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
2018

Abstract

Job shop scheduling (JSS) problems are a group of optimisation problems which involves processing utilising a limited number of machines to process jobs as effectively as possible, where a machine can process one job at a time. Many manufacturing systems in real-world can be modelled as JSS problems. Approaches to JSS ranges from computationally intensive exact optimisation techniques, to *dispatching rule* heuristics that can generate good but not necessarily the best solutions to JSS problem instances. Hyper-heuristics are high level heuristics which can automatically generate a smaller heuristic to a problem, and recent studies have applied hyper-heuristics to JSS problems. However, many of the hyper-heuristic approaches from the literature are limited in that they 1) only generate a single heuristic that are myopic in nature, 2) require a long computation time to generate a heuristic, 3) have mainly been applied to problems with no machine breakdowns, and 4) have no way of automatically updating a generated rule. The overall goal of this thesis is to develop a genetic programming based hyper-heuristic approach that aims to address the four limitations of existing hyper-heuristic approaches in the literature. This approach will combine novel dispatching rule representations with evolutionary computation mechanisms to meet all of these requirements.

Acknowledgments

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 have 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 like to thank my parents for their constant support and encouragment 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.

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 [38, 102]. The problems have garnered attention from both academics and leading industry experts. From an academic perspective, JSS problems are complex and difficult problems. Most JSS problems with fixed properties are NP-hard [10]. 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* [99]. This results in an issue where large problem instances are too computationally intensive to generate optimal solutions using a mathematical optimisation technique. JSS problems are considered to be among the most difficult combinatorial optimisation problems [36]. On the other hand, JSS is a good model of many manufacturing scenarios ranging from semiconductor manufacturing to automobile assembly lines [99] from a manufacturing and production perspective. In the United States alone, manufacturers contribute trillions of dollars to the US economy [114].

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* [99]. A job has a predetermined sequence of *operations* which need to be completed 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 [56, 57]), or to minimise the time when the last job in a batch is completed (i.e. minimisation the makespan [6, 112]). 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 [7, 56, 57, 115].

JSS problems are divided into two separate subsets of problems. The first subset of JSS problems are the *static* JSS problems [99]. 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, 14, 20, 38, 99]. 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 or higher than the makespan of that optimal solution. However, for many JSS problems, the search spaces for the problem instances do not scale well to the size of the problem instances. Therefore, mathematical optimisation techniques have been limited to very small problem instances. In one study in 2014, it took approximately 7 hours to find an optimal solution to a problem instance with 9 jobs and 5 machines [4]. For larger JSS problems, such as problem instances with up to 100 jobs and 20 machines [112], it is too computationally difficult to find the optimal solutions to the problem instances. In many real-world scenarios, although optimal solutions are preferred, they are often not necessary, with solutions that are "good enough" being sufficient to allow the manufacturer to have a competitive edge in the market. Because of this, it is sometimes more advantageous to tradeoff 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 [50]. 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 smaller heuristics [89]. Examples of meta-heuristics approaches to JSS include techniques such as Tabu Search [7] and Genetic Algorithm [125]. Specific meta-heuristic approaches [62] have been shown to handle large JSS problem instances very effectively.

The second subset of JSS problems are dynamic JSS problems [99]. Unlike static JSS problem instances, where the properties of jobs and machines are known *a priori*, unforeseen events occur in dynamic JSS problem instances which affect the properties of the problem instance. In many dynamic JSS problems, the job and the job's attributes are unknown before they arrive on the shop floor. Other dynamic JSS problems can focus on unforeseen machine breakdowns [90], and changes in the job properties [90]. Developing approaches for dynamic JSS problems are a way to address the issues bridging the gap between the theoretical approaches to JSS and the industrial applications of the various approaches [83]. In a real-world scenario, dynamic changes to the shop floor are unavoidable, and the rules that are applied in an industrial setting needs to be robust to changes in the environment [84].

Approaches to solving dynamic JSS problems differ from static JSS problems. Because dynamic JSS 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 do not work for dynamic JSS problem instances [111]. In addition, if the dynamic JSS problem also contains stochastic elements, then planning for future events becomes even more difficult. Instead, many effective approaches to dynamic JSS problems use small heuristics, such as dispatching rules [11]. This is because dispatching rules have short reaction times and can cope with the unforeseen changes in the manufacturing environment [86]. 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 [99] selects the job with the shortest processing time waiting at the available machine. Other approaches use a dispatching rule which combines multiple job 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 (CDRs) [56, 57]. 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 [125], they are very effective for dynamic environments [86].

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 [11], and no single heuristic is more effective than other heuristics for all JSS problems [119]. 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, change in the objectives of the problems, 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* [15]. Hyper-heuristics are heuristics which generate smaller heuristics instead of outputting solutions to the problem instances. The heuristics generated are often dispatching rules [11], 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) [67], and are denoted as Genetic Programming based hyper-heuristic (GP-HH) approaches [11]. 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-based dispatching rules. From the literature, GP-HH approaches generally outperform manually designed dispatching rules for both static and dynamic JSS problems [11, 83], making it a very promising approach to investigate.

The overall goal of this thesis is to develop new and effective **GP-HH** approaches to **automatically** design high quality dispatching rule heuristics for **dynamic job shop scheduling environments** that aims to improve over the standard GP approach in a number of different dynamic JSS problems 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 dynamic JSS problems by exploiting the problem properties.

1.2 Motivations

Dispatching rules are relatively simple heuristics in comparison to some meta-heuristics. However, dispatching rules can range widely in complexity of the job selection procedure [99]. 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 [56] [57]. Because of this, most of the previous studies [33] [37, 46, 55] which utilise GP-HH focus on evolving single priority-based CDRs. In a priority-based 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.

A problem arises, however, from the standard approach of evolving single priority-based dispatching rules for dynamic JSS problems. The prominent method of evaluating the effectiveness of rules for dynamic JSS problems [46, 45, 98] is to use stochastic discrete-event simulation. In these simulations, rules need to make many complex decisions which can impact performance later down the line. As dispatching rules are myopic in nature [12] and only consider local decisions, it is unlikely a single priority-based dispatching rule can effectively handle the decisions that need to be made. Hunt et al. [51] showed that by incorporating features which "look-ahead" to future decisions, higher quality rules can be evolved by GP. Therefore, an approach which separates out the different "lookahead" decisions into multiple rules may be more effective than evolving single rules. In other words, such a hyper-heuristic approach could potentially generate higher quality rules for the problem domain more consistently than a hyper-heuristic approach which evolves single rules. In addition, it may be possible that the evolved multi-rules can handle different dynamic JSS problem domains more effectively than evolved single rules. In essence, an approach which evolves multi-rules may potentially be more *robust* to the differences between the target domain (the problem domain that the evolved rules are applied to) and source domain (the problem domain that the rules are trained on) than an approach which evolves single rules, as future decisions may be more important than local decisions in certain problem domains than others.

Although many hyper-heuristic approaches have been proposed for dynamic JSS problems with unforeseen job arrivals, not many have explored dynamic JSS problems with the possibility of *machine breakdowns*. In a JSS problem with machine breakdowns, a stochastic event occurs where a specific machine is inactive for a period of time. Any job being processed at the time the machine becomes inactive needs to be processed again from the start of the specific operation. In JSS problems where the number of jobs arriving at the shop floor is known in advance, there are three categories of approaches when unforeseen events, such as machine breakdowns, occur in scheduling problems. The first is *completely reactive scheduling* [90], which includes dispatching rule approaches. In completely reactive scheduling, the schedule is built iteratively as the jobs are processed by the machines at each dispatching decision. The second is *predictive-reactive scheduling* [121]. A predictive-reactive algorithm predicts the final schedule beforehand, and 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. The third is *robust pro-active scheduling* [77], where the probability of the machine breaking down is known in advance, and the scheduling algorithm attempts to accommodate for the possible breakdowns and schedule jobs accordingly. However, if machine breakdowns are incorporated into dynamic JSS problems, predictive-reactive algorithms do not work, as there is always the possibility of a new job arriving sometime in the near future. In addition, the probability of machine breakdowns is not known in advance, which makes predicting when breakdowns occur difficult. Machine breakdowns have been studied extensively as extensions to static JSS problems. One GP-HH example is an approach by Yin et al. [123], where they evolve dispatching rules for a static single machine problem. However, not much research has been carried out for machine breakdowns for dynamic JSS problems in general.

Finally, dynamic JSS problems with difference sources of dynamic events brings up a difficult challenge where the GP evolved rules need to be flexible enough to accommodate different strategies for the different types of problems to be effective [83]. A dynamic JSS problem instances with both *dynamic job arrivals* and *machine breakdowns* have different properties to dynamic JSS problem instances with only dynamic job arrivals. For example, coping with a dynamic JSS problem with both dynamic job arrivals and machine breakdowns is difficult with GP, but the evolved rules also need to be effective in the problems where there is no machine breakdowns. However, it may be possible that these problems can be handled effectively by incorporating concepts from *transfer learning* [91], as the both problems (with and without machine breakdowns) likely share common features between the two problems likely could be exploited by a GP approach to improve *generalisation* ability of the evolved rules over the dynamic JSS problem. This is a major motivation behind the concept of *multitask learning* [21, 91]. In multitask learning, goal is to learn multiple problem domains (i.e. *tasks*) simultaneously so that the knowledge acquired from each problem domain can be shared to improve the overall generalisation ability of the algorithm [21]. In addition, *multitask optimisation*, where multitask learning is applied to optimisation problems [42], and evolutionary multitasking [28] have been proposed in the literature, but not for dynamic JSS problems. In addition, no GP approach for dynamic JSS problem have yet incorporated multitask optimisation to evolve high quality rules.

1.3 Research Goals

The overall goal of this thesis is to develop new and effective **GP-HH** approaches to **automatically** design **robust** dispatching rule heuristics for **dynamic job shop scheduling environments**. 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 combining techniques used to develop novel **dispatching rule representations** using **evolution search mechanisms**. By doing this, we can potentially evolve a good set of **reusable** heuristics which are competitive over a wide range of JSS problem domains compared to more specialised heuristics, and can better handle difficult dynamic JSS problems. This research will be broken down into the following key objectives.

1.3.1 Objective One: Develop GP-HH approaches to evolve robust ensembles of dispatching rules for dynamic JSS problems

The first objective is to develop GP-HH approaches to evolving robust rules for dynamic JSS problems that are effective for both the current problem domain and *unseen* problem domains. Robustness is a very important factor when it comes to automated heuristic design with hyper-heuristics, as information about other problem domains are not always available.

Therefore, evolved rules need to be able to handle problem domains other than the one that they were trained on. Otherwise, approaches which evolve rules that are not robust will likely quickly lose effectiveness as properties of the problem domain changes, requiring new rules to be evolved. Therefore, we will investigate the following sub-objectives to handle this objective:

1. First, we will investigate GP-HH approaches to evolving ensemble systems. In ensemble learning, a group of smaller constituent heuristics are developed such that they cover for each other's errors. They then work cooperatively together to form a larger heuristic that can work effectively over a wider range of problem domains. Ensemble learning has been applied extensively to difficult classification problems [100], and is able to deal with complex decisions that single classifiers cannot handle. Because of this, a technique which combines multiple smaller dispatching rules together is very promising for JSS problems. However, careful considerations will need to be made when developing the approach, as rules need to be able to provide good coverage for each others' errors for the ensemble to be effective.

2. Second, we will develop new diversity measures to improve the effectiveness of GP-HH evolved rules. It is possible that certain diversity measures can help prevent GP from converging too quickly to a local optima, resulting in higher quality and more robust rules being evolved. Research has shown that diversity can improve the effectiveness of GP [16, 29]. In addition, diversity is a key component to ensemble learning [100], meaning that a good diversity measure can be combined with an ensemble system to potentially develop a good GP-HH approach to JSS problems.

1.3.2 Objective Two: Develop new GP-HH approaches for dynamic JSS problems subject to machine breakdowns

The second objective is to develop GP-HH approaches for dynamic JSS problems subject to machine breakdowns. Note that GP-HH approaches to JSS with machine breakdown have not been covered significantly in the literature. This means that several key concepts need to be addressed to be able to evolve effective heuristics for this problem. These concepts are addressed in the following sub-objectives to the second objective:

1. First, we will investigate which features and attributes are important when dealing with machine breakdowns and how they affect the quality of the rules evolved using GP-HH. Research has shown that for dynamic JSS problems with machine breakdowns the effectiveness of different man-made dispatching rules depend on the machine failure rate and the times required to repair broken machines [48]. Therefore, it is likely that incorporating features 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 features such as the number of times a machine has failed, along with the average repair time of the machine, we can potentially avoid processing jobs with long processing times just before a machine failure is "expected" by the evolved rule. Therefore, feature selection and construction techniques, in conjunction with manual design of features, will be used to filter out the good features to deal with this problem.

2. Second, we will investigate whether an approach which decomposes dynamic JSS problems into standard scheduling and breakdown induced scheduling is effective. Predictive-reactive scheduling is very popular for JSS problems with machine breakdown [90]. It may be possible to adapt ideas from predictive-reactive scheduling techniques into GP-HH approaches to evolve good rules for this prob-

lem. In other words, an approach which incorporates multiple rules to handle different scenarios, e.g., when the machine is functioning normally, when a machine has broken down, etc., will be explored. In a static single machine problem with breakdowns, a GP-HH approach which split the decision of selecting rules and adding idle time to two separate decision makers proved very effective [123]. This approach will need to determine how to split the problem into specific scenarios, and how to evolve effective rules for each scenario. In addition, a suitable objective to represent 'disruptions', or changes which need to be made due to the sudden machine breakdowns, will need to be explored, such as an objective for minimising the number of disruptions or the sum durations of times jobs were processed for before they are disrupted by the failure.

1.3.3 Objective Three: Develop multitask GP-HH approaches that evolves a generalist rule and a portfolio of rules

The third objective is to develop multitask GP-HH approaches for dynamic JSS problems that is able to *generalise* well over the dynamic JSS problem with dynamic job arrivals and machine breakdowns that is described in Objective Two. To evolve effective generalist rule, we will be incorporating concepts from multitask optimisation [91]. First, the dynamic JSS problem will be divided into smaller tasks by separating out the problem instances by their properties, e.g., whether the machine breakdown occurs in a problem instance or not. Afterwards, by directing the GP search to handle the specific task as effectively as possible, it may be possible to discover useful features in one task that can be effectively applied to other tasks as well [91]. The discovered features would then help GP evolved rules improve their effectiveness on *unseen data* as well. To incorporate multitask optimisation to GP for the dynamic JSS problem, this is broken down into two sub-objectives:

1. First, we propose incorporating multitask optimisation to the GP approach by developing a *niching* technique [106, 78]. Niching techniques are used in the literature to promote diversity between the GP individuals in a single GP run [78]. When combined with GP as a multitask optimisation method for the dynamic JSS problem, niching may be able 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. This will also feed into the overall best (i.e. the "generalist") individual that performs well over the entire problem, as the individuals that are not cleared are likely to have sufficiently high performances in the tasks and are likely to have useful features to improve the quality of the best generalist individual found so far. Finally, the niched GP approach can be used to output two separate sets of rules for different situations: 1) a generalist rule that is likely to be effective on the dynamic JSS problem instances where the properties of the problem instances are unknown, and 2) specialist rules for situations where the properties of the problem instances are known where a more specific rule than the generalist rule can be applied.

2. Second, we propose a novel multitask GP approach to the dynamic JSS problem with dynamic job arrivals and machine breakdowns which uses the neighbourhood relations between the different tasks in the dynamic JSS problems to best determine the tasks that the GP individuals in the population should be assigned to. 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 the they perform poorly on, and prioritise the evaluation of good individuals. Similar to the niched GP approach mentioned above, this approach will also output a generalist and specialist rules for the different situations.

1.4 Major Contributions

This section will be later down the line after the thesis structure is fully finalised.

1.5 Organisation of Thesis

This section will be later down the line after the thesis structure is fully finalised.

Chapter 2

Literature Review

This literature review is composed of two major components and a summary. The first major component are sections which cover the background. Section 2.1 covers basic concepts for this research. Section 2.2 covers job shop scheduling (JSS), including the difference between active and non-delay schedules, and a description of JSS problems with machine breakdowns. Section 2.3 covers genetic programming (GP) in detail, and Section 2.4 defines hyper-heuristics. The second major component of the literature review are the related work into JSS (Section 2.5) and GP based hyper-heuristic (GP-HH) approaches to scheduling problems (Section 2.6). A summary of the literature review is given in Section 2.7.

2.1 Basic Concepts

This section covers the basic concepts for topics that are covered for this research. This includes a basic definition for sequencing and scheduling, machine learning, and the differences between heuristics, meta-heuristics and hyper-heuristics. Finally, definitions for evolutionary computation (EC), ensemble learning and multi-agent systems, and domain adaptation and concept drift are detailed.

2.1.1 Sequencing/Scheduling

In *sequencing* and *scheduling* problems, there are limited resources which can be used for activities over time, and the goal is to find the best way to utilise these limited resources [71]. Research in sequencing and scheduling is motivated by utilising the limited resources as effectively as possible. Sequencing and scheduling research ranges from studying how to plan production so that the manufacturing systems generate output with minimal waste in time and money, to studying computer controls [71]. In general, sequencing and scheduling problem can be used to model a wide range of practical real-world scenarios. JSS is an example of a type of scheduling problem, which is discussed in further detail below. Other types of scheduling problems involve flowshop scheduling problems, open shop scheduling problems and flexible scheduling problems [99].

2.1.2 Machine Learning

Machine learning is a field of computer science, and is an area of artificial intelligence [5]. Machine learning has been described as a field of study that gives computers the ability to learn without being explicitly programmed [5]. This means that at the base level, machine learning techniques are independent of the problem domain, e.g., neural networks (NN), a machine learning technique, can be applied to a wide range of classification problems [124].

2.1.3 Heuristics/Meta-heuristics/Hyper-heuristics

There are a number of definitions for heuristics and meta-heuristics. For this thesis, we define heuristics as 'rules-of-thumb' algorithms which can be applied to problems directly without adaptation [105]. For a particular problem, they often incorporate existing knowledge about the problem to find good solutions for the problem instances [11]. This is useful in cases where attempting to solve computational problems exactly is impractical, such as exact methods taking too much time [80]. However, it also means that heuristics are problem specific [118]. For example, a heuristic designed for one scheduling problem may not be effective for another scheduling problem.

On the other hand, we define meta-heuristics as problem independent methods which can be adapted to solve different problems [89]. Meta-heuristics solve problems by having a general procedure which can be used to search the solution space of the problem, by incorporating lower level, problem specific heuristics which facilitate the search [89]. Aside from the low level heuristics that are incorporated into a meta-heuristic method, the base meta-heuristic makes little assumptions about the problem domain which it is applied to. Therefore, a number of machine learning techniques and local search procedures can be considered as meta-heuristic approaches [89]. Meta-heuristics are often designed to deal with difficult optimisation problems.

Finally, hyper-heuristics are problem independent techniques which do not directly solve problems unlike heuristics and meta-heuristics. Instead, hyper-heuristics incorporate low level heuristic components which can be combined to construct a heuristic for a specific problem [15, 11]. It then searches for good heuristics, which can be used to solve the problem. More details on hyper-heuristics will be given in Section 2.4.

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 [30] and swarm intelligence [65].

Evolutionary algorithms (EAs) are influenced by the idea of Darwinian evolution and the science of genetics [30]. An evolutionary algorithm consists of a population of individuals representing a biological species in nature, and the population evolves over time to solve the problem more effectively. This is done by the use of genetic operators such as crossover and mutation (described in Section 2.3.4), along with applying selective pressure to the individuals. Examples of evolutionary algorithms are genetic algorithm (GA) [30] and genetic programming (GP) [67].

On the other hand, swarm intelligence focuses on the idea of cooperation between the individual members of a swarm, which represents group of biological organisms such as birds (particle swarm optimisation [64]), ants (ant colony optimisation [34]) or bees (artificial bee colony optimisation [63]). The individuals provide feedback to each other to better work towards solving the problem.

2.1.5 Ensemble Learning/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 [100], whereas multi-agent systems are associated with both single and multi-task problems [92]. 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 [100]. The idea of multiple experts that work in tandem to solve a problem have been shown to be very effective in classification [13, 35].

A key component of ensemble learning is diversity between the constituent subcomponents [100]. Without diversity, the subcomponents cannot sufficiently cover for each other's errors, and can potentially result in a bad decision. Therefore, number of different approaches have been proposed for diversifying the subcomponents of an ensemble [9, 13, 22, 35, 74, 75]. One notable approach of constructing diverse set of classifiers for classification problems simultaneously is negative correlation learning (NCL) [9, 22, 74, 75]. 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.

2.1.6 Domain Adaptation/Concept Drift

Pan and Yang [91] describe *domain adaptation* as a special case of *transfer learning*. In transfer learning, a learning algorithm (e.g. GP) is trained on a source domain, and applied to a target domain which may or may not have the same properties as the source domain [91]. In addition, transfer learning can also include problems where the target task is different from the source task. For domain adaptation, the target and the source tasks are the same (denoted as *transductive transfer learning*), but only the source domain contains labelled data for which the learning algorithm can compare to. In optimisation, a parallel can be drawn where a hyper-heuristic model is trained over a specific set of training instances which do not exactly match the test instances.

On the other hand, *concept drift* is described as a phenomenon where the problem is non stationary, with the underlying properties of the problem changing over time [118]. Learning under concept drift needs to handle changes in the properties of the instances in the problem domain by either being robust or by updating the learned model over time. Unlike transfer learning, where the labels for the target instances are either known or unknown, in concept drift the labels are revealed over time after the learned model has been applied to the instances, allowing the learned model to make adjustments if necessary. In addition, the learning algorithm handles instances one-by-one, and the full dataset is not revealed.

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 are defined. In addition, various techniques from the literature for JSS, ranging from exact mathematical optimisation to meta-heuristics, are detailed. Finally, machine breakdowns in JSS problems are covered, along with approaches specifically focused on problems with 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 at the shop floor, and N jobs arrive at the shop floor. In static JSS problems, N is known in advance, whereas in dynamic JSS problems it is unknown. 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 *flowshop* 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 $C_{\max} = \max_{j=\{1,\ldots,N\}} C_j$ [99], 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 [99]. 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. In addition, if the jobs have a weight/penalty factor w_j, then the tardiness and the weight of each job are used in JSS problems with total weighted tardiness (TWT) minimisation objective, which is given by Equation (2.3).

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

Job shop scheduling (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. [26] and then refined by Lawler et al. [70], 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), flowshop 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 $Jm|| \sum_j w_j T_j$.

In dynamic JSS problems which involves the unforeseen arrival of jobs, certain objectives, such as makespan, are not suitable objectives. Makespan is the completion of the "last" operation of all jobs. However, in a dynamic JSS problem instance there is always the possibility of a new job arriving sometime in the near future, which means that we do not know what the "last" job is. On the other hand, objectives such as minimising TWT or mean flowtime are based on the jobs completed up to a certain point, meaning that they are suitable for dynamic JSS problems. Therefore, the two objectives have been studied extensively in the literature [46, 45, 52, 86, 98] for dynamic JSS problems, and will be the main objectives for this thesis.

2.2.2 Active Schedules and Non-delay Schedules

When describing scheduling algorithms for JSS problems, key definitions which need to be covered are active and non-delay scheduling [99]. 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 [99, 109]. 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 [99]. 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 (2.4) [86].

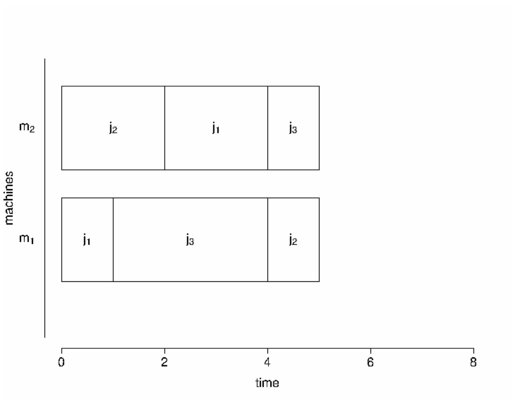
$$A' = \{j \in A \mid r(o_j) \leq \alpha(C'_{\text{earliest}} - r_{\text{earliest}}) + r_{\text{earliest}}\} \tag{2.4}$$

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 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. If a job j's arrival time $r(o_j)$ is greater than the earliest completion time C'_{earliest}, then the job with the earliest completion time could have been processed before job j's arrival. Giffler and Thompson [38] showed that the optimal solution to JSS problem instances belonged to the set of solutions which are active schedules.

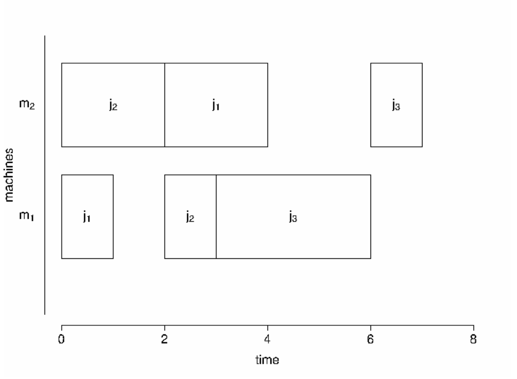
For example, consider the JSS problem instance with the makespan minimisation objective shown in Table 2.1. In this problem, there are $N = 3$ jobs and $M = 2$ machines, denoted as m_1 and m_2. The SPT dispatching rule will generate different solutions depending on whether it is active or non-delay. The non-delay schedule 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, as 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. For this 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 $M = 2$ machine 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	$3, 1$



(a) Schedule generated by non-delay SPT



(b) 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

2.2.3 Machine Breakdowns

In a JSS problem instance with machine breakdowns, there is an unknown point in time where a machine m in the problem instance becomes unavailable, needs to be repaired for time b. If preemption is not allowed, then any operation for a job that machine m was processing before the breakdown needs to be started from the beginning of the operation. The idea of machine breakdowns being an important factor was brought up by McKay et al. [76]. They mention that in a 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 [121] to failures occurring stochastically [77].

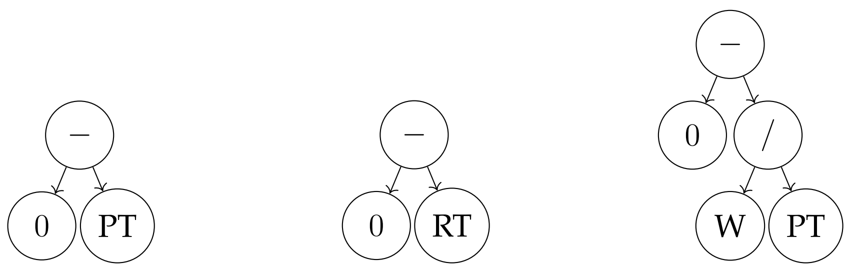
A significant amount of the literature on machine breakdown apply *predictive-reactive* schedules for static JSS problems with a fixed number of jobs which arrive at the shop floor with their properties known in advance [90]. In a predictive-reactive scheduling algorithm, an initial schedule is generated for a problem instance, and then adjustments to the schedule are made as machine breakdowns occur. Adjusting the initial schedule to accommodate for the unknown breakdown event is called *rescheduling*. In the literature [2, 121, 123], 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. Another method of handling breakdowns is to incorporate idle times into machines schedules when breakdowns are expected. These are denoted as *robust pro-active* scheduling [90]. Similar to predictive-reactive scheduling approaches, the literature which use robust pro-active techniques have an additional objective of minimising disruption [77] for the scheduling problem. The final set of approaches are *completely reactive* scheduling [90], where the decisions are made locally in real-time as machines become available. Therefore, dispatching rules are considered as completely reactive scheduling algorithms [90].

2.3 Genetic Programming

Genetic programming (GP) [67] 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 takes no arguments, whereas the non-terminal set consists of symbols which take one or more arguments [67]. In JSS, the most prominent method is to use a tree-based GP, where the individuals represent priority function trees which can be used as priority-based dispatching rules [11]. 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 arrives on the shop floor plus the constant 0. The non-terminals are $+$, $-$, $\times$ and $/$, which



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

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

are basic arithmetic operators. In most cases [11], 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.

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* [67]. 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* [67] is a hybrid of the full and 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.

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-based 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 [11]. The individual's performance over the simulation, based on the objective of the problem, is used as 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 [45, 88] is one such approach, which will be discussed in more detail in Section 2.3.7.

2.3.4 Selection

After the fitness values have been assigned to the individuals in the GP population, a selection procedure is carried out. A selection procedure is a method of filtering out a proportion of the individuals using their fitnesses. Two examples of selection procedures are roulette wheel selection (or fitness-proportionate selection) [8] and tournament selection [8]. 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 selected. The individual with the best fitness out of the group 'survives' to the next generation, 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 [67], and will be used for this research.

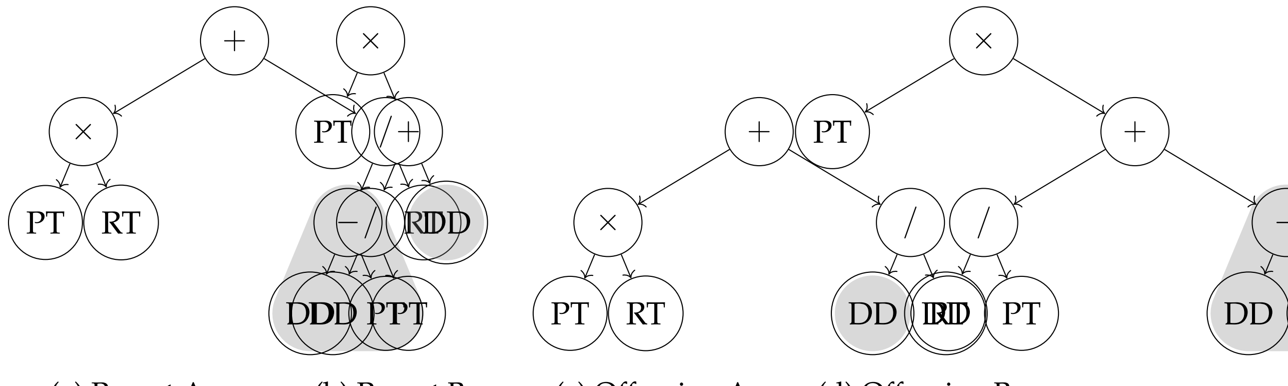
2.3.5 Genetic Operators

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, 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 $\mathrm{PT} \times \mathrm{RT} + (\mathrm{DD} - \mathrm{PT})/\mathrm{RT}$ is removed, and is replaced by the randomly generated replacement shown in Figure 2.4b, resulting in an offspring representing a function $\mathrm{DD} \times \mathrm{RT}^2 + (\mathrm{DD} - \mathrm{PT})/\mathrm{RT}$.

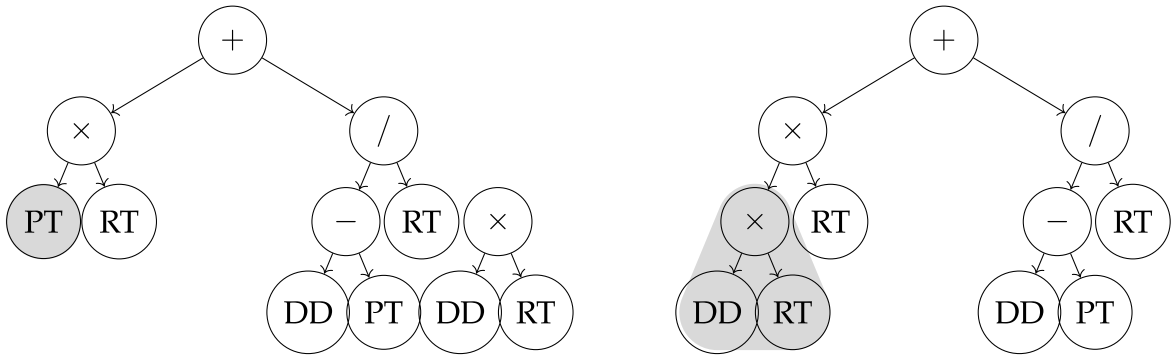
2.3.6 Multi-objective GP

Multi-objective evolutionary algorithm (MOEA) [31], also known as evolutionary multi-objective optimisation (EMO) [126], aims to evolve a *front*



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

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



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

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

of best tradeoff solutions simultaneously to optimise over multiple conflicting objectives using evolutionary computation techniques [60]. In particular, a MOEA technique incorporated as part of a GP system is denoted as multi-objective genetic programming (MOGP). In MOEA, a solution S_i is said to *dominate* another solution S_j if S_i is better or equal on all objectives to S_j, and is better on at least one objective. This is shown in Equation (2.5) for minimisation objective. A solution is said to be *nondominated* if it is not dominated by any solution in the population. Therefore, the solutions in a front of best tradeoff solutions found by a MOEA technique are all nondominated.

$$S_i \succ S_j \leftrightarrow \forall_{Score} \left[(S_i)_{Score} \leq (S_j)_{Score} \right] \wedge \exists_k \left[(S_i)_k < (S_j)_k \right] \tag{2.5}$$

Two very popular MOEA approaches have been used extensively in the literature. The first popular MOEA approach is NSGA-II by Deb et al. [31]. NSGA-II first ranks solutions using a dominance rank, and then uses the crowding distance, which is the solution's proximity to other solutions in the population, to determine the partial ordering of the solution. The solutions which are non-dominated in the population are assigned the dominance rank of one, and then the next set of solutions which are only dominated by solutions with dominance rank of one is assigned dominance rank of two, and so forth. The second is SPEA2 by Zitzler et al. [126]. SPEA2 uses dominance rank and dominance strength to determine the partial ordering of a solution. Dominance strength of a solution is the number of solutions in the population the solution dominates.

2.3.7 Coevolution in GP

Coevolutionary algorithms (CEA) are techniques where multi-agent behaviours are incorporated with evolutionary computation techniques [92]. 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 subproblems, and to allow different individuals of the population to fill in different 'ecological niches' [101], i.e., specifically handle the different subproblems. The evolved individuals are then combined together to a cohesive solver.

A popular CEA approach is cooperative coevolution [101], where a population of individuals is partitioned into subpopulations, and individuals from each subpopulation interact only with representatives of other subpopulations. A representative is selected randomly when the EA process is initialised, but is an individual which has the best fitness out of the other individuals in the subpopulation after the first evaluation procedure. One example which uses cooperative coevolution is a GP approach by Nguyen et al. [85], where it is used to evolve multiple scheduling policies to handle dynamic JSS problem with three objectives. Another CEA approach includes orthogonal evolution of teams (OET) [104, 108], 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.

2.3.8 Surrogate Modelling

In GP and other EAs, the most time consuming part of the process is often the fitness evaluation procedure [58]. Therefore, much research has been carried out in being able to make fitness evaluation more efficient, which includes surrogate modelling [59]. In a surrogate modelling approach, a surrogate function is used in place of full fitness evaluations to *approximate* the fitness of individuals in the population [59]. An effective surrogate function is computationally inexpensive in comparison to carrying out full fitness evaluations, and allows the surrogate assisted EA process to generate an output is competitive or outperforms an equivalent

EA process which does not use a surrogate model [45] [88].

2.4 Hyper-heuristics

Described by Cowling et al. [27] as "heuristics to choose heuristics", hyper-heuristics have gained the attention of researchers whose goals were to design generic but effective methods to problems [15]. Burke et al. [15] describes 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. Hyper-heuristics are provided low level heuristic components, operators to combine low level heuristics together, and a method of evaluating the performance of heuristics. The hyper-heuristic then searches in the heuristic space [18] to output a heuristic. This process distinguishes hyper-heuristics from meta-heuristics, as meta-heuristics search in the solution space. In other words, hyper-heuristics search for the method of solving problem instances, whereas meta-heuristics directly search for solutions to the problem instances.

One popular hyper-heuristic approach is genetic programming based hyper-heuristic (GP-HH) [15]. In GP-HH, individuals in the GP population represent heuristics which can solve problem instances in the problem domain the GP-HH system is designed for. Figure 2.5, shows a high-level overview of a tree-based GP being applied to a JSS problem. In the figure, a tree-based GP-HH takes in the base heuristics $\{\mathrm{PT}, \mathrm{RT}, \mathrm{DD}\}$ and the operators $\{+, -, \times\}$ to initialise the individuals in the population. The GP system then passes the heuristic onto the JSS problem domain, and get a goodness measure back in the form of the individual's fitness. The final output is a heuristic which represents a priority-based dispatching rule $\mathrm{RT} + \mathrm{PT} \times \mathrm{DD}$, which can be reapplied the JSS problem.

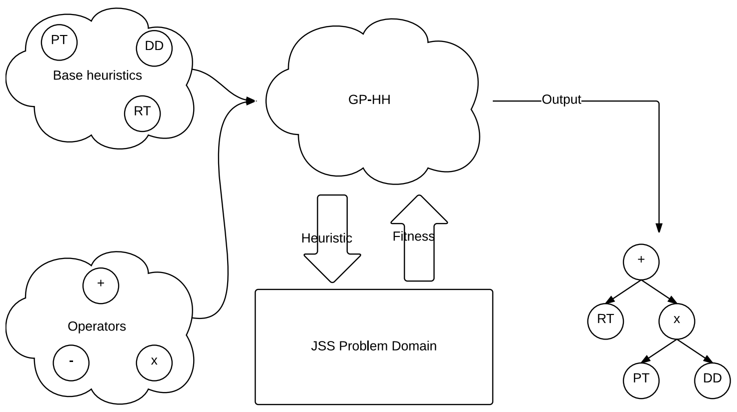


Figure 2.5: Overview of a tree-based GP-HH applied to JSS

2.5 Job Shop Scheduling 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

The initial techniques to scheduling problems in general were to build models for specific scheduling problems. This resulted in the algorithms which could solve specific problems directly, such as Jackson's algorithm [53], which could solve the two machine JSS problem with makespan minimisation optimally. Jackson's algorithm is an extension of Johnson's algorithm [61] for two machine flowshop problems with makespan minimisation. However, Garey et al. [36] 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 for makespan minimisation JSS problems with more than two machines that runs in polynomial time in the worst case scenario. In addition, other objectives for JSS problems, such as TWT, are NP-hard [99]. Researchers have suggested using search techniques to handle more difficult JSS problems. Branch-and-bound [69] is one of the search techniques which have been used significantly in the literature [4, 6, 14, 19, 20]. A notable branch-and-bound approach has been proposed by Carlier and Pinson [20] in 1989, where they were able to find an optimal solution for a JSS problem instance proposed by Muth and Thompson [81] 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 [102].

Dynamic programming is an exact optimisation technique which has been applied to static JSS problems [41, 72, 99]. Dynamic programming approaches divide a JSS problem instance into constituent subproblems, and attempts to solve over the subproblems to solve the problem instance. Lawler and Moore [72] 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. [41] in 2012, where they adapt an approach proposed by Held and Karp [44] 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.

2.5.2 Heuristic Techniques

Heuristic approaches have been applied to large static JSS problems, where problem instances have up to $N = 8700$ jobs and $M = 9$ machines [93]. They have also been applied to dynamic JSS problems, where it is not appropriate to use exact optimisation techniques, as a schedule may not be optimal as new unforeseen events occur during processing [111]. Examples of heuristics are small dispatching rules such as SPT, FIFO and earliest due date (EDD) [99]. The EDD rule prioritises jobs with the earliest due date time. Vepsalainen and Morton [117] covers multiple dispatching rule heuristics for JSS problems with TWT minimisation objective, including the COVERT (cost over time) and ATC (apparent tardiness cost) priority-based dispatching rules. They first identify "look-ahead" features from the COVERT and ATC rules, and modify the two rules to improve their performances over the JSS problem. Jayamohan and Rajendran [56] compare various dispatching rules which have been proposed in the literature, such as rules proposed by Holthaus and Rajendran [48], and also present new dispatching rules. They provide a computational analysis of the rules on dynamic JSS problems for minimising various aspects of flowtime and tardiness. Jayamohan and Rajendran extended the comparative study further [57] to incorporate weighted COVERT and weighted ATC rules [117] and to propose new dispatching rules. Holthaus and Rajendran [49] 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) [103], which was previously the best rule for minimising mean tardiness for dynamic JSS problem instances.

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 reoptimise each machine using an algorithm developed by Carlier [19], 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 reoptimised. Adams et al. shows that the shifting bottleneck 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. Cheng et al. [23] provides 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. [125]. Heuristics for JSS, such as weighted COVERT rule, are integrated into GA in a hybrid approach. They show that the hybrid GA outperformed pure GA approaches. Other meta-heuristic approaches which use EC techniques include artificial bee colony (ABC) optimiation [24], ant colony optimisation (ACO) [25] and particle swarm optimisation (PSO) [107, 120]. Other meta-heuristic approaches include neighbourhood search techniques such as Tabu Search (TS) [39]. Examples include an approach by Dell'Amico and Trubian [32], where they apply TS to various benchmark JSS problems [3, 6, 81] with the makespan minimisation objective. They show that the TS approach outperforms various benchmarks from the literature [116], and can find an optimal solution in many cases. Other neighbourhood search techniques include simulated annealing [66]. Kreipl [68] proposed a large step random walk (LSRW) method. This is considered one of the best local search method for dealing with JSS problems with TWT minimisation objective, where even the hybrid GA approach proposed by Zhou et al. [125] does not perform as well as Kreipl's LSRW approach according to Nguyen et al. [86].

2.5.4 Machine Breakdown Techniques

There are several examples of predictive-reactive scheduling algorithms in the scheduling literature. Wu et al. [121] 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 [19]. 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 [90], and show that the

AO algorithm provides a final schedule which has significantly lower deviation.

On the other hand, Mehta and Uzsoy [77] 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.

Finally, Holthaus [47] compares the application of various dispatching rules in dynamic JSS problems under various flowtime and tardiness minimisation objectives with machine breakdowns. Dispatching rules can be applied to machine breakdown problems as a completely reactive scheduling heuristic [90], as quick decisions can be made locally in real time. 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 performance of the rules depend on the properties of machine breakdowns, such as the failure rate of a machine. Subramaniam et al. [110] 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. Finally, Ouelhadj and Petrovic [90] provides a comprehensive survey of JSS problems with machine breakdowns, whereas Suresh and Chaudhuri [111] covers machine breakdown in brief detail along with JSS problems where unknown events occur.

2.6 GP-HH for Scheduling Problems

Many GP-HH approaches to scheduling focus on evolving dispatching rules. Dimopoulos and Zalzala [33] use GP to evolve priority based dispatching rules for a static single machine problem with total tardiness minimisation. The GP individuals represent priority function trees, and a job's attributes are used to calculate the priority of the job. They showed that the rules evolved using GP performed better than man-made benchmark dispatching rules. Yin et al. [123] use GP to evolve rules for a single machine static JSS 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, and 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. Geiger et al. [37] use GP-HH to evolve priority rules for multiple single machine JSS problems. Some of the JSS problems they deal with can be solved optimally within polynomial time, such as the single machine problem with makespan minimisation ($1||C_{\max}$), whereas other problems are NP-hard. They show that GP can evolve optimal rules for non NP-hard problems, and perform better than man-made dispatching rule benchmarks for the NP-hard problems. Jakobović and Budin [54] use GP in a single machine problem and a JSS problem, both with non-zero release times and the TWT minimisation objective. For the single machine problem, they use a single tree to represent a priority-based dispatching rule. On the other hand, they propose a multiple tree system for the JSS problem, denoted as GP-3, where a decision tree is used to determine which out of the two remaining trees representing scheduling heuristics to apply. GP-3 outperforms the approach used for the single machine problem. Jakobović et al. [55] then use the same GP system for a parallel machine problem. In a parallel machine problem, a job can be processed on any of the M machines on the shop floor. However, each machine has different speed, meaning that a job will be processed for different length of time on different machines. They show that GP evolved rules generally perform better than man-made dispatching rules such as ATCS (ATC with setups) [73], but do not compare the rules evolved by their GP-HH to rules evolved by other GP-HH approaches.

Tay and Ho [113] proposed a GP-HH approach for a multi-objective JSS problem. The objectives are to minimise the makespan, the mean tardiness and the mean flowtime. To evaluate the GP individuals, they combine the three objective values together into a single objective value by taking the weighted sum of the three values. They show that the evolved rules outperform other simple dispatching rules. However, Hildebrandt et al. [46] showed that Tay and Ho's approach performs poorly in dynamic environments, and proposed a GP approach to dynamic JSS problem with mean flowtime minimisation. They first calculate the "due date" of a job based on the time which job arrive on the shop floor and the sum processing time required to complete the job, and use a number of due date related terminals. This is because Holthaus and Rajendran [48] observed that even though flowtime minimisation problems do not contain due date related attributes, rules which use the artificial due dates of jobs and due date related terminals perform better than rules which do not use due date related terminals. Hildebrandt et al. [46] showed that the GP evolved rules perform well over the dynamic JSS problem, and are robust to changes in the problem. Pickardt et al. [98] use a two-step approach to a dynamic scheduling problem for semiconductor manufacturing, where the rules evolved using GP are combined with other dispatching rules which have been proposed in the literature using an Evolutionary Algorithm (EA). The hybrid approach performs better than the two approaches by themselves, and outperforms the best benchmarks for the particular problem from the literature.

Nguyen et al. [86] proposed three tree-based GP representations that can be used to evolve rules for static JSS problems with makespan and TWT minimisation, and a dynamic JSS problem with TWT minimisation. The first GP representation evolves decision trees, the second GP representation evolves priority functions which can be used in a priority-based 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 [125] for static JSS problems. Nguyen et al. [87] then proposed a method of evolving iterative dispatching rules (IDRs) using GP. For a JSS problem instance, an IDR first constructs a schedule, and gets the expected 'meta-attributes' such as tardiness and completion times of jobs from the generated schedule, and iteratively generates better schedules using the values of the meta-attributes of jobs from previous iterations. The evolved IDRs perform significantly better than rules evolved by standard GP, and outperform the man-made rules from the literature, such as WSPT, COVERT and ATC. Hunt et al. [52] first tests whether GP can be used to evolve rules that can handle the two machine JSS problem with makespan minimisation optimally. They evolve a large number of rules, and show that some of the rules function equivalently to Jackson's algorithm [53], i.e., generates optimal solutions to the problem. They then apply GP to a dynamic JSS problem with TWT minimisation, and show that the GP evolved rules outperforms the benchmark rules. Hunt et al. [51] then extended their work to incorporate lookahead terminals for GP when evolving rules. This is done to reduce the myopic nature of dispatching rules. They show that the "less-myopic" evolved rules performed significantly better than standard evolved rules over various dynamic JSS problem instances. A comprehensive survey of approaches which use evolutionary scheduling, including GP-HH approaches to scheduling problems, is covered by Branke et al. [11].

On the other hand, there are limited number of studies that use a surrogate model in conjunction with GP for scheduling problems. Hildebrandt and Branke [45] investigate using the genotype, the structure of the individuals, and the phenotype, the output decisions made by the individuals, for developing an effective surrogate model for a JSS problem with mean flowtime minimisation objective. In addition, Holthaus' rule ($2\text{PT} + \text{WINQ} + \text{NPT}$) [49] is used as a *reference rule*, where the genotype or the phenotype of the individual is compared against to calculate the approximate fitnesses of the individuals. They found that using the phenotype of the individual in the surrogate model is more effective than using the genotype, producing higher quality individuals, and that the surrogate assisted GP system evolves better solutions within a smaller number of generations than a standard GP system.

Nguyen et al. [88] proposed a surrogate assisted GP (SGP) system for dynamic JSS problems which is adapted from the Hildebrandt and Branke's [45] approach. In Nguyen et al.'s approach, they compare two different selection schemes. The first selection scheme removes any individuals with duplicate phenotypic output from the population. The second selection scheme removes individuals with the same output as the individuals in the previous population. They show that the two SGP generally produce higher quality individuals than standard GP for dynamic JSS problems. In addition, the additional computations required for the selection schemes compared to the standard GP are not significant except for the dynamic JSS problem with the mean flowtime minimisation objective.

2.7 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 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-based 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-based 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 [12], a single rule is more likely to make a particularly poor decision than a group of rules working together [100].

- There are a limited number of papers which use surrogate models to assist a GP system for scheduling problems. Both Hildebrandt and Branke [45] and Nguyen et al's [88] works use the same reference rule (Holthaus' rule [49]) to calculate the approximate fitness of individuals in the GP population. Therefore, there are opportunities to investigate the use of new reference rules and other comparison measures between an individual and the reference rule to calculate the approximate fitness of the individual.

- Although there are many meta-heuristic approaches to scheduling problems with breakdowns, there are very limited number of hyper-heuristic approaches to handling scheduling problems with machine breakdowns. Yin et al. [123] have proposed a GP-HH approach to a static JSS problem with machine breakdown, and Holthaus [47] 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, the literature covered in this chapter which use GP-HH for dynamic scheduling problems only deal with evolving rules for problems where the underlying properties of how jobs are generated remain the same throughout a simulation. Scenarios where the underlying properties of the problem changes over time during a single simulation run have not been covered extensively, and there are no methods to automatically detect when the properties of the problem has changed over time so that the evolved rules can be updated.

Chapter 3

Ensemble based GP Approaches for DJSS

3.1 Chapter Goal

Job shop scheduling (JSS) problems are types of combinatorial optimisation problems that model manufacturing environments [102]. In a JSS problem, there is a shop floor with machines that are used to process arriving jobs. Although both academics and industry experts have interest in JSS problems, there has always been a gap between the classical research to JSS (from an academic perspective) and its application (from an industrial perspective) [76]. In classical research to JSS, many approaches handle *static* JSS problems, where the properties of the shop are known a priori [99]. However, in practice the properties of the shop are extremely variable and it is commonly believed that any change to the shop floor can cause ripple effects [76]. To bridge the gaps between the static JSS problems that have been handled by academics (where the problems are predictable and can be optimised in advance) and unpredictable real-world scenarios encountered in the industry, researchers have focused on matching the problem more closely with real-world manufacturing environments by incorporating unforeseen events into the problem [90]. JSS problems that have real-time unforeseen events that affect the properties of jobs, machines and shop floor are called *dynamic* JSS problems [90]. Examples of unforeseen events include dynamic job arrivals, where job arrivals are unknown until they reach the shop floor, and machine breakdowns [90, 46, 11]. In real-world manufacturing environments, it is likely that last minute (and potentially urgent) jobs can arrive that require attention [90, 46]. In general, dynamic JSS problems are much more difficult than static JSS problems, and conventional optimisation methods cannot solve dynamic JSS problems due to the unpredictable changes in the shop floor [111]. Instead, *dispatching rules* [99] are studied by both academics and industry experts for dynamic JSS problems due to their interpretability [11], short reactions times and their ability to cope well with the unforeseen events in dynamic JSS problems [86]. To automate the design of effective dispatching rules, many genetic programming based hyper-heuristic (GP-HH) approaches to dynamic JSS problems have been proposed in the literature [11]. GP-HH approaches have successfully evolved dispatching rules for various dynamic JSS problems which are more effective than the man-made counterparts [11].

In addition to the primary motivation of automatically generating effective dispatching rules for dynamic JSS problems [11], many GP-HH approaches have focused on evolving *robust* dispatching rules for the dynamic JSS problems [85], i.e., rules that function reliably and effectively despite noise and unexpected changes in the problem domain. However, many approaches focus on evolving dispatching rules with a single constituent component, and are often not sufficiently robust for dynamic JSS problems. This issue was addressed by evolving *ensembles* [100] of dispatching rules [96, 95, 43]. Ensemble learning has been shown to be effective at training robust high quality rules for JSS problems [96, 95, 43] and problems outside of JSS (e.g. classification problems [100]) because ensemble members are able to minimise errors made by other ensemble members [100]. This makes ensemble approaches a promising direction to improve the robustness of rules evolved by GP-HH for dynamic JSS problems. However, the existing ensemble GP-HH approaches to JSS [96, 95, 43] only use majority voting combination scheme [100] to combine the outputs of the subcomponents of the ensembles together. Design of interactions between the ensemble members is an important factor in ensemble learning [100]. It is clearly evidenced on some classification problems that different combination schemes, such as linear combination and weighted combination schemes, can be more effective than majority voting [100]. Therefore, it may be possible that better rules can be evolved by GP using combination schemes besides majority voting. In addition, by analysing the rules evolved by the different combination schemes, one can observe the behaviour of ensembles that are applied to dynamic JSS problem instances. This allows for future ensemble GP-HH approaches which may generate higher quality and more robust rules than the current state-of-the-art GP-HH approaches for dynamic JSS problems.

3.2 Investigated Ensemble Approaches for DJSS

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

EGP evolves dispatching rules which are used in an ensemble of priority rules to determine which job to process for a ready machine. However, using a single population for ensembles will require a carefully designed grouping scheme to group the individuals together, along with a complementary evaluation scheme to evaluate the grouped individuals. Instead of doing this, we consider an approach where we partition the population into S smaller subpopulations. Each subpopulation has size K. EGP-JSS groups the individuals from the different subpopulations together to form an ensemble. This approach of splitting the population into smaller subpopulations that work together to solve a problem is known as cooperative coevolution [101]. By using cooperative coevolution, we allow for the subcomponents of the ensemble to apply crossover, mutation and reproduction separately, and allow for diversity between the different subcomponents.

In cooperative coevolution, individuals in a subpopulation only interact with representatives of the other subpopulations when they are being evaluated for their fitness. A representative is defined as the individual with the best fitness in a subpopulation. Initially, before the first fitness evaluation, the representative of each subpopulation is chosen randomly. Unlike Potter and De Jong's [101] cooperative coevolution approach, we do not destroy unproductive subpopulations, as destroying and regenerating a new subpopulation of individuals will require a large number of generations for it to be effective.

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

3.2.2 Coevolutionary GP Process with Fitness Sharing

There are a various niching algorithms which have been proposed in the literature to promote the diversities of individuals in a GP population. However, the focus of this paper is to investigate effective phenotypic distance measures used for niching. Therefore, in our study we focus on a specific niching algorithm called fitness sharing which have been used extensively in the literature [40].

Figure 3.1 shows a general overview of a coevolutionary GP process, and how the fitness sharing algorithm is incorporated. First, an initial population of GP individuals is generated randomly. Afterwards, a grouping procedure is carried out. The grouping procedure groups the individuals into ensembles temporary, i.e., the ensembles are discarded after the evaluation procedure [101], or peramently, e.g., the ensembles belong part of

Figure 3.1: An overview of a coevolutionary GP that uses fitness sharing.

the GP population [122]. After the ensembles have been generated, non-delay dispatching rules are formed from the ensembles and potentially the GP individuals, where the individuals represent single priority-based dispatching rules. The rules are then applied to the DJSS training instances to generate schedules. The performances of the ensembles either directly [96] or indirectly [95] affect the chances of the individuals surviving to the next generation. In one approach [96], the fitness of an individual directly calculated from the performance of the ensemble over the training instances that the individual belong to. In another approach [95], the "fitness" of the ensembles are not used to calculate the fitnesses of the individuals, but the probability of the individual being selected as a parent depends on the fitnesses of the ensembles that it belongs to. However, when fitness sharing is incorporated into the coevolutionary GP process, the fitnesses of the individuals in an ensemble are adjusted based on the decisions they made as the ensemble is applied to the training instances. After the fitnesses of the individuals and the ensembles are calculated, the selection and the breeding procedures occur to produce the next generation of individuals, and the process is repeated until the termination criteria is reached.

3.2.3 MLGP-JSS Process Overview

A key component of MLGP-JSS is *groups*. A group is a set of individuals in the population that cooperate with each other. For our approach, we use groups as ensembles to solve JSS problem instances. This is discussed in further detail in Section **??**. The MLGP-JSS process is broken down into three major steps. The first step is to carry 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, i.e., the maximum number of generations, is reached, the final output is the best group of individuals found so far, an ensemble of dispatching rules that can be applied to dynamic JSS problem instances. The overall MLGP-JSS process is shown in Algorithm 2, where G_B is the number of groups bred at each generation, and I_B 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 group. The entities for cooperation are selected using roulette wheel selection over all entities. An example is shown in Fig. 3.2a, 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. The two parents randomly exchange one individual to produce the child groups. Individuals in a parent group have equal probabilities of being exchanged. In Fig. 3.2b, 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. 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

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

Figure 3.2: Examples of group operators used to breed new groups.

is randomly selected with uniform probability. An example of mutation operator adding an individual is shown in Fig. 3.2c, where individual I5 is added to G2 to produce offspring G4.

Evolution on the Individual Level:

Crossover and mutation operators used to breed the individuals are the standard operators for tree-based GP [67]. To select the parent individuals to breed new offspring, a group grp is selected based on a probability proportional to the group's fitness. It is expected that individuals in a group contribute to achieve cooperation despite their fitnesses [122]. Therefore, an appropriate number of parent individuals are selected from group grp 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.

3.3 Modified EGP-JSS Approach and Combination Schemes

The combination schemes investigated are majority voting, linear combination, weighted majority voting and weighted linear combination [100]. Ensemble GP-HH approaches that use linear combination, weighted majority voting and weighted linear combination schemes have not been investigated in the literature for dynamic JSS problems. An existing ensemble GP-HH called Ensemble Genetic Programming for JSS (EGP-JSS) adapted from the literature so that the investigated combination schemes can be incorporated to the GP's training process [96]. However, the original EGP-JSS approach needs to be modified to evolve the weighted major-

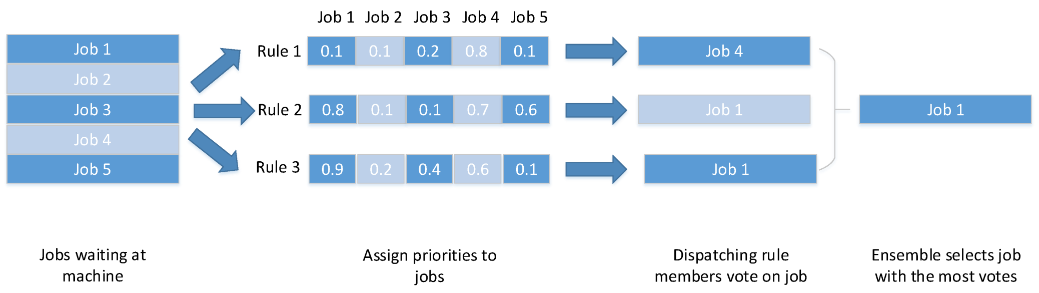

Job 1
Job 2
Job 3
Job 4
Job 5
Job 1 Job 2 Job 3 Job 4 Job 5
Rule 1 0.1 0.1 0.2 0.8 0.1
Rule 2 0.8 0.1 0.1 0.7 0.6
Rule 3 0.9 0.2 0.4 0.6 0.1
Job 4
Job 1
Job 1
Job 1
Jobs waiting at machine
Assign priorities to jobs
Dispatching rule members vote on job
Ensemble selects job with the most votes

Figure 3.3: Example of majority voting for ensembles being applied to a decision situation.

ity voting and linear combination schemes. Therefore, the modified EGP-JSS (mEGP-JSS) extends the GP-HH approach by simultaneously evolving the weights for the weighted ensemble by incorporating a genetic algorithm (GA) [**?**] to the cooperative coevolutionary procedure. This section first provides descriptions of the combination schemes and how they are applied to the decision situations. Afterwards, we cover the changes made by mEGP-JSS for weighted combination schemes and to the fitness evaluation.

3.3.1 Job Selection by Combination Schemes

The combination schemes are used by the ensembles during job sequencing decisions at decision situations. The majority voting scheme is adapted from existing ensemble GP-HH approaches to dynamic JSS [96, 95, 43], and the apparent tardiness cost (ATC) rule [117] is used as a tiebreaker. Figure 3.3 shows an example of majority voting with three ensemble members being applied jointly to a decision situation with five jobs.

For the linear combination scheme, the members of the ensemble assign "scores" to the different jobs, and the job with the highest sum score is selected to be processed. The score of a job assigned by a member is calculated from the assigned priorities using min-max normalisation. Given that a member 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 (3.1). In the Equation, $\delta_{\min}$ and $\delta_{\max}$ are the minimum and maximum priorities assigned to any jobs waiting at the machine by member x.

$$Score_{j,x} = \frac{\delta_{j,x} - \delta_{\min}}{\delta_{\max} - \delta_{\min}} \tag{3.1}$$

For the weighted majority voting scheme, each member x has a weight ν_x. This means that the member assigns a score equal to its weight to the job that it votes for. For the weighted linear combination scheme, an intermediate value is first calculated for a job j by ensemble member x by normalising the priorities (Equation (3.1)). Afterwards, the intermediate value is multiplied by the member's weight ν_x to get the final score for the job.

When applied to a decision situation, different combination schemes can potentially select different jobs for processing. For example, Table 3.1 shows ensembles with combination schemes being applied to a decision situation with five jobs. In the tables, the six ensemble members have the weights $3.5, 1.0, 2.0, 1.2, 2.5, 6.5$. For the majority voting scheme, job 1 is selected as rules 2, 3 and 4 vote on job 1. For the linear combination scheme, job 4 is selected because rules 1 and 5 heavily favour job 4 and job 4 has the second highest priorities for rules 2 and 3, resulting in job 4 having a higher score than job 1 overall. For the weighted majority voting scheme, job 3 is selected because rule 6 has a high weight compared to the other rules, resulting in a higher score. Likewise, job 3 is selected for the weighted linear combination because of the high weight of rule 6 contributing towards the total score towards the job. The different outcomes for the decision situations encountered by the ensembles using the different combination schemes may lead to the ensembles generating significantly different schedules and performances overall. It may also lead to significantly different rule structures and behaviours being evolved by mEGP-JSS.

3.3.2 Incorporating Weighted Combination Scheme to EGP-JSS

To evolve ensembles that use weighted combination schemes, EGP-JSS's GP process is modified to evolve weights for the members of the ensembles and the ensemble members simultaneously in a single evolutionary run. This allows us to evolve weighted ensembles with similar computation times as the computation times required to evolve unweighted ensembles, as other approaches (such as a two-step procedure) would require added computation after the GP process is completed. Given that there are Ψ GP subpopulations of size Φ, an additional $(\Psi + 1)$th GA subpopulation of size Φ_ν are also initialised along with the GP population. The GA individuals in the additional subpopulation are real-valued vectors of length Ψ, where the gene value at index i corresponds to the weight ν_i of an individual from subpopulation i. The gene values have a lower bound of zero, which prevents the member weights from being negative. On the other hand, from pilot experiments we found that the choice in the upper bound did not make a significant difference in the performance of the weighted ensembles during the preliminary experiments. Therefore, the upper bound of ten is selected as a rule of thumb. During the evaluation procedure, the GP individual being evaluated is grouped with the representatives from other subpopulations, i.e., the individuals with the best fitnesses from each subpopulations [101], and the representative of the GA subpopulation to form a weighted ensemble.

Similar to how a GP individual in EGP-JSS is evaluated, a GA individual is evaluated by grouping it up with the representatives from the GP subpopulations to form a weighted ensemble. An example of this is shown in Fig. 3.4, where the ensemble consists of GP individuals Ind 1, Ind 2 and Ind 3 from the three GP subpopulations. Afterwards, Ind 1, Ind 2 and Ind 3 are weighted using the values from GA 1. The weighted ensemble is then applied to the training instances. However, the fitness cal-

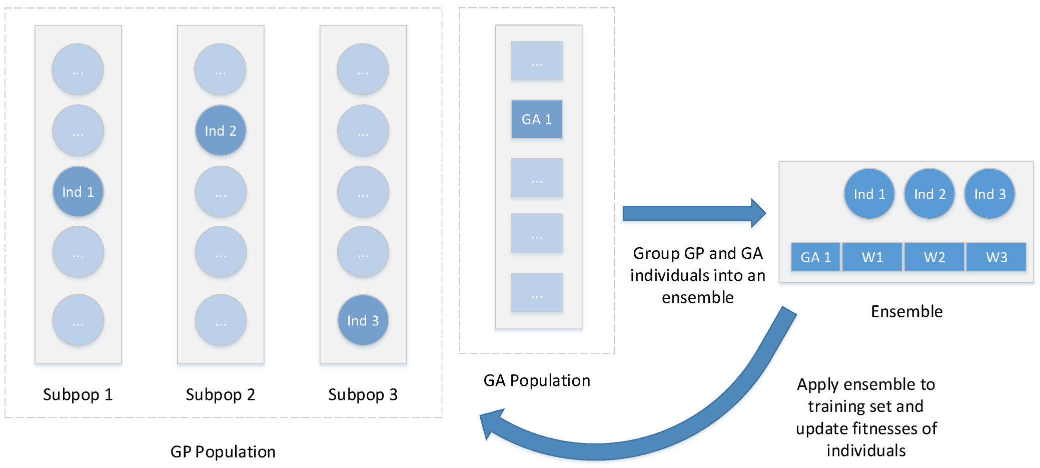

Ind 1
Ind 2
Ind 3
Subpop 1
Subpop 2
Subpop 3
GP Population
GA 1
GA Population
Group GP and GA individuals into an ensemble
Ind 1
Ind 2
Ind 3
GA 1
W1
W2
W3
Ensemble
Apply ensemble to training set and update fitnesses of individuals

Figure 3.4: Example of mEGP-JSS generating an ensemble with weighted members.

culation for the GP and the GA individuals is modified from the original EGP-JSS approach, and is described below (Section **??**). After all GP and GA individuals have been evaluated, the GA representative is updated to the individual with the best performance. Afterwards, standard one point crossover operator and a gaussian mutation operator [**?**] are used for breeding the next generation of individuals. The output of the GP process is the representatives from the GP subpopulations that form an ensemble, along with the representative from the GA subpopulation that assigns the weights to the members of the ensemble.

3.3.3 mEGP-JSS Performance and Fitness Calculation

The fitness calculation in mEGP-JSS is modified from the original EGP-JSS. After an ensemble E containing a GP individual x is applied to the dynamic JSS training instances, the fitness $f(x)$ of individual x is calculated by normalising the performances of the ensemble over the problem instances using the ATC as a *reference rule*. First, both the ensemble E and the reference rule R is applied to a problem instance I to obtain the schedules with the objectives values $Obj(E, I)$ and $Obj(R, I)$ respectively. Afterwards, the normalised objective value $Obj'(E, I)$ of the ensemble over the instance is given in Equation (3.2).

$$Obj'(E, I) = \frac{Obj(E, I)}{Obj(R, I)} \tag{3.2}$$

In the literature, normalisation of performances over the problem instances have been adopted in fitness calculation for GP-HH approaches to dynamic JSS to reduce the bias towards specific training instances [46, 79]. If a particular JSS problem instance has a greater optimal mean tardiness value than another JSS problem instance, we can expect many dispatching rules to obtain higher mean tardiness values on this problem instance. This results in the first problem instance having a greater effect on the fitness of an individual than the second problem instance if the mean tardiness values are not normalised. After the normalisation, the normalised performance of the ensemble E over the instances in the training set T_{train} is the average normalised mean tardiness values of the schedules as shown in Equation (3.3), which is also used as the fitness $f(x)$ of the individual x.

$$Perf(E, T_{train}) = \frac{1}{|T_{train}|} \sum_{I \in T_{train}} Obj'(E, I) \tag{3.3}$$

3.4 New Measures of Behaviour Analysis of the Evolved Ensembles

Based on the mEGP-JSS algorithm, we carry out several behavioural analyses of the evolved ensembles that are evolved with the different combination schemes. Many GP-HH approaches in the literature to dynamic JSS have both analysed the structures (i.e. genotype [67]) and the behaviours (i.e. phenotype [67]) of evolved rules [11]. However, due to the limited ensemble evolutionary scheduling approaches to dynamic JSS in the literature [96, 95, 43], the amount of analysis on the behaviours of evolved ensembles is limited. For example, Hart and Sim [43] have carried out both structural and behavioural analysis of ensembles evolved by their hyper-heuristic approach. They performed extensive analysis on ensembles' performances when they are limited (or not limited) in size and analyse the relation between the structural make-up of the ensembles (using the terminal distributions) and how well it solves specific problem instances. While the structural analysis is quite extensive, the analysis of the behaviours of the ensembles evolved by Hart and Sim is limited, with the scope of the paper being focused on the effectiveness of particular ensembles for specific problem instances.

Analysing and observing the behaviours of the evolved ensembles is important for understanding exactly how ensemble GP-HH can outperform standard GP-HH approaches, allowing us to exploit the advantages of ensemble based approaches while avoiding the disadvantages. In an ensemble, the dispatching rule members that collaborate with each other need to behave in specific ways for the ensemble to be effective. The members of the ensemble need to be able to sufficiently buffer for each other [100]. In other words, for complex decision situations, where the selection of a job that may lead to good solutions is ambiguous, the ensembles need to be able to make diverse sets of good decisions. In classification, complex decisions occur at the class boundaries, where it is ambiguous whether an instance belongs to one class or another [100]. 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 [100]. 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 [100, 96]. However, it is unlikely that analysing the ensembles through structural analysis will give clear results on the interactions between the different members of the ensembles. The arithmetic trees in the GP evolved dispatching rules often have redundancies in the tree [11], 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 [45] and diversity measures [82]. 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.

3.4.1 Measuring Behaviours of Evolved Ensembles

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

3.4.2 New Measure 1 – Decision Conflict (M1)

M1 calculates the proportion of sampled decision situations where the members of the ensembles have assigned highest priorities equally between two or more jobs in decision situations, i.e., how often the top decisions "conflict" with each other. For a decision situation 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 (3.4). From this, we get M1(E) in Equation (3.5), 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.

$$V_{j,j^*} = |\{x \in E | j^* = j_{j,x}\}| \tag{3.4}$$

$$\text{M1}(E) = \frac{1}{\Omega}|\{j \in [1, \ldots, \Omega], \exists i, j[i \neq j \wedge \max_k\{V_{j,k}\} = V_{j,i} = V_{j,j}]\}| \tag{3.5}$$

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

3.4.3 New Measure 2 – High Contribution Members (M2)

M2 calculates the proportion of sampled decision situations where the decisions of the highest contributing member match-up with the decisions made by the ensemble E, i.e., member whose decisions most align with the decisions made by the ensemble. For a decision situation j out of Ω decisions, suppose that a member x of an ensemble E assigns the highest priority to job $j_{j,x}$. If 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 $q^{\text{M2}}_{x,E}$ between member x and ensemble E over Ω decisions is given by Equation (3.6). Afterwards, we get M2(E) in Equation (3.7), where it is equal to the member with the most overlap with the ensemble over Ω decisions.

$$q^{\text{M2}}_{x,E} = \frac{1}{\Omega}|\{j \in [1, \ldots, \Omega]|j_{j,x} = j_{j,E}\}| \tag{3.6}$$

$$\text{M2}(E) = \max_{x}\{q^{\text{M2}}_{x,E}\} \tag{3.7}$$

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

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

M3 calculates the worst average ranks of the ensemble members using a rule ranking system modified from Hildebrandt and Branke [45], 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 3.2.

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

$$\text{M3}(E) = \max_{x}\{\frac{1}{\Omega}\sum_{j=1}^{\Omega} r'_{j,x}\} \tag{3.8}$$

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

3.5 Experiment Design

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

3.5.1 Baseline Approach

We modify a GP-HH approach that has been proposed by Park et al. [95], denoted as GP-JSS, that evolves single dispatching rules as the baseline approach for evaluating the mEGP-JSS approach. Although EGP-JSS has been shown to outperform GP-HH that evolve single rules [96, 95], evolving single dispatching rules using GP-HH is very prominent for DJSS problems [84]. GP evolved single rules have also shown to consistently outperform man-made dispatching rules [11]. For consistency, the GP-JSS uses the same fitness function as the mEGP-JSS approaches (Equation (3.2)), and the same terminals, arithmetic operators and GP parameters provided below to evolve the single rules.

3.5.2 GP Terminal and Function Sets

The terminal set used by GP-JSS and mEGP-JSS approaches consist of a mixture of terminal sets used by existing GP-HH approaches in the literature [46, 86, 51]. These terminals range from common attributes (e.g. operation processing time $GPpt$) to more complex terminals that utilise multiple common attributes (e.g. remaining processing time of job $GPrt$) and the previous states of the shop floor (e.g. average wait time at next machine $GPnqw$). These terminals have been shown to evolve high quality dispatching rules in the literature [51]. 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. `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 5.1.

3.5.3 GP and GA Parameter Settings

The parameters used for GP-JSS, mEGP-MV, mEGP-LC, mEGP-wMV and mEGP-wLC approaches are shown in Table 4.4. The GP parameters for mEGP-MV and mEGP-LC are kept consistent as the parameters used by Park et al. [96] for the mEGP-JSS approach, where the total population size (1024) is the same as the population size of GP-JSS. These parameters are modified from Koza's parameter setting [67], which is a standard set of parameters used for GP. For GP process in mEGP-wMV and mEGP-wLC, the four GP subpopulation sizes at 231 and the GA subpopulation size at 100 add to total population size of 1024. Since the number of GP subpopulations is four, the sizes of the GA individuals are also four. The GA parameters for mEGP-wMV and mEGP-wLC are based on common GA parameters used in the literature [**?**]. Afterwards, select parameters, i.e., the lower and the upper bounds, were set to 0 and 10 respectively after a sensitivity analysis was carried out. However, as long as the lower bound was non-negative, i.e., a member in a mEGP-wMV rule or a mEGP-wLC rule did not have negative weight, the choice of the lower and upper bounds did not significantly affect the qualities of the rules.

3.5.4 Dynamic JSS Simulation Model

An existing simulation model proposed by Hunt et al. [51] is used in this paper, where discrete-event simulations are used to represent dynamic JSS problem instances. Discrete-event simulations have been used to evaluate various GP-HH approaches for dynamic JSS in the literature [46, 85, 98, 86, 52, 51]. In a discrete-event simulation, the jobs arriving on the shop floor are generated stochastically. This means that a dynamic JSS problem instance is generated from a seed value and a set of *simulation configurations*. In Hunt et al.'s simulation model [51], two training sets $4op$ and $8op$ are used to evolve the rules. At every generation, a different seed is used in conjunction with the training set's simulation configuration, resulting in different dynamic JSS training instances being used to evaluate the GP individuals. This has been shown to improve the quality of the rule output from the GP process compared to fixing the seed [46]. The parameter values for Hunt et al's simulation model [51] are given in Table 4.3.

3.6 Results and Analysis

This section covers the evaluation of mEGP-JSS approaches and the combination schemes integrated with the mEGP-JSS processes in this paper. The rules are evolved by the benchmark GP-JSS approach and the different combination schemes of the mEGP-JSS approach. First, we provide the plot of the fitnesses of the individuals in the different GP approaches as they are applied to the two training sets $4op$ and $8op$ to evolve the dispatching rules. This is done thirty times to evolve a set of dispatching rules that are compared based on their performance over the simulation model described in Section **??**. Afterwards, the analysis procedure described in Section **??** is carried out on mEGP-JSS to measure the values of M1, M2 and M3 from the evolved rules. The discussion of the results from Sections **??** and **??** can also be found in Section **??**, i.e., after all the results have been provided.

3.6.1 Training Fitness Convergence Curves

To get the training performances of the different GP approaches, we first get the training performances of the independent runs. For a GP-JSS run, the training performance at a specific generation is the individual with the best fitness in the population. For a mEGP-JSS run, the training performance at a specific generation is the average performance of the individuals with the best fitness in each subpopulation, i.e., the individuals that will be updated as the representative of the next generation. The training performances over the generations are then averaged out over the multiple independent runs, and are shown in Figure 3.5.

For the rules evolved over $4op$, we can see that the different GP approaches except mEGP-wMV roughly converge to the same output, whereas mEGP-wMV consistently has worse training performance over the generations than the other approaches. On the other hand, for the rules evolved over $8op$, mEGP-LC rules have significantly worse training performance than the other rules over generations between 10 and 30. However, mEGP-LC rules converge to similar training performances as the other GP rules after generation 30.

3.6.2 Test Performance

The sets of evolved rules are used to generate schedules for the instances in the training and the test sets. We examined 20 different test scenarios, each with different processing times (μ), utilisation rates (ρ) and number

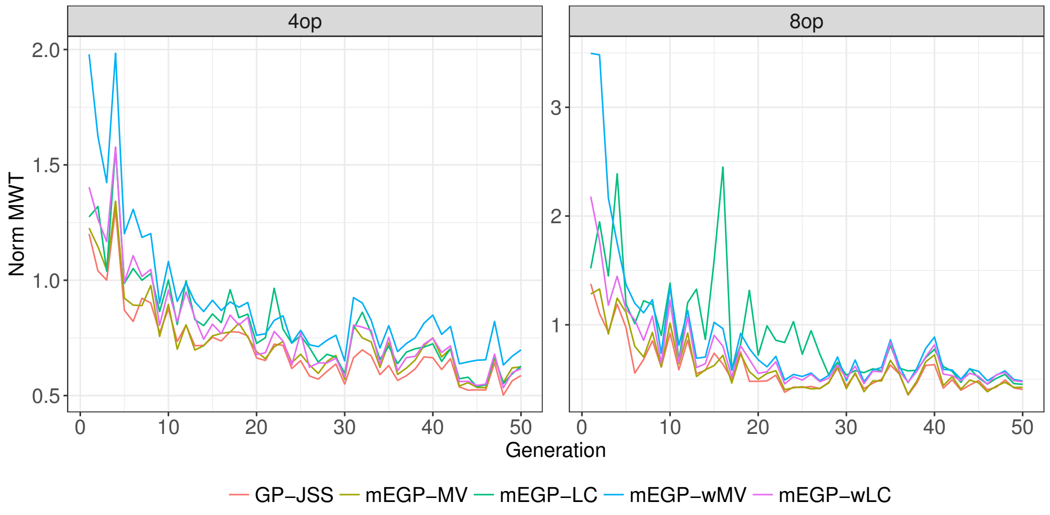


Figure 3.5: The average fitnesses of the GP approaches over the training sets.

of operations per job (N_j). For each test scenario, thirty replications were randomly generated. The test performance of a rule on a test scenario is defined as the average mean tardiness obtained by applying the rule to the thirty corresponding replications. In addition, for each test scenario, we test whether one set of evolved rules is *significantly* better than another by using two tailed t-test at $\alpha = 0.05$ for the pair of results. The performances of rules over the entire simulation model is shown in Table 3.6 for the rules evolved from $4op$ and in Table 3.7 for the rules evolved from $8op$. In the tables, "Training Set" shows the average mean tardiness over the four simulation configurations from both $4op$ and $8op$. The test set is partitioned into four subsets of configurations based on the expected processing time (μ) and the utilisation rate (ρ). In the tables, $\langle x, y, z \rangle$ categories denote that the configuration has $\mu = x$, $\rho = y$ and $N_j = z$. The sections highlighted with *red* mean that the mEGP-JSS with the particular combination scheme is significantly worse than the benchmark GP-JSS approach for the simulation configuration, and the sections highlighted with *blue* mean that mEGP-JSS rules performed significantly better than the GP-JSS rules.

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

In addition to the comparisons to the benchmark GP-JSS rules, results in Tables 3.6 and 3.7 also enables us to perform pairwise comparisons between the mEGP-JSS rules. The results of the pairwise comparison are given in Appendix **??**. The pairwise results show that the unweighted mEGP-JSS approaches (mEGP-MV and mEGP-LC) generally perform better than the weighted counterpart (mEGP-wMV and mEGP-wLC).

3.6.3 Behavioural Analysis and Further Discussion

For each test simulation configuration in the simulation model, a problem instance is generated and M1, M2 and M3 are calculated for mEGP-JSS approaches using the procedure described in Section **??**. 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 parameter tuning, $\epsilon = 10$ and $\Omega = 50$, i.e., 50 decision situations sampled from a problem instance are used to calculate M1, M2 and M3 and the decision situations have at least 10 jobs. The results of applying the analysis measures are shown in Fig. 3.6.

From the figure, the M1 values, which is used to measure the level of conflict between the different ensemble members, do not significantly differ from each other. In addition, with the exception of the mEGP-MV rules evolved over $4op$ and the mEGP-wMV rules evolved over $8op$, the GP

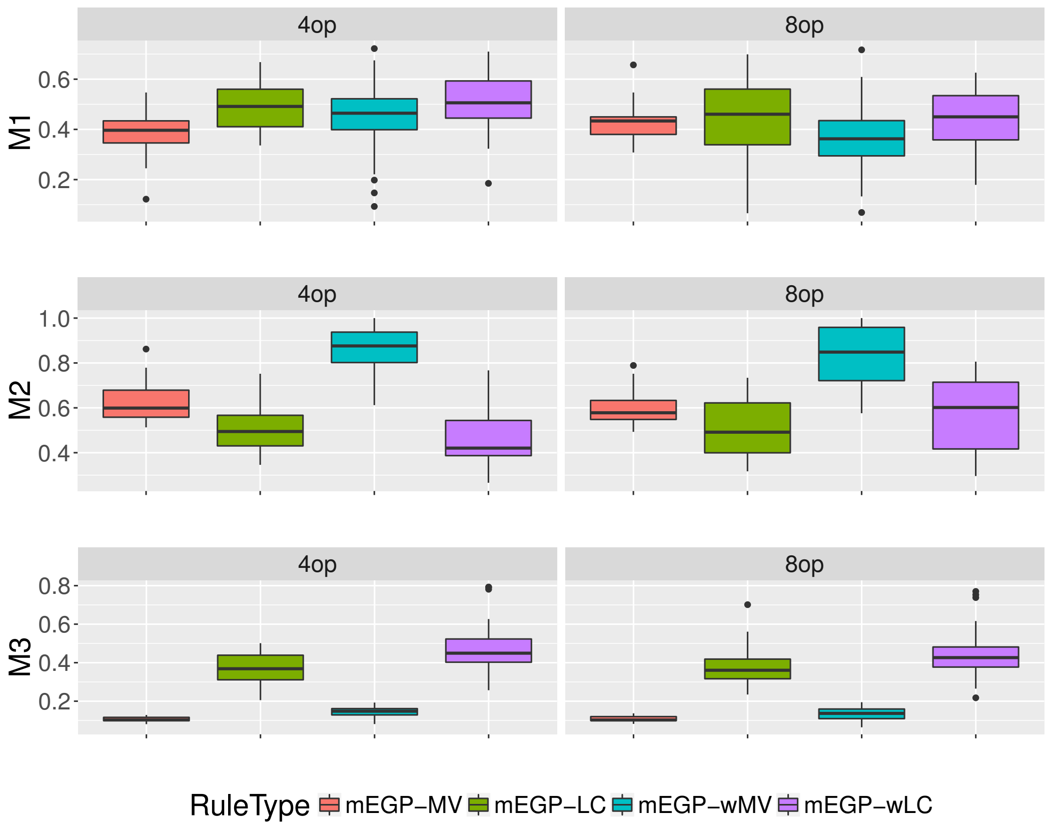


Figure 3.6: The analysis measures for the mEGP-JSS approaches plotted against the performance over the problem instances. $4op$ and $8op$ denotes that the GP rules are evolved from the respective training sets.

rules evolved over both training sets have M1 values over 0.39 on average. This means that for all decision situations, the average proportions of conflicting decisions made by members of the ensembles are above 39%. For example, mEGP-MV rules evolved from $4op$ have an average M1 value of 0.39 and mEGP-MV evolved from $8op$ have an average M1 value of 0.43. This means that out of decision situations sampled for the $8op$ rules, the majority voting scheme resulted in a tie for approximately 43% of the decision situations, where this results in the ATC tiebreaker rule being used.

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

On the other hand, mEGP-wMV rules have high M2 compared to other GP rules evolved over the training sets, whereas the other GP rules have similar M2 values. M2 is used to measure the bias of an ensemble towards a specific ensemble member. Therefore, for mEGP-wMV rules there is a single member that participates highly actively in the majority of the decision making process. Further considering the performance results from Section **??** (Tables 3.6 and 3.7), where the mEGP-wMV rules generally perform worse than other mEGP-JSS combination schemes, it is clear that having high M2 value negatively affects the quality of the GP evolved ensembles. As expected, we found that the mEGP-wMV rules behave similarly to single rules given their high M2 values. It increases the difficulty for other members of an mEGP-wMV ensemble to cover for the "high-contribution" rules's errors. This is reflected in the performance results, where mEGP-wMV generally perform worse than the other GP evolved rules. In addition, it is likely that the unweighted combination schemes perform better than the weighted combination schemes because our combination of GP and GA is unable to explore the search space effectively and find a good configuration of weights and ensemble members simultaneously from the search space. This may potentially cause an underfitting problem, which is evidenced by the convergence curves for mEGP-wMV and the test performance of mEGP-wMV rules. In the results for the training performances of the GP individuals, mEGP-wMV shows generally worse training performances when trained over $4op$, and has slightly worse training performances near the end of the generation when trained over $8op$. In other words, the current results show that evolving individuals that contribute equally towards job selection may be more beneficial than having the individuals be weighted during the evolutionary process.

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

One additional observation made from the performance evaluation is that the performance of mEGP-MV rules have comparable performance to GP-JSS rules. This is contrary to previous results in the literature [96, 95], which state that EGP-JSS with majority voting generally outperform standard GP-HH approaches. However, this is due to the case that previous works considered mainly to train the GP rules on static JSS problem instances [96], or use the same dynamic JSS training instances in every generation [95]. Generating dynamic JSS training instances that are different for any consecutive generations have been shown to improve the generalisation ability of the rules evolved by the GP-HH approaches in the literature [46], and may be the reason why mEGP-MV rules have comparable performances to GP-JSS rules. Additional supplementary experiments in Appendix **??** confirms that using same dynamic JSS instances every generation results in better evolved mEGP-MV rules than GP-JSS rules.

3.6.4 Summary

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

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

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

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

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

3.7 Conclusions and Future Work

In this paper, we investigate the effect of combination schemes in evolving ensembles of dispatching rules for dynamic JSS using GP-HH. Four different combination schemes have been investigated: majority voting, linear combination, weighted majority voting and weighted linear combination [100]. The modified EGP-JSS, denoted as mEGP-JSS, which incorporate the combination schemes are denoted as mEGP-MV, mEGP-LC, mEGP-wMV and mEGP-wLC. The results show that the mEGP-JSS which uses the linear combination scheme (mEGP-LC) generally performs better compared to the other mEGP-JSS approaches and the baseline GP-HH approach. In addition, further analysis shows that the current weighted combination schemes have worse performance than the unweighted combination schemes. This is likely because the weighted ensembles are too biased towards specific ensemble members (particularly for mEGP-wMV rules), which results in the ensembles behaving like single dispatching rules.

There are many further directions that can be investigated from the findings made in this paper. To circumvent the issue where there can be too much bias towards specific members of an ensemble, a two-step approach where good members are first evolved before the weights are assigned to the members could be promising. Another method that may be promising is to assign weights depending on the properties of the shop floor environment at decision situations. Certain rules may be more useful in decision situations with a large number of urgent jobs than a decision situation with less number of jobs, and vice versa. This would likely require a function that takes into account properties of the shop floor as a whole. In addition, linear combination schemes may be effective for other ensemble GP-HH approaches to dynamic JSS problems outside of the EGP-JSS approach. For example, another ensemble GP-HH approach that has evolved effective rules for dynamic JSS is Multilevel Genetic Programming for Job Shop Scheduling (MLGP-JSS) [95], which uses the majority voting to combine the ensemble members' priority assignments. Finally, the analysis measures used in the experiments may potentially be used as a diversity measure in an ensemble GP-HH approach to improve the quality of the evolved ensembles.

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

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

Algorithm 2: MLGP-JSS

```
Initialise Population $\mathcal{P}$ with $I_R$ Individuals;
Evaluate Fitnesses of Individuals $x_1, \ldots, x_{I_R} \in \mathcal{P}$;
while Maximum generation is not reached do
    for $g \leftarrow 1$ to $G_B$ do
        Breed a Group $grp$ from Population $\mathcal{P}$;
        Evaluate the Fitness of Group $grp$;
        Add Group $grp$ to Population $\mathcal{P}$;
    end
    for $i \leftarrow 1$ to $I_B$ do
        Breed an Individual $x$ from Population $\mathcal{P}$;
        Evaluate the Fitness of Individual $idv$;
        Add Individual $idv$ to Population $\mathcal{P}$;
    end
    Retain the $G_R$ Best Groups and the $I_R$ Best Individuals from
      Population $\mathcal{P}$;
end
```

Table 3.1: Examples of the combination schemes for an ensemble with three members being applied to a decision situation with five jobs waiting at the machine. The weights of the members are $\nu = 3.5, 1.0, 2.0, 1.2, 2.5, 6.5$ respectively.

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

Table 3.2: 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	

Table 3.3: The terminal and function sets used for EGP-JSS, where job j is one of the jobs waiting at the machine m^* to process operation o_{ij}.

Terminal	Description
$GPrj$	Operation ready time
$GPro$	Remaining number of operations of job j
$GPrt$	Remaining total processing times of job j
$GPpt$	Operation processing time of job j
$GPrm$	Machine m^* ready time
$GPnj$	Non-delay jobs waiting at machine m^*
$GPdd$	Due date of job j
$GPnpt$	Next operation processing time
$GPnnq$	Number of idle jobs waiting at the next machine
$GPnqw$	Average waiting time of last 5 jobs at the next machine
$GPaqw$	Average waiting time of last 5 jobs at all machines
#	Constant real-value in the interval $[0, 1]$
Function	$+$, $-$, $\times$, $/$, `if`, $\max$, $\min$

Table 3.4: GP parameters used by the GP-HH approaches for evolving ensembles of dispatching rules

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

Table 3.5: Simulation configurations used for the generating arriving jobs in dynamic JSS problem instances.

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

Table 3.6: Comparison of the performances of mEGP-JSS (with the different combination schemes) and GP-JSS over the simulation model. Rules are evolved from $4op$.

Table 3.7: Comparison of the performances of mEGP-JSS (with the different combination schemes) and GP-JSS over the simulation model. Rules are evolved from $8op$.

Chapter 4

GP for dynamic JSS problems subject to machine breakdowns

4.1 Framework for Investigating the Generality of GP

In the field of operations research, job shop scheduling (JSS) and other scheduling problems have been extensively researched for the past 50 years [?]. In a JSS problem, there is a *shop floor* that usually contains a fixed number of *machines* and the machines are used to process incoming *jobs* [?]. A job needs to be processed on a sequence of specific machines and machines on the shop floor can only process one job at a time. The goal in a JSS problem is to find a *schedule*, a solution that gives the sequences of times that the jobs are processed at the machines, that is the optimal given an *objective function* [?]. For example, in a JSS problem with makespan as the objective, the goal is to find a schedule which completes all jobs as early as possible [?].

In a real-world scenario, it is likely that unforeseen events such as dynamic job arrivals and machine breakdowns affect the properties of the shop floor in a JSS problem instance [90]. A JSS problem instances with dynamic job arrivals is called a *dynamic* JSS (DJSS) problem instance [90]. In a DJSS problem instance with dynamic job arrivals, the properties of the arriving jobs are unknown (until they arrive on the shop floor) and the number of jobs that arrive on the shop floor is unknown. To handle DJSS problems with dynamic job arrivals, researchers have proposed various dispatching rule approaches. Dispatching rules [**?**] are iterative heuristics that determine the job that is selected to be processed by the machine when it is available. This decision process for determining the job that is selected by the available machine is called a *decision situation* [45]. Dispatching rules are effective for DJSS problems with dynamic job arrivals because they can react quickly to the arrival of new jobs and can cope with the dynamic environment [86]. In addition, because they are easy to interpret by operators on the shop floor [86], they are used extensively in real-world manufacturing environments, e.g., semi-conductor manufacturing [46]. However, a limitation of dispatching rule approaches is that they are tailored to a specific JSS problem. Although humans are very good at identifying good building blocks for heuristics [17], constructing effective heuristics from the building blocks require extensive trial-and-error testing [86, 46]. Therefore, genetic programming based hyper-heuristic (GP-HH) approaches have been proposed in the literature to automatically evolve dispatching rules for DJSS problems [11]. GP-HH is provided heuristic building blocks, DJSS problem instances for training and searches in the heuristic space to find high quality solutions for the DJSS problem [17]. It has been shown in the literature that GP evolved rules generally perform better than man-made rules for different DJSS problems [11].

There are several factors that need to be considered to develop an effective GP-HH approach to DJSS problems. One of the factors is that evolved rules need to *generalise* well [17, 11]. In other words, the evolved rules trained over a specific problem domain needs to perform well on unseen problem instances both within and outside the problem domain. Generality has been covered in the literature for DJSS problems with dynamic job arrivals [11]. However, although generality of GP for DJSS with dynamic job arrivals have been investigated [86, 46, 11], it is not known how well GP can generalise for over DJSS problem instances with no machine breakdowns and with machine breakdowns. When a machine breakdown occurs, any job that is being processed on the machine is interrupted and the machine needs to be repaired for a specific amount of time before it is back “online” again. Machine breakdowns can severely disrupt the processing that occurs on the shop floor.

The goal of this paper is to investigate the *generality* of GP for DJSS problems with dynamic job arrivals and machine breakdowns, and to analyse the terminals that are effective for DJSS problem instances with machine breakdowns. By analysing the generalisation ability of the evolved rules, it may be possible to determine whether the standard GP-HH approach is suitable for the DJSS problem with machine breakdowns. Otherwise, if the standard GP-HH approach cannot generalise well over the DJSS problems with machine breakdowns, then the analysis of the terminals may provide insight for developing new extensions to the standard GP-HH approach that are more effective for the DJSS problem. To achieve the goal, this paper carries out the following objectives:

(a) Develop a new DJSS dataset for generating problem instances with dynamic job arrivals and machine breakdown.

(b) Investigate the generality of an existing GP-HH [86, 51] by evolving and evaluating the rules over different combinations of machine breakdown scenarios.

(c) Analyse the structure of the GP rules to extract information on the distributions of the terminals for the evolved rules.

First, we cover the background to DJSS in Section **??**, which provides the problem definitions and outlines sample GP-HH approaches to DJSS problems. Afterwards, Section **??** describes the testing framework that is used to test the generality of GP-HH approach for the DJSS problem. Section **??** describes the benchmark GP-HH approach that is used to evolve the rules, the fitness function and the GP parameters. Finally, Section **??** gives the results and an analysis of the findings, and Section **??** gives the concluding remarks and future works.

This section describes the framework that is used to investigate the generality of GP-HH for DJSS problems with dynamic job arrivals and machine breakdowns. This covers the DJSS dataset containing the problem instances with machine breakdowns, how the rules are evolved from problem instances with different machine breakdown scenarios from the dataset, and the procedure for analysing the structures of the GP evolved rules.

4.1.1 Generating DJSS Problem Instances using Simulations

The standard approach in the literature to generate DJSS problem instances is to use discrete-event simulations [45, 46, 51, 47]. This means that the job arrivals, the machine breakdown events and the repair times for the breakdowns are generated stochastically. For this paper, the dataset Δ used to evaluate the generality of GP is modified from the dataset proposed by Holthaus [47], which has been used effectively to evaluate and analyse man-made dispatching rules in DJSS problems with dynamic job arrivals and machine breakdowns. The following parameters are kept consistent as the ones originally used by Holthaus. The problem instances have $M = 10$ machines on the shop floor. The processing times for an operation for a job is generated from a uniform distribution between $[1, 49]$. In other words, the mean processing time of the operations is $\mu = 25$. For generating an arriving job, the arrival times of the jobs are generated according to a Poisson process with mean λ. The utilisation rate is a standard parameter used in DJSS discrete-event simulations that defines the expected proportion of time that the machines are occupied processing the jobs against the total duration of simulation [46, 47]. Because of this, the mean arrival time is often defined by the utilisation rate of the problem instances, and is given in Equation (4.1) [46, 47]. In the equation, ρ is the utilisation rate and p_M is the expected number of operations per job divided by the number of machines. The utilisation rate is set to $\rho = 0.9$, which is consistent with Holthaus's dataset. For our paper, there is no reentry, i.e., a job cannot have at least two separate operations on the same machine [**?**]. This means that a job can have at most 10 operations, and the number of operations per job is modified to be random between 2 to 10 operations, i.e., $p_M = (2 + 10)/2 = 6$.

$$\lambda = \frac{\rho \times p_M}{(1/\mu)} \tag{4.1}$$

The due date of arriving job j $d_j = r_j + h \sum_{i=1}^{N_j} p_{ij}$, where h multiplied to the sum processing times of the job is the due date tightness factor. 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 [47], as preliminary experiments found that due date tightness $h = 8$ resulted in GP evolved rules generating schedules for problem instances where the MWT values are zero. 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 [86, 51]. For each problem instance, there is a "warm-up" period of 500 jobs which do not contribute towards the objective value, and jobs continue arriving until the 2500th job has been completed. However, all jobs that have arrived on the shop floor need to be completed before the problem instance is completed. From Holthaus's dataset [47], 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 between machine breakdowns (η) are the same for all machines on the floor. In addition, for the configuration used to generate a problem instance, 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 processing, e.g., if $\pi = 0.025$ and the all jobs took 2500 time units to process, then the total repair time for all machines is approximately $0.025 \times 2500 = 62.5$ time units. In other words, the machine breakdown level is given by $\pi = r/(\eta + r)$, which means that $\eta = r/\pi - r$ [47]. The dataset has variable configurations for the following parameters: due date tightness (h), mean repair times of machines (r) and breakdown level (π). The configurations can have $r \in \{\mu, 5\mu, 10\mu\}$ and $\pi \in \{0, 0.025, 0.05\}$. Overall, the two due date tightness configurations and the configurations for the machine breakdowns results in a total of 18 different configurations.

4.1.2 GP-HH Training Procedure

To evolve and evaluate the GP rules, different subsets of DJSS problem instances in the dataset are used to evolve different sets of GP rules. Figure 4.1 shows an overview of how the dataset Δ is used to evolve different sets of GP rules that are either "generalists" or "best-fit" over the machine breakdown level (π). The generalist rules are designed to be effective for the different machine breakdown scenarios, whereas the best-fit rules [17] are designed to be effective for specific machine breakdown scenarios. For the scope of this paper, the machine breakdown level parameter allows us to analyse the generality of GP rules in DJSS problems with dynamic job arrivals and machine breakdowns. First, the dataset Δ containing 18 different configurations for generating problem instances is partitioned into three subsets based on the machine breakdown level. In the subsets, machine breakdown level $\pi = 0$ means that the generated problem instances do not have machine breakdowns, $\pi = 0.025$ and $\pi = 0.05$ means that

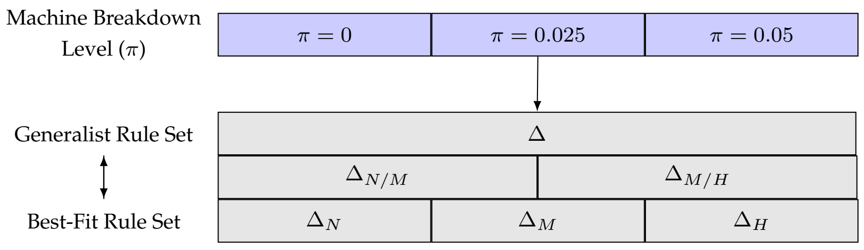


Figure 4.1: Overview of how the dataset used for the DJSS problem is partitioned to train GP rules specialised for different machine breakdown configurations.

the generated problem instances have "medium" and "high" levels of machine breakdowns respectively. The subsets are denoted as Δ_N, Δ_M and Δ_H respectively and contain six different configurations. The best-fit rules are evolved from Δ_N, Δ_M and Δ_H and are designed to cope with the specific 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 best-fit rules, e.g., rule evolved from Δ_H does not perform significantly worse than $\Delta_{M/H}$ for problem instances with $\pi = 0.05$, 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 dataset 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 best-fit over the DJSS problem. This procedure was first covered by Burke et al. [17] 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-' 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., problem instances with no machine breakdowns. In addition, at each generation, the seeds used to stochastically generate the jobs and the machine breakdowns are rotated for the training procedure of the GP-HH. This means that the problem instances used in one generation will be different to the problem instances used for the next generation. This has shown to improve the generalisation ability of the evolved rules for DJSS problems [46].

4.1.3 Rule Terminal Analysis Procedure

For analysing the effectiveness of GP rules, existing literature have proposed methods where small numbers of rules are sampled from the sets of evolved rules and the tree structures of the rules are analysed [46, 86, 51]. However, for investigating the generality of GP over different machine breakdown scenarios, it may be more effective to analyse the structures of entire sets of rules instead of sampling specific rules from the sets, as GP needs to be able to evolve good rules consistently. However, it is too cumbersome to directly analyse the tree structures of the sets of rules directly. Therefore, the distributions of the terminals that make up the GP rule is analysed. For example, if the due date terminal occurs more frequently for the rules in DR-H than the rules in DR-N, then it means that processing urgent job is more important for problem instances with high level of machine breakdowns than problem instances with no machine breakdowns. To calculate the distribution, the proportion of the terminals that make up an evolved rule is first calculated. For example, suppose that an evolved rule has 23 PT terminals out of 150 terminals that make up the rule. Then the $23/150$ is the proportion of PT terminals that make up the rule. Afterwards, the proportions are normalised over the set of evolved rules.

Table 4.1: Terminal set for GP

Standard	RJ	operation ready time of job j
	PT	operation processing time of job j
	RO	remaining number of operations of job j
	RT	remaining total processing times of job j
	RM	machine m^*'s ready time
	DD	due date d_j
	W	job's weight w_j
	#	random number from 0 to 1
Look-ahead	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

4.2 Experimental Design

This section covers the benchmark GP-HH approach that is investigated for this paper, which is based off existing GP-HH approaches. This includes the GP representation, terminal set, function set and fitness measure used for the individuals. Afterwards, the parameters used for the GP-HH is detailed.

4.2.1 GP Representation, Terminals and Function Sets.

The most prominent method of evolving priority-based dispatching rule using GP is to use a tree-based GP, where the individuals represent arithmetic function trees [11]. For this paper, we modify the arithmetic representation proposed by Nguyen et al. [86] to evolve priority-based dispatching rules. In addition, a look-ahead terminal set proposed by Hunt et al. [51] that have effectively been applied to a DJSS problem with unforeseen job arrivals will be incorporated into the terminal set. The list of terminals used for the GP-HH process are shown in Table 4.1 for a job j waiting at machine m^*.

The function set includes of the arithmetic operators $+$, $-$, $\times$, and protected /, where protected / returns one if the denominator is zero. The rest of the operators for the function set are `if`, which returns the value of the second branch if the value of the first branch is greater than or equal to zero or returns the value of the third branch otherwise, max and min operators.

4.2.2 Calculating a GP Individual's Fitness

For the evaluation procedure, the individuals are applied to the DJSS training instances as *non-delay* dispatching rules [?]. A non-delay dispatching rule greedily attempts to minimise the idle time from when a machine is available to when it starts processing the next job [?]. From this, the MWT over the training instances is normalised using the ATC rule, where a standard $k = 3$ value is used for the ATC parameter [117]. The normalisation procedure have been used in the literatures [46] to reduce the bias towards specific problem instances that are more likely to have a higher optimal MWT values than other problem instances in the training set. Given that $\text{MWT}_{x,I}$ and $\text{MWT}_{ref,I}$ are the MWT of the schedule generated by individual x and the reference rule for training instance I respectively, the fitness f_x of an individual x is given in Equation (4.2).

$$f_x = \frac{1}{|\Delta_{train}|} \sum_{I \in \Delta_{train}}^{T_{train}} \frac{\text{MWT}_{x,I}}{\text{MWT}_{ref,I}} \tag{4.2}$$

4.2.3 GP Parameter Settings

The parameters used for GP are modified from the parameters used by GP-HH approaches to DJSS problems in the literature [86, 51] after carrying out parameter tuning on the population size and the crossover, mutation and reproduction rates. After the parameter tuning, the population size is set to 256 to reduce the computational cost, and the number of generations is set to 51. The crossover, mutation and reproduction rates are

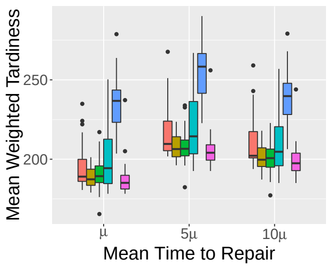
80%, 10% and 10% respectively. The maximum depth of an individual is 8, and the maximum depth of an individual that can be initialised is 2. Tournament selection of size 7 is used during the selection process. Finally, the parameter value used for the ATC reference rule is set to $k = 3.0$ [117].

4.3 Experimental Results

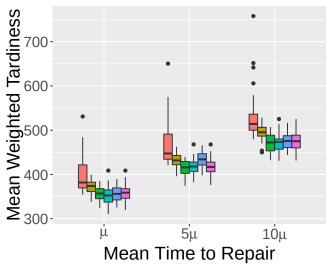
This section covers the evaluation of the GP-HH approach over the DJSS problem with dynamic job arrivals and machine breakdowns. First, 30 independent runs of the GP processes are carried out over the training sets, resulting in DR-N, DR-M, DR-H, DR-N/M, DR-M/H and DR-All each consisting of 30 rules. The sets of GP evolved rules are applied to the problem instances in the test set, and the qualities of the schedules are compared against each other as part of the general evaluation procedure. Afterwards, an analysis on the structures of the evolved rules is carried out.

4.3.1 Evolved Rule Performance Evaluation

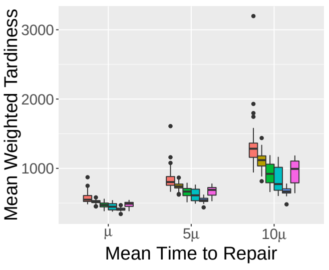
Each configuration in the test set is used to generate 30 different problem instances as part of the test set. In total, this results in a total of $18 \times 30 = 540$ test instances using the 18 different configurations. The sets of GP evolved rules are then applied to the test instances to generate schedules for the problem instances, 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 Student's t-test at $p = 0.05$. The performances of the rule sets over the different problem instances are shown in Figure 4.2. Each box plot shows the results over problem instances in a configuration, and the configurations are categorised by the breakdown level and due date tightness, where $\langle \pi = 0.025, h = 3 \rangle$ denotes that breakdown level is 2.5% and due date tightness is 3.



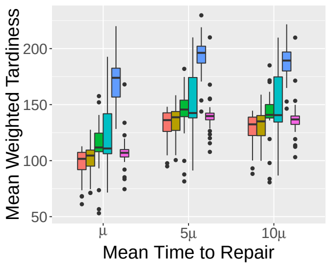
(a) $\langle \pi = 0, h = 3 \rangle$



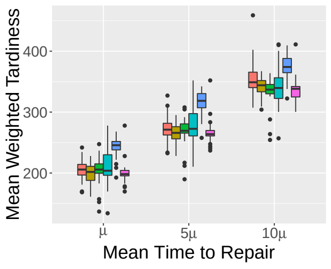
(b) $\langle \pi = 0.025, h = 3 \rangle$



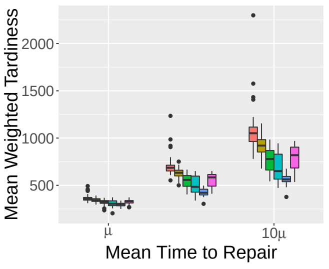
(c) $\langle \pi = 0.05, h = 3 \rangle$



(d) $\langle \pi = 0, h = 5 \rangle$



(e) $\langle \pi = 0.025, h = 5 \rangle$



(f) $\langle \pi = 0.05, h = 5 \rangle$

🟥 DR–N/M 🟨 DR–N 🟩 DR–M/H 🟦 DR–M 🟦 DR–H 🟪 DR–All

Figure 4.2: The comparisons of the mean weighted tardiness performances of the GP rules evolved over different training sets over the problem instances in the test set.

From the results, we can see that for the problem instances generated with $\pi = 0$ and $\pi = 0.05$ DR-N and DR-H generally perform well over the respective problem domain they are trained on, but perform poorly on problem instances with high level of machine breakdowns (for DR-N) and problem instances 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 significant. Finally, the generalist rule DR-All perform well over problem instances with no machine breakdowns and problem instances with machine breakdown level $\pi = 0.025$, but performs significantly worse than DR-H and DR-M/H for problem instances 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 to sensitive to the proportion of time that the machine is broken down during the simulation.

4.3.2 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 make up of the rules. For the sets of GP rules, the proportion of the rule structure made up of the terminals are shown in Figure 4.3.

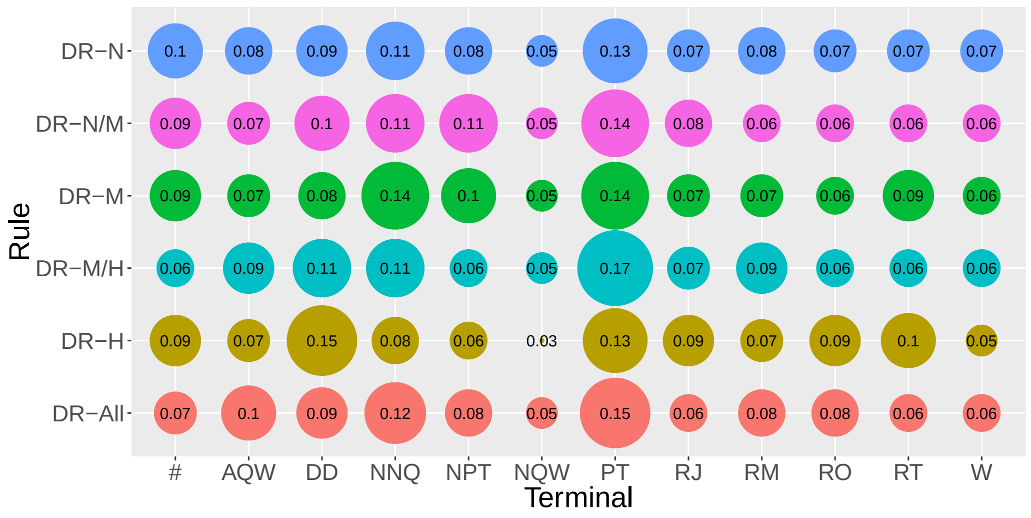


Figure 4.3: The proportion of terminals used by the sets of rules evolved by the GP-HH approaches.

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 have a higher proportion of due date terminal compared to the other sets of evolved rules. This is likely due to the fact that in problem instances with high machine breakdown level (e.g. $\pi = 0.05$) processing time becomes unreliable in determining on predicting the expected duration of time that a job requires on the machine for the operation to complete. Compared to problem instances with lower machine breakdown levels (e.g. $\pi = 0$ and $\pi = 0.025$), it is more likely in the problem instances 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 may be more reliable method of generating good schedules for problem instances with high level of machine breakdowns. Therefore, this may potentially result in individuals that use high proportion of the processing time terminal, that prioritise processing shorter job processing times, generating worse schedules for the problem instances than individuals that use high proportion of the due date terminal, that prioritise processing urgent jobs. 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 [51].

This paper investigates the generality of a standard GP-HH approach to a DJSS problem subject to dynamic job arrivals and machine breakdowns. This is done by first developing a DJSS dataset for evaluating GP-HH approaches. Afterwards, a standard GP-HH approach that evolves priority-based dispatching rules is applied to a new dataset for generating DJSS problem instances. Finally, the distributions of the terminals in the GP rules evolved from different machine breakdown scenarios are analysed. From the performance results and the results of analysing the terminal distributions, this paper makes the following findings:

(a) GP-HH approaches in the literature have been shown to be generalise well over different problem domains (including JSS) [17, 11]. However, for the DJSS problem with dynamic job arrivals and machine breakdowns, the results show that a standard GP-HH approach is sensitive to the level of machine breakdowns. In addition, a generalist set of rules evolved by a standard GP-HH approach is unable to cover for high level of machine breakdowns.

(b) Analysis of the distribution of the terminals for the evolved rules show that there are higher proportions of DD terminal and lower proportion of NNQ, NPT, NQW and PT terminals in rules evolved on training instances with high levels of machine breakdowns compared to the other evolved rules. This is likely due to the added uncertainty associated with the duration of time required to process a job.

For future work, a GP-HH approach that incorporates terminals that use machine breakdown specific attributes could potentially evolve rules that can outperform rules evolved by a standard GP-HH approach and improve the generality of GP rules over different machine breakdown scenarios. For example, it is likely that incorporating terminals such as the next time the machine is expected to break down and the expected true processing time may generate rules which perform well over both DJSS problem instances with and without machine breakdowns.

4.4 Developing Machine Breakdown Specific GP Terminals

Job shop scheduling (JSS) problems are combinatorial optimisation problems that have been studied over the past 60 years [102]. Due to their direct application in important real-world manufacturing systems, extensive research has been carried out for JSS problems to find effective and practical techniques which may be incorporated to a real-world scenario for the manufacturers so that they gain a competitive edge in the respective markets [102]. In a JSS problem instance, there are *machines* on the *shop floor* that are used to process arriving *jobs*, and the manufacturer needs to make intelligent decisions to process the jobs as effectively as possible. In other words, machine resources need to be optimally allocated (given a specific criterion) by determining the *sequence* in which the jobs are processed. However, optimal allocation of machines can be a difficult task. Most job shop scheduling (JSS) problems are NP-hard [102], and mathematical optimisation techniques that return optimal solutions for problem instances do not scale effectively with the problem size. In addition, in a dynamic JSS problem instance there are unforeseen events that affect the properties of the shop floor, e.g., dynamic job arrivals and machine breakdowns [90]. To handle dynamic JSS problems, various heuristic approaches have been proposed to generate good solutions to problem instances while coping with the unforeseen events. For this paper, we handle dynamic JSS problems with dynamic job arrivals, where the jobs' properties and their arrival times are unknown until the job arrival times are reached during processing [46]. Dispatching rule approaches are the most prominent method of handling dynamic JSS problems with dynamic job arrivals due to their short reaction times and their ability to cope with the dynamic environment [86].

In addition to manually designing effective dispatching rules for dynamic JSS problems with dynamic job arrivals, researchers have proposed various genetic programming based hyper-heuristic (GP-HH) approaches to automatically evolving dispatching rule from heuristic subcomponents [11]. GP evolved dispatching rules generally perform better than man-made dispatching rules for JSS problems [83]. However, GP approaches that have been proposed for dynamic JSS problems have mainly focused on dynamic job arrivals [86, 46, 11, 83, 51]. In a real-world scenario, it is likely that there are different types of dynamic events that occur during processing. An example is machine breakdown, where the machines need to be serviced and repaired [90]. It is likely that disruptions caused by machine breakdowns can likely impact the performance of the scheduling algorithm if they are not specifically accounted for. The only GP approach that explicitly accounts for machine breakdowns in the literature deals with a single machine JSS problem with no dynamic job arrivals [123]. Developing machine breakdown specific GP approaches may allow us to improve the overall quality of rules evolved by GP for dynamic JSS problems with dynamic job arrivals and machine breakdowns (DJSS-MB).

The goal of this paper is to develop new machine breakdown terminals for a GP terminal set commonly used in the literature [86, 51] to handle a DJSS-MB. By incorporating machine breakdown terminals into the GP terminal set, it may be possible to evolve rules that can account for machine breakdown information. 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, and generate better solutions overall. In other words, developing new machine breakdown terminals may allow GP to consistently evolve high quality rules for DJSS-MB. Afterwards, by analysing specific machine breakdown GP evolved rules, it may be possible to develop an insight into how the rules behave in DJSS-MB, allowing us to potentially develop more effective machine breakdown GP approaches in the future. Overall, this paper carries out the following objectives:

(a) Develop and evaluate new machine breakdown terminals for an existing GP approach [86, 51].

(b) Carry out a structural analysis of the machine breakdown GP rules to gain an understanding of the useful features and properties of GP rules that are evolved under machine breakdown.

First, we cover the background to dynamic JSS in Section **??**, which includes the problem definition and outlines existing GP approaches for dynamic JSS problems. Afterwards, Section **??** describes the existing GP approach used in the literature [86, 51], the benchmark GP terminals, and the machine breakdown GP terminals investigated in this paper. Section **??** describes the dynamic JSS simulation model used in this paper, and the GP parameters. Finally, Section **??** gives the results and an analysis of the findings, and Section **??** gives the concluding remarks and the future works.

As machine breakdown GP approach for DJSS-MB have not been proposed, this paper proposes simple but novel machine breakdown terminals which are incorporated to a GP approach. 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 GP representation, the benchmark terminal and function sets. 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 "react" to the machine breakdowns happening on the shop floor, to the existing set of GP terminals.

4.4.1 GP Representation, Terminal Set and Function Set

For this paper, we use a tree-based GP representation [67]. The GP individuals represent arithmetic function trees that calculate the priorities of jobs during decision situations. Arithmetic GP representation has been used prominently in the literature to evolve effective priority dispatching rules for JSS problems [11]. A GP terminal corresponds to a job, machine and shop floor attribute value at a decision situation, or combines multiple base level shop floor attributes as a part of the terminal. For example, the RT terminal returns the sum remaining total processing times of job j waiting at a machine to process operation o_{ij}, i.e., RT$(j) = \sum_{k=i}^{N_j} p_{kj}$. The GP terminals and the arithemtic operators used by the benchmark GP approach are listed in Table 5.1, which is based off existing terminal and function sets used by GP approaches to dynamic JSS problems in the literature [51, 94]. The function set consists of the arithmetic operators $+$, $-$, $\times$, protected /, binary operators $\max$, $\min$ and a ternary operator `if`. The protected / returns one if the denominator is zero, and carries out the standard division operator otherwise. $\max$ and $\min$ returns the maximum and the minimum value of the two arguments respectively. `if` returns the value of the second argument if the value of the first argument is greater than or equal to zero, or returns the value of the third argument otherwise.

A GP individual x is evaluated over a set of dynamic JSS training instances to calculate its fitness as follows. GP individual x is applied to a JSS problem instance I as a *non-delay* dispatching rule [99]. During a simulation when a machine m becomes available, the jobs that are currently waiting at machine m are assigned priority values by the dispatching rule. The job that has the highest priority value is selected to be processed at machine m. This continues until the termination criteria for the simulation has been reached (e.g. after a certain number of jobs have been completed), at which point the objective function value $Obj(x, I)$ is calculated from the solution generated by individual x. The individual x is applied to all problem instances in the training sets, obtaining objective values $Obj(x, I_1), \ldots, Obj(x, I_{T_{train}})$.

After the GP individual x is applied to the problem instances in the training set, the objective values obtained are normalised to reduce the likelihood that the GP individuals are biased towards specific instances

Table 4.2: Terminal set for GP, where a job j is waiting at the available machine m at a decision situation.

Terminal	Description
RJ	operation ready time of job j
PT	operation processing time of job j
RO	remaining number of operations of job j
RT	remaining total processing times of job j
RM	machine m's ready time
WINQ	work in next queue for job j
DD	job's due date d_j
SL	slack of job j
W	job's weight w_j
NPT	next operation processing time of job j
NNQ	number of idle jobs waiting at the next machine
NQW	average waiting time of last 5 jobs at the next machine
AQW	average waiting time of last 5 jobs at all machines

[46, 79]. In other words, an objective value $Obj(x, I)$ for a solution generated by individual x for instance I is normalised using *reference* objective value $Obj(R, I)$ as shown in Equation (4.3). The reference objective value $Obj(R, I)$ is calculated from the solution generated by applying a *reference rule* R to the problem instance I. The reference rule used is the weighted apparent tardiness cost (wATC) rule [117], a man-made dispatching rule effective for weighted tardiness related objectives. This was also used by Park et al. [94] in the fitness function to evolve GP rules for the DJSS-MB.

$$Obj'(x, I) = \frac{Obj(x, I)}{Obj(R, I)} \tag{4.3}$$

4.4.2 Augmented GP Terminals

The following terminals in the original GP terminal set (as described in Table 5.1) 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). 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 'MB-' (e.g. MBPT for machine breakdown adjusted processing time). The 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) in Table 5.1. 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 by factoring 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 (4.4).

$$p'_{ij} = \begin{cases} p_{ij} & \text{if } t + p_{ij} < b_t^m \\ p_{ij} + r_t^m & \text{otherwise} \end{cases} \tag{4.4}$$

Machine breakdown adjusted next processing time (MBNPT):

The machine breakdown adjusted next processing time terminal (MBNPT) replaces the next processing time terminal (NPT) in Table 5.1. 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', the repair time of m' is added to the next processing time $p_{(i+1)j}$ if it is expected to be interrupted by a breakdown at machine m' at operation $o_{(i+1)j}$ earliest possible completion at machine m'. The earliest possible time that job j can be completed is if operation o_{ij} is selected immediately by machine m, and then the successive operation $o_{(i+1)j}$ is then processed by machine m' 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 (4.4)). In other words, the earliest time operation $o_{(i+1)j}$ can be processed at machine m' is at $t + p'_{ij}$ after operation o_{ij} is expected to complete. Therefore, if machine m' breaks down before time $t + p'_{ij} + p_{(i+1)j}$, then repair time $r_t^{m'}$ of machine m' is added to the operation $o_{(i+1)j}$'s processing time $p_{(i+1)j}$ as shown in Equation (4.5).

$$\text{MBNPT}(j) = \begin{cases} p_{(i+1)j} & \text{if } t + p'_{ij} + p_{(i+1)j} < b_t^{m'} \\ p_{(i+1)j} + r_t^{m'} & \text{otherwise} \end{cases} \tag{4.5}$$

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) in Table 5.1. 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' is required by operation $o_{(i+1)j}$, the standard WINQ terminal returns the sum processing times of the jobs that are currently waiting at machine m' plus the remaining time required to process the operation currently being processed by machine m', i.e., the work remaining. MBWINQ modifies the work remaining time calculated by WINQ, and adds the machine m''s repair time if the work is interrupted by machine breakdown at time $b_t^{m'}$. In other words, MBWINQ$(j) = wr'_{m',j'} + \sum_{i=1}^{N_{m'}} p_{j_i}$, where $p_{j_1}, \ldots, p_{j_{N_{m'}}}$ are the processing times of jobs waiting at machine m', $wr'_{m'j'}$ is the actual work remaining required on j' being processed on machine m' before it becomes available. The calculation for actual work remaining $wr'_{m'j'}$ is given in Equation (4.6), where $s_{j'}$ denotes the time when j' started, $p_{j'}$ is the processing time required by j' at machine m' and t is the current time.

$$wr'_{m'j'} = \begin{cases} s_{j'} + p_{j'} - t & \text{if } s_{j'} + p_{j'} < b_t^{m'} \\ s_{j'} + p_{j'} - t + r_t^{m'} & \text{otherwise} \end{cases} \tag{4.6}$$

In summary, the augmented GP approach replaces three existing terminals (PT, NPT and WINQ) with equivalent terminals that incorporate information about future machine breakdowns. 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 a machines during decision situations.

4.4.3 Reactive GP Terminals

Reactive machine breakdown terminals are added to the GP terminal set described in Table 5.1 and incorporate information about current machine status. As the two terminals incorporate informations 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 time compared to jobs with high expected wait time. This may lead to jobs spending less time waiting at busy machines, and the evolved rules may generate higher quality schedules. The GP approach that incorporates the RTR terminal is denoted as GP-RTR, and the 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' visited by j is currently not broken down. Otherwise, given that machine m' broke down at time $b_t^{m'}$ and the repair time is $r_t^{m'}$, the value given by RTR $= b_t^{m'} + r_t^{m'} - t$.

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' 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' to be available. If machine m' is currently processing a job j' or is broken down with an interrupted job, then it returns the actual work remaining $wr'_{m'j'}$ which is given in Equation (4.6). Otherwise, if the m' is broken down and a job was not interrupted by the machine breakdown, WT returns the remaining repair time of machine m' as given by the terminal RTR.

4.5 Experimental Design

This section describes the setup used to evaluate the different GP approaches to tackle the DJSS-MB. To evaluate the machine breakdown GP approaches, a simulation model that is slightly modified from existing simulation models in the literature [47, 94] is used to both evolve and evaluate the evolved rules. Afterwards, we provide the parameter used by the GP approaches.

4.5.1 Dynamic Simulation Model with Machine Breakdown

Discrete-event simulations are the most prominent method of generating dynamic JSS problem instances [11, 83]. In a discrete-event simulation, the dynamic events such as the job arrivals and the machine breakdowns are stochastically generated from a set of parameters. A simulation configuration is the set of parameters required, along with a seed value, to generate a dynamic JSS problem instance. In a dynamic JSS problem instance, there are $M = 10$ machines on the shop floor. The problem instance has a "warm-up" period of 500 jobs, where the first 500 jobs completed do not contribute towards the objective. The simulation is terminated after 2500 jobs are completed, and the objective function value is calculated from the 2000 jobs completed after the warm-up phase. The job arrivals times follow a Poisson process with arrival rate $\lambda = \rho \times \mu \times p_M$. In the equation, ρ is the utilisation rate, μ the mean processing time, and p_M the mean number of job operations to machine ratio. Utilisation rate (ρ) is the expected proportion of time the machines are spent processing job operations, and $\rho = 80\%$ for all problem instances. If the utilisation rate plus the machine breakdown level is too high, it is very likely that the shop will be *unstable* [99], i.e., job arrival rate is greater than the rate at which the shop floor can process them. Therefore, 80% utilisation rate is used instead of higher utilisation rates used in the literature (e.g. 90% or 95% [47, 86]) to accommodate for the high level of machine breakdown used by the simulation model (described below). The mean processing time (μ) is used in a uniform distribution with the interval $[1, 2\mu - 1]$ that the jobs' processing times follow, and $\mu = 25$ for all problem instances. The mean number of job operations to machine ratio (p_M) is the expected number of machines that a job will visit divided by the total number of machines. The number of operations a job has follows a uniform distribution in the interval $[2, 10]$, i.e., the minimum and the maximum number of operations that a job can have is 2 and 10 respectively. Therefore, the expected number of operations is 6 for a job arriving on the shop floor and $p_M = 0.6$ for all problem instances. In addition, a job's weight has the value of 1, 2 or 4 with 20%, 60% or 20% probabilities respectively. Given a job j's arrival time r_j, total processing time $\sum_{i=1}^{N_j} p_{ij}$ and the due date tightness simulation parameter h, the due date of job j is $d_j = r_j + h \times \sum_{i=1}^{N_j} p_{ij}$.

For generating machine breakdown events, the inter-breakdown times of the machines follow an exponential distribution, and the expected breakdown rate is given by $\eta = r_t/\pi - r_t$. In the equation, π is the breakdown level and r_t is the machine repair time. The breakdown level is the expected proportion of time for the machine to be broken down over the course of the simulation, and varies between the different simulation configurations used to generate the problem instances. For a problem instance the machine repair time is the same across all machines for a problem instance.

The dataset parameters for generating job arrivals and machine breakdowns are shown in Table 4.3. First, the simulation configurations have the possible breakdown levels $\pi = 0\%$, 2.5%, 5%, 10%, or 15% for a simulation configuration. Second, fixed repair times for the machine breakdowns are either $r_t = 37.5$, 137.5, or 262.5 for a simulation configuration. These parameters were selected after running the benchmark GP approach on different breakdown levels and durations of repair times as part of a preliminary experiment. The simulation configuration consists of a combination of the dataset parameters, which means that there are $3 \times 5 \times 2 = 30$ different simulation configurations available in the dataset. We use the simulation configuration with $\pi = 15\%$, $r_t = 262.5$ and $h = 3$ to generate the training problem instances. In addition, a different seed is used with the training simulation configuration every generation during the GP

Table 4.3: Dynamic JSS Parameter Settings

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

Table 4.4: GP Parameter Settings

GP Parameter	Value
Population size	1024
Number of generations	51
Crossover rate	80%
Mutation rate	10%
Reproduction rate	10%
Max initial depth	2
Max depth	8
Initialisation method	Ramped half-and-half
Selection method	Tournament selection
Selection size	2

process, resulting in different dynamic JSS problem instances being used every generation.

4.5.2 GP Parameters

The GP parameters are used by the GP approaches are shown in Table 4.4. The GP parameters are the same as the parameters that are same as the ones used by Park et al. [94] in their investigation into GP approaches for a DJSS-MB, which allows our benchmark GP approach to be consistent with the GP approach that was used during their investigation.

4.6 Experimental Results

In this investigation, we first carry out a performance evaluation of the GP approaches. The performance evaluation first compares the GP approaches and how consistently they can evolve high quality dispatching rules for the dynamic JSS problem, i.e., measures the effectiveness of the GP approaches. This is done by evolving a set of rules for each approach and applying them over the dynamic JSS simulation model. Afterwards, the best rules are extracted from the sets of evolved rules for the GP approaches and compared individually to determine whether an individual machine breakdown rule can outperform an evolved rule generated by the benchmark GP approach. Finally, we carry out a structural analysis of the best rules evolved by the machine breakdown GP approaches to find out the useful properties from the evolve rules.

4.6.1 Performance Evaluation

For the performance evaluation, each GP approach is applied to a training set (described in Section **??**) thirty times to evolve thirty independent rules. Afterwards, each of the rule is applied to the dynamic JSS simulation model as follows.

Performance Measure:

First, an evolved rule x is run multiple times over each simulation configuration in the simulation model. A single run consists of a seed value and a simulation configuration, which are used to generate a dynamic JSS problem instance. The rule is then applied to the problem instance and generates a schedule, which has a MWT objective value. Afterwards, the subsequent runs over the simulation configuration use unique seeds so that new problem instances are generated from the same simulation configuration. In other words, given a simulation configuration sim and rule x, sched-

Table 4.5: Comparison of the rule sets evolved by the GP approaches over the simulation configurations. Rules are evolved from $\langle 0.15, 262.5, 3 \rangle$.

ules with MWT values $Obj(x, I_{(sim)1})$, ..., $Obj(x, I_{(sim)30})$ are generated by the rule for the given simulation configuration. These are used slightly differently for the rule set evaluation and best rule evaluation, which are described below.

Rule Set Results:

In the rule set evaluation, the MWT values $Obj(x, I_{(sim)1})$, ..., $Obj(x, I_{(sim)30})$ generated by a rule x for a simulation configuration sim is averaged out to obtain the "performance" $Perf$ of the rule over the simulation configuration, i.e., $Perf(x) = \frac{1}{30}\sum_{i=1}^{30} Obj(x, I_{(sim)i})$. The rule performances are then used to compare between the different sets of rules evolved by the GP approaches.

The results of the performance evaluation is shown in Table 4.5. 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 $\mu \pm \sigma$ is the mean (μ) and standard deviation (σ) of the performance $Perf$ of the rules for the simulation configuration respectively. If a set of GP evolved rules that use the machine breakdown terminals is *significantly* better than the set of benchmark GP rules for a simulation configuration by satisfying the two sided Student's t-test at $p = 0.05$, then the particular entry is highlighted.

Although the differences are not significant, the results show that the three machine breakdown approaches (GP-Aug, GP-WT and GP-RTR) have slightly better performances than the benchmark GP for 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 configura-

Table 4.6: Comparison of the best rules over the simulation configurations.

tions $\langle 0, 37.5, 5\rangle$and $\langle 0.15, 37.5, 5\rangle$. Finally, the results of the comparison between the GP-Aug rules and the benchmark GP rules is most mixed, where GP-Aug rules are slightly better or worse than the benchmark rules on roughly the equal number of simulation configurations. However, due to the lack of statistical significance, no significant conclusions can be drawn on whether the machine breakdown GP approaches is more consistent in evolving higher quality dispatching rules than the benchmark GP approach. However, by analysing the rules further, it may be possible to gain a better understanding of how GP can be applied effectively to the machine breakdown problem.

Best Rule Results:

The best rule from each GP approach are compared against each other after the performances of the rules are compared. The best rule is defined to be the rule that has the lowest average performance values over all the simulation configurations out of the evolved rules. The best rules are then compared on each simulation configuration by the MWT values from the generated schedules. In other words, best rule comparison uses the results $Obj(x, I_{(sim)1})$, ..., $Obj(x, I_{(sim)30})$ from the rules being applied to the 30 problem instances generated by each simulation configuration sim. The results of the best rules being applied to each simulation configuration is shown in Table 4.6, where each entry $\mu \pm \sigma$ is the mean (μ) and standard deviation (σ) of the MWT values generated by the best rule after being applied to 30 independent problem instances generated from the simulation configuration.

The best rules from the machine breakdown GP approaches show 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 $\langle 0.15, 262.5, 3\rangle$simulation configuration. Therefore, it is likely for GP approaches with the machine breakdown terminals to evolve high quality individual rules than the benchmark GP approach.

4.6.2 Rule Analysis

The evolved machine breakdown GP rules are analysed further by carrying out a qualitative analysis based on the structures of the evolved rules. First, 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 Fig. 4.4a, 4.4b, and 4.4c respectively.

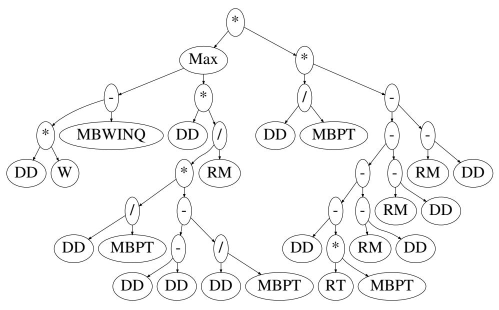
An important observation from the best rules evolved by GP-WT and GP-RTR is the lack of machine breakdown terminals that make up the best rules. The best rule from GP-WT has *no* occurrence of the WT terminal that is incorporated into the terminal set, and the best rule from GP-RTR has *one* occurrence of the RTR terminal. Therefore, it may be the case that the machine breakdown terminals do not directly contribute towards the qualities of the final evolved rules. Instead, the machine breakdown terminals may facilitate the evolution of good GP rules, and are discarded from the best GP individuals near the end of the GP process. This may explain the lack of machine breakdown terminals in best GP-WT and GP-RTR rules, but why the best rules generally perform better than the best benchmark rule In addition, it may also explain why GP-WT and GP-RTR rules also perform slightly better than the benchmark GP 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 evolved rules. Intuitively, important jobs with short processing time should be prioritised out of the jobs waiting at the available machine. 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 date (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 steady 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 remains relatively the same over the whole duration of the simulation. Therefore, the relative differences in the due date between an urgent job and a non-urgent job waiting on a machine late in the simulation may not be as big as the differences in their processing time, due to the large due date values of both the urgent and non-urgent jobs. This may result in the due date of a job for long simulations being used as an arbitrary large value that can be combined with the processing time terminal using the protected / operator to form a composite that prioritises short processing times. Further experiments can be carried out to determine whether the same phenomenon occurs by replacing the processing time terminal with 1/PT terminal in future GP approaches.

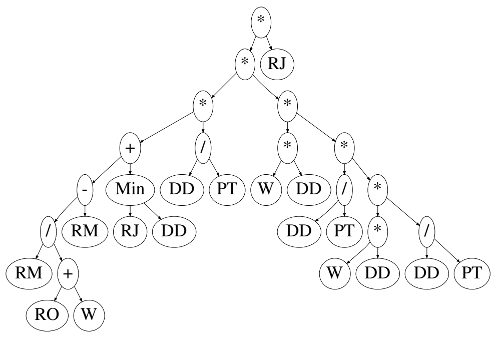
4.7 Conclusions and Future Work

This paper is a very first piece of work that develops new machine breakdown GP terminals to improve the qualities of GP evolved rules for a DJSS-MB. 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 gives information on current state of the shop floor. The machine breakdown GP approach does not evolve significantly better rules overall, but the best rules evolved by the machine breakdown GP significantly outperform the best rule evolved by the benchmark GP. The analysis shows very interesting results and insights, where the machine breakdown terminals appear infrequently in the best rules for GP-WT and GP-RTR. Hypotheses have been raised to explain why this is the case, and further work will be needed in this direction. We hope that this work can attract more people to start their work in this direction in the near future.

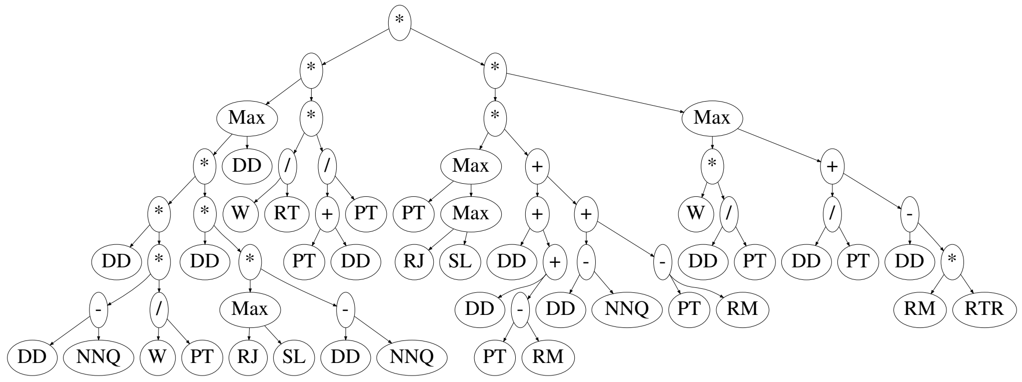
For the future work, further analysis based on the behaviours of the evolved rules will be carried out. Analysis of evolved rule behaviours in JSS problems have been carried out in the literature [45, 43], and further investigation into the behaviours of rules evolved for DJSS-MB may allow better machine breakdown specific approaches to be developed. In addition, the relation between the utilisation rate of job shop scheduling problems and the machine breakdown level will be explored further by analysing rule behaviours in different shop environments. For example, a rule evolved for shop with low utilisation rate and high machine breakdown will be compared against a rule evolved for shop with high utilisation rate and low machine breakdown. This relation may help us develop further insight into machine breakdowns and how the properties of the shop changes with such disruptions.



(a) Best GP-Aug rule



(b) Best GP-WT rule


*
* *
Max * * Max
* DD / / Max + * +
* * W RT + PT PT Max + + W / / -
DD * DD * PT DD RJ SL DD + - - DD PT DD PT DD *
- / Max - DD - DD NNQ PT RM RM RTR
DD NNQ W PT RJ SL DD NNQ PT RM

(c) Best GP-RTR rule

Figure 4.4: The structures of the best rules found by the GP approaches.

Chapter 5

Developing Multitask GP-HH Approaches that evolves a Generalist Rule and a Portfolio of Rules

The goal of this paper is to develop a novel machine breakdown specific GP approach that is able to cope with a DJSS problem with dynamic job arrivals and various severity of machine breakdowns that occur during processing. To do this, we propose a niched GP approach that evolves multiple specialist rules simultaneously for the DJSS problem to handle the different machine breakdown scenarios and a generalist rule for all machine breakdown scenarios. Compared to a standard GP approach that evolves specialist rules for the different machine breakdown scenarios, a niched GP approach that evolves specialist rules simultaneously has the potential to significantly save on the computation cost of evolving effective rules for machine breakdown scenarios while being competitive with the specialist rules evolved by the standard GP approach. In addition, by analysing the rules evolved by the niched GP approach, e.g., by determining the behaviours of different specialist rules, we can observe the overlap between the machine breakdown scenarios based on the behaviours of the evolved specialist rules.

5.1 Niched GP Approach to Handling Dynamic JSS Problems

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

5.1.1 Niched GP Process

The niched GP approach is modified from a niched GP approach proposed by Mei et al. [78] used to evolve a diverse set of rules. The niched GP keeps track of the GP individuals that are the best for the different machine breakdown scenarios during training. This may help guide the GP process as it explores the search space for effective specialist rules.

For the niched GP process, the GP individuals are first randomly initialised and the set of specialist rule $\mathcal{S}$ is empty. For each generation when a GP individual x is being evaluated, the individual is first evaluated to the "general" training set $\mathcal{T}$ to calculate its fitness $f(x)$. The problem instances in training set $\mathcal{T}$ consist of N machine breakdown scenarios. In addition, if an individual x has the best performance for problem instances in niche n (i.e. under a specific machine breakdown scenario) in the training set $\mathcal{T}$, then current generation niche individual t_I for niche n is updated to individual x. After all GP individuals in the current population have been evaluated, the current generation niched individuals $t_1, \ldots, t_N$ are then compared against the overall niched individuals $s_1, \ldots, s_N$. To compare the current generation niched individuals t_n to overall niched individual

s_n, the individuals are applied 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. If the $f'(t_n)$ of the current generation niched individual t_n is better than the fitness $f'(s_n)$ of the overall niched individual s_n over the niched training set $\mathcal{V}_n$, then s_n is updated to the current generation's niche individual t_n. Otherwise, the individual t_n is kept the same. After the set of niched individuals has been updated, the clearing algorithm `Clearing`$(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ is carried out before the individuals undergo the standard tournament selection procedure. The clearing algorithm is adapted from the algorithm used by Mei et al. [78]. In the algorithm, the individuals with poor performances within distance σ from either a specialist GP rule from the rule set $\mathcal{S}$ or a generalist rule g are removed from the GP population if the niche has reached its capacity κ. This continues until the maximum number of generations has been reached, where output is the best overall rule is the generalist rule g and the set of specialist rules $\mathcal{S}$. The pseudocode that summarises the niched GP process is shown in Algorithm 3.

The niched GP approach likely a greater computation time than a standard GP approach that uses a single population because it evaluates the niched individuals on the niched training sets on top of the standard evaluation procedure. When evolving dispatching rules for DJSS problems using GP, the evaluation procedure and the application of the individuals on the training instances is the most computationally intensive step of the GP process [84]. This means that the niched GP approach will have additional $\mathcal{O}(N|\mathcal{V}|)$ computation time compared to a standard GP approach.

5.1.2 GP Representation, Terminal Set and Function Set

The GP representation, terminals and function sets are adapted from the GP approach used by Park et al. [94] to investigate the DJSS problem with dynamic job arrivals and machine breakdowns. For the niched GP approach, the GP individuals are arithmetic function trees that are used to calculate the priorities of jobs waiting at an available machine m^* during a decision situation. The terminals listed in Table 5.1 for a GP individual's tree correspond to job, machine and shop floor attributes. The non-terminals consist of arithmetic operators $+$, $-$, $\times$, protected /, binary operators $\max$, $\min$ and a ternary operator `if`. Protected / returns 1 if the denominator is zero, and returns the output of a standard division operator otherwise. `if` operator returns the value of the second child branch (representing the "then" condition) if the first child branch (representing the input into the "if" condition) is greater than or equal to zero, but returns the value of the third child branch (representing the "else" condition) otherwise.

5.1.3 Evaluation Procedure

The GP individual x is applied to the DJSS problem instances in the training sets as a *non-delay* [99] dispatching rule. The individual x is applied to a problem instance I to generate a schedule. Afterwards, the MWT value $Obj(x, I)$ of the schedule is normalised using a *reference rule* to reduce bias towards specific DJSS problem instances [45]. The reference rule R, which is the weighted apparent tardiness cost (wATC) rule [99], is applied to problem instance I to generate a schedule with $Obj(R, I)$. Afterwards, the normalised MWT value is calculated as $Obj'(x, I) = \frac{Obj(x,I)}{Obj(R,I)}$. From the normalised objective values, the fitness of individual x is given by $f(x) = \sum_{I \in \mathcal{T}} Obj'(x, I)$ after the individual has been applied to all problem instances in the training set $\mathcal{T}$.

5.2 Neighbourhood-based GP Approach to Handling Dynamic JSS Problems

5.3 Experimental Design

This section describes the simulation model used to evolve and evaluate the specialised rules for the niched GP approach, followed by a description of baseline GP used for comparison during evaluation. Afterwards, the details for GP and niching parameters are provided.

5.3.1 Dynamic JSS Simulation Model

Discrete-event simulations are the standard method of simulating job shop scheduling problem instances [11]. A discrete-event simulation stochastically generates the dynamic events, i.e., the job arrivals and the machine breakdowns. The simulation model is adapted from the simulation model used by Holthaus [47] and Park et al. [94].

In the simulation model proposed by Holthaus [47], the machine breakdowns are generated stochastically from two simulation parameters: machine breakdown level π and mean repair time r. The breakdown times of the machines follow an exponential distribution with breakdown rate η, and the time required to repair the broken down machines follow an exponential distribution with mean repair time r. The breakdown rate is calculated from the machine breakdown level π, which is the proportion of time during processing that the machines are expected to be broken down, and the mean repair time r. From breakdown level and repair time, the breakdown rate is given by $\eta = r/\pi - r$. There are three different values for π, three different values for r and two different values for the due date tightness h. Due date tightness is a simulation parameter used to determine how the due date is generated for a job arrival [47]. Since the repair times are not a factor when the breakdown level is zero, i.e., there is no machine breakdown, there are problem instances can be generated under $2 \times 3 \times 3 = 14$ different scenarios. These parameters are listed below in Table 5.2.

At each generation, the training set $\mathcal{T}$ simulates a DJSS problem instance using each simulation configuration scenarios (i.e. different combinations of due date tightness, breakdown level and mean repair time), resulting in a GP individual being applied to 14 DJSS problem instances. The simulations used in training set $\mathcal{T}$ are grouped up into the seven groups of two problem instances based on their breakdown level and mean repair time, e.g., a group with breakdown level $\pi = 2.5\%$ and mean repair time $r = 25$ is denoted as $\langle 2.5\%, 25 \rangle$. The group with breakdown level $\pi = 0\%$ is simply denoted as $\langle 0 \rangle$. In other words, the seven groups are the "niches" that are filled up by GP individuals that perform the best for the different machine breakdown level and mean repair time parameter values (i.e. $N = 7$). The seed used to simulate the problem instances from the simulation configurations are rotated every generation to help improve the generalisation ability of the evolved rules [78].

After the current generation niched individuals have been found, they are applied to the niched training sets to update the set of specialist GP rules $\mathcal{S}$. A niched individual $t_{\langle \pi, r \rangle}$ from the current generation for the scenario $\langle \pi, r \rangle$ is applied to the niched training set $\mathcal{V}_{\langle \pi, r \rangle}$. The niched training set $\mathcal{V}_{\langle \pi, r \rangle}$ has the configurations with due date tightness $h = 3$ or $h = 5$. For each due date tightness, the niched individual is applied to a simulation five times with different seeds, resulting in the individual being applied to the ten different simulation runs overall in the niched training set. Finally, to ensure that the comparisons between the rules between different generations are kept consistent, the niched training sets are fixed over every generation for the niched GP process.

5.3.2 GP Benchmarks

To evaluate the niched GP's evolved rules, we use a standard single-tree GP representation [11, 84] with the same terminal and function set used by the niched GP for consistency (Table 5.1). Afterwards, the benchmark GP is applied to the machine breakdown scenarios independently to evolve the generalist and the specialist rules. The entire training set $\mathcal{T}$ is used to evolve the generalist rules from the benchmark GP approach. To evolve the specialist rules, instead of using the entire training set $\mathcal{T}$ described above, the benchmark GP only uses specific machine breakdown scenarios to evolve dispatching rules, e.g., a GP process is run with training instances being generated from $\langle 2.5\%, 25\rangle$. Since there are two possible due date tightness parameters, an individual in the benchmark GP process is applied to two training problem instances during the evaluation procedure. The best individual of the last generation before maximum number of generations is reached is the output dispatching rule for the benchmark GP process.

5.3.3 GP and Niching Parameters

The niched and the benchmark GP approaches follow parameters used by an existing GP approach for DJSS problems [94]. The GP population sizes are 1024, and the number of generations is 51. The crossover, mutation and reproduction rates are 80%, 10% and 10% respectively. The maximum depth of the individuals during initialisation is 2, and 8 during the duration of the GP process. Tournament selection of size 7 is used by the two GP approaches. For the clearing algorithm $\texttt{Clearing}(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ used by the niched GP approach, the two parameters niche radius σ and niche capacity κ is kept consistent as the parameters used by Mei et al., i.e., $\sigma = 1$ and $\kappa = 1$. Finally, $k = 3.0$ is used for the wATC reference rule.

5.4 Experimental Results

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

5.4.1 Computation Costs

The rules evolved by the niched and the benchmark GP approaches are evolved from a Java program ran on Intel(R) Core(TM) i7 CPU 3.60GHz. The time taken to evolve the rules is given in Table 5.3. In the table, "Specialist Total" denotes the sum of the times required to evolve the specialist rules with the benchmark GP approaches over the different machine breakdown scenarios. This is to compare the overall computation time required to evolve the specialist rules with the benchmark GP approach to the niched GP approach, as the niched GP approach evolves the generalist rule and the specialist rules simultaneously over a single run. "Total" denotes the sum of the time required to evolve the generalist and the specialist rules for the niched and the benchmark GP approaches. Since niched GP approach evolves the generalist and the specialist rules simultaneously, its time is only given in the "Total" category.

From the tables, we can see that the niched GP approach takes significantly less time, i.e., less than half the time required, to evolve all the rules than the benchmark GP approach. In addition, the niched GP approach also requires significantly less time ($\sim 27\%$ less time) to evolve the specialist rules than the total time required for benchmark GP approach.

In addition, the niched GP approach has a comparable computation time to evolving generalist rules using the benchmark GP approach. In other words, although it is expected that the niched GP approach is expected to have a greater computation time than the standard GP approach (as discussed in Section **??**), the results show that this difference is negligible.

The results also show that computation time for evolving rules differ based on the breakdown level and the mean repair time of the simulation models. As seen in the results, simulation models with no machine breakdowns or low mean repair time have low computation time required to evolve the rules. On the other hand, simulation models with mean repair time $r = 125$ generally have a higher computation cost than simulation models with mean repair time $r = 250$. For example, benchmark GP specialist rules takes around 6274 seconds (~ 1.74 hours) to evolve on machine breakdown scenario $\langle 2.5\%, 25\rangle$, which is significantly shorter than 47140 seconds (~ 13.09 hours) to evolve on machine breakdown scenario $\langle 2.5\%, 125\rangle$. A possible explanation is that for simulation models with $r = 125$ the time in between two consecutive machine breakdowns on the same machine may be long and frequent enough to cause an overall increase in the number of jobs that are waiting at the machines. Therefore, the priority function of the rule may likely need to calculate more job priority values in general for the simulation models with high machine breakdown level and mean repair time $r = 125$, leading to an overall increase in the computation time required to generate a schedule for a simulation model. In addition, although simulation models with $r = 250$ will have a longer durations of times when the machines are not available than simulation models with $r = 125$, the duration in between two consecutive machine breakdowns is also longer. This means that there is an opportunity for the backlog jobs to be processed during the times when the machines are not broken down.

5.4.2 Performance Comparison of Evolved Rules

The performances of a set of rules are calculated by applying the evolved rules to simulation models generated from the simulation configurations provided in Section **??**. An evolved rule is applied to DJSS simulation model to generate a schedule and get a MWT objective value. This is then repeated 30 times with different seeds for the simulation model to get an average performance of the evolved rule over the simulation configuration. The evolved generalist rules are applied to all simulation configurations, whereas the evolved specialist rules are applied to the machine breakdown scenarios they are designed for, e.g., NGP-$\langle 2.5\%, 25\rangle$is applied to simulation models with $\pi = 2.5\%$ and $r = 25$. The performances of the specialist and the generalist rules are given in Table 5.4. In the table, $\mu \pm \sigma$ for each set of rules denotes that the mean average performance is μ and the standard deviation is σ. In addition, $\langle \pi, r, h\rangle$ in the tables denotes that the particular simulation model has π breakdown level, r mean repair time and h due date tightness factor.

From the tables, we can see that for the specialist rules, there are only a few simulation configurations ($\langle 2.5\%, 25, 3\rangle$, $\langle 5\%, 250, 5\rangle$ and $\langle 5\%, 250, 3\rangle$) where the evolved niched rules perform slightly better than the evolved benchmark rules. In the other simulation configurations, the benchmark GP performs slightly better than the niched GP approach. However, the differences between the evolved rules are not statistically significant. Therefore, specialist niched rules are highly competitive in terms of their performance to the specialist benchmark rules despite the fact that they took less time to evolve. In addition, the generalist rules evolved by the niched GP approach also perform slightly better (but not significantly better) than the generalist rules evolved by the benchmark GP approach for most of the simulation configurations. Combined with the earlier result (in Section **??**) that the niched GP approach is significantly faster than evolving the specialist rules individually using the benchmark GP, the results show that the niched GP is more suitable than benchmark GP at handling the difficulty of DJSS problems with dynamic job arrivals and machine breakdowns. In other words, although the niched GP approach does not significantly outperform a standard GP approach it can evolve new competitive rules more quickly than the standard GP approach as unseen machine breakdown scenarios are encountered in the future.

5.4.3 Diversity Analysis

For the analysis procedure, the goal is to find differences between the simulation models that are in the different machine breakdown scenarios. To do this, we calculate the phenotypic distances between the rules evolved by the niched GP approaches using the job rank distance measure proposed by Hildebrandt and Branke [45] and used by the clearing algorithm `Clearing`$(\mathcal{P}, \mathcal{S}, \sigma, \kappa)$ [78]. The distances between a single rule in a rule set is compared against the 30 rules of another rule set to obtain an average distance of the single rule to the rule set. The means and the standard deviations of the average distances of the rule for the generalist and the specialist rule sets are shown in Table 5.5. In the tables, we highlight the average distances of two sets of rules as visual aids. The two sets of rules that have low mean average distances are highlighted red, whereas the two sets of rules that have high mean average distances are highlighted blue.

From the table, we can see that from the specialist rule comparisons the rules evolved on no machine breakdown ($\langle 0\%, 0 \rangle$) have low distances, i.e., strong similarities, to the rules evolved on $\pi = 2.5\%$ with mean repair times $r = 125$ and $r = 250$ ($\langle 2.5\%, 125 \rangle$ and $\langle 2.5\%, 250 \rangle$). In addition, the generalist rules behave similar to the specialist rules evolved on $\langle 0\%, 0 \rangle$, $\langle 2.5\%, 125 \rangle$ and $\langle 2.5\%, 250 \rangle$. Finally, the rules evolved on breakdown level $\pi = 5\%$ are more similar to the rules evolved on lower breakdown levels than the rules evolved on the same breakdown level with different mean repair times, e.g., rules evolved on $\langle 5\%, 25 \rangle$ have lower average distances to rules evolved on $\langle 0\%, 0\rangle$ than rules evolved on $\langle 5\%, 125\rangle$ or $\langle 5\%, 250\rangle$. The above observations likely indicate that DJSS scenarios with no machine breakdown or $\pi = 2.5\%$ have strong similarities to each other, and that the DJSS scenarios start having significantly different properties as the breakdown level increases (i.e. scenarios with $\pi = 5\%$).

5.5 Conclusions and Future Work

This paper proposes a novel niched GP approach to evolve effective dispatching rules for a DJSS problem with dynamic job arrivals and machine breakdowns. The proposed niched GP approach evolves multiple specialist rules for the different machine breakdown scenarios simultaneously which are competitive with a standard GP approach that are specifically trained on each machine breakdown scenario. By doing this, the niched GP approach significantly saves on computation cost required to evolve the rules. Overall, instead of improving the effectiveness and generality of standard GP approach on the DJSS problem, the niched GP approach provides an alternative solution of covering the different machine breakdown scenarios by evolving specialist rules that can handle the different machine breakdown scenarios effectively.

For the future work, it may be promising to investigate an approach that aggregates the specialist rules evolved by the niched GP approach, where the aggregated rule automatically determines which rule is best suited for a specific problem instance. This allows us to apply the specialist rules to DJSS problem instances where the machine breakdown scenario that the problem instances belong to are not known in advance. In addition, further adjustments to the niched GP approach may allow the rules evolved by the niched GP approach to outperform the rules evolved by the standard GP approach while being significantly faster to evolve. One possible example is to investigate niching techniques other than clearing, such as crowding and sequencing niching [97].

Algorithm 3: $\mathcal{S} \leftarrow$ `NichedGP`(G)

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

Table 5.1: Terminal set for GP, where a job j is waiting at the available machine m at a decision situation.

Terminal	Description
RJ	operation ready time of job j
PT	operation processing time of job j
RO	remaining number of operations of job j
RT	remaining total processing times of job j
RM	machine m's ready time
WINQ	work in next queue for job j
DD	job's due date d_j
SL	slack of job j
W	job's weight w_j
NPT	next operation processing time of job j
NNQ	number of idle jobs waiting at the next machine
NQW	average waiting time of last 5 jobs at the next machine
AQW	average waiting time of last 5 jobs at all machines

Table 5.2: 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 (w_j)	Random from 1, 2, 4 with probabilities 20%, 60%, 20%
	Due date tightness (h)	3.0 or 5.0
Machine breakdown parameters	Breakdown level (π)	0%, 2.5% or 5%
	Mean repair time (r)	25, 125 or 250

Table 5.3: Comparison of the computation time required to evolve the rules for the GP approaches.

Table 5.4: Comparison of the performances for the specialist and the generalist rules evolved by the niched and the benchmark GP approaches.

Table 5.5: Pairwise mean and standard deviations of the average distances between the rules evolved by the niched GP approach.

Chapter 6

Conclusions

Write the conclusions here.

6.1 Achieved Objectives

Will be written later down the line.

6.2 Main Conclusions

Will be written later down the line.

6.3 Discussion

Will be written later down the line.

6.4 Future Work

Will be written later down the line.

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] 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.

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

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

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

[15] BURKE, E. K., GENDREAU, M., HYDE, M., KENDALL, G., OCHOA, G., OZCAN, 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.

[16] BURKE, E. K., GUSTAFSON, S., AND KENDALL, G. Diversity in genetic programming: an analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation 8*, 1 (2004), 47–62.

[17] 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.

[18] BURKE, E. K., MCCOLLUM, B., MEISELS, A., PETROVIC, S., AND QU, R. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research 176*, 1 (2007), 177–192.

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

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

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling*. Addison-Wesley, 1967.

[27] 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.

[28] 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).

[29] De Jong, E. D., Watson, R. A., and Pollack, J. B. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the 2001 Conference on Genetic and Evolutionary Computation* (2001), pp. 11–18.

[30] De Jong, K. A. *Evolutionary Computation: A Unified Approach*. MIT press, 2006.

[31] 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.

[32] DELL'AMICO, M., AND TRUBIAN, M. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research 41*, 3 (1993), 231–252.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

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

[40] GOLDBERG, D. E., AND RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms* (1987), pp. 41–49.

[41] 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.

[42] 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.

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

[44] 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.

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

[46] 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.

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

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

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

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

[51] 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.

[52] 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.

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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] JIN, Y. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing 9*, 1 (2005), 3–12.

[59] JIN, Y. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation 1*, 2 (2011), 61–70.

[60] JIN, Y., AND SENDHOFF, B. Pareto-based multiobjective machine learning: An overview and case studies. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 38*, 3 (2008), 397–415.

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

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

[63] 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.

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

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

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

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

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

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

[70] 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.

[71] 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.

[72] 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.

[73] LEE, Y. H., BHASKARAN, K., AND PINEDO, M. A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions 29*, 1 (1997), 45–52.

[74] LIU, Y., AND YAO, X. Negatively correlated neural networks can produce best ensembles. *Australian Journal of Intelligent Information Processing Systems 4*, 3–4 (1997), 176–185.

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

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

[77] 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.

[78] 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.

[79] 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.

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

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

[82] 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.

[83] 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.

[84] 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.

[85] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A co-evolution 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.

[86] 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.

[87] 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.

[88] 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.

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

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

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

[92] 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.

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

[94] PARK, J., MEI, Y., CHEN, A., AND ZHANG, M. Investigating the generality of genetic programming based hyper-heuristic approach to dynamic job shop scheduling with machine breakdown. In *Proceedings of the 2017 Australasian Conference on Artificial Life and Computational Intelligence* (2017), vol. 10142 of *Lecture Notes in Artificial Intelligence*, Springer International Publishing, pp. 301–313.

[95] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., JOHNSTON, M., AND ZHANG, M. Genetic programming based hyper-heuristics to dynamic job shop scheduling: Cooperative coevolutionary approaches. In *Proceedings of 19th European Conference on Genetic Programming (EuroGP 2016)* (April 2016), vol. 9594 of *Lecture Notes in Computer Science*, Springer International Publishing, pp. 112–127.

[96] 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.

[97] 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.

[98] 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.

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

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

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

[102] 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.

[103] 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.

[104] 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.

[105] Russell, S. J., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 2 ed. Pearson Education, 2003.

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

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

[108] 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.

[109] 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.

[110] SUBRAMANIAM, V., LEE, G. K., RAMESH, T., HONG, G. S., AND WONG, Y. S. Machine selection rules in a dynamic job shop. *The International Journal of Advanced Manufacturing Technology 16*, 12 (2000), 902–908.

[111] SURESH, V., AND CHAUDHURI, D. Dynamic scheduling – a survey of research. *International Journal of Production Economics 32*, 1 (1993), 53–63.

[112] TAILLARD, E. Benchmarks for basic scheduling problems. *European Journal of Operational Research 64*, 2 (1993), 278–285.

[113] 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.

[114] 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.

[115] VALLADA, E., AND RUIZ, R. Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega 38*, 1–2 (2010), 57–67.

[116] 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.

[117] VEPSALAINEN, A. P. J., AND MORTON, T. E. Priority rules for job shops with weighted tardiness costs. *Management Science 33*, 8 (1987), 1035–1047.

[118] WIDMER, G., AND KUBAT, M. Learning in the presence of concept drift and hidden contexts. *Machine Learning 23*, 1 (1996), 69–101.

[119] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation 1*, 1 (1997), 67–82.

[120] 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.

[121] 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.

[122] 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.

[123] 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.

[124] 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.

[125] 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.

[126] 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.