., AND YANG, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359.

[117] PANAIT, L., AND LUKE, S. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems 11*, 3 (2005), 387–434.

[118] PANWALKAR, S. S., AND ISKANDER, W. A survey of scheduling rules. *Operations Research 25*, 1 (1977), 45–61.

[119] PARK, J., NGUYEN, S., JOHNSTON, M., AND ZHANG, M. Evolving stochastic dispatching rules for order acceptance and scheduling via genetic programming. In *Proceedings of 26th Australasian Joint Conference on Artificial Intelligence (AI 2013)* (2013), Lecture Notes in Computer Science, pp. 478–489.

[120] PARK, J., NGUYEN, S., ZHANG, M., AND 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)* (April 2015), vol. 9025 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 92–104.

[121] PÉREZ, E., HERRERA, F., AND HERNNDEZ, C. Finding multiple solutions in job shop scheduling by niching genetic algorithms. *Journal of Intelligent Manufacturing 14*, 3-4 (2003), 323–339.

[122] PÉREZ, E., POSADA, M., AND HERRERA, F. Analysis of new niching genetic algorithms for finding multiple solutions in the job shop scheduling. *Journal of Intelligent Manufacturing 23*, 3 (2012), 341–356.

[123] PETROWSKI, A. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation* (1996), pp. 798–803.

[124] PICKARDT, C. W., HILDEBRANDT, T., BRANKE, J., HEGER, J., AND SCHOLZ-REITER, B. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics 145*, 1 (2013), 67–77.

[125] PINEDO, M. L. *Scheduling: Theory, Algorithms, and Systems*, 4 ed. SpringerUS, 2012.

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

[127] POTTER, M. A., AND DE JONG, K. A. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation 8*, 1 (2000), 1–29.

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

[129] RAGHU, T., AND RAJENDRAN, C. An efficient dynamic dispatching rule for scheduling in a job shop. *International Journal of Production Economics 32*, 3 (1993), 301–313.

[130] RAMASESH, R. Dynamic job shop scheduling: A survey of simulation research. *International Journal of Management Science 18*, 1 (1990), 43–57.

[131] RUBINI, J., HECKENDORN, R. B., AND SOULE, T. Evolution of team composition in multi-agent systems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation* (2009), ACM, pp. 1067–1074.

[132] RUSSELL, S. J., AND NORVIG, P. *Artificial Intelligence: A Modern Approach*, 2 ed. Pearson Education, 2003.

[133] SARENI, B., AND KRAHENBUHL, L. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation 2*, 3 (1998), 97–106.

[134] SCHAPIRE, R. E. The strength of weak learnability. *Machine Learning 5*, 2 (Jun 1990), 197–227.

[135] SHA, D., AND HSU, C.-Y. A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering 51*, 4 (2006), 791–808.

[136] SOULE, T., AND KOMIREDDY, P. Orthogonal evolution of teams: A class of algorithms for evolving teams with inversely correlated errors. In *Genetic Programming Theory and Practice IV*, vol. 5 of *Genetic and Evolutionary Computation*. Springer, 2007, pp. 79–95.

[137] SPRECHER, A., KOLISCH, R., AND DREXL, A. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research 80*, 1 (1995), 94–102.

[138] SUBRAMANIAM, V., LEE, G. K., RAMESH, T., HONG, G. S., AND WONG, Y. S. Machine selection rules in a dynamic job shop. *The In-*

ternational Journal of Advanced Manufacturing Technology 16, 12 (2000), 902–908.

[139] SUDDARTH, S. C., AND KERGOSIEN, Y. Rule-injection hints as a means of improving network performance and learning time. In *Neural Networks*. Springer, 1990, pp. 120–129.

[140] SURESH, V., AND CHAUDHURI, D. Dynamic scheduling – a survey of research. *International Journal of Production Economics 32*, 1 (1993), 53–63.

[141] TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research 64*, 2 (1993), 278–285.

[142] TAILLARD, E. Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing 6*, 2 (1994), 108–117.

[143] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering 54*, 3 (2008), 453–473.

[144] THE WORLD BANK. Manufacturing, value added (current US$). `https://data.worldbank.org/indicator/NV.IND.MANF.CD?locations=US&year_high_desc=true`. Accessed: 25 June 2015. Available.

[145] TOMASSINI, M. *Spatially structured evolutionary algorithms: Artificial evolution in space and time*. Springer, 2006.

[146] VALLADA, E., AND RUIZ, R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega 38*, 1–2 (2010), 57–67.

[147] VAN LAARHOVEN, P. J. M., AARTS, E. H. L., AND LENSTRA, J. K. Job shop scheduling by simulated annealing. *Operations Research 40*, 1 (1992), 113–125.

[148] VEPSALAINEN, A. P. J., AND MORTON, T. E. Priority rules for job shops with weighted tardiness costs. *Management Science 33*, 8 (1987), 1035–1047.

[149] WIDMER, G., AND KUBAT, M. Learning in the presence of concept drift and hidden contexts. *Machine Learning 23*, 1 (1996), 69–101.

[150] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82.

[151] WONG, T. C., AND NGAN, S. C. A comparison of hybrid genetic algorithm and hybrid particle swarm optimization to minimize makespan for assembly job shop. *Applied Soft Computing 13*, 3 (2013), 1391–1399.

[152] WU, S., STORER, R. H., AND PEI-CHANN, C. One-machine rescheduling heuristics with efficiency and stability as criteria. *Computers & Operations Research 20*, 1 (1993), 1–14.

[153] WU, S. X., AND BANZHAF, W. Rethinking multilevel selection in genetic programming. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation* (2011), pp. 1403–1410.

[154] YENISEY, M. M., AND YAGMAHAN, B. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega 45* (2014), 119–135.

[155] YIN, W. J., LIU, M., AND 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), pp. 1050–1055.

[156] YSKA, D., MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In *Proceedings of the 21st European Conference on Genetic Programming (EuroGP 2018)* (April 2018), Lecture Notes in Computer Science, Springer International Publishing, pp. 306–321.

[157] ZHANG, G. Neural networks for classification: a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 30*, 4 (2000), 451–462.

[158] ZHOU, H., CHEUNG, W., AND LEUNG, L. C. Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research 194*, 3 (2009), 637–649.

[159] ZITZLER, E., LAUMANNS, M., AND 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)* (September 2001), pp. 1–21.