Automatic Design of Dispatching Rules for Job Shop Scheduling with Genetic Programming

by

Su Nguyen

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
2013

Abstract

Scheduling is an important planning activity in manufacturing systems to help optimise the usage of scarce resources and improve the customer satisfaction. In the job shop manufacturing environment, scheduling problems are challenging due to the complexity of production flows and practical requirements such as dynamic changes, uncertainty, multiple objectives, and multiple scheduling decisions. Also, job shop scheduling (JSS) is very common in small manufacturing businesses and JSS is considered one of the most popular research topics in this domain due to its potential to dramatically decrease the costs and increase the throughput.

Practitioners and researchers have applied different computational techniques, from different fields such as operations research and computer science, to deal with JSS problems. Although optimisation methods usually show their dominance in the literature, applying optimisation techniques in practical situations is not straightforward because of the practical constraints and conditions in the shop. Dispatching rules are a very useful approach to dealing with these environments because they are easy to implement (by computers and shop floor operators) and can cope with dynamic changes. However, designing an effective dispatching rule is not a trivial task and requires extensive knowledge about the scheduling problem.

The overall goal of this thesis is to develop a genetic programming based hyper-heuristic (GPHH) approach for automatic heuristic design of reusable and competitive dispatching rules in job shop scheduling environments. This thesis focuses on incorporating special features of JSS in the representations and evolutionary search mechanisms of genetic programming (GP) to help enhance the quality of dispatching rules obtained.

This thesis shows that representations and evaluation schemes are the important factors that significantly influence the performance of GP for evolving dispatching rules. The thesis demonstrates that evolved rules which are trained to adapt their decisions based on the changes in shops are better than conventional rules. Moreover, by applying a new evaluation scheme, the evolved rules can effectively learn from the mistakes made in previous completed schedules to construct better scheduling decisions. The GP method using the new proposed evaluation scheme shows better performance than the GP method using the conventional scheme.

This thesis proposes a new multi-objective GPHH to evolve a Pareto front of non-dominated dispatching rules. Instead of evolving a single rule with assumed preferences over different objectives, the advantage of this GPHH method is to allow GP to evolve rules to handle multiple conflicting objectives simultaneously. The Pareto fronts obtained by the GPHH method can be used as an effective tool to help decision makers select appropriate rules based on their knowledge regarding possible trade-offs. The thesis shows that evolved rules can dominate well-known dispatching rules when a single objective and multiple objectives are considered. Also, the obtained Pareto fronts show that many evolved rules can lead to favourable trade-offs, which have not been explored in the literature.

This thesis tackles one of the most challenging issues in job shop scheduling, the interactions between different scheduling decisions. New GPHH methods have been proposed to help evolve scheduling policies containing multiple scheduling rules for multiple scheduling decisions. The two decisions examined in this thesis are sequencing and due date assignment. The experimental results show that the evolved scheduling rules are significantly better than scheduling policies in the literature. A cooperative coevolution approach has also been developed to reduce the complexity of evolving sophisticated scheduling policies. A new evolutionary search mechanisms and customised genetic operations are proposed in this approach to improve the diversity of the obtained Pareto fronts.

Through this thesis, following major contributions have been made: (1) the first study comparing different representations of dispatching rules in GP, (2) the development of a novel type of dispatching rules referred to as iterative dispatching rules and a new GPHH for automatic design of these rules, (3) the first multi-objective GPHH method for dynamic JSS problems to find Pareto fronts of non-dominated dispatching rules, (4) two new GP methods to generate reusable due date assignment rules, and (5) a new multi-objective GPHH method for automatic design of scheduling policies to handle multiple scheduling decisions and multiple conflicting objectives.

List of Publications

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* (2012). DOI:10.1109/TEVC.2012. 2227326.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming". *IEEE Transactions on Evolutionary Computation* (2013). DOI:10.1109/TEVC.2013.2248159.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Genetic programming for evolving reusable due-date assignment models in job shop environments". *Evolutionary Computation* (2013). DOI:10.1162/EVCO_a_00105.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Hybrid evolutionary computation methods for quay crane scheduling problems". *Computers and Operations Research* (2013), Vol. 40, Issue. 8., pp. 2083-2093.

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* (2012). DOI:10.1007/s00170-013-4756-9.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Dynamic job shop scheduling problems: a MO-GPHH approach". In *Automated Scheduling (Eds.)*, Springer. (2012). (accepted).

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Evolving reusable

operation-based due-date assignment models for job shop scheduling with genetic programming". In *EuroGP'12: Proceedings of European Conference on Genetic Programming* (2012), pp. 121–133.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems". In *CEC '12: Proceedings of the IEEE Congress on Evolutionary Computation* (2012), pp. 3261–3268.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Automatic discovery of optimisation search heuristics for two dimensional strip packing using genetic programming". In *SEAL '12: Proceedings of International Conference on Simulated Evolution And Learning* (2012), pp. 341–350.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Learning reusable initial populations for multi-objective order acceptance and scheduling problems with genetic programming". In *EuroGP'13: Proceedings of European Conference on Genetic Programming* (2013), pp. 157-168.

Nguyen, S., Zhang, M., and Johnston, M.. "A genetic programming based hyper-heuristic approach for combinatorial optimisation". In *GECCO '11: Proceedings of Genetic and Evolutionary Computation Conference* (2011), pp. 1299-1306.

Park, J., **Nguyen, S.**, Zhang, M., and Johnston, M. "Genetic programming for order acceptance and scheduling". In *CEC '13: Proceedings of the IEEE Congress on Evolutionary Computation* (2013), (To appear).

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "A two stage learning/optimising system for multi-objective order acceptance and scheduling". *European Journal of Operational Research* (2013), (under review).

Acknowledgments

I would like to express my very great appreciation to my supervisors, Prof Mengjie Zhang, Dr Mark Johnston and Prof Kay Chen Tan, for their guidance and support during the course of my PhD study. Prof Zhang has spent many dedicated hours reading this thesis and my research articles; his feedback is always full of encouragement but also challenges that help improve my research work. Dr Johnston is always good to talk to, and the discussions with him are a great source for novel ideas. Prof Tan always gave very fast, useful and constructive recommendations on this study, despite the time zone.

I am grateful for the Victoria Doctoral Scholarship, the Marsden Fund of New Zealand (VUW0806 and 12-VUW-134), and the University Research Fund (200457/3230) at Victoria University of Wellington for their financial support over the past three years.

I would like to offer my thanks to my friends in the Evolutionary Computation Research Group (ECRG) for creating an active and interesting research environment. Thank you to Bing and Bruce for sharing their delicious food and enjoyable pingpong games.

I wish to thank my family for their constant support and encouragement throughout my study. Thank you to my parents who have always been there for me, offering their unconditional love and care. Last, but not least, I would like to thank my beloved wife Hue (and the little Tory) for her love, patience and support. You have always been the source of love that helps me complete this journey.

Contents

List of Tables

List of Figures

List of Notations

p_j total processing time of job j

d_j due-date of job j

w_j weight of job j

m_j number of operations of job j

r_j release time of job j

C_j completion time of job j

f_j flowtime of job j

$\hat{f}_j$ estimated (predicted) flowtime of job j

e_j error in flowtime estimation of job j

σ considered operation

δ machine that processes operation σ

$p(\sigma)$ processing time of operation σ

$\hat{f}_o$ estimated operation flowtime of operation σ

List of Abbreviations

CDR composite dispatching rule

DDAR due-date assignment rule

DJSS dynamic job shop scheduling

DMOCC diversified multi-objective cooperative co-evolution

DR dispatching rule

EA evolutionary algorithm

EC evolutionary computation

EMO evolutionary multi-objective optimisation

GA genetic algorithm

GP genetic programming

GPHH genetic programming based hyper-heuristic

HaD-MOEA harmonic distance based multi-objective EA

HH hyper-heuristic

IDR iterative dispatching rule

JSS job shop scheduling

MO-GPHH multi-objective GPHH

MTO make-to-order

NSGA-II non-dominated sorting genetic algorithm II

SI swarm intelligence

SPEA2 strength Pareto evolutionary algorithm 2

SP scheduling policy

VNS variable neighbourhood search

Chapter 1

Introduction

1.1 Problem Statement

Job Shop Scheduling (JSS) is a hard problem in the scheduling literature and it has received a lot of attention because of its computational challenges for the research community as well as applicability in real world situations. The term *job shop* is used to indicate companies that produce customer-specific components in small batches [108]. One feature that sets job shops apart from other production environments is the large variety of routings with different operation processing times through a set of machines. As reported by Johns and Rabelo [92], there are thousands of factories with billions of dollars worth of products in the United States that can be classified as job shops and production scheduling in this environment is considered as one of the most popular research topics due to its potential to dramatically decrease costs and increase throughput [92]. In the current highly competitive market, scheduling also plays a crucial role to improve customer satisfaction. It has been shown that consistently high priority has been given to the speedy and on time delivery capabilities of manufacturing managers in Europe, Japan, and the United States [100].

In JSS, given a set of machines and a set of jobs with various predetermined routes through the machines, the objective is to find a sched-

ule of jobs that minimises certain criteria such as makespan, maximum lateness, and total weighted tardiness [153]. When dealing with a fixed number of jobs with known processing information, the problem is called **static**. On the other hand, if the processing information of jobs is not known prior to their arrival, the problem is called **dynamic**. Different approaches have been proposed to solve these problems and they can be classified into two main categories [145]: (1) theoretical studies of optimisation methods, which are usually restricted to static problems and (2) experimental studies of **heuristics** or **dispatching rules** to deal with both static and dynamic problems.

Although dispatching rules are not guaranteed to provide optimal solutions for the problems, they have been applied extensively in research and practice because of their simplicity and ability to cope with the dynamic environment [169]. Different from conventional optimisation methods which represent the scheduling solution in a very sophisticated way in order to employ specialised techniques to solve the scheduling problem, a dispatching rule provides a way to perform the scheduling task which is understandable to shop floor operators. Normally, a dispatching rule is considered as a function that determines the *priorities* of jobs in the queue of a machine and decides which one should be processed next. The popularity of the dispatching rule is derived from the fact that it can be easily modified when real world aspects such as setup time, release time, machine breakdowns or parallel machines are considered. Another aspect that makes dispatching rules attractive to both researchers and practitioners is that they do not have the scalability problems which are a big issue for almost all optimisation methods. Moreover, effective dispatching rules can also be used to create initial solutions for optimisation procedures to fine-tune. In practice, dispatching rules have been applied in different manufacturing environments such as semiconductor factories [150] and printing companies [149] and shown promising results.

The process of designing dispatching rules is quite time consuming and complicated because the researchers have to manually synthesise various relevant elements to create the rules and design experiments to evaluate these rules. Recently, with the improvements of computing power, some machine learning methods [54, 112, 68, 87, 181, 78, 84] have been used to facilitate the design process of new dispatching rules. These methods are referred as hyper-heuristics, which aim at automating the design and tuning of heuristic methods to solve hard computational search problems [30]. **Genetic Programming (GP)** [103] has been shown to be a promising approach for this purpose because GP is capable of **evolving** complex priority functions with its flexible representation. In addition, the rules evolved by GP also provide good potential interpretability which is very useful for scheduling applications. However, many key aspects of GP have not been investigated that may enhance the performance of this method for evolving dispatching rules, such as representations and evolutionary search mechanisms. Besides, the past studies only focus on simplified JSS problems and practical requirements, e.g., multiple conflicting objectives and multiple scheduling decisions have not yet been explored.

The overall goal of this thesis is to develop a genetic programming based hyper-heuristic approach for **automatic heuristic design** of **reusable** and **competitive** dispatching rules in job shop machine scheduling environments by incorporating special features of JSS in the representations and evolutionary search mechanisms.

1.2 Motivations

Previous studies have suggested some approaches to improve the effectiveness of dispatching rules. One of the most straightforward ways to improve the performance of dispatching rules, without affecting their simplicity, is to use a combination of simple dispatching rules. One way to employ different dispatching rules is to monitor the status of jobs in the sys-

tem and make a change from one dispatching rule to another as planned. For example, FIFO/SPT will apply first-in-first-out (FIFO) when the jobs in the queue of the considered machine have been waiting for more than a specific time and shortest-processing-time (SPT) will be applied otherwise. This combination, even though very simple, can take advantage of each rule at the appropriate decision making moments and is normally better than the application of a single dispatching rule. Another approach to improving the performance of dispatching rules is to create composite dispatching rules (CDR) [88, 89], which provide heuristic combinations of simple rules basically in the form of sophisticated human-made priority functions of various scheduling parameters (processing times, waiting times, etc.).

Existing GP methods [87, 181, 78] for automatic discovery of dispatching rules for JSS only focus on generating CDRs by evolving sophisticated priority functions. However, the incorporation of the machine and system status has not been investigated in these GP methods even though it is a key point to create more **adaptive** dispatching rules. CDRs may have some implicit adaptive behaviours; however it is very difficult for a GP method to evolve rules with such a property because these adaptive behaviours will require complex arithmetic combinations. For this reason, there is a need to devise a new representation for GP to cope with this problem by making the evolved dispatching rules adaptive to shop changes. In addition, it is important that GP can help evolve rules that overcome the myopic behaviours of the traditional dispatching rules to improve the quality of the obtained schedules.

The existing research on the automatic design of dispatching rules mainly concentrates on single objective problems while multiple conflicting objectives are a natural feature of most production scheduling environments. Tay and Ho [181] have suggested a GP method to evolve dispatching rules for multi-objective flexible job shop scheduling problems. In their method, the multi-objective problem is transformed into a single objective problem

by linearly combining the objectives. However, since the scales of objectives in scheduling problems are very different (e.g. the scale of tardiness values is different from that of makespan values), it is very difficult to assign a suitable weight for each objective in a single fitness function in advance. Even in the case that normalised objective functions are used, the weights still need to be pre-defined by the decision makers even though they do not have good knowledge about the trade-offs among different objective functions. Finding the Pareto front of non-dominated dispatching rules will naturally be a good approach to this multi-objective problem because it does not require the decision makers to pre-determine the weights and will also help decision makers make better choices of dispatching rules based on the trade-offs represented by the obtained Pareto fronts. However, this research direction has not been explored in the past research.

Another issue with the design of new dispatching rules is the interaction of these rules with other scheduling and planning decisions within the scheduling system. This is important in order to ensure that scheduling systems perform properly and effectively. Due to the complexity of these decisions, the existing research on these interactions of scheduling decisions [160, 62, 151, 35, 41] is normally performed by examining different combinations of well-known rules and trying to find the combinations of rules that can achieve the best performance. A limitation of this approach is that it restricts us from exploring new potential combinations of rules for all related decisions. One of the advantages of GP is that the GP programs can represent multiple rules to handle multiple scheduling decisions in comprehensive scheduling systems. However, since the search space of GP will be significantly increased, the determination/development of suitable evolutionary search mechanisms in this case will be very important to improve the performance of GP.

1.3 Research Goals

The overall goal of this thesis is to develop **Genetic Programming based Hyper-heuristic (GPHH)** methods to evolve **reusable** and **competitive dispatching rules** for **job shop machine scheduling environments**. The focus of this research is to investigate how to utilise genetic programming to learn new dispatching rules for both **static** and **dynamic** JSS problems through the studies of **representations** and **evolutionary search mechanisms**. This research aims to use GP to evolve new effective dispatching rules that are capable of incorporating machine/system attributes and related scheduling decisions to enhance the productivity of job shops. It is expected that the evolved rules can be effective in unseen situations (reusable) and competitive as compared to other dispatching rules proposed in the literature. The research in this thesis will help answer the following research questions:

(i) *Which elements of JSS problems are significant/relevant when evolving effective dispatching rules with GP and how can these elements be represented by GP programs?*
Job shop scheduling is known as a very hard problem and many elements of the problem are needed to make effective scheduling decisions. However, it is not obvious what is a good way to employ these elements in order to improve the performance of dispatching rules. The manual development process is too time consuming to investigate this issue. Therefore, GP as an automatic heuristic generation method is more suitable in this case. Moreover, the flexibility of GP representations provides various ways to synthesise and evaluate complex dispatching rules. Studying GP representations in this case not only helps improve the performance of GP for evolving dispatching rules but also provides important patterns that could be useful in good dispatching rules and helps improve the interpretability of the evolved rules.

(ii) *How can GP be used to deal with multi-objective job shop scheduling problems?*
One of the challenges when designing dispatching rules for a job shop scheduling environment is to handle multiple conflicting objectives. However, this issue has been mostly ignored in the past research on automatic discovery of dispatching rules. In cases where GP is used, it is important to know which multi-objective evolutionary search mechanism is most helpful in tackling this issue. It would be also interesting to know how representations help GP explore trade-off (non-dominated) dispatching rules for the multi-objective job shop scheduling problems.

(iii) *Which evolutionary search mechanism of GP is effective for evolving a complex scheduling system involving different scheduling decisions?*
Even though dispatching rules are a key component in a scheduling system, their performance is affected by other planning and scheduling decisions. Due date assignment is one of the most direct scheduling decisions that can influence the performance of a dispatching rule. Creating combined rules to be evolved by GP for both dispatching and due date assignment tasks would be the most straightforward option to deal with this issue. However, it also increases the complexity of the problem. In this case, different evolutionary search mechanisms need to be investigated and evaluated to find the most suitable one for evolving such complex rules.

In order to fulfill the overall goal stated above and to find the answers for these research questions, a set of research objectives have been established to guide this research.

1. Developing a new GP method that provides *adaptive behaviours* for the evolved dispatching rules by incorporating the machine and system attributes; with an expectation of improving the effectiveness of these rules.

There are some existing works [86, 78, 181] using GP to evolve new dispatching rules for different manufacturing environments but the influences of representations of these dispatching rules within the GP population have not been carefully investigated. This thesis aims to evaluate and analyse these representations, which will help gain more understanding of how different representations can influence the performance of GP. New representations which incorporate the machine and system attributes will be proposed and compared with existing representations. Different aspects of representations such as the interpretability, overfitting issue and scalability will also be investigated to point out the advantages and disadvantages of rules evolved by using each representation. The thesis will also compare the performance of the dispatching rules evolved with these representations with the well-known dispatching rules proposed in the literature as well as existing heuristics and meta-heuristics.

2. Extending and improving the GP method to deal with *multi-objective* job shop scheduling problems.

   In practice, job shop scheduling is a multi-objective problem since many objectives are often simultaneously desired and they are normally conflicting. When dealing with multi-objective problems, a study needs to be done in order to know whether the proposed representations are still effective. This thesis will concentrate on using GP to evolve a Pareto front of evolved solutions which allow the decision makers to choose the suitable dispatching rules on the Pareto front that can satisfy their interests. Studying the Pareto front in this case will also help the scheduling researchers understand more about the factors that influence the trade-offs among different objective functions. To carry out this research effectively, evolutionary search mechanisms to find the Pareto front will need to be investigated. Popular methods such as Nondominated Sorting Ge-

netic Algorithm (NSGA-II) [52], Strength Pareto Evolutionary Algorithms (SPEA2) [195], Harmonic Distance Based Multi-Objective Evolutionary Algorithm (HaD-MOEA) [187], and a new proposed method based on the special characteristics of JSS will be considered to find the effective strategy for exploring the Pareto front in this case.

3. Developing a *hyper-heuristic method* that allows relevant factors influencing the performance of dispatching rules to be considered and evolved through the automatic heuristic generation process governed by GP. This is a crucial issue for the development of a comprehensive production planning and control system.

   Dispatching rules play an important role in most production planning and control systems. However, there are several factors that can affect the performance of dispatching rules. One of the most important interactions investigated in the job shop scheduling literature is between due date assignment rules and dispatching rules since these rules can directly influence the performance of the scheduling system. In this thesis, the proposed GP method will be used to evolve these two rules simultaneously. Since exploring the heuristic search space of these combined rules would be more difficult, a special evolutionary search mechanism may need to be used. Cooperative coevolution [156] seems to be a natural option in this case since each rule, either dispatching rules or due date assignment rules, can be evolved in a subpopulation and then combined with the rules in another subpopulation to create comprehensive rules for the scheduling system.

In order to evaluate the performance of the proposed GP methods, we will measure the performance of the evolved rules. For the static JSS problems, the evolved rules will be trained and tested on a set of well-known benchmark instances [53, 179, 5]. To deal with the dynamic JSS problem,

stochastic simulation models [94, 176, 81, 88, 78] will be used for both training and testing purposes. In this thesis, common criteria (e.g. makespan, mean tardiness, etc.) are used to evaluate the performance of a dispatching rule because of their popularity in both theoretical studies and practical applications. When dealing with the due date assignment rules in the third research objective, mean absolute percentage error (MAPE) between estimated flow times and actual flow times will be used to measure the performance of these rules.

Completing the first and second objectives will address research questions (i) and (ii). Meanwhile, the third research objective is investigated to help tackle the last research question.

1.4 Major Contributions

This thesis makes the following major contributions.

1. The thesis presents the first study that has compared different representations of dispatching rules used in GP and investigated how representations influence the performance of GP when evolving dispatching rules for the static JSS problems. Experimental results show that the representation which integrates system and machine attributes can improve the quality of the evolved rules. In addition, the proposed representations in this thesis also provide a convenient way to incorporate special system features of real world environments into the dispatching rules. Two fitness functions were also investigated and the rules that are evolved to optimise the average performance across all training instances provide better results than those evolved to optimise worst-case performance.

   Part of this contribution has been published in:

   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* (2012). DOI:10.1109/TEVC.2012.2227326.

2. This thesis develops a novel type of dispatching rule for the static JSS problems which can iteratively improve the schedules by utilising the information from completed schedules obtained in previous iterations. A new GP method has been used to help evolve such iterative dispatching rules (IDRs) and the results showed that these evolved rules outperform the existing rules in the literature and composite dispatching rules evolved by other existing GP methods. The analysis also showed that the evolved iterative rules are efficient and interpretable. Different aspects of IDRs are also investigated and the insights from these analyses are used to significantly enhance the performance of IDRs.

   Part of this contribution has been published in:

   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* (2012). DOI:10.1007/s00170-013-4756-9.

3. This thesis proposes the first multi-objective GPHH method for dynamic JSS problems to find the Pareto fronts of non-dominated dispatching rules to deal with multiple conflicting objectives. The extensive computational results have helped confirm the need of using multi-objective approaches to design effective and practical dispatching rules. The results showed that the Pareto fronts evolved from the proposed method contain rules that outperform existing rules when multiple objectives are considered simultaneously. The analysis of the obtained Pareto front provides much better knowledge about the search space of dispatching rules and shows that the proposed method is a good tool to support the decision making pro-

cess. New performance measures are also proposed to help assess the quality of the evolved rules for a simulation scenario and the robustness of the evolved rules across different simulation scenarios. These measures are useful not only for this study but also for future studies on multi-objective GPHH methods.

Part of this contribution has been published in:

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Dynamic job shop scheduling problems: a MO-GPHH approach". In *Automated Scheduling (Edited)*, Springer. (2012). (accepted).

4. This thesis develops two new GP methods to generate reusable due date assignment rules to estimate the due dates of new jobs in dynamic job shops with a particular dispatching rule. The novelty of these methods are the evaluation scheme employed to estimate flowtimes of jobs. Different factors of job shops consisting of utilisations, distributions of operation processing times, etc, are used to evaluate the reusability of the evolved rules. The results show that the evolved rules based on detailed information of each operation of a job are better than those that are only based on aggregate information in the shop. It is noted that the training scenarios play an important role on the testing performance of the rules and appropriate choices of trainings scenarios can help reduce the computational times of the proposed GP methods. Meanwhile, the number of simulation replications did not make a large impact on the performance of the evolved rules.

   Part of this contribution has been published in:

   Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming". In *EuroGP'12: Proceedings of European Conference on Genetic Programming* (2012), pp. 121–133.

Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Genetic programming for evolving reusable due-date assignment models in job shop environments". *Evolutionary Computation* (2013). DOI:10.1162/EVCO_a_00105.

5. This thesis proposes a new multi-objective GPHH method to generate a Pareto front of scheduling policies for dynamic job shops. This is the first work that focuses on multiple scheduling decisions and multiple objectives in JSS. The evolved Pareto fronts generated by the proposed methods contain scheduling policies that dominate existing scheduling policies developed manually in the literature. Moreover, the scheduling policies are interpretable, which helps explain how the trade-offs between different conflicting objectives can be obtained. A new evolutionary search mechanism, diversified multi-objective cooperative coevolution (DMOCC), is proposed based on the cooperative coevolution framework [156] to help find the Pareto front. The novelty of this method, as compared with the existing GPHH methods, is that it allows specific scheduling rules to be evolved in a subpopulation to reduce the complexity of evolving the whole scheduling polices.

   Part of this contribution has been published in:

   Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems". In *CEC '12: Proceedings of the IEEE Congress on Evolutionary Computation* (2012), pp. 3261–3268.

   Nguyen, S., Zhang, M., Johnston, M. and Tan, K. C. "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming". *IEEE Transactions on Evolutionary Computation* (2013). DOI:10.1109/TEVC.2013.2248159.

1.5 Organisation of Thesis

The remainder of this thesis is organised as follows. Chapter 2 presents the literature review of related works. Chapters 3 and 4 address the first research objective. Chapters 5–7 presents works to fulfill the second and third research objectives. Chapter 8 concludes the thesis.

Chapter 2 presents detailed descriptions of the JSS problem and the methodologies used for this problem. The basic concepts of genetic programming are given to provide the essential background for the readers and make this thesis self-contained. This chapter then gives a review of the current research on hyper-heuristics for heuristic generation, with a special focus on the genetic programming based hyper-heuristics for learning new dispatching rules.

Chapter 3 proposes new representations for dispatching rules which allow them to systematically incorporate useful JSS features. Different representations and fitness functions of dispatching rules are evaluated and the evolved rules are analysed to show how they can solve the problems.

Chapter 4 presents a new evaluation scheme to help iteratively construct better schedules for static JSS problems. Based on this evaluation scheme, a new form of dispatching rule called iterative dispatching rules are proposed which are able to iteratively improve the quality of the schedules by learning from the mistakes of the previous or existing schedules. Different aspects of iterative dispatching rules are investigated to enhance the performance of the evolved rules.

Chapter 5 develops a new multi-objective GPHH method to deal with dynamic job shops. Five popular objectives of job shop scheduling are simultaneously considered when the proposed method is used to evolve the Pareto front of non-dominated dispatching rules. An extensive comparison between the evolved rules and a range of rules from the literature is performed. New measures to help assess the Pareto dominance and the robustness of evolved rules are also provided.

Chapter 6 develops new GPHH methods to generate due date assignment rules for JSS. Two evaluation schemes of due date assignment rules based on the aggregate information of the shop and detailed information of each operations are proposed. Different factors of dynamic job shops are considered to assess the reusability of evolved rules. The performance of the evolved rules are compared with the analytical dynamic due date assignment rules proposed in the literature. A specialised terminal set is also proposed to enhance the quality of the evolved rules when due date oriented dispatching rules are used in the shop.

Chapter 7 proposes new multi-objective GPHH methods for automatic design of scheduling policies, i.e., combinations between due date assignment rules and dispatching rules. Two representation/evaluation concepts for scheduling policies are investigated in this chapter. Based on these concepts, four multi-objective GPHH methods are proposed. Reusability of the evolved Pareto fronts and the performance of the proposed methods are provided.

Chapter 8 summarises the key findings and provides the overall conclusions from this thesis. Key research points and the contributions of this thesis are ascertained. Finally, the opportunities for future works are discussed.

Chapter 2

Literature Review

This chapter starts by introducing basic concepts in scheduling problems and related methodologies used and discussed in this thesis. Then, detailed descriptions of the job shop scheduling (JSS) problem including the formal definitions of terminology and notations used by job shop researchers are provided. The concepts of active and non-delay schedules in JSS will also be presented. A review of traditional optimisation, heuristic, and hybrid methods to solve the static and dynamic JSS problems is given in this chapter to provide the readers a summary of the research topics in this field. This chapter then reviews key concepts of genetic programming (GP) such as the representation and genetic operators. The rest of this chapter provides an overview of GP based hyper-heuristic (GPHH) methods for heuristic generation with a special focus on heuristic generation methods for scheduling problems.

2.1 Basic Concepts

The purpose of this section is to provide basic concepts of general scheduling problems, artificial intelligence (AI), evolutionary computation (EC), and heuristics/meta-heuristics/hyper-heuristics.

2.1.1 Sequencing/Scheduling

Sequencing and scheduling is a research field motivated by practical needs in production planning, in computer control, and in most situations in which scarce resources have to be allocated to jobs/tasks over time [110]. Because of its practical importance, sequencing and scheduling has attracted great attention from researchers in operations research and computer science in the last few decades. In the literature, sequencing and scheduling are two steps of the process for generating a schedule/plan. In the first step, sequencing decides which jobs should be processed next. Scheduling then specifies the detailed schedules such as the resource (e.g. machine, work centre) to handle each task, the start times, and completion times of each task. Due to the complexity of the scheduling problems and the techniques employed to solve the problems, it is sometimes not easy to separate these two steps. For example, dispatching rules discussed in this thesis perform sequencing and scheduling iteratively at each decision moment. Meanwhile, some heuristics/meta-heuristics decide the processing sequence of jobs before mapping the sequence to a complete schedule.

Scheduling problems are classified based on the resource configurations and the nature of the jobs [13]. For instance, scheduling problems concern allocating jobs to a single machine or multiple machines in a manufacturing system. Meanwhile, if the set of jobs remain unchanged over time, it is a static scheduling problem. Otherwise, the scheduling problem is dynamic when new jobs arrive over time. Another important aspect to be considered in scheduling problems is the uncertainty of the processing information (processing times, release times, etc.). If all information is known in advance, we deal with deterministic problems. If uncertainty exists, the scheduling problems are called stochastic.

A wide range of optimisation techniques have been applied to deal with scheduling problems. Optimal solutions are found for some simplified scheduling problems. However, scheduling is considered NP-hard problems in most practical situations [110]. Therefore, approximating ap-

proaches and heuristics/meta-heuristics/hyper-heuristics have been developed to find near optimal solutions for scheduling problems in practical computational times. The descriptions of popular scheduling techniques can be found in Baker and Trietsch [13] and Pinedo [153]. A more detailed review of scheduling techniques for job shop scheduling problems is presented in Section 2.2.

2.1.2 Machine Learning

Machine learning can be defined as computational methods using experiences to improve performance or to make accurate prediction of specific tasks [126]. The goal of machine learning is to design computer programs which learn to solve problems without explicitly being programmed or instructed [3].

Machine learning methods are classified into three main categories: (1) supervised learning, (2) unsupervised learning, and (3) reinforcement learning [167]. In supervised learning, the actions or desired outputs for a problem are known in advance (e.g. classification, regression). A supervised learning algorithm tries to create an inferred function to map inputs to desired outputs. On the other hand, unsupervised learning deals with unlabelled training examples (e.g. clustering, dimensionality reduction). Since the correct actions are not provided, it is difficult to quantitatively evaluate the performance of learners [126]. In reinforcement learning, the learner interacts with the environment and receives an immediate reward for each action. The goal of the learner in this case is to maximise its reward over the course of actions and interactions with the environment. Different from supervised learning, no long-term reward feedback is provided by the environment and desired outputs are not explicitly provided. Some popular machine learning methods are decision tree, support vector machines, Bayesian learning [3, 126], and genetic programming [103].

2.1.3 Heuristics/Meta-heuristics/Hyper-heuristics

There are a few definitions of heuristics in the literature. In this thesis, heuristics are considered in the scope of optimisation, particularly in the field of scheduling. Heuristics refer to experience-based techniques to find good solutions for computational problems. Usually, heuristics are developed to deal with the cases where exact methods for solving a computational problem are impractical (e.g. too time-consuming) [121, 153]. For example, one of the most famous heuristics in scheduling is the shifting bottleneck for JSS problems. Since JSS is too complicated to be solved to optimality using exact methods such as branch-and-bound, the shifting bottleneck heuristic uses a clever definition of the bottleneck value of a machine to iteratively construct a schedule by solving a number of one machine scheduling problems [164].

Meta-heuristics are optimisation methods designed to deal with hard optimisation problems. Meta-heuristics are search methods containing general low level heuristics that help explore the solution search space to find near-optimal solutions. Different from heuristics, meta-heuristics are more abstract and usually make no or very few assumptions about the problems to be solved. Similar to heuristics, meta-heuristics also do not guarantee to find the global optimal solutions. There are many meta-heuristic techniques developed in the literature and they can be classified into two main categories: (1) local search based or single solution based and (2) population-based. Local search based methods employ local search heuristics and acceptance criteria to explore the search space. Typical examples of these methods are tabu search (TS) [71], simulated annealing (SA) [101] and variable neighbourhood search (VNS) [76]. On the other hand, population-based methods maintain multiple solutions that interact with each other to explore the search space. Most population-based methods belong to evolutionary computation, which will be discussed in the next section. When a problem domain knowledge is available, meta-heuristics can be also combined with heuristics to create hybrid methods.

Hyper-heuristics (HH) are a relative new research area which focuses on exploring the "heuristic search space" [30] of the problems instead of the solution search space in the cases with heuristics and meta-heuristics. Currently, there are two main research directions for hyper-heuristics which are (1) HH for heuristic selection and (2) HH for heuristic generation. The first research direction aims to "raising the level of generality" at which optimisation systems can operate [28]. Heuristic selection has mainly focused on developing HH frameworks that are able to adaptively select suitable pre-existing heuristics based on the problem solving states and the historical records obtained from the problem solving process [30, 34]. Many HH frameworks have been developed for heuristic selection such as the choice function [47, 46], tabu search based HH [34], and simulated annealing based HH [57]. Some new powerful HH frameworks for heuristic section have also been discovered from the Cross-Domain Heuristic Search Competition (CHeSC) [140]. Meanwhile, the objective of heuristic generation methods is to fabricate a new heuristic (or meta-heuristic). The obtained heuristic can be either an improving or constructive heuristic. In order to generate a new heuristic, the hyper-heuristic framework must be able to combine various small components (normally common statistics or operations used in pre-existing heuristics) and these heuristics are trained on a training set and evolved to become more effective [130]. Genetic programming and its variants (e.g. linear GP, grammatical evolution) are currently the most popular approaches for this heuristic generation.

2.1.4 Evolutionary Computation

Evolutionary Computation (EC) is a sub-field of artificial intelligence that focuses on nature inspired algorithms and iterative population-based systems to deal with optimisation or machine learning problems. The two main categories in this research area are (1) evolutionary algorithms [93] and (2) swarm intelligence [99].

Evolutionary Algorithms (EAs)

EAs are a subset of EC that imitate Darwinian biological evolution such as natural selection, reproduction, crossover, and mutation. EAs are usually characterised by the use of population(s) of individuals through the course of evolution. Similar to natural evolution, individuals in the population(s) have to compete for survival. The concept of fitness is used extensively in EAs to reflect the ability of an individual to survive and reproduce [93]. EAs work based on the notion of dynamically changing population due to the birth of new individuals inheriting genetic materials from parent individuals with high fitness, and the death of individuals with low fitness. Some popular EAs in the literatures are:

- Genetic Algorithms (GAs) [80]: is one of the earliest EC techniques. In GAs, each individual or chromosome is represented (encoded) by a fixed-length array of bits, integer numbers, or real numbers. These arrays carry the information that can be decoded to solutions for the problem needed to be solved. New offspring are generated by exchanging the genetic material between two selected parents or by mutating a selected parent. Parents in GAs are usually randomly selected and the individuals with higher fitness are more likely to be chosen.

- Genetic Programming (GP) [103]: is an extended form of GAs where individuals are represented as variable length computer programs. Different data structures (e.g. tree, linear, grammar) have been employed to construct computer programs. This provides GP with the flexibility to perform different tasks such as machine learning and optimisation.

- Evolution Strategy (ES) [22]: is an optimisation method which adopts the concept of biological evolution. Different from GAs and GP, ES mainly uses only mutation to generate new individuals and the indi-

vidual selection in ES is deterministic and only based on the fitness rankings.

- Evolution Programming (EP) [60]: is similar to GAs, but focuses more on the behaviour linkage between parents and offspring rather than trying to imitate the genetic operators in nature. In EP, both populations of parents and offspring are placed into the same pool to select individuals for the next generation.

Swarm Intelligence (SI)

SI is a research area that focuses on the emergent collective intelligence of groups of simple agents [21, 26] such as ant colonies, bird flocks, fish schools, and bacterial growth. In SI, the agents follow simple rules, but the interaction between agents can lead to "intelligent" global behaviour by the swarm even though there is no centralized control structure dictating how individual agents should behave. Two typical SI techniques in the literature are:

- Particle Swarm Optimisation (PSO) [99]: is a simple SI technique based on simulated social behaviours. In PSO, a swarm or population of particles (candidate solutions) moves in the solution search space. Each solution is usually encoded as a vector of real numbers that are treated as the positions of a particle in the swarm. Each particle is assigned a fitness value based on its performance. Then particles update their positions by being accelerated towards referenced particles such as global best, and local best (based on certain topologies) with higher fitness. The key idea is that this movement will help guide the swarm towards the global optimal solutions in the search space.

- Ant Colony Optimization (ACO) [55]: is an algorithm that simulates the behaviour of ants seeking a path between their colony and a

source of food. In ACO, the artificial ants find the path (solution) to the food source and leave on its path a trail of pheromone. Gradually, the pheromone on the shorter paths (better solutions) are strengthen and become more attractive to the ants. Based on this idea, many extensions of ACO have been developed and shown to provide very good results.

These two techniques have been applied successfully into a wide range of optimisation problems such as vehicle routing [2, 20] and job shop scheduling [173, 189]. Other well-known SI techniques include artificial immune system (AIS) [48], and artificial bee colony algorithm (ABC) [96].

2.2 Job Shop Scheduling Problems

A scheduling problem is traditionally described by the triplet $\beta|\gamma|\delta$, where β represents the machine environment, γ provides the processing characteristic (it may contain no entry at all or multiple entries), and δ describes the objective to be minimised [153]. The general JSS problem could be simply defined as the scheduling of different jobs to be processed on different machines [61] to satisfy certain objectives. In this case, a job is a sequence of operations, each of which is to be performed on a particular machine. In JSS, the routes of jobs are fixed, but not necessarily the same for each job [153]. An example of a job shop studied in this thesis is shown in Figure 2.1. For the static JSS problem, the shop (or the working/manufacturing environment) includes a set of $\mathcal{M}$ machines and $\mathcal{N}$ jobs that need to be scheduled. Each job j has its own pre-determined route through a sequence of machines to follow and its own processing time at each machine it visits. In static JSS, processing information of all jobs is available. In the dynamic JSS problem, jobs arrive randomly over time and the processing information of jobs is unknown before their arrival. Some basic definitions and notations in JSS are as follows.

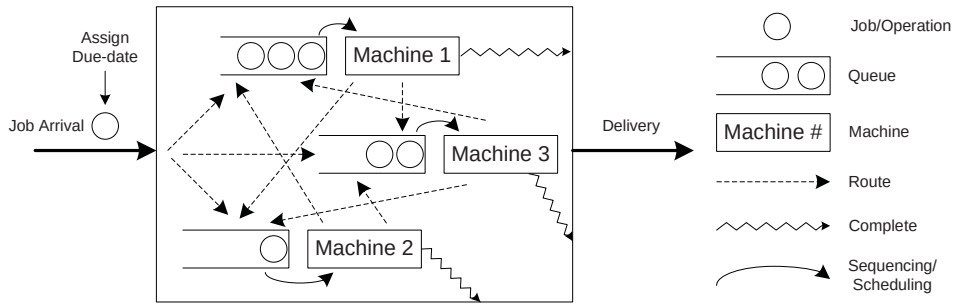


Figure 2.1: Job Shop Scheduling (shop with 3 machines).

Parameters:

- $O_j = \{o_{j,1}, \ldots, o_{j,l}, \ldots, o_{j,N_j}\}$: the set of all operations of job j where $o_{j,l}$, is the l^{th} operation of job j and N_j is the number of operations of job j.
- w_j: the weight of job j in the weighted tardiness objective function.
- d_j: the due date assigned to job j.
- $p(\sigma)$: the processing time of operation σ.
- $m(\sigma)$: the machine that processes operation σ.
- $next(\sigma)$: the next operation of the job that contains operation σ or $null$ if σ is the last operation of that job (if $\sigma = o_{j,l}$ then $next(\sigma) = o_{j,l+1}$).

Variables:

- U_k: the ready time of machine k, which is the time that the machine becomes idle; in this study, all machines are idle at the beginning.
- r_j: the release time when job j is available to be processed.
- $r(\sigma)$: the ready time of operation σ, which is the release time r_j of job j for the first operation or the completion time of its preceding operation for other operations.

Table 2.1: Performance measures for JSS problems

Mean Flowtime	$\mathrm{F} = \frac{\sum_{j\in\mathbb{C}} f_j}{\lvert\mathbb{C}\rvert}$
Maximum Flowtime	$\mathrm{F}_{max} = \max_{j\in\mathbb{C}}\{f_j\}$
Percentage of Tardy Jobs	$\%\mathrm{T} = 100 \times \frac{\lvert\mathbb{T}\rvert}{\lvert\mathbb{C}\rvert}$
Mean Tardiness	$\mathrm{T} = \frac{\sum_{j\in\mathbb{T}} (C_j - d_j)}{\lvert\mathbb{T}\rvert}$
Maximum Tardiness	$\mathrm{T}_{max} = \max_{j\in\mathbb{T}}\{C_j - d_j\}$
Makespan	$\mathrm{C}_{max} = \max_{j\in\mathbb{C}}\{C_j\}$
Total Weighted Tardiness	$\mathrm{TWT} = \max_{j\in\mathbb{T}}\{w_j \times (C_j - d_j)\}$

- C_j: the completion time of job j.
- f_j: the flowtime of job j calculated by $f_j = C_j - r_j$.
- T_j: the tardiness of job j calculated by $T_j = \max(C_j - d_j, 0)$.

Table 2.1 gives formal definitions of the seven objective measures considered throughout this thesis. In this table, $\mathbb{C}$ is the collection of jobs recorded to calculate the objective values ($\mathbb{C}$ is all the jobs in static JSS problem instances or a set of jobs recorded after the warm-up period of the simulation of the dynamic job shops). Meanwhile, $\mathbb{T} = \{j \in \mathbb{C} : C_j - d_j > 0\}$ is the collection of tardy jobs. These objectives are selected since they are very popular performance measures in JSS, which have been used regularly in previous studies [162, 88, 81].

2.2.1 Active schedules and non-delay schedules

In JSS, a schedule is called *active* if it cannot be altered to make any operations complete earlier without delaying the completion time of other operations. Active schedules were first proposed by Giffler and Thompson [69] in their seminal work and they have proven that an optimal solution

1: $\Omega \leftarrow \{o_{1,1}, o_{2,1}, \ldots, o_{\mathcal{N},1}\}$
2: **repeat**
3: let $t(\Omega) = \min_{\sigma \in \Omega}\{\max\{r(\sigma), U_{m(\sigma)}\} + p(\sigma)\}$
4: let σ^* be the operation that minimum is achieved, $m^* = m(\sigma^*)$, and $\Omega^* = \{\sigma \in \Omega | m(\sigma) = m^*\}$
5: let $S(m^*) = \max\{\min_{\sigma \in \Omega^*}\{r(\sigma)\}, U_{m^*}\}$
6: let $\Omega' = \{\sigma \in \Omega^* | r(\sigma) \leq S(m^*) + \alpha(t(\Omega) - S(m^*))\}$
7: apply dispatching rule Δ on Ω' to find the next operation σ' to be scheduled on m^*
8: remove σ' from Ω' and include $next(\sigma)$ into Ω if $next(\sigma) \neq null$
9: **until** all operations have been scheduled

Figure 2.2: A Generic procedure to construct a schedule for JSS.

of JSS with makespan as the objective to be minimised must be an active schedule. Schedules are called *non-delay* schedules if no machine is allowed to be idle when there are jobs in its queue ready to be processed. A non-delay schedule is more restricted than an active schedule and the set of non-delay solutions may not necessarily include the optimal solution. Non-delay schedules are often applied in experiments using simulation since a machine will immediately process all jobs in its queue in the order determined by some sequencing rule to avoid loss of utilisation of manufacturing resources.

Figure 2.2 shows a generic procedure to construct an active schedule, a non-delay schedule or a hybrid of both active and non-delay schedules with a dispatching (priority) rule Δ. In this procedure, Ω contains all the operations that are ready to be scheduled. The procedure will first determine the next operation with the earliest completion time and its corresponding machine m^* (ties are broken arbitrarily). The non-delay factor $\alpha \in [0, 1]$ controls the look-ahead ability of the algorithm and determines the set Ω' of jobs which should be considered to be processed next

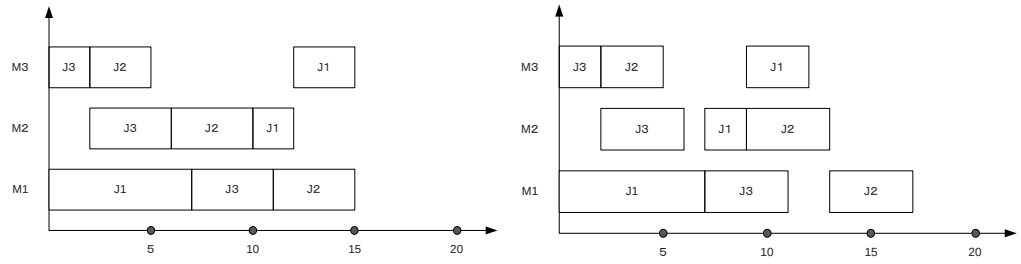
Table 2.2: Example of a static JSS problem instance ($\mathcal{N} = 3, \mathcal{M} = 3$)

job	machine sequence	processing time	weight	due date
#1	1,2,3	7,2,3	4	13
#2	3,2,1	3,4,4	2	10
#3	3,2,1	2,4,4	1	12

on machine m^* by the dispatching rule Δ. If $\alpha = 0$, the procedure can only construct non-delay schedules and only the operations that have already joined the queue are considered for scheduling. On the other hand, if $\alpha = 1$, the procedure will consider all the potential jobs that are ready to join the queue of machine m^* before the earliest completion time of m^*. In general, α determines the interval of time that a machine is allowed to wait even when there are operations in the queue. As a result, there are fewer operations to be considered in each step of the algorithm when α is small than when α is large. The purpose of dispatching rule Δ is to calculate the priorities for all operations in Ω'. Usually, the job with the highest priority in Ω' will be scheduled next on machine m^*.

Example: The following is an example of a static JSS problem instance with three jobs and three machines. Table 2.2 shows the machine sequence and processing time corresponding to each job. Figure 2.3 shows the schedule (solution) using shortest processing time (SPT), where operations with smaller processing times have higher priorities, as the dispatching rule Δ with both $\alpha = 0$ (non-delay) and $\alpha = 0.5$ (look-ahead).

For this example, the first machine to be scheduled is machine $m^* = 3$ since it is the one with the earliest completion time (from operation $o_{3,1}$) within $\Omega = \{o_{1,1}, o_{2,1}, o_{3,1}\}$. Because $o_{3,1}$ is also the operation with the shortest processing time (SPT) in the queue of machine 3, it is scheduled in this step, and removed from Ω. The updated set of ready operations is then $\Omega = \{o_{1,1}, o_{2,1}, o_{3,2}\}$. Similarly, $o_{2,1}$, $o_{3,2}$ and $o_{1,1}$ will be



(a) SPT/non-delay ($\alpha = 0$) (b) SPT/look-ahead ($\alpha = 0.5$)

Figure 2.3: Example of schedules in JSS ($\mathcal{N} = 3, \mathcal{M} = 3$).

scheduled in the next three steps. In the fifth step, $\Omega = \{o_{1,2}, o_{2,2}, o_{3,3}\}$ and $m^* = 2$. In this step, if $\alpha = 0$, only $o_{2,2}$ is considered to be scheduled. If $\alpha = 0.5$ then $\Omega' = \{o_{1,2}, o_{2,2}\}$ and $o_{1,2}$ is scheduled in this step because it has a smaller processing time on machine 2. In the case of SPT/non-delay (SPT is used with non-delay factor $\alpha = 0$), the makespan $C_{max} = \max\{C_1, C_2, C_3\} = \max\{15, 15, 11\} = 15$ and the total weighted tardiness $\sum w_j T_j = 4 \times 2 + 2 \times 5 + 1 \times 0 = 18$. Similarly, $C_{max} = 17$ and $TWT = 14$ in case of SPT/look-ahead. In this example, SPT/non-delay outperforms SPT/look-ahead on the makespan but SPT/look-ahead is better than SPT/non-delay when the objective is to minimise the total weighted tardiness.

2.2.2 Job shop scheduling methods

Over the last few decades, a large number of methods have been developed and applied to JSS, ranging from simple heuristics to artificial intelligence and mathematical optimisation methods [92]. Dispatching rules are perhaps the most straightforward method to deal with both static and dynamic JSS problems [171, 162, 88]. Meanwhile, optimisation is the main research stream to deal with the static JSS problems [153]. A review of these methods is shown in this section. For a broader review of scheduling methods, the readers are encouraged to read Ouelhadj and Petrovic [143] and Potts and Strusevich [158].

Dispatching rules

Although there have been many breakthroughs in the developments of exact and approximate methods for JSS; these methods are mainly focused on static problems and simplified job shop environments. General methods like genetic algorithm (GA) can be extended to solve problems with realistic constraints, but the major drawback is its weak computational efficiency. Moreover, as pointed out in [118], the conventional operations research and artificial intelligence methods are often not applicable to the dynamic characteristics of the actual situation because these methods are fundamentally based on static assumptions. For that reason, simple dispatching rules have been used consistently in practice because of their ability to cope with the dynamic changes of the shop.

There have been a large number of rules proposed in the literature and they can be classified into three categories [92]: (1) simple priority rules, which are mainly based on the information related to the jobs; (2) combinations of rules that are implemented depending on the situation that exists on the shop floor; and (3) weighted priority indices which employ more than one piece of information about each job to determine the schedule. Composite dispatching rules (CDR) [88, 89, 153] can also be considered as a version of rules based on weighted priority indices, where scheduling information can be combined in more sophisticated ways instead of linear combinations. Panwalkar and Iskander [145] provided a very comprehensive survey on scheduling (dispatching) rules used in research and real world applications using a similar classification. Pinedo [153] also showed various ways to classify dispatching rules based on the characteristics of these rules. The dispatching rules in this case can be classified as *static* and *dynamic* rules, where dynamic rules are time dependent (e.g. minimum slack) and static rules are not time dependent (e.g. shortest processing time). Another way to categorise these rules is based on the information used by these rules (either local or global information) to make sequencing decisions. A *local* rule only uses the information available at the machine

where the job is queued. A *global* rule, on the other hand, may use the information from other machines.

The comparisons of different dispatching rules have been continuously done in many studies [171, 162, 88, 81, 78]. The comparison was usually performed under different characteristics of the shop because it is well-known that the characteristics of the shop can significantly influence the performance of the dispatching rules. Different objective measures were also considered in these studies because they are the natural requirements in real world applications. Although many dispatching rules have been proposed, it is still a challenge for scheduling researchers to develop rules that can perform well on multiple objectives.

Meta-heuristic methods

Since the static JSS is a NP-hard problem [65], finding optimal solutions by mathematical programming methods can be very time-consuming even for reasonable small instances. The research on meta-heuristics [70] for scheduling has been very active in the last two decades, mostly with makespan as the objective. Local search based methods [4] such as simulated annealing [182], large step optimisation [115], tabu search [139], and guided local search [14] have shown very promising results. The focus of these methods is on the development of efficient neighbourhood structures (mainly based on the concept of critical paths and critical blocks) and diversifying strategies to escape from local optima. Since the neighbourhood structures play an important role in these methods, they and their related operators have to be redesigned in order to incorporate real world constraints; even then it is still questionable whether they produce good results.

A more general alternative for solving JSS problems is the use of evolutionary computation methods. GA is one of the most popular methods in this line of research (refer to [38] for a review of GA methods for JSS). More recently, many hybrid algorithms have been proposed to combine the advantages of GA and local search heuristics. Yamada and Nakano

[190] presented a GA with the multi-step crossover (MSX) for JSS. In this method, MSX was used in combination with a local search heuristic. The preliminary experiments using benchmark instances showed promising performance of the proposed approach. Goncalves et al. [74] proposed a hybrid GA method for JSS to minimise makespan. In this method, the chromosome representation is based on random keys and represents the priorities of operations. An active/non-delay parameter is also applied to restrict the delay time of operations. During the GA search, the schedule is further improved by the neighbourhood local search procedure from [139].

Swarm intelligence methods [21, 26] have also been applied to JSS problems and show very promising results. Sha and Hsu [173] developed a hybrid PSO algorithm (HPSO) that modified the particle position based on preference list-based representation and employed Giffler and Thompson algorithm [69] to decode particle positions into schedules. Moreover, tabu search is also applied to further improve the solution quality. The experimental results showed that HPSO is competitive compared to other meta-heuristics proposed in the literature. Xing et al. [189] proposed a sophisticated ant colony optimisation method in which a knowledge model is used to learn some available knowledge from the optimisation of ACO. The existing knowledge will be used to guide the current heuristic searching. The proposed knowledge-based ant colony optimization (KBACO) algorithm outperformed some current approaches.

Research on other objectives have also been considered in the literature, especially due date related objectives due to the need to improve the delivery performance in modern manufacturing systems. Pinedo and Singer [152] presented a heuristic to minimise the total weighted tardiness in JSS, which was based on the shifting bottleneck procedure [153] that schedules one machine at a time and used a branching tree to find a good order to schedule the machines. Every node of the tree represents a partial order in which the machines are scheduled. From the experiments, this

method yielded solutions that were near optimum on some problems with 10 jobs and 10 machines. Asano and Ohta [6] introduced another heuristic for the minimisation of the total weighted tardiness in JSS that is based on the tree search procedure, also with very promising results. Kreipl [105] proposed an efficient large step random walk (LSRW) method for minimising total weighted tardiness. This method employed different neighbourhood sizes to perform a small step or a large step. The small step consists of iterative improvement while the large step consists of a Metropolis algorithm (similar to the simulated annealing algorithm but with a constant temperature). Essafi et al. [59] proposed a hybrid genetic algorithm which employed the iterative local search and the hybrid scheduling construction procedure to solve this problem. The results showed that the proposed method is very competitive as compared to LSRW [105].

Dispatching rules based meta-heuristics

Some efforts to combine both meta-heuristics and dispatching rules into the same framework have also been proposed in the literature. Storer et al. [178] proposed a simple hill climbing method to find the good assignments of heuristics/dispatching rules into different scheduling windows. Even though some good results were obtained, this method was still restricted to known dispatching rules and their linear combinations, which may not be sufficient to solve hard JSS problems. Dorndorf and Erwin [56] employed GA as a meta-strategy to guide an optimal design of local decision rule sequences. They proposed two GA methods in their paper. The first method aimed at finding the optimal sequence of dispatching rules to handle the conflicting operations in each decision step of the Giffler and Thompson algorithm [69]. The second method was used to control the selection of nodes in the enumeration tree of the shifting bottleneck heuristic [153]. The results showed that the proposed methods can find better solutions compared to the shifting bottleneck heuristic and simulated annealing. Hart and Ross [77] developed a hybrid GA method which is quite

similar to the dispatching rules based GA proposed by Dorndorf and Erwin [56], except that the methods to calculate the set of conflicting operations are also encoded into the genes. This method was quite promising compared to other similar methods in the literature. However, the performance of this method was governed by the choice of dispatching rules to deal with different objectives. Zhou et al. [194] introduced a general framework using a genetic algorithm and dispatching rules to solve a similar problem and showed that it was better than the pure genetic algorithm. Different from those proposed in [77] and [56], this method encoded the dispatching rules for each machine in combination with the first job to be scheduled on that machine. Even though the proposed hybrid GA provided solutions that are inferior to those obtained in [105], it was still very promising since it is capable of solving a variety of scheduling problems without major redesign. For other similar methods, the reader is referred to [43], [149] and [155].

Multi-objective optimisation methods for JSS

Since many conflicting objectives usually exist in JSS environments, there is a practical need to design a method to deal with these multi-objective problems. Kacem et al. [95] proposed a Pareto approach based on the hybridisation of fuzzy logic and evolutionary algorithms to minimise makespan, total workload of machine, and the workload of the most loaded machine. The results from the experiments were very encouraging. Xia and Wu [188] used particle swarm optimization (PSO) to assign operations to machines and simulated annealing to schedule operations on each machine. This study also considered the three objectives used in [95]; however, the problem is converted to a single objective problem by using an aggregated weighted objective function. The results from this hybrid method were shown to be better than some evolutionary algorithms. Also using an aggregated weighted objective function, Lam et al. [106] introduced an enhanced genetic algorithm for multi-objective JSS with ob-

jectives (makespan, mean of weighted tardiness and mean of weighted earliness). The algorithm evolved three individual objectives in a modified parallel GA system with migration to reach their own objectives. After that they are combined to continue the evolutionary process with the compromise combined objective. More recently, Vázquez-Rodríguez and Petrovic [184] introduced a new dispatching rule based GA for the multi-objective JSS problem. In their algorithm, they encoded both dispatching rules and the number of successive steps that these rules were applied to. The proposed representation showed very good performance on a wide range of problem instances.

2.2.3 Holistic view of job shop scheduling

It has been a common assumption for many people that job shop scheduling is equivalent to sequencing that determines the order in which jobs waiting in the queue of a machine are processed [1]. A large number of studies on JSS, as shown in the previous section, also mainly focus on the sequencing part. However, sequencing is only one of several scheduling decisions. Other important scheduling decisions include job order release and due date assignment. For job order release, the task is to control the amount of work to be released into the shop in order to balance the throughput and the congestion of the manufacturing system. Meanwhile, due date assignment decisions are made whenever orders (jobs) are received from customers and good due date assignments are needed in order to maintain high customer satisfaction. Figure 2.4 shows the connections between these decisions and the locations where these decisions are applied in a production planning and control system. Most studies on job shop scheduling only considered one of the many decisions and fixed the others in order to reduce the complexity of the scheduling problems. These approaches are valid when there is no interaction among the scheduling decisions, which is often not the case for real world applica-

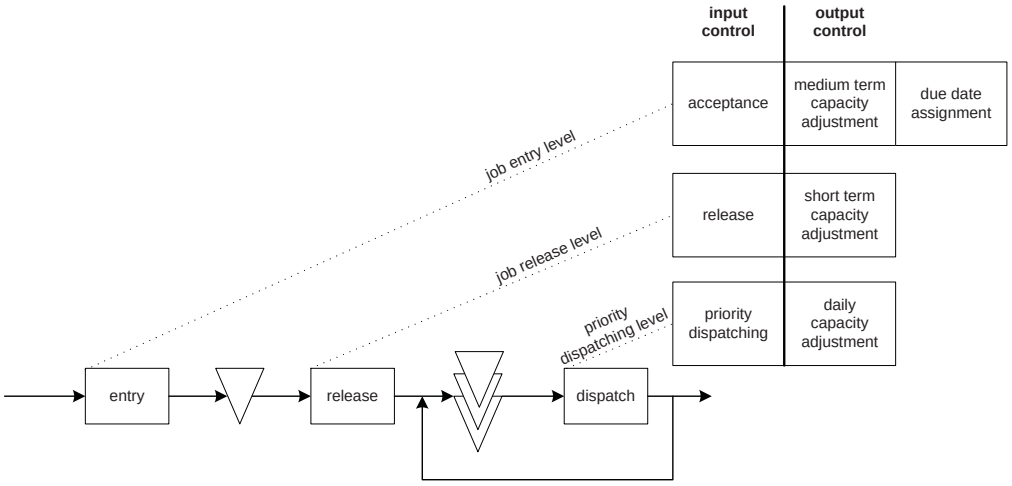


Figure 2.4: Production planning and control (PPC) decisions [107].

tions. The focus of thesis is mainly on due date assignment and dispatching/scheduling decisions. Other planning tasks (as shown in Figure 2.4) will be simplified in order to make it easier to analyse the interactions between the two considered scheduling decisions. Following are reviews on other scheduling decisions and their interactions with dispatching rules.

Due date assignment

Due date assignment decisions are made whenever jobs (customer orders) are received from customers. Good due date assignments are needed in order to maintain high delivery performance (delivery speed and delivery reliability). Generally, due dates can be set: (1) exogenously, or (2) endogenously [40, 163]. In the former case, due dates are decided by independent agencies (sellers, buyers). This thesis only focuses on the second case, where the due dates are internally set based on the characteristics of the jobs and shop [163], to improve the delivery performance of job shops. The endogenous due date assignment is especially important when manufacturers need to "promise" a delivery date to customers and it is also

useful for better management of shop floor activities. Basically, the due date of a new job is calculated as:

$$d_j = r_j + \hat{f}_j \tag{2.1}$$

where $\hat{f}_j$ is the estimated (predicted) flowtime of job j. The value of $\hat{f}_j$ is calculated by the due date assignment rule (DDAR). In the ideal case, we want the calculated due date d_j to be equal to the completion time of the job C_j. The performance (with respect to missing the due date) is normally measured by the error (lateness) between the completion time and due date $e_j = C_j - d_j = f_j - \hat{f}_j$, where f_j is the actual flowtime.

Some criteria to evaluate the performance of DDARs [41, 19] in the JSS literature are shown in Table 2.3. In this table, $\mathbb{C}$ is the set of jobs collected from the simulation runs to calculate the performance measures, e_j is the lateness of job j, $\bar{e}$ is the mean lateness and $\mathbb{T}$ is the set of tardy jobs ($C_j - d_j > 0$). `MAPE` and `MAE` measure the accuracy of the flowtime estimation. Smaller `MAPE`s or `MAE`s indicate that the DDAR can make better predictions. `MPE` measures the bias of the DDAR. If the DDAR results in a negative (positive) `MPE`, it means that the DDAR tends to overestimate (underestimate) the due date. `STDL` measures the delivery reliability of the DDAR. A smaller `STDL` indicates that the estimated due dates are more reliable. Another delivery performance measure is $\%$`T`, which shows the percentage of jobs that fail to meet the due date.

Many DDARs have been proposed in the JSS literature. The DDARs in early studies are mainly based on creating a simple model that employed aggregate information from the new job and the shop. Examples of these methods are Total Work Content (TWK) where $d_j = r_j + kp_j$, Number of Operations (NOP) where $d_j = r_j + km_j$, and Processing Plus Waiting (PPW) where $d_j = r_j + p_j + km_j$. In these methods, p_j and m_j are the total processing time and the number of operations of job j, and k is a coefficient that needs to be determined. Other more sophisticated models

Table 2.3: Performance measures of DDARs

Mean Absolute Percentage Error	$\texttt{MAPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{\lvert e_j\rvert}{f_j}$
Mean Percentage Error	$\texttt{MPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{e_j}{f_j}$
Mean Absolute Error	$\texttt{MAE} = \frac{\sum_{j\in\mathbb{T}}\lvert e_j\rvert}{\lvert\mathbb{C}\rvert}$
Standard Deviation of Lateness	$\texttt{STDL} = \sqrt{\frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}(e_j-\bar{e})^2}$
Percent Tardiness	$\%\texttt{T} = 100\times\frac{\lvert\mathbb{T}\rvert}{\lvert\mathbb{C}\rvert}$

have also been proposed that incorporate more information of jobs and the shop to make better predictions of flowtimes. These include Job in Queue (JIQ), Work in Queue (WIQ), Total Work and Number of Operations (TWK + NOP), Response Mapping Rule (RMR), and Operations-based Flowtime Estimation (OFE). Comparisons of these DDARs [160, 62, 151, 35, 41, 168] show that the DDARs which employ more useful information can lead to better performance. However, the main drawback of these methods is that they depend strongly on the determination of the corresponding coefficients for factors used in the prediction models. The most popular method to determine the coefficients is using linear regression models.

Because of the complexity and stochastic nature of dynamic job shops, nonlinear models will be needed [151], which make it computationally more expensive to solve for regression methods. For this reason, many artificial intelligence methods have been applied to solve this problem. Philipoom et al. [151] proposed a neural network (NN) method for due date assignment and showed that their NN method can outperform conventional methods and nonlinear models. Also in this direction, Sha and Hsu [174] developed an NN method for due date assignment in a wafer fabrication system that showed very good results. Patil [147] enhanced the NN method by using ensemble learning and bagging/boosting concepts. A GA method was also employed to search for neural network architectures that develop a parsimonious model of flowtime prediction.

The computational results showed that the enhanced NN method outperformed other simple NN methods. Although different shop environments were considered, the paper only focused on training and testing the NNs on the same shop environments and the reusability of the obtained NNs on unseen different shop environments have not been examined. Baykasoglu and Gocken [17] applied gene expression programming (GEP) to evolve a symbolic regression model for DDA in a specific multi-stage job shop. The results showed that the evolved DDAR was better than the previous proposed DDARs. However, only aggregate information from the shop was employed to estimate the job flowtimes and detailed information of operations has not been considered. Also, similar to [147], there is no analysis on the reusability of the evolved DDARs. Other data-mining methods such as decision trees [144], regression trees [175], and a regression based method with case-based tuning [176] have also been proposed, showing very promising results.

Although the DDARs described above have shown good results in simulation studies, determining good model coefficients is not an easy task, especially with the dynamic changes in the shop floor. To overcome this problem, some dynamic DDARs have been proposed, in which the coefficients are adjusted based on the information of the new job and state of the system. Cheng and Jiang [41] proposed Dynamic Total Work Content (DTWK) and Dynamic Processing Plus Waiting (DPPW) by applying Little's law [114] from queueing theory:

- DTWK:
$$d_j = r_j + \max\left\{1, \frac{N_{s_t}}{\lambda\mu_p\mu_g}\right\}\sum_{i=1}^{m_j} p_{ji} \tag{2.2}$$

- DPPW:
$$d_j = r_j + \sum_{i=1}^{m_j} p_{ji} + \frac{N_{q_t} m_j}{\lambda\mu_g} \tag{2.3}$$

  where N_{s_t} is the number of jobs in the shop at the moment a new job arrives, λ is the average arrival rate of jobs, μ_p and μ_g are respectively

the average processing time and average number of operations, p_{ji} is the processing time of the i^{th} operation of job j, and N_{q_t} is the total number of jobs in the queue of each machine.

In another study, Baykasoglu et al. [19] developed ADRES, a new dynamic DDAR, which uses a simple smoothing method to estimate the waiting time of the next job. In this model, the due date can be calculated as follows (assuming zero transportation times):

$$d_j = r_j + \sum_{i=1}^{m_j} w'_{ji} + \bar{p}_j \tag{2.4}$$

where $w'_{(j+1)i} = \alpha_{ji} + (1 - \alpha_{ji})w'_{ji}$ is the formula to estimate the waiting time of job j at its i^{th} operation, $\alpha_{ji} = |\frac{S_{ji}}{A_{ji}}|$ is the smoothed value, $S_{ji} = \beta e_{ji} + (1-\beta)S_{(j-1)i}$ is the smoothing error, $A_{ji} = \beta|e_{ji}| + (1-\beta)A_{(j-1)i}$ is the absolute smoothed error, β is a smoothing constant, $e_{ji} = w_{ji} - w'_{ji}$ is the error of the waiting time estimation when w_{ji} is the actual waiting time, and $\bar{p}_j$ is the sum of mean processing times at stations on the route of job j.

Previous research has shown that DTWK, DPPW and ADRES are very effective DDARs as compared to static regression-based DDARs. Another advantage of these DDARs is that no preliminary runs to obtain the parameter estimations are necessary. Therefore, they have been used as good candidates for comparison purposes [175, 19, 17, 18, 186].

One of the problems with the dynamic DDARs is that they are still mainly based on the aggregate information of jobs and the shop to make the prediction and ignore the detailed operation information, while it is shown that this information can help improve the quality of the prediction [168]. However, development of such operation-based DDARs would be very difficult since these models involve many different factors (variables). Thus, there is a need to have an automatic method to facilitate the design of such models. Also, there is no previous study on the reusability of the proposed models in the JSS literature, so it is questionable whether the

proposed/evolved models can be applied, without major revisions, when there are changes in the shop.

Other scheduling policies

Ragatz and Mabert [161] performed a large number of experiments to evaluate the combinations of five releasing mechanisms and four dispatching rules under different levels of due date tightness. The results suggested that the use of job release mechanisms can enhance the performance of simple dispatching rules. Rohleder and Scudder [166] performed another study to compare the performance of dispatching rules and order release mechanisms. The simulation experiments showed that the job release mechanisms are less important than dispatching rules when dealing with early/tardy problems, especially when the shop is at high utilisation level (the percentage of time that machines are employed). Lu et al. [116] performed similar research for the assembly job shop environment. The experimental research on both job release mechanism and dispatching rules is highly relevant to due date and flow time based performance measures. The study also emphasised the incorporation of the order release mechanism into dispatching rules to enhance the performance of the job shop scheduling system.

Ahmed and Fisher [1] were the first to investigate three-way interaction between due date, job release and due date assignment rules. The average total cost was used as the performance measure. This study concluded that the combination of rules used in a job shop is at least as important as the choice of individual scheduling decisions. Moreira and Alves [129] further investigated the interactions between these three scheduling decisions and included an order acceptance decision to form a four-way decision making problem for the general job shop environment. The study concluded that simultaneously considering all four decisions can improve mean tardiness and total flow time of jobs in the system.

Although job shop scheduling has been popular for decades, the studies on the interactions between scheduling decisions are still quite limited. One of the reasons for the lack of research in this direction is because dealing with each scheduling decision is itself already hard. Therefore, the studies on the interactions can only depend on the combinations of well-known rules for each scheduling decision, making the problem even harder. Since the 1990s, optimisation has been the dominant research trend in job shop scheduling. While there have been many great achievements, the focus on optimisation methods (mainly for one scheduling decision in a static environment) also restricts the studies of the interactions between scheduling decisions. In order to effectively tackle this problem, there is a need for a new methodology for improving scheduling decisions, which can cope with dynamic features of job scheduling problems.

2.3 Genetic Programming

Genetic programming (GP) [103] is an evolutionary computation method, inspired by biological evolution, to automatically find computer programs for solving a specific task. In genetic programming, a population of computer programs (individuals) is created and these programs are evolved to gain higher (better) fitnesses through an evolutionary process. In each generation of the evolutionary process, each program is evaluated by using a pre-defined fitness function, which assesses the ability of the program to perform a specific computational task. The fitness values obtained by programs in the population decide the chance of each program to survive and reproduce in the next generation. Different from GAs [80], each individual of a GP population is not represented by a fixed length string of genes (bits, real numbers, or symbols). Because the shape and length of the final program is normally not known by the user, individuals in GP usually represent programs in a tree form of various lengths [103]. Other rep-

resentations are also developed in grammar-based GP [119], linear-based GP [27], cartesian genetic programming [122] and achieve very promising results.

Since its first introduction more than thirty years ago, GP has been applied to solve a wide range of real world applications. Moreover, because of its ability to evolve novel and promising solutions for solving problems, GP has produced a large number of human-competitive results and even patentable new inventions [154]. In the rest of this section, key concepts of GP are presented to explain how GP works. This review only focuses on tree based GP as this form is used in this thesis. Since multi-objective optimisation and coevolution are also incorporated into GP in this thesis, we briefly review these aspects in this section.

2.3.1 Representation

Figure 2.5 shows an example of a GP individual, which is a tree-based representation of the program $x + \min(y - 3, 0)$. The variables $\{x, y\}$ and constants $\{3, 0\}$ are called terminals of the program and can be found at the leaves of the tree. Meanwhile, the arithmetic operations $\{+, -, \min\}$ are called functions in GP; each requires at least one argument and cannot be located at the leaves of the tree. The collections of these terminals and functions are known as the terminal set and the function set, respectively. Basically, a GP individual is a specific combination of elements selected from these two sets. In the GP literature, in order to easily observe the relationship between a function and its subtrees, the GP programs are usually presented to the human users by using the *prefix* notation similar to a Lisp expression. For example, the program $x + \min(y - 3, 0)$ can be presented as $(+\,(x\,(\min\,(-\,y\,3)\,0)))$.

In the above example, the return values of all functions as well as terminals are numerical. However, in some applications of GP, it is useful to have programs that contain different components (or subroutines) with

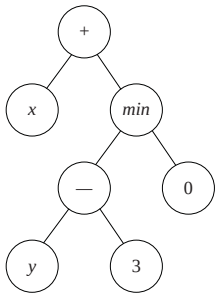


Figure 2.5: GP program that represents $x + \min(y - 3, 0)$.

different return types. For example, when the conditional function if is used, the condition requires the return type to be boolean, but other arguments of this function can be numerical. This issue, if not considered, will cause serious errors in the evolution process of GP. Poli et al. [154] presents a summary of these approaches, including strongly typed genetic programming, and grammar based approaches. The basic idea of these methods is to ensure the *closure* of GP programs. In strongly typed GP (STGP) [127], types and their constraints are incorporated into the GP system. Therefore, each terminal possesses a type and each function will only accept the arguments of certain types and it also has its own return type. Another approach to deal with this problem is to use grammars to express constraints [75, 79, 141]. In this type of system, the grammar is used to ensure that the initial population is made up of legal "grammatical" programs. Moreover, the genetic operators in this GP system also need to follow the grammar. An interesting discussion about the advantages and disadvantages of these solutions is presented in [154].

2.3.2 Initialisation

Similar to other evolutionary computation methods, GP starts its evolution process with a randomly generated population. Various initialisation approaches have been proposed in the literature. Two of the earliest and

most popular approaches are *full* and *grow* [103]. In these approaches, a maximum depth is determined for each GP individual to restrict the program tree depth (the largest number of edges required to reach a leaf node). For the full approach, the purpose is to generate full trees whose terminals are all located at the maximum depth of the trees. In this approach, nodes are randomly selected from the function set until the maximum depth of the tree is reached. However, the fact that all programs generated by the full approach will have the same depth does not mean that the lengths of these programs (the total number of nodes) are the same because functions may require different number of arguments (arities). On the other hand, the grow approach does not require the generated program to have all terminal nodes at the maximum depth. In this case, nodes can be selected either from the function set or the terminal set and the approach only forces the node to be selected from the terminal set when the maximum depth is reached. In order to diversify the initial population with individuals of various depths, lengths and shapes, these two approaches are often combined and the hybrid approach is known as *ramped half-and-half*. The purpose of this method is to generate half the initial population with the full approach and the other half with the grow approach.

2.3.3 Evaluation

Evaluation is an important step in GP in order to determine the fitness (goodness) of evolved programs. If the programs are in a tree form, they are normally executed (interpreted) by traversing the tree recursively starting from the root node and postponing the evaluation of each node until the values of its children (arguments) are known [154]. Since an evolved program is usually planned to be able to handle different situations, its fitness will depend on the performance of the program when it runs with different inputs (scenarios). A pre-defined fitness function will use the re-

sults obtained from the application of an evolved program to calculate the fitness for that program. The fitness function plays an important role in a GP system because it helps guide the search to find good programs. Depending on a specific task, the fitness function can be very different. For example, the fitness can be the aggregate errors between the outputs and the target outputs of the evolved programs; the amount of time or cost required for a program to complete a task; or the performance measures of the programs when they are used to solve a computational problem. Since the fitness function is used to assess the ability of the evolved programs to achieve a specific objective, it is sometimes also referred as the objective function.

2.3.4 Selection

After individuals in the GP population are evaluated, they will be assigned fitnesses to show how well these individuals perform on a specific task. The fitness of an individual decides its chance to be selected for genetic operators. Similar to the natural selection process, good individuals will be more likely to be chosen to generate offspring (child programs). The most popular selection methods in EC methods are fitness proportionate (or roulette wheel selection) and tournament selection [154].

For roulette wheel selection, individuals are randomly selected based on the distribution determined by their fitness. Individuals with good fitness have higher probabilities to be chosen and the poor individuals have lower probabilities to be selected. A variant of this method is ranking selection where the proportionate selection is performed based on the ranks of individuals in the population. One problem with roulette wheel selection is that poor individuals with low fitness may have very small probabilities to be selected while the selection pressure for good individuals can be very high.

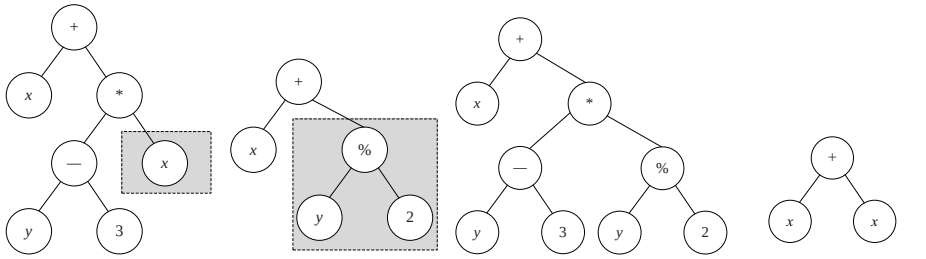
Tournament selection is another alternative to deal with this problem. In tournament selection, a number of individuals (the tournament size)

are first randomly sampled from the population. Then, the best individual among these individuals will be selected. The first step of the tournament selection uses equal probabilities for all individuals to be selected, including the poor individuals [117]. For this selection method, a smaller tournament size will give higher probabilities for the bad individuals to be selected for the genetic operators.

2.3.5 Genetic operators

The role of genetic operators is to utilise the genetic materials from existing individuals (parents) to generate new individuals (children or offspring) for the next generation. Because of the special representation, genetic operators in GP are significantly different from those used in other evolutionary computation methods. Like in genetic algorithms, the basic genetic operators in GP are *crossover*, *mutation* and *reproduction*. To perform crossover, two parent individuals are selected from the population by using the selection methods above. The most commonly used form of crossover in GP is *subtree crossover* [103] as shown in Figure 2.6. A node is selected randomly as the crossover point from each of the selected parents and two offspring are typically created by replacing the subtree rooted at the crossover point in the copy of the one parent with the copy of the subtree rooted at the crossover of the another parent [154]. Since a large number of nodes in GP trees are leaves (terminals), uniformly selecting nodes for crossover may lead to exchanges of very small amounts of genetic materials. To deal with this problem, Koza [103] suggested to choose function or non-terminal nodes for 90% of the time and terminal nodes or leaves 10% of the time.

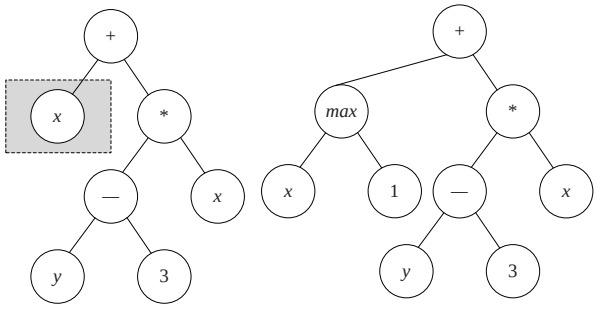
For mutation, only one parent is needed to generate a new offspring. The most popular type of mutation in GP is the *subtree mutation*, in which a random mutation point is chosen from the selected parent. The subtree rooted from this node is then removed and replaced by a newly generated



(a) Parent 1 (b) Parent 2 (c) Offspring 1 (d) Offspring 2

Figure 2.6: Subtree crossover in GP.

subtree. An example of subtree mutation is shown in Figure 2.7. Another common form of mutation in GP is *point mutation*, which only replaces a random node with an equivalent node from either function or terminal sets. Multiple nodes can also be mutated in one application of point mutation. The difference between the subtree mutation and point mutation is that the shape of the program may be changed when subtree mutation is used while the point mutation still preserves the shape of the parent program.



(a) Parent (b) Offspring

Figure 2.7: Subtree mutation in GP.

Reproduction in GP is quite similar to that in other evolutionary computation methods. When reproduction is applied, an individual is selected from the population by the selection mechanism (e.g. tournament selection) and copied to the population of the next generation. In each gen-

eration of GP, the applications of each genetic operator are governed by the probability assigned to each type of genetic operator and referred to as crossover rate, mutation rate and reproduction rate. Different from some evolutionary computation methods where these genetic operators can be applied sequentially, these operators are mutually exclusive in GP [154]. Besides these basic genetic operators, an *elitism* mechanism is also often used in many evolutionary computation methods. This operator just simply picks the top individuals of the current generation and inserts them into the population of the next generation. The objective of using elitism is to ensure that the best obtained individuals will not be lost through the probabilistic selection process.

2.3.6 Basic GP algorithm

Figure 2.8 shows a basic GP algorithm. As mentioned, the aim of GP is to evolve a program that is capable of solving a problem, so the output of this algorithm is the best program found by GP. The algorithm starts by loading all the data required to evaluate the fitness of a program and randomly generating the initial population of size S. At a generation, each program Δ_i will be applied and the fitness value of each program is calculated by a pre-determined fitness function. If the evaluated program is better (has smaller fitness value since we focus on minimisation problems in this research) than the best program Δ^*, it will be assigned to the best program found so far Δ^* and the best fitness value $fitness(\Delta^*)$ is also updated. After all individuals in the population are evaluated, the GP system will apply genetic operators such as reproduction, crossover and mutation to the selected programs (using the selection methods in Section 2.3.4) in the current population to generate new individuals for the next generation. When the maximum number of generations is reached, the GP algorithm will stop and return the best found rule Δ^*.

Inputs: problem data
Output: the best evolved program Δ^*

randomly initialise the population $P \leftarrow \{\Delta_1, \ldots, \Delta_S\}$
set $\Delta^* \leftarrow null$ and $fitness(\Delta^*) = +\infty$
$generation \leftarrow 0$
while $generation \leq maxGeneration$ **do**
    for all $\Delta_i \in P$ **do**
        evaluate $fitness(\Delta_i)$
        if $fitness(\Delta_i) < fitness(\Delta^*)$ **then**
            $\Delta^* \leftarrow \Delta_i$
            $fitness(\Delta^*) \leftarrow fitness(\Delta_i)$
        end if
    end for
    $P^{new} \leftarrow \{\}$
    while $|P^{new}| < S$ **do**
        $\Delta^{new} \leftarrow$ apply either reproduction, crossover, or mutation to selected programs from P
        $P^{new} \leftarrow P^{new} \cup \Delta^{new}$
    end while
    $P \leftarrow P^{new}$
    $generation \leftarrow generation + 1$
end while
return Δ^*

Figure 2.8: Basic GP algorithm.

2.3.7 Multi-objective GP

Multi-objective GP (MOGP) has been very active in the last decade [154] to solve problems with different conflicting objectives. The applications of MOGP are quite wide, ranging from reducing bloat [58, 50, 49, 11], simplifying solution trees [109] within the GP evolution process, learn-

ing boolean queries in information retrieval systems with precision and recall as objectives [45], to designing autonomous navigation controllers for unmanned aerial vehicles with high stability and efficiency [16].

In general, there are two basic ways to deal with multiple objectives. The first option is to combine these objectives into an aggregate scalar fitness function. One of the easiest ways is to linearly combine these objectives into a function $f = \sum_i w_i f_i$, where f_i is the i^{th} objective to be optimised and w_i is the weight assigned to objective f_i. The advantage of this method is that the existing single objective GP methods can be used without any (or with very little) modification. However, the users need more effort to find suitable ways to combine conflicting objectives. Even when the simple linear combination is employed, determining weights for different objectives is not easy because it requires the users to have some knowledge about the landscapes of the objective functions, which is often not available.

Another way to handle multiple objectives is based on the Pareto dominance concept. From the Pareto dominance perspective, a solution dominates another solution when it is not inferior to the other solution in all objectives and there is at least one objective where this solution performs better. In this case, the ultimate goal is not to find a single high fitness solution but a set of non-dominated solutions or Pareto front instead. Because the Pareto front may contain a large number of non-dominated solutions and heuristic methods such as GP cannot guarantee the optimal solution, the aim of the search methods is to find the set of non-dominated solutions to approximate the true Pareto front. For this reason, many performance measures have been proposed to measure the quality of the approximated Pareto front such as (inverted) generational distance convergence to the true Pareto front [52, 197] and (generalised) spread [52, 193] to indicate the diversity of solutions in the obtained Pareto front. An excellent review and analysis of different quality indicators for multi-objective problems are presented by Zitzler et al. [197]. Many general approaches in evolu-

tionary computation have been proposed for searching the Pareto front. The following are some popular approaches which can work well with genetic based methods (see [44] for a more comprehensive review):

- Non-dominated Sorting GA (NSGA, NSGA-II) [52]: evolve an archive population which is updated in each generation by collecting the non-dominated solutions from both parent and offspring populations. Pareto dominance is used in the traditional tournament selection and a "crowding distance" is used to break the ties in these tournaments as well as the updating process of the archive population.

- Harmonic Distance Based Multi-Objective Evolutionary Algorithm (HaD-MOEA) [187]: is an improved version of NSGA-II by introducing a new crowding distance measure called harmonic average distance and changing the selection scheme used in NSGA-II to prevent the inappropriate selection of solutions based on non-dominated ranks.

- Strength Pareto Evolution Algorithms (SPEA, SPEA2) [195]: maintain a fixed separate archive and update this archive in each generation. The fitness of an individual in this case is the combination between the dominance or strength of that individual and its area density. When updating the archive, if the size of the union of non-dominated solutions from both population and archive exceeds the size of the archive, the most similar solutions will be eliminated from the archive.

- ε-Multi-Objective Evolutionary Algorithm (ε-MOEA) [51]: is a steady state algorithm which is different from generational evolutionary algorithms like NSGA, NSGA-II, SPEA, and SPEA2. In each generation of ε-MOEA, one solution is produced by using one parent from the population and one parent from the archive. The offspring will

be compared with other individuals in the population to see whether it can be accepted to the population and replace the old individual. Moreover, this offspring is also considered to be included in the archive by checking the ε-dominance criterion and different strategies are also used to decide whether the offspring is accepted.

- Pareto Archived Evolution Strategy (PAES) [102]: is also a steady state algorithm. However, different from ε-MOEA, PAES accepts an offspring into the population and the archive of non-dominated solutions. The new solution is generated by applying a simple mutation operator to the current solution. If the current solution is dominated by the new solution, it will be replaced and the archive will also be updated. In PAES, the archive is used for two purposes: (1) storing non-dominated solutions, and (2) supporting for accurately select between current and candidate solutions. If the archive is full, the solution in the least crowded region will be preferred.

2.3.8 Co-evolution with GP

Another interesting topic in GP is co-evolution [157], where multiple populations are evolved and the fitness of an individual depends on its interactions with other individuals. There are two types of co-evolution called *competitive* co-evolution and *cooperative* co-evolution. In competitive co-evolution, individuals have to co-evolve against other individuals (within a population or in opponent population) and the fitness of an individual depends on the fitness of opponent individuals. On the other hand, cooperative co-evolution evolves different populations each of which contains partial solutions of a problem. Different from competitive co-evolution, the fitness of an individual is measured by combining it with one member from each of the other populations to form a complete solution.

Co-evolution with GP has been very successful in many applications. Azaria and Sipper [9] applied a competitive co-evolution method to learn strategies for playing the game of backgammon. The results show that

the co-evolution approach, which allows individuals to play against each other, can lead to much better results compared to those learned with fixed external opponents. Jin [90] used a cooperative co-evolution GP to evolve the strategies for bargaining problems under incomplete information and time preference. This method evolves two populations each of which contains the bargaining strategies for a player. The fitness of a strategy is the average utility of the strategy gaining from the agreements with strategies in the co-evolving opponent's population. The co-evolution method was shown to provide reasonably good solutions for several difficult bargaining problems. Gagné and Parizeau [64] proposed a cooperative co-evolution method to design nearest neighbour classifiers. In this method, both GA and GP are used to evolve the nearest neighbour prototypes and neighbourhood proximity measures. The experimental results from this method are very promising when compared with other traditional methods. In another study, Mendes et al. [120] tried to discover fuzzy classification rules with a cooperative co-evolution system, in which GP is used to evolve the fuzzy rule sets and evolution strategies (ES) [170] is used to evolve membership function definitions. The experimental results of this method on five data sets are very good compared with two existing methods. In bioinformatics, co-evolution GP was also employed for automatic knowledge discovery in annotated sequence data. Both amino acid-based regular expressions and keyword-based logical expressions were co-evolved by genetic programming. The proposed approach has been shown to be very useful and it was able to discover unexpected links between biological processes. Many other applications of co-evolution GP also achieved reasonable successes [113, 192, 83, 172].

2.4 Hyper-Heuristics for Heuristic Generation

Hyper-heuristics (HHs) are a relatively new search method and have attracted a lot of attention from the research community. Different from

heuristics and meta-heuristics, hyper-heuristics aim at exploring the heuristic search space to find a logical and efficient way to solve hard computational search problems. Heuristic selection and heuristic generation are currently the two main research methodologies in HH [30]. Even though they both work in the *"heuristic search space"* and try to *"raise the level of generality"* [30] at which optimisation can operate, their methodologies are quite different. The key idea of heuristic selection is to find the right existing heuristics or combinations of existing heuristics for problem solving. Most of the related studies for heuristic selection have been based on evolutionary methods and local search methods. Examples of evolutionary methods are memetic algorithms [104, 85, 142], where genetic search and local search are employed adaptively through the searching process. Some interesting aspects of memetic automation can be found in [36]. On the other hand, local search methods are mainly based on the principles of reinforcement learning [47] and local search based meta-heuristics such as tabu search [34] and simulated annealing [57].

The focus of this study is on hyper-heuristics for heuristic generation where the aim is to generate new heuristics. In order to generate a new heuristic, the HH framework must be able to combine various small components (normally common statistics or operators used in pre-existing heuristics) and these heuristics are trained on a training set and evolved to become more effective.

2.4.1 Genetic programming based hyper-heuristics

Recently, GP has become popular in the field of hyper-heuristics and it is known as genetic programming based hyper-heuristics (GPHH) [31]. Because GP is able to represent and evolve complex programs or rules, it naturally becomes an excellent candidate for heuristic generation. Bolte and Thonemann's work [25] can be considered the first to successfully adopt GP to learn new heuristics. They proposed a GP system to evolve an-

nealing schedule functions in simulated annealing to solve the quadratic assignment problem (QAP). The paper also considered the case when the GP system was used as a meta algorithm for simulated annealing where the annealing schedules found were tailored to a specific problem, and the case where the GP system is used to evolve a more generalised annealing schedule. The experimental results showed that the method with GP as a meta algorithm can find near optimal solutions for QAP and the improved simulated annealing algorithm based on the generalised annealing schedule found by GP outperformed existing simulated annealing algorithms.

Fukunaga [63] used GP to evolve variable selection heuristics in each application of a local search algorithm for the satisfiability (SAT) problem. Existing components that are used in popular local search methods are used as primitives of the GP system. The experimental results showed that the evolved heuristics are very competitive when compared with other heuristics. Burke et al. [29, 32, 33] proposed a GPHH framework to evolve construction heuristics for online bin packing. The basic idea of this GP system is to generate a priority function from static and dynamic statistics of the problem such as the size of piece p, and the fullness and capacity of bin i. If the output of this function is greater than zero, piece p is placed in bin i. The experimental results showed that human designed heuristics can be obtained by GP.

Keller and Poli [98, 97] proposed a grammar based linear genetic programming method to solve the travelling salesman problem. Several grammars are introduced, including ones with a loop construct. Bader-El-Den et al. [10] introduced a sophisticated grammar based GP for evolving timetabling heuristics. Their GPHH is based on a grammar derived from a collection of graph colouring heuristics that have previously been shown to be effective in constructing timetables. The grammar in this method provided a flexible way to evolve a heuristic that can react to different states of the construction process of a solution. Even though the proposed GPHH produced competitive results when compared with some existing

search methods in the literature, it was not shown whether the evolved heuristics can be reused on new problem instances.

2.4.2 GPHH for scheduling problems

GPHH is currently the most popular approach to automatic design of dispatching/scheduling rules. This section provides a review of GP applications for scheduling problems, which has been classified based on manufacturing environments.

Single machine environments

Dimopoulos and Zalzala [54] used GP to evolve dispatching rules for the one-machine scheduling problem with a standard function set and a terminal set of scheduling statistics (processing time, release time, due date, number of jobs, etc.). The evolved dispatching rules are better than traditional rules even for large and unseen instances. Jakobovic et al. [87] employed the same method for developing dispatching rules for the parallel machine scheduling problem in both static and dynamic environments. However, the evolved rules obtained from these studies have not considered the effects of different representations on the performance of the GP system, and the machine and system attributes were not included in the evolved rules.

Also trying to learn new dispatching rules for the single machine environment, Geiger et al. [68] presented a learning system that combines GP with a simulation model of an industrial facility. The proposed GP is used to create the priority rule for a single machine in static and dynamic environments. The terminal set of GP includes system attributes, job attributes, and machine attributes, and the function set consists of basic operators. The paper also proposed a method to learn dispatching rules for multiple machine problems in which GP will evolve multiple trees simultaneously with modified crossover and mutation operators. Comparison with

the optimal rule in a simple two machine environment showed that the evolved rules are quite competitive. However, the use of an independent dispatching rule for each machine may rapidly increase the complexity of the scheduling systems and make it difficult to generate generalised rules for large manufacturing systems. A similar GP system is also applied to solve the batch processor scheduling problem [67], where a machine can simultaneously process multiple jobs in a batch and all of these jobs will begin and finish at the same time. Both static and dynamic cases are considered in their study and the experimental results show that evolved rules can approximate the optimal rule in the static case when total flow time is the objective to be minimised. In dynamic cases, the evolved rules are better than the existing rules for minimising the total tardiness and total flow time. However, no system or machine status is considered in these studies to improve the adaptiveness of the evolved rules.

Nei et al. [138] proposed a gene expression programming (GEP) method to solve the single machine scheduling problem. In their study, many objective functions are considered and the rules evolved by GEP show very competitive results compared with existing rules. The comparison with GP also showed that GEP can perform slightly better than GP and with less computational time. However, no statistical test was performed to support these results. A disadvantage of the proposed GEP system is that it requires more parameters than the standard GP. Yin et al. [191] employed GP with bi-trees programs that include the priority rules to sequence jobs in the queue and the functions to calculate the required idleness to deal with the machine breakdown problem in a single-machine environment. The evolved rules are shown to be much better than the existing rules in this field.

Job shop environments

Miyashita [123] developed an automatic method using GP to design customised dispatching rules for a job shop environment and viewed JSS as

a model of a multi-agent problem where each agent represents a resource (machine or work station). Three multi-agent models are proposed: (1) a homogeneous model where all resources share the same dispatching rule, (2) a distinct agent model where each resource employs its own evolved rule, and (3) a mixed agent model where two rules can be selected to prioritise jobs depending on whether the resource is a bottleneck or not. Although the multi-agent models perform better, the use of these models depends on some prior-knowledge of the job shop environment, which can be changed in dynamic situations. A similar system was also proposed by Atlan et al. [7] but the focus was on finding the solution for a particular problem instance.

Jakobovic and Budin [86] applied GP to evolve dispatching rules for both single machine and job shop environments. The results for the single machine environment are shown to be better than existing rules. For the job shop environment, a meta-algorithm is defined to show how the evolved rules are used to construct a schedule. This study also proposed an interesting way to provide some adaptive behaviours for the evolved rules. They proposed a GP-3 system that evolves three components, a discriminant function and two dispatching rules. The discriminant function aims at identifying whether the considered machine to be scheduled is a bottleneck or not. This function serves as the classifier in binary classification problems. Based on the classification decision obtained from the discriminant function, one of two dispatching rules will be selected to sequence jobs in the queue of that machine. Even though the purpose of the discriminant function in this case is to identify the bottleneck machine, there is no guarantee that the classification can help indicate the bottleneck machine or just (some) other useful attributes of the shop or machines. The results show that the GP-3 system performed better than traditional GP with a single tree. Unfortunately, no analysis of the evolved rules or demonstrations of the evolved rules are given. Tay and Ho [181] proposed a GP system to evolve dispatching rules for a multi-objective

job shop environment. The multi-objective problem was converted into a single objective problem by linearly combining all objective functions. The proposed GP program can be considered as a priority function which is used to calculate the priority of operations in the queue of a machine based on a set of static and dynamic variables. The set of instances was randomly generated and it was shown that the evolved dispatching rules can outperform other simple dispatching rules. However, they did not consider the use of machine attributes in the priority function.

Hildebrandt et al. [78] re-examined this system in different dynamic job shop scenarios and showed that rules evolved by Tay and Ho [181] are only slightly better than the earliest release date (ERD) rule and quite far away from the performance of the SPT rule. They explained that the poor performance of these rules is caused by the use of the linear combination of different objectives and the fact that the randomly generated instances cannot effectively represent the situations that happened in a long term simulation. For that reason, Hildebrandt et al. [78] evolved dispatching rules by training them on four simulation scenarios (10 machines with two utilisation levels and two job types) and only aimed at minimising mean flow time. Some aspects of the simulation models were also discussed in their study. The results indicated that the evolved rules were quite complicated but effective when compared to other existing rules. Moreover, the evolved rules are also robust when tested with another environment (50 machines and different processing time distributions).

2.4.3 Other techniques for learning dispatching rules

Other methods have also been applied for discovering new dispatching rules for JSS. Li and Olafsson [112] applied the decision tree method on production data to generate dispatching rules. For further improvement, the authors used data engineering to create more useful attributes besides ones recorded as part of the raw production data. The results show that

the discovered decision rules can accurately replicate the dispatching list obtained by specific rules. However, since the rules are learned from the historical records of the production system, it is difficult to generate new dispatching rules and the performance of the evolved rules may only be as good as the actual rules. Most recently, Ingimundardottir and Runarsson [84] proposed a logistic regression approach which tries to discover new dispatching rules using the characteristics of optimal solutions. The learned linear priority dispatching rules showed better results than simple rules. One of the drawbacks is that the proposed supervised learning approach tries to learn from the optimal solutions which are normally not available in many cases.

2.5 Chapter Summary

Dispatching rules have been a very practical tool for scheduling in real world situations. However, manual design of new dispatching rules is still a very time consuming process. Several machine learning methods have been proposed to ease this task. Genetic Programming is one of the more popular methods because of its flexibility which helps GP easily cope with different problem environments. The results reported in the literature have confirmed the effectiveness of GP for designing new dispatching rules. However, the research in this direction is relatively new and many aspects still need to be explored to enhance the quality of GP methods and cope with practical requirements. The following are some remarks regarding this line of research.

- Although several works have been done to investigate the use of GP for automatic design of dispatching rules, studies of representations and evaluation schemes of the evolved rules have not been provided. It is noted that existing works mainly aim to evolve priority functions with the same behaviours as the traditional dispatching rules. However, with the flexibility of GP representation, special as-

pects of JSS can be used to enhance the representation of dispatching rules. Moreover, since the design process is automatic, GP can apply a more sophisticated evaluation scheme to evolve the dispatching rules which are difficult to be explored manually.

- In real-world applications, scheduling decisions need to take into account multiple conflicting objectives. However, dispatching rules evolved by existing GP methods mainly focus on a single objective. Although, Tay and Ho [181] considered multiple objectives in their study, all objectives are combined into a single objective function and rules are evolved in the way similar to other GP methods. Since the weights for each objective need to be determined prior to GP runs, without any knowledge about the trade-offs between these objectives the evolved rules may be undesirable as pointed out by Hildebrandt et al. [78]. Evolving Pareto fronts of non-dominated dispatching rules is a more suitable method to deal with this issue.

- With the current focus on the delivery performance, it is important that all due date related decisions in the scheduling system are considered. GP is only used to handle sequencing/scheduling decisions and other decisions such as due date assignment are simplified. This restricts the evolved dispatching rules from adapting to other related decisions to ensure that the scheduling system runs smoothly. It is noted that manually designing different scheduling decisions and investigating different interactions between these decisions are very challenging tasks in the scheduling literature. However, the use of automatic design methods can help overcome these issues and open a new research direction to construct comprehensive scheduling systems.

- Existing GP methods employed in the past studies may not be sufficient to deal with practical issues such as multiple conflicting objective and multiple scheduling decisions. The main reason is the

high complexity of the problems which makes it much more difficult to find effective scheduling rules. For that reason, it is important to utilise special features of the scheduling problems to develop more effective GP methods. The focus should be on developing new evolutionary search mechanisms that can effectively find competitive scheduling rules. Moreover, the representation of each individual rule and sets of scheduling rules needs to be investigated to help reduce the search space that GP will explore.

- The reusability of the evolved rules is an interesting issue in previous studies in automatic design of heuristics. In scheduling problems, the reusability or the extent to which the evolved rules can be used effectively is of great interest for scheduling researchers and production/operation managers. Understanding the reusability of the evolved rules will help decide how rules can be employed given the practical situations of the shops. For example, the shop utilisation often fluctuates over time and it is important to know whether the employed rules are able to cope with different utilisation levels. While examining the reusability of the evolved rules for the single objective problem is quite straightforward, dealing with multiple objectives is much more challenging. In the case of multiple objective problems, the resuability of the evolved rules needs to be examined by checking the Pareto relationships from different objectives instead of each individual objective. This is a crucial issue which needs to be investigated to have a proper assessment of the evolved rules.

The following chapters in this thesis will show how we can employ GP to tackle these issues.

Chapter 3

Program Representations of Dispatching Rules

The purpose of this chapter is to investigate the influence of different representations on the performance of rules evolved by GP. Through these representations, we aim to evolve *adaptive dispatching rules* (ADR) for the static JSS with makespan and total weighted tardiness as objective functions. Basically, an ADR is a combination of different dispatching rules and it is "adaptive" because the master rule will choose a specific dispatching rule to sequence jobs in the queue of a machine based on the status of the machines at each decision making step. Three representations of dispatching rules are proposed and tested with the GP system. Different fitness functions used to measure the performance of an evolved dispatching rule are also compared.

The objectives of this chapter are to:

1. Investigate the performance of the GP system with different types of representations.

2. Compare the performance of the evolved rules with well known dispatching rules both on the training set and the test set.

3. Analyse the performance of the proposed algorithm as well as the evolved rules.

The remainder of this chapter is organised as follows. Sections 3.1 and 3.2 provide detailed descriptions of the proposed representations and fitness functions. Section 3.3 shows the overall GP algorithm employed in this chapter. The experimental setting is presented in Section 3.4 and the results will be shown in Section 3.5. Some insights regarding the proposed method and the evolved rules will be discussed further in Section 3.6. Finally, Section 3.7 concludes the findings of this chapter.

3.1 Representations

Three representations of dispatching rules are considered. The first representation (R_1) provides a way to incorporate machine attributes into the GP program along with simple dispatching rules and the hybrid scheduling strategy (between non-delay and active scheduling). The second representation (R_2) is the traditional arithmetic representation like that employed in [181]; the purpose of this representation is to generate composite dispatching rules. The last representation (R_3) is a combination of R_1 and R_2, in which different composite dispatching rules exist and are logically applied to JSS based on the machine and system attributes.

3.1.1 Decision-tree like representation (R_1)

The key idea of this representation is to provide dispatching rules with the ability to apply different simple dispatching rules based on machine attributes. In this case, dispatching rules are represented in a decision-tree form. To make the rules more readable and explainable, the proposed grammar in Figure 3.1 is used when building the GP programs and performing the genetic operators (e.g. dispatching nodes must contain two

```
Start ::= <action>

<action> ::= <if> | <dispatch>

<if> ::= if <attributetype> <op> <threshold>

then <action> else <action>

<op> ::= ≤ | >

<attributetype> ::= WR | MP | DJ | CMI | CWR | BWR

<threshold> ::= 10%|20%|30%|40%|50%|60%|70%|80%|90%|100%

<dispatch> ::= assign <nondelayfactor> assign <rule>

<nondelayfactor> ::= uniform[0,1]

<rule> ::= FIFO | SPT | LPT | LSO | LRM | MWKR | SWKR | MOPR | EDD | MS | WSPT
```

Figure 3.1: Grammar for the proposed GP system with R_1.

arguments, which are a value of the non-delay factor and a single dispatching rule). Two example rules based on this grammar are shown in Figure 3.2 (it is noted that the numbers in this figure only for the demonstration purpose). In Figure 3.2(a), the rule is SPT which is applied with the non-delay factor $\alpha = 0.084$. The rule in Figure 3.2(b) is a bit more sophisticated. The rule firstly checks the workload ratio WR (the ratio of the total processing times of jobs in the queue to the total processing times of all jobs that have to be processed at the machine) of the considered machine m^* (refer to the generic schedule construction procedure in Figure 2.2 on page 27); if the workload ratio is less than or equal to 20%, dispatching rule SPT is applied with $\alpha = 0.221$; otherwise, dispatching rule FIFO is applied with $\alpha = 0.078$. This rule can be considered as a variant of FIFO/SPT, in which the workload of the machine is used as the switch instead of the waiting times of jobs in the queue. Different from other applications [91, 148] where a single non-delay factor is evolved, the proposed GP system using this representation allows different values of non-delay factors to be employed based on the status of the shop.

In this study, we will consider six attributes which indicate the status of machines in the shop. Let Λ be the set of operations that are planned to

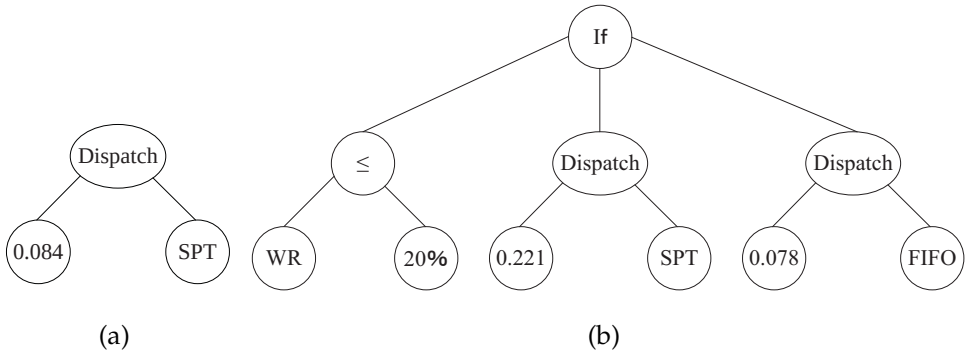


Figure 3.2: Example program trees based on representation R_1.

visit the considered machine m^*, and K and I are the sets of all operations that have and have not yet been processed by m^*, respectively ($\Lambda = K \cup I$). In the shop, we call a machine *critical* if it has the greatest total remaining processing time $\sum_{\sigma \in I} p(\sigma)$ and a machine is called *bottleneck* if it has the largest workload $\sum_{\sigma \in \Omega'} p(\sigma)$ in Ω'. The following definitions of the machine and system attributes are used in this study:

- *Workload ratio, WR* $= \frac{\sum_{\sigma \in \Omega'} p(\sigma)}{\sum_{\sigma \in I} p(\sigma)}$: indicates the workload in Ω' compared to the total remaining workload that m^* has to process (including the operations in the queue and operations that have not yet visited m^*).

- *Machine progress, MP* $= \frac{\sum_{\sigma \in K} p(\sigma)}{\sum_{\sigma \in \Lambda} p(\sigma)}$: indicates the progress of m^*, calculated as the ratio of the total processing time that m^* has processed to the total processing time of all operations in Ω' that have to visit m^*.

- *Deviation of jobs, DJ* $= \frac{\min_{\sigma \in \Omega'}\{p(\sigma)\}}{\max_{\sigma \in \Omega'}\{p(\sigma)\}}$: is a simple ratio of minimum processing time to the maximum processing time of operations in Ω'.

- *Critical machine idleness, CMI*: is the workload ratio WR of the critical machine.

- *Critical workload ratio, CWR* $= \frac{\sum_{\sigma \in \Omega^c} p(\sigma)}{\sum_{\sigma \in \Omega'} p(\sigma}$: is the ratio of the workload of operations in Ω^c to the workload in Ω' where $\Omega^c \subset \Omega'$ is the set of operations belonging to the jobs that have operations that still need to be processed at the critical machine after being processed at m^*.

- *Bottleneck workload ratio, BWR* $= \frac{\sum_{\sigma \in \Omega^b} p(\sigma)}{\sum_{\sigma \in \Omega'} p(\sigma)}$: is the ratio of the workload of operations in Ω^b to the workload in Ω' where $\Omega^b \subset \Omega'$ is the set of operations belonging to the jobs that have operations that still need to be processed at the bottleneck machine after being processed at m^*.

While the first three attributes provide the local status at m^*, the last three attributes indicate the status of the shop with a special focus on the critical and bottleneck machines. The machine and system attributes here appear in the scheduling literature in different forms. There are also other attributes in the literature but they are mainly designed for special manufacturing environments which are not useful (or applicable) for this study. The key difference between our attributes and the attributes used in other studies is that our attributes have been scaled from 0 to 1. The scaled (normalized) attribute values aim to enhance the generality of the evolved rules and also make the evolved rules easier to understand. For example, two scheduling instances can have very different processing times. If the machine progress is important for our scheduling decisions, it is very difficult to design a general rule for two instances with the unnormalised value $\sum_{\sigma \in K} p(\sigma)$ (total processing time that a machine has processed). Also, reading the rule with normalised attributes are much easier (e.g. "50% of workload has been done" rather than "100 processing hours have been done").

Jakobovic and Budin [86] employed attributes similar to ours without normalization (e.g. remaining work at the machine is similar to workload ratio in our study). The definition of attributes for bottleneck and critical machines are adapted from the bottleneck concept [82] for static problems

and these attributes are used to adjust the rules to react appropriately to the changes of the shop.

For representation R_1, eleven simple dispatching rules are considered as the candidate rules in the ADR. The aim of these rules is to determine which operation σ in Ω' will be processed next. Let $n(\sigma)$ be the job which operation σ belongs to, $j = n(\sigma)$ and $o_{j,h} = \sigma$ (h is the index of the current operation of job j). The following are brief descriptions of the candidate dispatching rules. Detailed discussion of these rules can be found in [145] and [185].

- *FIFO*: operations are sequenced *first-in-first-out*.
- *SPT*: select the operation with the *shortest processing time* $p(\sigma)$.
- *LPT*: select the operation with the *longest processing time* $p(\sigma)$.
- *LSO*: select the operation belonging to the job that has the *longest subsequent operation* $p(next(\sigma))$.
- *LRM*: select the operation belonging to the job that has the *longest remaining processing time* (excluding the operation under consideration) $\sum_{l=h+1}^{N_j} p(o_{j,l})$.
- *MWKR*: select the operation belonging to the job that has the *most work remaining* $\sum_{l=h}^{N_j} p(o_{j,l})$.
- *SWKR*: select the operation belonging to the job that has the *smallest work remaining* $\sum_{l=h}^{N_j} p(o_{j,l})$.
- *MOPR*: select the operation belonging to the job that has the *largest number of operations remaining* $N_j - h + 1$.
- *EDD*: select the operation belonging to the job that has the *earliest due date* d_j.

- *MS*: select the operation belonging to the job that has the *minimum slack* $MS_j = d_j - \sum_{l=h}^{N_j} p(o_{j,l}) - t$. Value $t = R_{m^*}$ is the time at which the sequencing decision needs to be made.

- *WSPT*: select the operation that has the maximum *weighted shortest processing time* $w_j/p(\sigma)$.

The function set for this representation contains *If*, *Dispatch*, $\leq$,$>$ to help construct the logic of the adaptive dispatching rule as demonstrated by the examples in Figure 3.2. *Dispatch* represents the combination of a single dispatching rule and its corresponding non-delay factor. The non-delay factor is treated as Ephemeral Random Constants (ERC) in GP [103]. The values of the non-delay factor will initially be a random number from 0 to 1. Meanwhile, attribute type, attribute threshold and dispatching rule terminals are randomly chosen from their candidate values as described in the previous section with equal probabilities.

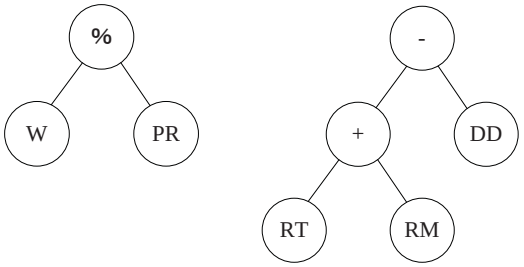
3.1.2 Arithmetic representation (R_2)

For this representation, the focus is to formulate composite dispatching rules that include different pieces of information from jobs and machines. Basically, the GP programs are priority functions that can be used to calculate the priorities for operations in Ω' and the operation with the highest priority will be scheduled to be processed next on the machine m^*. Different from R_1 rules that become more effective by logical choices of single dispatching rules, R_2 rules create their sophisticated behaviour by arithmetically combining various terms into the priority functions. The advantage of this representation is that more information can be directly considered to determine the priorities of operations when sequencing decisions need to be made.

In our GP system, the function set consists of $+$, $-$, $*$, $\%$ (protected division), If, min, max, and abs. The terminal set contains popular terms

Table 3.1: Terminal set for R_2 ($j = n(\sigma)$ and $o_{j,h} = \sigma$)

Notation	Description	Value
RJ	operation ready time	$r(\sigma)$
RO	number of remaining operations of job j	$N_j - h + 1$
RT	work remaining of job j	$\sum_{l=h}^{N_j} p(o_{j,l})$
PR	operation processing time	$p(\sigma)$
W	weight	w_j
DD	due date	d_j
RM	machine ready time	R_{m^*}
#	constant	Uniform[0,1]



(a) (b)

Figure 3.3: Example program trees based on representation R_2.

that are used in existing dispatching rules. The descriptions of the terminals used for calculating the priority of operation σ are shown in Table 3.1. Figure 3.3 shows two simple examples when WSPT and MS are represented by R_2 rules. The non-delay scheduling strategy will be used along with this representation like common applications of composite dispatching rules.

3.1.3 Mixed representation (R_3)

This representation tries to combine the advantages of R_1 and R_2 to create sophisticated adaptive dispatching rules. Within R_3, the incorporation of both system/machine status and composite dispatching rules are considered. The representation R_3 inherits the grammar in R_1 and the composite dispatching rules will be used to calculate the priorities of operations for sequencing decisions besides the use of simple dispatching rules. An example of an R_3 rule is shown in Figure 3.4. The function set for R_3 representation is the combination of function sets of R_1 and R_2.

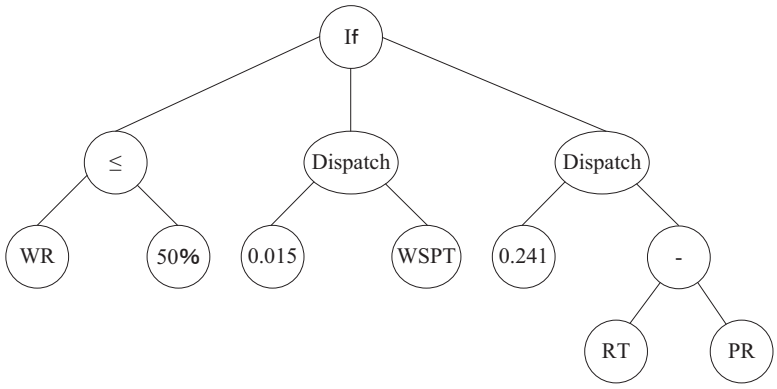


Figure 3.4: An example program tree based on representation R_3.

3.2 Fitness Evaluation

The focus of this study is to learn effective rules for JSS to minimise the makespan or total weighted tardiness. In order to estimate the effectiveness of an evolved dispatching rule, it will be applied within the construction procedure in Figure 2.2 (page 27) to solve a set of instances in the training set and the resulting objective values from all instances are recorded. Since the objective values obtained by a dispatching rule Δ for each instance are very different, we will measure the quality of an ob-

tained schedule by the relative deviation of its objective value from its corresponding reference objective value as shown in equation (3.1).

$$dev(\Delta, I_n) = \frac{Obj(\Delta, I_n) - Ref(I_n)}{Ref(I_n)} \tag{3.1}$$

In this equation, $Obj(\Delta, I_n)$ is the objective value obtained when applying rule Δ to instance I_n, and $Ref(I_n)$ is the reference objective value for instance I_n. The fitness of rule Δ on the training set is calculated by either equation (3.2) or (3.3); $dev_{average}(\Delta)$ and $dev_{max}(\Delta)$ will correspondingly measure the average performance and worst-case performance of Δ across T instances in the data set.

$$dev_{average}(\Delta) = \frac{\sum_{I_n \in \{I_1, \ldots, I_T\}} dev(\Delta, I_n)}{T} \tag{3.2}$$

$$dev_{max}(\Delta) = max_{I_n \in \{I_1, \ldots, I_T\}} \{dev(\Delta, I_n)\} \tag{3.3}$$

The objective of the GP system is to minimise these fitness functions. In the case of $Jm||\mathrm{C}_{max}$, the reference objective value is the lower bound obtained by other approaches (refer to [146] for a list of lower bound values obtained for popular benchmark instances). Since the lower bounds are used in this case, the fitness values for the GP programs are always non-negative. If the fitness value is close to zero, it indicates that the evolved rules can provide near optimal solutions. For $Jm||\sum w_j T_j$, because the lower bound values are not available for all instances, we will use objective values obtained by EDD as the reference objective values for all instances in the data set since it is a widely used dispatching rule for due date related problems. Because EDD is just a simple rule, it can be dominated by more sophisticated rules. For that reason, the fitness value of the GP programs for $Jm||\sum w_j T_j$ can be negative, which means that the evolved rules perform better than EDD with a given fitness function.

3.3 Proposed GP algorithm

Figure 3.5 shows the GP algorithm used in this study to evolve dispatching rules for JSS. The GP system first sets up the training set D and randomly initialises the population. At a generation, each dispatching rule (or individual) Δ_i will be applied to solve all instances in the training set D to find the relative deviation $dev(\Delta_i, I_n)$ for each instance. Then, the fitness value of each rule is calculated by using $dev_{average}(\Delta_i)$. If the evaluated rule is better (has smaller fitness value) than the best rule Δ^*, it will be assigned to the best rule Δ^* and the best fitness value $fitness(\Delta^*)$ is also updated. After all individuals in the population are evaluated, the GP system will apply genetic operators such as reproduction (elitism), crossover and mutation to the programs in the current population to generate new individuals for the next generation. More details of the genetic operators used in this study will be provided in the next section. When the maximum number of generations is reached, the GP algorithm will stop and return the best found rule Δ^*, which will be applied to the test set to evaluate the performance of the GP system.

3.4 Design of Experiments

This section discusses the configuration of the GP system and the data sets used for training and testing.

3.4.1 Parameter setting

The GP system for learning ADRs is developed based on the ECJ20 library [117] (a java-based evolutionary computation research system). The parameter settings of the GP system used in the rest of this paper are shown in Table 3.2. The parameters in this table are commonly used for GP [103, 15] and also show good results from our pilot experiments. The

Inputs: training instances $D \leftarrow \{I_1, I_2, \ldots, I_T\}$
Output: the best evolved rule Δ^*

randomly initialise the population $P \leftarrow \{\Delta_1, \ldots, \Delta_S\}$
set $\Delta^* \leftarrow null$ and $fitness(\Delta^*) = +\infty$
$generation \leftarrow 0$
while $generation \leq maxGeneration$ **do**
    for all $\Delta_i \in P$ **do**
        for all $I_n \in D$ **do**
            $dev(\Delta_i, I_n) \leftarrow$ solve I_n with Δ_i
        end for
        evaluate $fitness(\Delta_i) \leftarrow dev_{average}(\Delta_i)$
        if $fitness(\Delta_i) < fitness(\Delta^*)$ **then**
            $\Delta^* \leftarrow \Delta_i$
            $fitness(\Delta^*) \leftarrow fitness(\Delta_i)$
        end if
    end for
    $P \leftarrow$ apply reproduction, crossover, mutation to P
    $generation \leftarrow generation + 1$
end while
return Δ^*

Figure 3.5: GP algorithm to evolve dispatching rules for JSS.

population size of 1000 is used to ensure that there is enough diversity in the population. The initial GP population is created using the ramped-half-and-half method [103]. Since the R_1 and R_3 rules are created based on the grammar in Figure 3.1, we use strongly typed GP to ensure that the GP nodes will provide proper return types as determined by the grammar. In this case, the crossover and mutation operators of GP are only allowed if they do not violate the grammar. For crossover, the GP system uses the subtree crossover [103], which creates new individuals for the next generation by randomly recombining subtrees from two selected

parents. Meanwhile, mutation is performed by subtree mutation [103], which randomly selects a node of a chosen individual and replaces the subtree rooted at that node by a newly randomly-generated subtree. The combinations of three levels of crossover rates and mutation rates will be tested in our experiments to examine the influence of these genetic operators on the performance of GP. When generating random initial programs or applying crossover/mutation, the maximum depth of eight is used to restrict the program from becoming too large. Greater maximum depths can also be used here to extend the search space of GP; however, we choose this maximum depth to reduce the computational times of the GP system and make the evolved rules easier to analyse. Tournament selection with the tournament size of seven is used to select individuals for genetic operations [103].

3.4.2 Data sets

There are many data sets in the JSS literature which are generated by different scheduling researchers [111, 5, 179, 53] to measure the performance of different heuristic and optimisation methods. The instances in these data sets are still very useful because they include a wide range of instances with different levels of difficulty. Moreover, lower bounds for these instances are available and can be used to calculate the fitness of the evolved dispatching rules as described in Section 3.2. Descriptions of the data sets used for the experiments are shown in Table 3.3 ($\mathcal{N}$ is the number of jobs and $\mathcal{M}$ is the number of machines). In this study, we combine these data sets and distribute them into the training set and test set used by the proposed GP system. The training set and the test set are created to include halves of the instances of each individual data set in Table 3.3. In particular, the training set will contain {la01, la03, ..., la39}, {orb01, ..., orb09},{ ta01, ..., ta79}, {dmu01, ..., dmu79}. The other (even index) instances will be included in the test set. This allows a fair distribution of

Table 3.2: Parameters of the proposed GP system

Population Size	1000
Crossover rate	95%,90%,85%
Mutation rate	0%,5%,10%
Reproduction rate	5%
Generations	50
Max-depth	8
Tournament size	7
Function set (R_1)	*If, Dispatch,* $\leq$,$>$
Terminal set (R_1)	*attribute type, attribute threshold, non-delay factor, dispatching rule*
Function set (R_2)	$+, -, *, \%$, If, min, max, abs
Terminal set (R_2)	*as shown in Table 3.1*
Function set (R_3)	*Function set (R_1) and Function set (R_2)*
Terminal set (R_3)	*Terminal set (R_1) and Terminal set (R_2)*
Fitness	$dev_{average}(\Delta)$

Table 3.3: JSS data sets

Data set	Notation	# of inst.	Size ($\mathcal{N} \times \mathcal{M}$)	Reference
LA	la01-la40	40	10×5 to 15×15	Lawrence [111]
ORB	orb01-orb10	10	10×10	Applegate and Cook [5]
TA	ta01-ta80	80	15×15 to 100×20	Taillard [179]
DMU	dmu01-dmu80	80	20×15 to 50×20	Demirkol et al. [53]

problems with different instance sizes into both the training set and the test set. There are 105 instances in each of the training set and test set. For the case of $Jm||\sum w_jT_j$, the due dates for jobs in each instance will be generated (following Baker [12]) by a due date assignment rule:

$$d_j = r_j + h \times \sum_{l=1}^{N_j} p(o_{j,l}) \tag{3.4}$$

The parameter h is used to indicate the tightness of due dates. We choose $h = 1.3$ for all instances in the training set and test set because it is the common value used in previous research [105, 194]. For the weights w_j of jobs, we employ the $4 : 2 : 1$ rule which has been used in [105] and [194]. This rule is inspired by Pinedo and Singer [152] when their research showed that 20% of the customers are very important, 60% are of average importance and the remaining 20% are of less importance. For that reason, in $Jm||\sum w_j T_j$, the weight of 4 is assigned to the first 20% of jobs, the next 60% are assigned a weight of 2 and the last 20% of jobs are assigned a weight of 1.

3.5 Influence of different representations

The proposed GP systems, with different settings, will be applied to evolve new dispatching rules. This section shows the results obtained from the GP system with three representations, three levels of crossover/mutation rates, two fitness functions and two objectives. In total, we need to run $3 \times 3 \times 2 \times 2 = 36$ experiments. For each experiment, 30 independent runs are performed with different random seeds. Table 3.4 and Table 3.6 show the means and standard deviations of fitness values obtained from all experiments on the training set and test set. The upper part and lower part of each table show the statistics of $dev_{average}(\Delta)$ and $dev_{max}(\Delta)$ when they are used as the fitness function for the GP system. The triple $\langle c, m, r \rangle$ indicates the GP parameters used in a specific experiment. For example, $\langle 85, 10, 5 \rangle$ represents the experiment where the crossover rate is 85%, the mutation rate is 10% and the reproduction rate is 5%. All statistical tests discussed in this section are the standard Wilcoxon tests and they are considered significant if the obtained p-value is less than 0.05.

3.5.1 Makespan

As shown in Table 3.4, when $dev_{average}(\Delta)$ is used as the fitness function, the evolved rules based on R_1 show a performance close to those obtained by R_2 and R_3 on the training set. It is also noted that the R_1 rules evolved with higher mutation rates are significantly better than those evolved without lower mutation (all p-values < 0.0163) on the training set. Since R_1 rules contain many ERCs, higher mutation rates seem quite useful to improve the performance of the GP system with the R_1 representation. However, the performances of R_1 rules are quite poor on the test set. This indicates the overfitting issue of R_1 rules when learning with the training set. The reason for this problem comes from the fact that the candidate rules used in R_1 are too simple, and therefore the rules have to depend strongly on the machine and system statuses to provide better sequencing decisions on the training instances. However, the overuse of the machine and system attributes make R_1 rules less effective when dealing with unseen instances in the test set.

Evolved R_2 rules show a more consistent performance on both the training set and test set. Different from R_1, the statistical tests indicate that the choice of GP parameters does not have significant influence (all p-values > 0.1511) when R_2 is used as the representation of the dispatching rule. These results indicate that mutation is not really useful in this case and the crossover operator is sufficient for the GP system to evolve good individuals. Since R_2 provides only one way to sequence operations, the effectiveness is obtained by good combinations of different terms. Hence, they are also less affected when working with unseen instances like R_1 rules.

Taking the advantages of R_1 and R_2, the evolved R_3 rules show very promising performance. Different from R_1, the incorporation of the machine and system attributes into the R_3 rules is supported by better composite dispatching rules, and therefore the R_3 rules do not need to depend heavily on the use of machine and system attributes to be effective. The

Table 3.4: Performance of evolved rules for $Jm||\mathsf{C}_{max}$ on training set and test set with different settings ($mean \pm standard\ deviation$)

Setting		R_1		R_2		R_3	
		Training	Testing	Training	Testing	Training	Testing
$dev_{average}(\Delta)$	$\langle 95, 0, 5\rangle$	0.188 ± 0.008	0.197 ± 0.007	0.181 ± 0.003	0.187 ± 0.004	0.183 ± 0.005	0.186 ± 0.005
	$\langle 90, 5, 5\rangle$	0.181 ± 0.003	0.192 ± 0.005	0.181 ± 0.003	0.188 ± 0.005	0.181 ± 0.005	0.184 ± 0.006
	$\langle 85, 10, 5\rangle$	0.180 ± 0.003	0.191 ± 0.006	0.180 ± 0.003	0.188 ± 0.004	0.179 ± 0.005	0.184 ± 0.004
$dev_{max}(\Delta)$	$\langle 95, 0, 5\rangle$	0.378 ± 0.011	0.460 ± 0.032	0.381 ± 0.009	0.474 ± 0.050	0.384 ± 0.010	0.457 ± 0.034
	$\langle 90, 5, 5\rangle$	0.372 ± 0.007	0.445 ± 0.025	0.386 ± 0.012	0.461 ± 0.032	0.376 ± 0.011	0.457 ± 0.033
	$\langle 85, 10, 5\rangle$	0.370 ± 0.008	0.457 ± 0.039	0.385 ± 0.009	0.472 ± 0.038	0.376 ± 0.010	0.452 ± 0.028

performance on the test set shows that the evolved R_3 rules also have good generalisation qualities like that of the R_2 rules. Mutation does not affect the GP system with R_3 as strongly as the GP system with R_1. The significant difference is only observed between the experiment with no mutation and the experiment with the mutation rate of 10% (p-value < 0.0042). Although there is no obvious difference in the performance of evolved rules from different configurations on the training set, R_2 rules and R_3 rules (evolved with non-zero mutation rate) are significantly better than R_1 rules on the test set (all p-values < 0.0421 over $(3 + 2) \times 3 = 15$ statistical tests). Also, the evolved R_3 rules from the GP system with non-zero mutation rates are significantly better than R_1 and R_2 rules on the test set (all p-values < 0.017 over $2 \times (3 + 3) = 12$ statistical tests).

When $dev_{max}(\Delta)$ is used as the fitness function, the evolved R_1 rules do not show overfitting issues compared to other representations as shown in the case when $dev_{average}(\Delta)$ is used. This is because the evolved rules tend to focus on hard instances that caused high relative deviations and integrate essential features into the evolved rules to avoid worst-case scenarios. In this case, R_1 and R_3 rules from the GP system with non-zero mutation rates are significantly better than R_2 rules in the training set (all p-values < 0.0213 over $(2 + 2) \times 3 = 12$ statistical tests). This suggests that evolved rules that employ the machine and system attributes are more effective to improve worst-case scenarios for $Jm||\mathrm{C}_{max}$. However, no obvious difference is recorded from the performance of evolved rules on the test set. This indicates that the GP system with $dev_{max}(\Delta)$ as the fitness function does not provide good generalisation for its evolved rules.

Table 3.5 shows the performance of the evolved rules with four selected dispatching rules for $Jm||\mathrm{C}_{max}$. These four rules are selected since they are known to reduce the flowtimes (completion time C_j in static JSS) of jobs. The values in this table are the statistics of relative deviations using equation (3.1) obtained when applying the evolved rules to the instances in the training set and test set. The values of Mean and Max can be calcu-

lated by using equation (3.2) and (3.3) to measure the average and worst-case performance of a dispatching rule on a given set of instances. The values of Min, like Max, indicate the performance of the evolved rule in extreme cases but they are used to show the best case performance instead. From each representation and each fitness function, two evolved rules that show the best fitness in the training stage are used here for comparison. Rule $\Delta_{R_x}^{yvz}$ is the v^{th} best rule that was evolved by using the representation R_x, fitness function z to minimise objective function y for the JSS. For example, $\Delta_{R_2}^{c2a}$ is the second best rule evolved with the representation R_2, $dev_{average}(\Delta)$ as the fitness function (m will be used to indicate $dev_{max}(\Delta)$), and C_{max} as the objective function (t will be used to indicate the total weighted tardiness).

Each simple dispatching rule is used to generate both active and non-delay schedules for all instances in the training set and test set. The results show that LRM with non-delay scheduling strategy is better than other simple rules. The performance of the evolved rules are better than all of the simple dispatching rules on the training set and test set. The results from the evolved rules show that the GP system can find the rules that minimise the fitness function. On the training set, rules evolved by $dev_{average}(\Delta)$ show better average relative deviations than those evolved with $dev_{max}(\Delta)$ as the fitness function. On the other hand, the rules evolved by $dev_{max}(\Delta)$ show better worst-case relative deviations than those evolved with $dev_{average}(\Delta)$ as the fitness function. However, not all evolved rules can produce good results when dealing with unseen instances. Rules $\Delta_{R_1}^{c2a}$ and $\Delta_{R_3}^{c2a}$ are the rules that provide the best performance on the test set even though they are not the best rules on the training set. The two R_1 rules show the best worst-case performance on the training set and test set among rules evolved with $dev_{max}(\Delta)$. These results again suggest that the incorporation of machine and system attributes plays an important role in improving the worst-case performance. Generally, it seems quite difficult to develop a rule that produces good average and worst-case perfor-

Table 3.5: Relative deviations obtained by evolved rules and other rules for $Jm||C_{max}$

Rules	If	Training Set			Test Set		
		Min	Mean	Max	Min	Mean	Max
FIFO	Active	0.012	0.325	0.691	0.000	0.325	0.654
	Non-Delay	0.012	0.325	0.691	0.000	0.325	0.654
SPT	Active	0.322	0.694	1.252	0.316	0.711	1.091
	Non-Delay	0.029	0.292	0.576	0.092	0.312	0.664
LRM	Active	0.020	0.321	0.745	0.000	0.319	0.723
	Non-Delay	0.000	0.224	0.556	0.000	0.225	0.529
MWKR	Active	0.047	0.323	0.736	0.000	0.328	0.713
	Non-Delay	0.000	0.253	0.590	0.000	0.254	0.584
$dev_{average}(\Delta)$	$\Delta_{R_1}^{c1a}$	0.000	0.173	0.448	0.000	0.192	0.525
	$\Delta_{R_1}^{c2a}$	0.000	0.174	0.428	0.000	0.183	0.479
	$\Delta_{R_2}^{c1a}$	0.000	0.174	0.479	0.000	0.190	0.442
	$\Delta_{R_2}^{c2a}$	0.000	0.175	0.457	0.000	0.191	0.446
	$\Delta_{R_3}^{c1a}$	0.000	0.171	0.490	0.000	0.185	0.572
	$\Delta_{R_3}^{c2a}$	0.000	0.172	0.447	0.000	0.177	0.428
$dev_{max}(\Delta)$	$\Delta_{R_1}^{c1m}$	0.000	0.194	0.356	0.000	0.193	0.416
	$\Delta_{R_1}^{c2m}$	0.000	0.190	0.361	0.000	0.189	0.412
	$\Delta_{R_2}^{c1m}$	0.023	0.220	0.362	0.000	0.219	0.482
	$\Delta_{R_2}^{c2m}$	0.000	0.187	0.367	0.000	0.198	0.420
	$\Delta_{R_3}^{c1m}$	0.000	0.185	0.360	0.000	0.184	0.457
	$\Delta_{R_3}^{c2m}$	0.000	0.179	0.361	0.000	0.185	0.478

mance. One explanation is that the rules evolved with $dev_{max}(\Delta)$ only aim at a group of hard instances (which cause high relative deviations) and cannot capture the general useful features. Meanwhile, the rules evolved with $dev_{average}(\Delta)$ focus on the overall performance, therefore they may ignore some extreme cases. Evolved rule $\Delta_{R_3}^{c2a}$ is one rare case when the best average performance and very good worst-case performance is achieved.

3.5.2 Total weighted tardiness

According to Table 3.6, the experiments with R_3 as the representation can produce rules with better fitness than experiments with R_1 or R_2 as the representation. When $dev_{average}(\Delta)$ is used as fitness, the statistical tests show that evolved R_3 rules from the GP system with non-zero mutation rates are significantly better than other evolved R_1 and R_2 rules on both training set and test set (all p-values < 0.0007 over $2 \times (3 + 3) \times 2 = 24$ statistical tests). These results again confirm the effectiveness of the R_3 representation. Also, the R_1 rules from the GP system with non-zero mutation rates are significantly better than R_2 rules in this case (all p-values $< 2 \times 10^{-16}$ over $2 \times 3 \times 2 = 12$ statistical tests). This suggests that the machine attributes and scheduling strategies are quite important when dealing with $Jm||\sum w_j T_j$. When $dev_{max}(\Delta)$ is used as the fitness function, the evolved rules cannot easily dominate EDD because the obtained $dev_{average}(\Delta)$ are not significantly smaller than zero. This observation shows that $Jm||\sum w_j T_j$ is a very hard objective for this problem and it is very difficult to create a dispatching rule that is generally better than other dispatching rules.

The comparison of the evolved rules and other dispatching rules are shown in Table 3.7. It is easy to see that non due date related rules such as FIFO and LRM have a very poor performance on $Jm||\sum w_j T_j$ even though LRM achieves good performance on $Jm||C_{max}$. MS and WSPT show better performance than FIFO and LRM because information about the due date and the weight of a job is considered in these rules. While MS is still much worse than EDD, WSPT shows good performance even though it still cannot totally beat EDD. Sophisticated due date related rules W(CR+SPT), W(S/RPT+SPT), COVERT and ATC (see [185, 125] for a detailed description of these rules) which have not been included in the list of candidate dispatching rules are also presented here. The expected waiting time in COVERT and ATC is calculated based on the standard method [185] in which the expected waiting time $W = b \times PR$, where PR is the

Table 3.6: Performance of evolved rules for $Jm||\sum w_jT_j$ on training set and test set with different settings ($mean \pm standard\ deviation$). Negative values mean the evolved rules are better than EDD

Setting		R_1		R_2		R_3	
		Training	Testing	Training	Testing	Training	Testing
$dev_{average}(\Delta)$	$\langle 95,0,5\rangle$	-0.227 ± 0.004	-0.221 ± 0.005	-0.216 ± 0.003	-0.203 ± 0.005	-0.240 ± 0.015	-0.233 ± 0.015
	$\langle 90,5,5\rangle$	-0.235 ± 0.006	-0.223 ± 0.003	-0.216 ± 0.004	-0.205 ± 0.006	-0.245 ± 0.012	-0.233 ± 0.014
	$\langle 85,10,5\rangle$	-0.237 ± 0.006	-0.222 ± 0.003	-0.216 ± 0.004	-0.204 ± 0.006	-0.247 ± 0.013	-0.235 ± 0.013
$dev_{max}(\Delta)$	$\langle 95,0,5\rangle$	0.002 ± 0.016	0.108 ± 0.139	0.000 ± 0.000	0.000 ± 0.000	0.006 ± 0.030	0.167 ± 0.148
	$\langle 90,5,5\rangle$	-0.001 ± 0.017	0.061 ± 0.085	0.000 ± 0.001	0.007 ± 0.036	-0.009 ± 0.026	0.132 ± 0.115
	$\langle 85,10,5\rangle$	-0.001 ± 0.008	0.018 ± 0.049	0.000 ± 0.000	0.000 ± 0.000	-0.017 ± 0.033	0.143 ± 0.125

Table 3.7: Relative deviations obtained by evolved rules and other rules for $Jm||\sum w_jT_j$

Rules		Training Set			Test Set		
		Min	Mean	Max	Min	Mean	Max
FIFO	Active	−0.318	0.455	1.600	−0.159	0.442	1.196
	Non-Delay	−0.318	0.455	1.600	−0.159	0.442	1.196
LRM	Active	0.193	0.832	2.129	−0.125	0.836	1.813
	Non-Delay	−0.189	0.507	1.571	−0.236	0.494	1.425
MS	Active	0.106	0.601	1.519	0.035	0.607	1.321
	Non-Delay	−0.040	0.387	1.205	−0.133	0.363	0.822
WSPT	Active	−0.133	0.148	0.874	−0.154	0.160	0.946
	Non-Delay	−0.394	−0.169	0.168	−0.459	−0.161	0.253
W(CR+SPT)	Active	−0.133	0.140	0.670	−0.234	0.147	0.858
	Non-Delay	−0.398	−0.173	0.177	−0.459	−0.165	0.435
W(S/RPT+SPT)	Active	−0.110	0.149	0.867	−0.154	0.161	0.853
	Non-Delay	−0.394	−0.168	0.168	−0.459	−0.161	0.253
COVERT	Active	−0.171	0.146	0.722	−0.235	0.147	0.853
	Non-Delay	−0.394	−0.173	0.177	−0.459	−0.160	0.253
ATC	Active	−0.258	0.145	0.757	−0.260	0.140	0.795
	Non-Delay	−0.394	−0.168	0.168	−0.459	−0.163	0.253
$dev_{average}(\Delta)$	$\Delta_{R_1}^{t1a}$	−0.586	−0.247	0.020	−0.560	−0.220	0.150
	$\Delta_{R_1}^{t2a}$	−0.531	−0.244	0.003	−0.507	−0.224	0.018
	$\Delta_{R_2}^{t1a}$	−0.467	−0.223	−0.003	−0.517	−0.199	0.326
	$\Delta_{R_2}^{t2a}$	−0.529	−0.223	0.151	−0.523	−0.206	0.116
	$\Delta_{R_3}^{t1a}$	−0.506	−0.265	−0.033	−0.616	−0.246	−0.002
	$\Delta_{R_3}^{t2a}$	−0.581	−0.265	−0.022	−0.555	−0.253	0.031
$dev_{max}(\Delta)$	$\Delta_{R_1}^{t1m}$	−0.469	−0.212	−0.043	−0.492	−0.211	0.129
	$\Delta_{R_1}^{t2m}$	−0.493	−0.206	−0.041	−0.559	−0.203	0.121
	$\Delta_{R_2}^{t1m}$	−0.456	−0.170	−0.005	−0.490	−0.177	0.198
	$\Delta_{R_2}^{t2m}$	0.000	0.000	0.000	0.000	0.000	0.000
	$\Delta_{R_3}^{t1m}$	−0.487	−0.221	−0.098	−0.448	−0.202	0.086
	$\Delta_{R_3}^{t2m}$	−0.497	−0.215	−0.072	−0.514	−0.212	0.065

operation processing time. For each method, two parameters needed to be specified which are k (look-ahead parameter) and b. 25 combinations of $b \in \{0.5, 1.0, 1.5, 2.0, 2.5\}$ and $k \in \{1.5, 2.0, 2.5, 3.0, 3.5\}$ are examined on the training set, and the combination that gives the best average performance is selected for comparison ($k = 2.5$, $b = 2.5$ for COVERT and $k = 1.5$, $b = 0.5$ for ATC). The performance of these two rules are quite good since they are customised to deal with weighted tardiness problems. However, in the worst case, they still cannot provide better schedules than those obtained by EDD.

On both the training set and the test set, all of these GP evolved rules show better average performance than the existing rules (except for the R_2 rules evolved with $dev_{max}(\Delta)$ as the fitness function). Similar to what has already been stated, R_1 and R_3 rules are also much better than R_2 rules. However, it is still not easy to find a rule that totally dominates EDD. Among all the evolved rules, the evolved R_3 rules are the most promising ones. The best two evolved R_3 rules obtained very good average relative deviations for both training set and test set compared to those obtained by R_1 and R_2 rules. Rule $\Delta_{R_3}^{t1a}$ is also the only evolved adaptive dispatching rule that totally dominates EDD on all training and testing instances. Meanwhile, $\Delta_{R_3}^{t2a}$ produces a very good average performance, even better than $\Delta_{R_3}^{t1a}$ on the test set and it is just slightly worse than $\Delta_{R_3}^{t1a}$ in the worst case. The impressive results of R_3 rules again confirm the need for integrating machine and system attributes with sophisticated dispatching rules to generate generalised and effective dispatching rules.

In the case where $dev_{max}(\Delta)$ is used as the fitness function, the evolved rules show a very poor generalisation quality. Their worst-case performance on the test set is sometimes worse than those obtained by rules using $dev_{average}(\Delta)$ as the fitness function. Some promising rules can be found when R_3 is used as the representation. However, these rules are still much worse than their counterparts $\Delta_{R_3}^{t1a}$ and $\Delta_{R_3}^{t2a}$. This suggests that $dev_{average}(\Delta)$ is better than $dev_{max}(\Delta)$ as the fitness function.

3.6 Analysis and Discussions

In this section, we will further investigate the evolved rules to figure out how they can produce good performance. An analysis on the components of the evolved rules is also given in order to gain more understanding of each factor within the proposed representations that may influence the ability of the GP system to generate better rules. A comparison between the evolved rules and some meta-heuristic approaches is then provided. Finally, the dispatching rules evolved by the proposed GP system are tested under a simulated JSS dynamic environment.

3.6.1 Insights from the evolved rules

In the previous experiment, there are 1080 rules evolved with different GP parameters and representations for $Jm||\mathrm{C}_{max}$ and $Jm||\sum w_jT_j$. The performance of the two best evolved rules for each representation and each objective function of JSS were shown in Table 3.5 and Table 3.7. As an example, we pick one evolved rule from each representation among these rules based on their overall performance on the training set and test set for further analysis (other evolved rules have a similar pattern).

Evolved rules for $Jm||\mathbf{C}_{max}$

Here, $\Delta^{c1a}_{R_1}$, $\Delta^{c1a}_{R_2}$ and $\Delta^{c2a}_{R_3}$ are chosen for analysis. The detailed representations of these rules are shown in Figure 3.6. For $\Delta^{c1a}_{R_1}$ (Figure 3.6(a)), the first observation is that even though the rule looks complicated, it is just a combination of four simple dispatching rules (LRM, SPT, LPT and WSPT) and three machine attributes (CMI, CWR, DJ). Since WSPT is less relevant to $Jm||\mathrm{C}_{max}$ and LPT is not very effective in this case, they only appear once in the entire adaptive dispatching rule. The root condition of $\Delta^{c1a}_{R_1}$ checks the critical machine idleness and it is noted that LRM is the main dispatching rule when the idleness of the critical machine is greater

than 10%. When CMI is small, the rule is more complicated and rules that favour small processing time operations like SPT and WSPT occur more in this case. This rule suggests that when the critical machine seems to be idle, the considered machine should focus on completing operations with small processing times in order to feed more work to the critical machine and keep it busy; otherwise LRM should be used to prevent certain jobs from being completed so late and increase the makespan.

Different from $\Delta_{R_1}^{c1a}$, $\Delta_{R_2}^{c1a}$ (Figure 3.6(b)) is a pure mathematical function. In order to make it easy for analysis, we will simplify the whole function by eliminating terms which appear to be less relevant. The simplification step is as follows:

$$\begin{aligned}\Delta_{R_2}^{c1a} &= \tfrac{(PR+RM)W}{PR} - \tfrac{(RJ+RM)PR}{RT} + RT + \tfrac{DD}{(PR+RM)}RT\tfrac{W}{(PR^2)} + \tfrac{RT}{PR}(RJ+RM) \\ &\approx RT + \tfrac{RT}{PR}\tfrac{DD}{(PR+RM)}\tfrac{W}{(PR)} + \tfrac{RT}{PR}(RJ+RM) \\ &\approx RT + k \times \tfrac{RT}{PR}\end{aligned}$$

Since the first and second terms of $\Delta_{R_2}^{c1a}$ do not make much sense in this case, we just drop them from the priority function. The rest of the function can be grouped in two parts. The first part has RT, like rule MWKR, and the second part contains $\frac{\mathrm{RT}}{\mathrm{PR}}$, which is a combination of SPT and MWKR. When considering other terms in the second part as a constant k, we have the approximation of $\Delta_{R_2}^{c1a}$ as a linear combination of MWKR and SPT/MWKR. This rule is actually not new. If we omit RT in the approximation function, the rest is known as shortest processing time by total work (SPTtwk) rule in the literature [43].

Rule $\Delta_{R_3}^{c2a}$ (see Figure 3.6(c)) is the most interesting rule in this case because both arithmetic rules and simple dispatching rules are employed. However, with the condition that MP has to be less than 100% at the second level, we can totally eliminate the sub-trees that contain the simple dispatching rules. After some simplification steps, it is also noted that the arithmetic rules are also variants of SPTtwk. With the support of the

```
(IF (> CMI 10%)
 (IF (> CWR 20%)
 (IF (> CWR 80%) (DISPATCH 0.131 LRM)
  (IF (≤ DJ 30%) (DISPATCH 0.198 SPT) (DISPATCH 0.102 LRM)))
 (IF (> CWR 10%) (DISPATCH 0.102 LRM) (DISPATCH 0.131 LRM)))
(IF (> CWR 10%)
 (IF (> CWR 80%) (DISPATCH 0.014 WSPT)
  (IF (≤ DJ 30%) (DISPATCH 0.198 SPT) (DISPATCH 0.131 LRM)))
 (IF (> CWR 80%) (DISPATCH 0.830 LPT)
  (IF (≤ DJ 20%) (DISPATCH 0.198 SPT) (DISPATCH 0.102 LRM)))))
```

(a) Evolved rule $\Delta_{R_1}^{c1a}$

```
(+ (+ (-
        (*(+ PR RM) (/ W PR))
        (/(+ RJ RM) (/ RT PR)))
     RT)
 (-(*
        (/ DD(+ PR RM))
        (*(/ RT PR) (/ W PR)))
 ((-1*) (*(/ RT PR) (+ RJ RM)))))
```

(b) Evolved rule $\Delta_{R_2}^{c1a}$

```
(IF (> CWR 90%)
 (DISPATCH 0.069 (/ (* (+ RJ 0.594) (+ RT PR)) (+ W PR)))
 (IF (≤ MP 100%) (DISPATCH 0.128 (/ (- RT PR) (+ W PR)))
 (IF (≤ CWR 100%) (IF (≤ MP 100%)
  (DISPATCH 0.166 WSPT) (DISPATCH 0.282 LPT))
  (IF(≤ MP 100%)(DISPATCH 0.044 LRM) (DISPATCH 0.736 FIFO)))))
```

(c) Evolved rule $\Delta_{R_3}^{c2a}$

Figure 3.6: Selected evolved rules for $Jm||C_{max}$.

machine attribute, the obtained arithmetic rules become much easier to analyse. The simplified version of $\Delta_{R_3}^{c2a}$ is as follows:

$$\Delta_{R_3}^{c2a} = \begin{cases} \langle \frac{(RJ+0.594845)(RT+PR)}{W+PR}, \alpha = 0.069 \rangle & \text{if } CWR > 90\% \\ \langle \frac{(RT-PR)}{(W+PR)}, \alpha = 0.128 \rangle & \text{otherwise} \end{cases}$$

$$\approx \begin{cases} \langle \frac{(RJ+0.594845)(RT+PR)}{PR}, \alpha = 0.069 \rangle & \text{if } CWR > 90\% \\ \langle \frac{(RT-PR)}{PR}, \alpha = 0.128 \rangle & \text{otherwise} \end{cases}$$

The notation $\langle \cdot, \cdot \rangle$ indicates the dispatching rule and α value as in the middle and right subtrees of Figure 3.4. Although both priority functions of $\Delta_{R_3}^{c2a}$ are variants of SPTtwk, the non-delay factor α for the case is smaller when CWR $> 90\%$. One explanation is that when Ω' contains many critical operations, it is reasonable to start the available operations right away instead of waiting for the operations that will be ready after the ready time of m^*.

Evolved rules for $Jm||\sum w_j T_j$

Here, $\Delta_{R_1}^{t1a}$, $\Delta_{R_2}^{t2a}$ and $\Delta_{R_3}^{t1a}$ are selected to represent the evolved rules for $Jm||\sum w_j T_j$. The three full rules obtained by the GP are shown in Figure 3.7. For $\Delta_{R_1}^{t1a}$, it is quite interesting that this rule can obtain such a good result (as shown in Table 3.7) without any due date related components. The two main simple dispatching rules used in this case are WSPT and LPT. While WSPT can be considered as a suitable rule for $Jm||\sum w_j T_j$, it does not make sense to include LPT in this case. The result when we replace LPT by WSPT shows that the refined rule can still produce the results as good as $\Delta_{R_1}^{t1a}$. For this reason, the contribution for the success of the rule comes from WSPT and other factors instead of the combination of different rules as we observed in the previous section. It is noted that most values of α in this case are about 0.4. Using these values for the WSPT alone shows that WSPT with appropriate choice of α can produce the results much better than the case when the non-delay scheduling strategy is used. In this rule, the contribution of the machine and system attributes are not very

significant and they are mainly employed to improve the worst-case performance.

```
(IF (>DJ 80%)
 (IF (>BWR 90%)
 (IF (≤ DJ 90%) (DISPATCH 0.426 WSPT)
 (IF (≤ MP 10%) (DISPATCH 0.436 WSPT) (DISPATCH 0.364 LPT)))
  (DISPATCH 0.065 WSPT))
 (IF (> BWR 20%)
 (IF (≤ DJ 30%) (DISPATCH 0.436 WSPT)
  (IF (≤ DJ 30%) (DISPATCH 0.364 LPT) (DISPATCH 0.389 WSPT)))
 (IF (≤ DJ 30%) (DISPATCH 0.436 WSPT)
  (IF (> DJ 80%) (DISPATCH 0.364 LPT) (DISPATCH 0.181 WSPT)))))
```

(a) Evolved rule $\Delta_{R_1}^{t1a}$

```
(- (+ (*
        (* PR (* 0.614577 PR))
        (- ((-1*) RM) (/ RM W)))
      ((-1*) (* (* RT PR) (/ RT W))))
   (+ (-
        (/ (* RT PR) (- W 0.5214191))
        (* (/ RM W) (* 0.614577 PR)))
      (* (/ (* RT PR) (- W 0.5214191)) (+ (/ RM W) (/ RM W)))))
```

(b) Evolved rule $\Delta_{R_2}^{t2a}$

```
(IF (≤ DJ 50%)
      (DISPATCH 0.331 ((-1*) (* DD (/ PR W))))
      (DISPATCH 0.163 ((-1*) (* (/ DD W) (/ RT W)))))
```

(c) Evolved rule $\Delta_{R_3}^{t1a}$

Figure 3.7: Selected evolved rules for $Jm||\sum w_jT_j$.

For $\Delta_{R_2}^{t2a}$, we perform the following steps to simplify the evolved rule:

$$\Delta_{R_2}^{t2a} = 0.614PR^2(-RM-\tfrac{RM}{W})-RT\times PR\times\tfrac{RT}{W}-(RT\tfrac{PR}{W}-0.521)-0.614\tfrac{RM}{W}PR+$$

$$
\begin{aligned}
&2RT\tfrac{PR}{(W-0.5214191)}\tfrac{RM}{W} \\
&\quad \approx -0.614RM \times PR^2(1+\tfrac{1}{W}) - RT\tfrac{PR}{W}(RT + 2\tfrac{RM}{(W-0.5214191)}) \\
&\quad \approx -k_1 \times PR^2(1+\tfrac{1}{W}) - k_2 \times RT\tfrac{PR}{W}
\end{aligned}
$$

The simplified rule is a linear combination of two sophisticated variants of WSPT where the first part includes PR^2 instead of PR and the second part includes RT. Repeating the experiment on this simplified rule shows that it can perform better than sophisticated rules like ATC and COVERT (just slightly different from the full rule) regarding the average relative deviation with appropriate choice of k_1 and k_2 (with $k_2 > k_1$, similar to the original rule).

It is very surprising that the best evolved rule for $Jm||\sum w_jT_j$ with the R_3 representation is also the smallest rule. Rule $\Delta^{t1a}_{R_3}$ can be formally described as follows:

$$
\Delta^{t1a}_{R_3} = \begin{cases} \langle -\frac{DD \times PR}{W}, \alpha = 0.331 \rangle & \text{if } DJ \leq 50\% \\ \\ \langle -\frac{DD \times RT}{W^2}, \alpha = 0.163 \rangle & \text{otherwise} \end{cases}
$$

The dispatching rule following the first priority function is a combination of EDD and WSPT and it is applied when the deviation of processing times of operations in Ω' is less than 50% (which means that the minimum processing time is less than half of the maximum processing time). When DJ is larger than 50% (which means that the gap between the minimum and maximum processing time is small), RT is used instead of PR and W^2 is used instead of W to increase the priority of jobs with small remaining processing times and high weights.

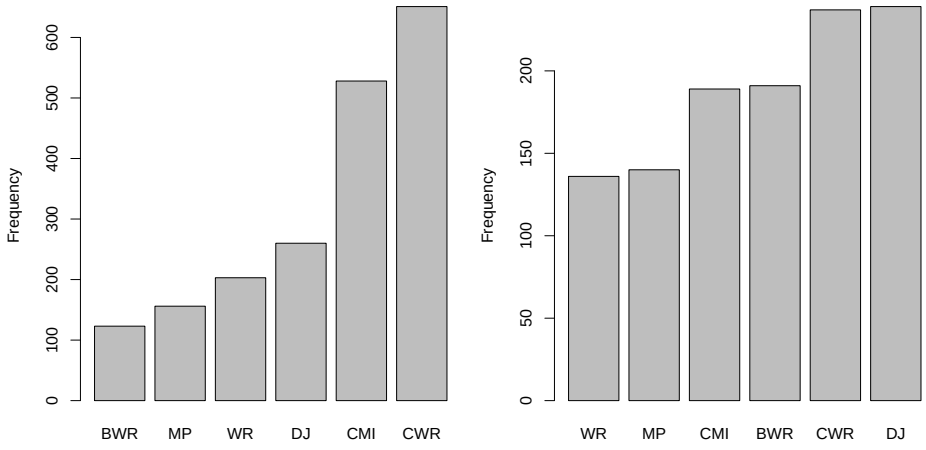
In general, even though the evolved rules can be very complicated sometimes, they contain some good patterns which are very useful to create good dispatching schedules. While R_1 rules can be easily explained with the support of the machine and system attributes, R_2 rules are quite sophisticated and often possess very interesting properties. It is surprising

that the R_3 rules presented here are quite straightforward even thought they contain both machine attributes and composite dispatching rules. Rule $\Delta_{R_3}^{t1a}$ could be quite tricky to represent as a pure mathematical function and it could be more difficult to discover its useful patterns; however, the use of attributes makes it much easier to identify and explain the rule.

3.6.2 Aggregate view of the evolved rules

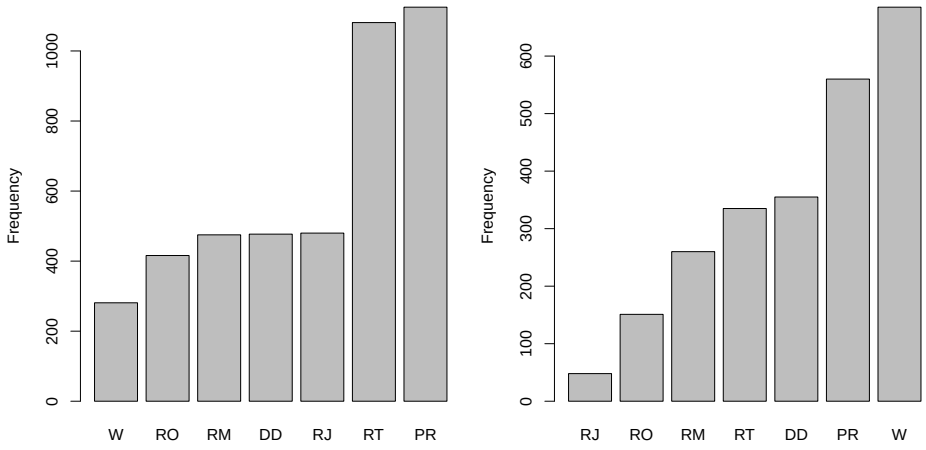
Figure 3.8 shows the frequency that machine and system attributes have occurred in all of the evolved R_1 and R_3 rules. For $Jm||\mathrm{C}_{max}$, critical machine related attributes are the most frequent ones while the bottleneck workload ratio has the lowest frequency. This result indicates that the information related to the critical machine is very important for the construction of a good dispatching rule to minimise the makespan. As shown in the previous section, suitable dispatching rules can be selected based on the idleness of the critical machine as well as the critical workload of m^*. In the case of $Jm||\sum w_jT_j$, the critical machine related attributes are still employed very often, and CWR along with DJ are the ones with the highest occurrence frequency. However, the distribution of attributes in $Jm||\sum w_jT_j$ are more "uniform" than that in $Jm||\mathrm{C}_{max}$. This observation suggests that different attributes should be used to construct good dispatching rules for $Jm||\sum w_jT_j$.

For the priority functions (for composite dispatching rules) within R_2 and R_3 rules, Figure 3.9 shows that RT and PR are the most used terms for $Jm||\mathrm{C}_{max}$. This explains the occurrence of SPTtwk variants in the best rules in the previous section. While W is the least used term for $Jm||\mathrm{C}_{max}$, it is the most used term in $Jm||\sum w_jT_j$. This emphasises the importance of weights for determining the priority of operations in $Jm||\sum w_jT_j$. It is quite surprising that the second most frequent term in evolved rules for $Jm||\sum w_jT_j$ is PR instead of DD. Even though due dates of jobs also depends on the processing times in aggregate form, the results here suggest



(a) $Jm||\mathrm{C}_{max}$ (b) $Jm||\sum w_j T_j$

Figure 3.8: Frequency of attributes in the evolved R_1 and R_3 rules.



(a) $Jm||\mathrm{C}_{max}$ (b) $Jm||\sum w_j T_j$

Figure 3.9: Frequency of terminals in the evolved R_2 and R_3 priority function.

that such local information as PR is still very useful for the dispatching tasks for due date related problems.

3.6.3 Performance in a dynamic JSS environment

This section examines the performance of the evolved rules in a dynamic environment. A simulation model of a simple job shop environment is built for the evaluation of the evolved rules. Table 3.8 shows the parameters of the simulation model. This simulated symmetrical job shop has been widely used in the literature to assess the performance of dispatching rules [81, 162, 78]. With this configuration, the utilisation of each machine $\frac{\lambda}{\mu}$ is equal to the arrival rate λ (since $\mu = 1$). The due date assignment rule is the same as that used in the previous section. Each simulation experiment consists of 30 replications and the *common random seeds* are used for each experiment in order to reduce the variance of the obtained results. The interval from the beginning of the simulation until the arrival of job 50,000 is considered as the warm-up time and all statistics are collected for the next 100,000 jobs. The results obtained from the experiments are shown in Figure 3.10. Since the JSS instances in the data set do not consider the release times of jobs (all release times of jobs are assumed to be zero in all instances) which are very important for the dynamic $Jm||\mathrm{C}_{max}$, the evolved rules for $Jm||\mathrm{C}_{max}$ are not suitable for these experiments. So, only the evolved rules for $Jm||\sum w_jT_j$ presented in Figure 3.7 are investigated here. For ease of presentation, the total weighted tardiness is normalised using the formula proposed by [185]. Thus, the normalised weighted tardiness $= \frac{\sum w_jT_j}{\mathcal{N}\times\mathcal{M}\times\frac{1}{\mu}\times\bar{w}}$, where the number of jobs $\mathcal{N} = 100000$, the number of machines $\mathcal{M} = 6$, the average processing time of an operation $\frac{1}{\mu} = 1$ and average weight of a job $\bar{w} = 2.2$.

In Figure 3.10, the performance of the evolved rules are quite consistent with what has been shown when dealing with the static problem shown in Section 3.5. The results show that simple rules such as FIFO and EDD are easily dominated by the sophisticated rules like ATC, WSPT, COVERT and the evolved rules. It is noted that rule $\Delta^{t1}_{R_1}$ and rule $\Delta^{t1}_{R_3}$ are significantly better (p-value $\ll 0.05$) than existing rules ATC, COVERT and WSPT when the utilisation is less than or equal to 0.8, but they are beaten by the existing

Table 3.8: Parameter setting of the simulation model

Parameter	Value
Number of machines	6
Arrival process	Possion with λ from 0.5 to 0.9
Processing time	Exponential with mean $\frac{1}{\mu} = 1$
Number of operations per job	6
Visiting order of jobs	randomly generated with no machine being revisited
Weight	randomly sampled from $\{4, 2, 1\}$ with the probabilities $\{0.2, 0.6, 0.2\}$

rules when utilisation is 0.9. On the other hand, the R_2 rule is significantly worse than the existing rules for all levels of utilisations. The R_2 rule is significantly worse than R_1 and R_3 rules when the utilisation is less than or equal to 0.8 and better than R_1 when utilisation is 0.9. One of the reasons for the inferiority of the evolved rules when the utilisation of the shop is high is that the evolved rules are trained on a set of static instances which usually reflect the situations in the dynamic job shop with a low utilisation.

The investigation of the evolved rules in the dynamic environment shows that there is still a difference in the characteristics between the static problem and the dynamic problem which sometimes prevent the evolved rules from being effective in the dynamic environment. Although the evolved rules show good results in the dynamic $Jm||\sum w_jT_j$, the fact that they are inferior to the existing rules suggests the lack of generality of the data set used to train the dispatching rules. The deteriorated performance of the evolved rules when utilisations of machines are high is not because the GP system is not able to evolve effective rules but because the static training instances cannot represent all possible scenarios happening in a dynamic environment. This is consistent with the observation from Hilde-

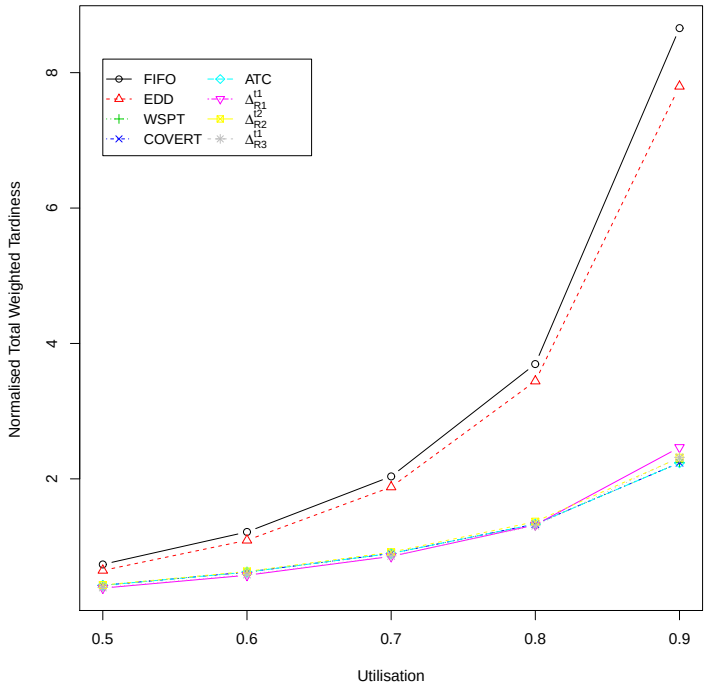


Figure 3.10: Performance of the evolved rules in the dynamic environment (FIFO and EDD curves are above all other curves)

brandt et al. [78]. Thus, if the ultimate objective of the dispatching rules is to be applied to the dynamic environment, it seems reasonable to train the rules in the dynamic situation (e.g. via a simulation model) in order to ensure that the evolved rules can capture all critical features of the dynamic system.

3.6.4 Further Discussion

Most of the proposed GP systems [181, 123, 191, 67] in the literature are very similar to ours when the R_2 representation is used. Miyashita [123] and Jakobovic and Budin [86] are the only works that focus on the use of system attributes, specifically bottleneck machines, to support the sequencing decisions. However, it is noted that the GP system proposed by

Miyashita [123] is only suitable when we try to evolve dispatching rules for a specific shop with a small number of machines in which the bottleneck machine is known. Therefore, it is difficult to generalise the rules based on that GP system. Jakobovic and Budin [86] improved that GP system [123] by using a dedicated GP tree to detect the bottleneck machine to apply suitable dispatching rules. The problem is that the bottleneck machine is not always a good feature to help decide which dispatching rules to be applied. When there are multiple bottleneck machines in the shop (especially with a symmetrical shop), applying a dedicated rule for the detected bottleneck is not very effective since the temporary bottleneck machine can change rapidly before that rule shows any noticeable effect. This explains why the proposed GP-3 system [86] performed significantly better than the simple GP system, but the gaps between the objectives obtained by GP-3 and the simple GP are not large. When examining the GP-3 system with our training and testing instances (using the same terminal sets for dispatching rules in R_2 and R_3), the experimental results show that there is no significant difference between the rules evolved by GP-3 and rules evolved with the R_2 representation (more detailed results are shown in Section 4.4.3 of Chapter 4). The analysis in Section 3.6 also showed that the workload ratio WR (an indicator for the bottleneck level of the considered machine) is not a major attribute in the best evolved rules. This suggests that different machine and system attributes should be used instead of depending solely on bottleneck machines.

One major advantage of the evolved rules is that they are quite ready to be applied to the dynamic environment as shown in the previous subsection, which makes these rules more practical than meta-heuristic methods. Therefore the evolved rules are suitable for shops with rapid dynamic changes [165, 169, 177]. In these shops, even when mathematical programming methods are able to find optimal schedules, these schedules can quickly become sub-optimal or even infeasible under dynamic conditions [169]. Besides, with its flexibility, the proposed GP system can

be easily applied to generate good rules for complex manufacturing processes (e.g. batch processing, assembly station, etc.), which are difficult to be handled by conventional optimisation methods. Another advantage of the evolved dispatching rules is that they are understandable to the users (managers, operators, and researchers), and therefore it is much easier for these people to incorporate these rules with other planning decisions compared to the detailed schedules produced by other methods.

3.7 Chapter Summary

In this chapter, we investigated the use of GP as a hyper-heuristic method for automatically discovering new dispatching rules for the JSS problem. New representations of dispatching rules which take advantage of the machine and system attributes to support the sequencing decisions were developed and examined. The experimental results suggest that the GP system can evolve effective dispatching rules for $Jm||C_{max}$ and $Jm||\sum w_j T_j$ with the representations that integrate system attributes and suitable composite dispatching rules. The evolved rules are also shown to outperform the existing rules on the training set and test set. The simulation experiments have also confirmed the effectiveness of the evolved rules compared to well-known dispatching rules in the dynamic environment. Also, compared to the solutions obtained by the conventional meta-heuristic approaches (as mentioned in Chapter 2.2.2), the evolved rules in this work provide much better interpretability of the sequencing decisions. Another advantage of the proposed GP system is that it only requires one run to generate dispatching rules which can be used to solve multiple problem instances.

This chapter also evaluates rules evolved based on static problem instances on the dynamic environment. It is observed that the evolved rules are most effective in shops with low utilisation levels. One of the reasons is that the evolved rules are trained on a set of static instances which

usually reflect the situations in the dynamic job shop with a low utilisation. Another reason is that the global information such as CWR and BWR can change rapidly in the dynamic shop, especially with a high utilisation level. In this case, the use of global information may be outdated very soon after the sequencing decisions are made, which can cause unexpected consequences. These reasons suggest two important points that need to be considered when GP is used to evolve rules for dynamic job shops. First, rules should be trained based on long term simulation instead of static problem instances to ensure that evolved rules can handle all possible situations. Second, the use of local information (as terminals to construct rules) is more suitable than global information which are very sensitive to the rapid changes in the dynamic shops.

In general, this is the first study that has compared different representations of dispatching rules used in a GP system and investigated how representations influence the performance of the GP system when evolving dispatching rules for the JSS. With their flexibility, the proposed representations in this study can be easily extended to deal with different manufacturing environments. Moreover, these representations provide a convenient way to incorporate special system features of real world environments into the adaptive dispatching rules. Therefore, the GP system proposed in this study can also be employed as a good tool to automatically discover effective dispatching rules for a particular manufacturing system.

However, similar to traditional dispatching rules, schedules constructed by the evolved rules using the proposed representations only depend on myopic information of jobs and machines. Therefore, these evolved rules did not taken into account the future impact of the sequencing decisions. To overcome this drawback, the next chapter proposes a new form of dispatching rules called iterative dispatching rules. The novelty of these (iterative) dispatching rules is that they can iteratively improve the schedules by utilising the information from completed schedules.

Chapter 4

Iterative Dispatching Rules

One of the issues with dispatching rules is that the schedules generated from these rules are often not as good as those obtained from optimisation methods when tested on static JSS instances. Even though this is a trade-off between the quality and the simplicity/understandability of the rules, it is still expected that the gap in quality between these rules and optimisation methods can be reduced to make dispatching rules more attractive. The most straightforward way to deal with this issue is to use different dispatching rules (or complex rules with different parameters) to generate multiple schedules and select the best generated schedule to apply. However, this is just a trial-and-error method and the quality of the obtained schedules depends strongly on the existing rules.

Some priority dispatching rules based meta-heuristic methods [37, 56, 77, 178, 184, 194], e.g., genetic algorithms (GAs), have also been proposed to determine how dispatching rules should be applied. The key idea of these methods is to find the best way to utilise existing dispatching rules or heuristics for JSS problems. Even though some promising results are obtained, these methods do not have good understandability as dispatching rules. Also, they are computationally expensive because many schedules need to be evaluated.

In this chapter, we propose *iterative dispatching rules* (IDR), a new type of dispatching rule for solving JSS problems. Different from traditional dispatching rules (DR) which provide one fixed schedule of jobs, IDR tries to iteratively improve the schedules by taking into account the information of scheduled jobs in previous steps. Therefore, these rules have a chance to correct their mistakes from their previous sequencing decisions. However, developing such rules is quite complicated and difficult using traditional methods because we have to synthesise many pieces of information. Therefore, we will develop a new GP method for automatic design of IDRs. This study will focus on using GP to learn/evolve new IDRs for JSS problems. Three key objectives for this chapter are:

1. Developing a new GP method for evolving IDRs.
2. Comparing the performance of evolved IDRs with evolved composite DRs and existing DRs in the literature.
3. Analysing the evolved IDRs and further improving their performance.

Section 4.1 provides the definition of the proposed IDRs. Development and the evaluation scheme of GP for IDRs are described in Section 4.2. Section 4.3 compares the performance of the evolved IDRs with existing dispatching rules and dispatching rules evolved with the proposed representations in Chapter 3. Different aspects of IDRs are investigated in Section 4.4 to help improve the performance of the evolved IDRs. Section 4.5 proposes a new representation for look-ahead strategies. An example of an evolved IDR is analysed Section 4.6.

4.1 Definition of IDR

A traditional dispatching rule (DR) can be defined as a priority function $\Delta(\mathcal{J}, \mathcal{W})$ which is used to assign a priority for each job $\mathcal{J}$ in the queue of a considered machine $\mathcal{W}$ based on the information from $\mathcal{J}$ (e.g. processing

time, due date) and $\mathcal{W}$ (e.g. ready time). After a priority has been assigned to all jobs in the queue, the one with the highest priority will be processed next. A limitation of these rules is that they can only provide sequencing decisions based on the available information of jobs and machines at the time the decision is made. Since these pieces of information come from a partial schedule, the future impact of the decisions made by a dispatching rule on the complete schedule is not considered in order to enhance the effectiveness of the rule. Different from these dispatching rules, an IDR is defined as $\Delta^I(\mathcal{J}, \mathcal{W}, \mathcal{R})$, where $\mathcal{R}$ is the recorded information of the previous generated schedule. The overall algorithm used to construct a schedule by $\Delta^I(\mathcal{J}, \mathcal{W}, \mathcal{R})$ is shown in Figure 4.1.

At the beginning, an initial recorded information $\mathcal{R}^0$ (more details on how to determine $\mathcal{R}^0$ will be shown in the next section) is assigned to $\mathcal{R}$. Steps from 4 to 12 are used to construct a schedule similar to the procedure in Figure 2.2 (on page 27). After a schedule is obtained, the objective is calculated and compared to the best objective Obj^* obtained in previous iterations. If the schedule obtained is improved, Obj^* will be updated and the information from scheduled jobs is used to update $\mathcal{R}$. This procedure will be repeated until no improvement is realised from the new generated schedule. Different from traditional dispatching rules which only make sequencing decisions based on the partial schedule, IDRs can use the information from a complete schedule to correct the mistakes made in previous iterations. However, designing such a rule would be very challenging using the traditional methods since many pieces of information will need to be considered. In this study, we apply GP to handle this problem by automating the design process.

Regarding the time complexity of IDRs, the maximum number of schedules (or iterations) $maxstep^{IDR}$ generated by IDRs is fewer than $\lceil \frac{Obj_I - LB}{\varepsilon} \rceil$, where Obj_I is the objective of the initial solution, LB is a lower bound of the problem and ε is the smallest improvement in the objective values. Obviously, LB and Obj_I are always finite and ε is always larger than zero

1: $\mathcal{R} \leftarrow \mathcal{R}^0$
2: $Obj^* = +\infty$
3: $\pi \leftarrow \emptyset$ and $\Omega \leftarrow \{o_{1,1}, o_{2,1}, \ldots, o_{N,1}\}$
4: **repeat**
5: let $t(\Omega) = \min_{\sigma \in \Omega}\{\max\{r(\sigma), U_{m(\sigma)}\} + p(\sigma)\}$
6: let σ^* be the operation that minimum is achieved, $m^* \leftarrow m(\sigma^*)$
7: $\Omega^* \leftarrow \{\sigma \in \Omega | m(\sigma) = m^*\}$
8: let $\Omega' = \{\sigma \in \Omega^* | r(\sigma) \leq S(m^*) + \alpha(t(\Omega) - S(m^*))\}$ where $S(m^*) = \max\{\min_{\sigma \in \Omega^*}\{r(\sigma)\}, U_{m^*}\}$
9: apply $\Delta^I(\mathcal{J}, \mathcal{M}, \mathcal{R})$ on Ω' to find the next operation σ' to be scheduled on m^*
10: remove σ' from Ω' and include it into π
11: include $next(\sigma')$ into Ω if $next(\sigma') \neq null$
12: **until** all operations have been scheduled
13: $Obj(\pi) \leftarrow$ calculate the objective value obtained from schedule π
14: **if** $Obj^* > Obj(\pi)$ **then**
15: $Obj^* = Obj(\pi)$
16: $\mathcal{R} \leftarrow get\_information(\pi)$
17: go back to step 3
18: **else**
19: end the procedure, and return $(Obj^*, \mathcal{R})$
20: **end if**

Figure 4.1: Generic procedure to construct a schedule for JSS problems with IDR.

(for all instances with C_{max} as the objective function and integer processing times, ε is always larger than or equal to one). Therefore, $maxstep^{IDR}$ is always a finite value and IDRs can always stop in a finite time.

4.2 Evaluation scheme for IDRs

The same GP algorithm as in Figure 3.5 (on page 76) will be used to evolve IDRs. However, there are some changes in the terminal set and the evaluation scheme of evolved IDRs. The terminal set for IDRs is shown in Table 4.1. Terminals in the upper part are the same as those used by the R_2 representation in Section 3.1. The three terminals in the lower part of this table are the recorded information $\mathcal{R}$ of scheduled jobs from the previous iteration. These terminals provide information about the previous schedule at the job level (RFT) and the operation level (RWT and RNWT). An illustration of how an IDR is represented and evaluated is shown in Figure 4.2.

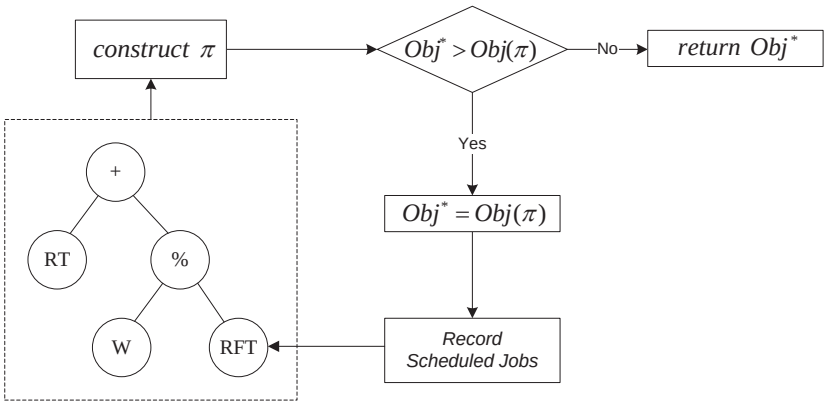


Figure 4.2: Representation and evaluation of an evolved IDR.

In this case, a GP individual will play the role of a $\Delta^I(\mathcal{J}, \mathcal{W}, \mathcal{R})$ to assign the priorities to jobs in the queues, which is step 9 in the schedule construction procedure in Figure 4.1. In the first time when a GP individual is used to construct the schedule, since there has been no recorded

Table 4.1: Terminal set for IDR

Notation	Description
RJ	operation ready time
RO	number of remaining operations
RT	work remaining of job
PR	operation processing time
W	weight
DD	due date
RM	machine ready time
#	constant from Uniform[0,1]
RFT	recorded finish time
RWT	recorded operation waiting time
RNWT	recorded waiting time of the next operation

information $\mathcal{R}$ from the previous schedule, so other pieces of information will be employed as the initial value of RFT, RWT, and RNWT, which represent the initial recorded information $\mathcal{R}^0$. The initial value for RFT is the total processing time of the considered job. For RWT and RNWT, they will take half of the workload (total operation processing time of jobs waiting in the queue) of the considered machine as their initial recorded waiting times. This implies that a job has to wait for half of the jobs in the queue to be finished before it can be processed, given that all of those jobs have the same processing times.

4.3 Comparison with other GP methods

Although the evolved IDRs use a different evaluation scheme compared to the rules in Chapter 3, the representation of IDRs in GP are the same as R_2.

Therefore, Algorithm 3.5 (page 76) can be employed to evolve IDRs. The only difference in this algorithm is that composite dispatching rules Δ_i are replaced by iterative dispatching rules Δ_i^I. Tables 4.2 and 4.3 show the performance of evolved IDRs with the non-delay factor $\alpha = 0$ (adaptive α is presented in Section 4.5) and rules evolved with R_1, R_2, and R_3. All of the rules in this table are evolved using $dev_{average}(\Delta)$ as the fitness function with the crossover rate and mutation rate of 90% and 5%, respectively.

Table 4.2: Performance of evolved rules for $Jm||\mathrm{C}_{max}$

	Data Sets	Min	Mean $\pm$ Stdev.	Max
Δ_{R_1}	Training	0.174	0.181 ± 0.004	0.190
	Testing	0.181	0.192 ± 0.005	0.201
Δ_{R_2}	Training	0.175	0.181 ± 0.003	0.186
	Testing	0.182	0.188 ± 0.004	0.199
Δ_{R_3}	Training	0.173	0.181 ± 0.004	0.187
	Testing	0.176	0.184 ± 0.005	0.195
Δ^I	Training	0.145	0.151 ± 0.004	0.160
	Testing	0.151	0.160 ± 0.005	0.171

These tables show that evolved rules Δ^I are significantly better than other rules evolved by using different representations when dealing with $Jm||\mathrm{C}_{max}$. For $Jm||\sum w_j T_j$, evolved IDRs are significantly better than rules evolved with R_1 and R_2, but not significantly different from rules evolved with R_3. The results from both the training set and the test set are very consistent, which indicates that the improvement of IDRs is made without deteriorating their reusability. A problem with the proposed GP method is that it usually need more time to evaluate evolved IDRs as compared to the evaluation of dispatching rules with R_1, R_2, and R_3 representations because of the iterative construction procedure. From the experi-

Table 4.3: Performance of evolved rules for $Jm||\sum w_jT_j$

	Data Sets	Min	Mean $\pm$ Stdev.	Max
Δ_{R_1}	Training	-0.245	-0.235 ± 0.007	-0.221
	Testing	-0.232	-0.221 ± 0.005	-0.207
Δ_{R_2}	Training	-0.224	-0.217 ± 0.004	-0.210
	Testing	-0.214	-0.205 ± 0.006	-0.193
Δ_{R_3}	Training	-0.263	-0.245 ± 0.013	-0.223
	Testing	-0.261	-0.236 ± 0.015	-0.209
Δ^I	Training	-0.258	-0.251 ± 0.004	-0.243
	Testing	-0.244	-0.238 ± 0.003	-0.231

ments, evolutionary training times of the simple GP method is about 1–2 hours while the evolutionary training times of the proposed method is 2–4 hours. However, since the evolved IDRs can be reused to solve unseen instances without the need to rerun GP, these running times are still very reasonable, especially when GP is used to deal with such a complicated design task.

4.4 Insights into IDRs

The previous results have shown that the evolved IDRs can effectively solve the JSS problems. This section will carefully investigate how an IDR works and how to enhance its performance. As shown in Figure 4.1, IDRs try to improve the quality of the schedule through some iterations. Different from a traditional dispatching rule that only provides a unique schedule π, an IDR will iteratively generate a sequence of schedules $\pi^0 \to \pi^1 \to \cdots \to \pi^n$ where $Obj(\pi^i) > Obj(\pi^{i+1})$ and π^{i+1} is a function $\mathcal{F}$ of π^i. This behaviour is very similar to that of a *local search*, which also

tries to improve the quality of the solution iteratively. However, instead of using a neighbourhood structure $\mathcal{N}(\pi^i)$ to find a new improved solution, an IDR employs a function $\mathcal{F}(\pi^i)$ to generate a new schedule. From the local search viewpoint, two key issues needed to be considered for developing effective IDRs are the choice of initial solution, and the strategy to escape from local optima. We will explore these issues in the rest of this section.

4.4.1 Initialisation for IDRs

Initialisation for an IDR involves the choice of $\mathcal{R}^0$ since this choice will decide how the initial schedule is created, and therefore also influence the final schedule obtained by that IDR. In the proposed GP system, a heuristic approach is used to generate $\mathcal{R}^0$, more specifically RFT, RWT, and RNWT. Since the evolved IDRs can be in very different forms, it is expected that they can be more effective if a more suitable $\mathcal{R}^0$ is used. To cope with this problem, we introduce an extension of the proposed GP method to take into account the initialisation of $\mathcal{R}^0$. In this method, the three terminals RFT, RWT, and RNWT will be turned into *pseudo* terminals **rft**, **rwt**, and **rnwt**. The difference between these pseudo terminals and the original ones is that the pseudo terminals are treated as a function which also includes a child node or a subtree. When the evolved IDRs are used in the first iteration, the values evaluated from the subtrees will be used as $\mathcal{R}^0$. In the later iterations, these subtrees will be ignored when the evaluation reaches the pseudo nodes and $\mathcal{R}$ will be used in this case instead. An illustration of this approach is shown in Figure 4.3. In this example, when evaluating node rft, we will check whether it is the first iteration of the scheduling construction procedure. If it is the first iteration, this node will return the output from its subtree which is (RT−PR) in this case. Otherwise, it will return the recorded finish time from the previous schedule.

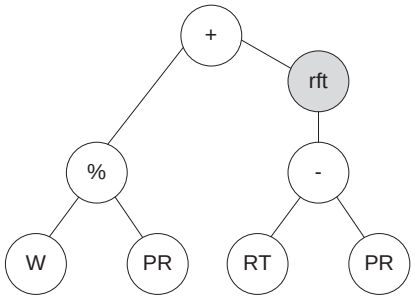


Figure 4.3: Illustration of a pseudo terminal.

4.4.2 Variable neighbourhood search with IDRs

Being trapped at a local optimum is a big issue with local search methods. For this reason, many approaches to escaping from the local optima have been proposed in the literature. Some popular approaches used for JSS problems are simulated annealing [182], tabu search [139], large step optimisation [115], and guided local search [14]. Even though these methods have been shown to be very effective when tested on benchmark JSS instances, they require extensive knowledge in the local search operators to make the approaches effective/efficient. Therefore, it would be hard to apply these approaches to evolve IDRs. For IDRs, it would be more suitable to consider high-level and more general approaches. Variable neighbourhood search (VNS) [76] seems a good candidate for this task since it mainly deals with the choice of neighbourhood structure or local search heuristics and their order in the search instead of low-level manipulations. One interesting idea in VNS is that a local optimum within one neighbourhood is not necessarily a local optimum within others, and a change of neighbourhoods has the potential to enhance the effectiveness of the search. In this study, we now propose a new local search method based on IDR and VNS. An overview of the new method IDR-VNS is shown in Figure 4.4.

The key idea of this method is to employ different IDRs to explore better schedules since the best schedule obtained by an IDR is not necessarily the best solution of the other IDRs. However, instead of the applying dif-

1: select a set of iterative dispatching rules $\mathcal{F}_k$, $k = 1, \ldots, k_{max}$
2: $Obj^{**} = +\infty$ and $k = 1$
3: $\mathcal{R}^* \leftarrow \mathcal{R}^0$
4: $improve \leftarrow false$
5: $\mathcal{R} \leftarrow \mathcal{R}^*$
6: $(Obj^*, \mathcal{R}) \leftarrow$ apply steps 2–20 of the algorithm in Figure 2.2 with $\mathcal{F}_k$ as $\Delta^I(\mathcal{J}, \mathcal{W}, \mathcal{R})$
7: **if** $Obj^{**} > Obj^*$ **then**
8: $\quad Obj^{**} = Obj^*$
9: $\quad \mathcal{R}^* \leftarrow \mathcal{R}$
10: $\quad improve \leftarrow true$
11: **end if**
12: **if** $k \neq k_{max}$ **then**
13: $\quad k = k + 1$ and return to step 5
14: **else**
15: $\quad$ **if** $(improve = true)$ **then** $k = 1$ and return to step 4;
16: $\quad$ **else** stop and return $(Obj^{**}, \mathcal{R}^*)$
17: **end if**

Figure 4.4: Local search IDR-VNS.

ferent IDRs independently, we will use the recorded information $\mathcal{R}^*$ from the best schedule obtained by an IDR to be the initial $\mathcal{R}^0$ of the next IDR to efficiently explore better schedules. Similar to VNS, in order to create effective IDR-VNS, we need to decide: (1) how many $\mathcal{F}_k$ to be used (k_{max}), (2) what $\mathcal{F}_k$ to be used, and (3) the order in which these $\mathcal{F}_k$ are applied. While the first factor has to be decided experimentally, the next two factors can be handled by GP. A GP method is proposed in which a local search IDR-VNS is represented by a GP individual with multiple trees. An example of this representation is shown in Figure 4.5. By evolving these GP individuals, GP can also help us find the effective IDRs and the order in which they will be applied.

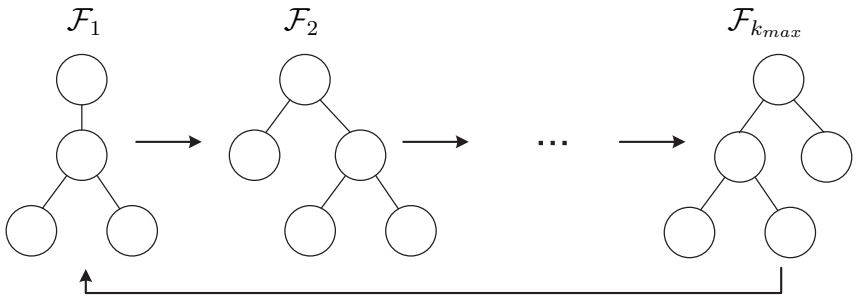
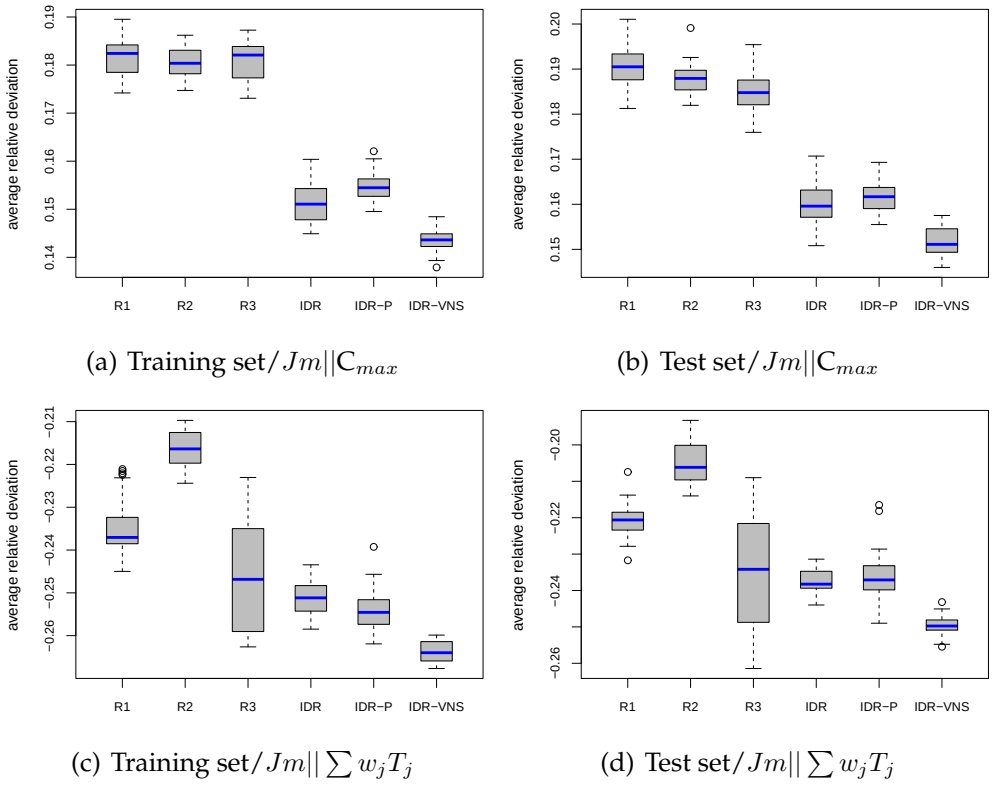


Figure 4.5: Representation of an IDR-VNS.

4.4.3 Performance of the enhanced IDRs

Figure 4.6 shows the performance of the enhanced IDRs using the two extensions discussed in the previous subsections. In this figure, IDR-P is used to indicate the evolved rules using pseudo terminals and $k_{max} = 2$ is used for IDR-VNS. It is easy to see that the evolved IDRs dominate the evolved DRs. For $Jm||\mathrm{C}_{max}$, there is no significant difference between IDR and IDR-P. These results indicate that our heuristic to initialise $\mathcal{R}$ (as described in Section 4.2) is good enough for these problems. For $Jm||\sum w_j T_j$, IDR-P rules are significantly better than IDRs on the training set but there is no significant difference between these two types of dispatching rules on the test set. One explanation for this is that the initialisation of $\mathcal{R}$ can be useful when dealing with sophisticated objectives such as $\sum w_j T_j$; however, each problem instance may require different $\mathcal{R}^0$ and it is really hard to create a general way to generate good inital $\mathcal{R}^0$.

IDR-VNS is the best approach in this comparison since the average relative performance of IDR-VNS is significantly smaller than those of all the other approaches. Obviously, the use of variable IDRs to search for good schedules is shown to be very effective. The fact that IDR-P has trouble improving the quality of IDRs and the success of IDR-VNS suggests two interesting points. First, the JSS instances in the data sets have very different characteristics, which cannot be handled easily by a single dispatching rule, even when the feedback from the previous schedule is used. Second,



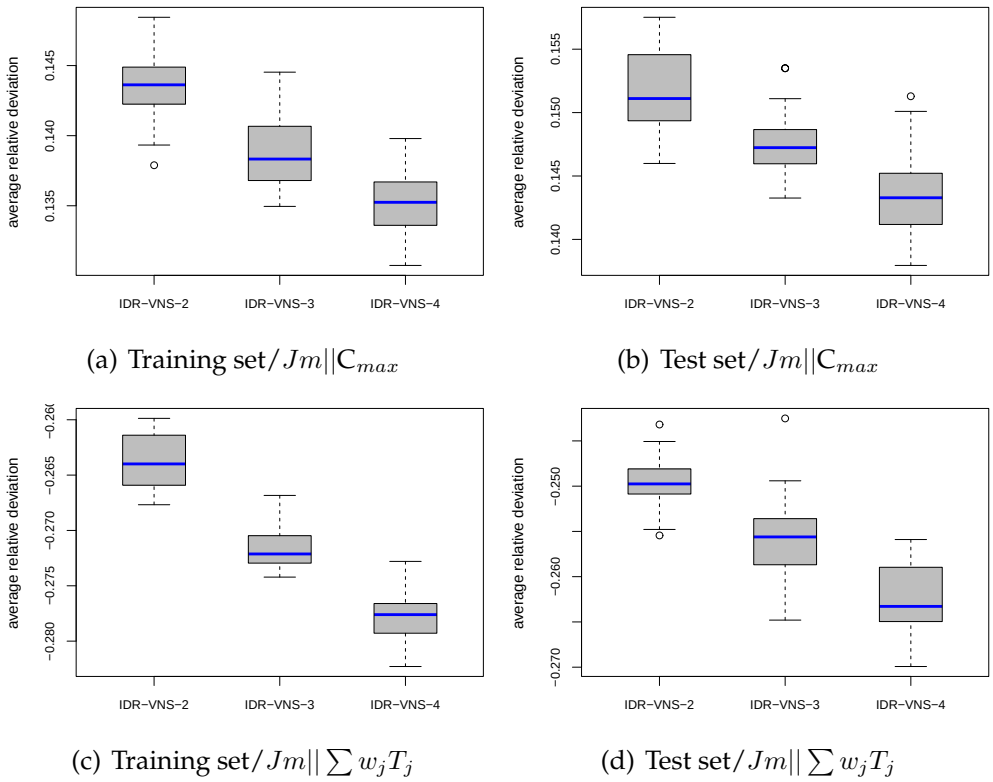
(a) Training set/$Jm||\mathrm{C}_{max}$ (b) Test set/$Jm||\mathrm{C}_{max}$

(c) Training set/$Jm||\sum w_j T_j$ (d) Test set/$Jm||\sum w_j T_j$

Figure 4.6: Comparison between evolved DR, IDR, IDR-P, and IDR-VNS.

from the optimisation search viewpoint, the search space of JSS problems is very complicated with many local optima. Therefore, the use of good $\mathcal{R}^0$ alone may not be useful because IDRs can be easily trapped at some local optimum. To enhance its performance, IDRs need to include some mechanism that helps escape from the local optima. VNS is an effective mechanism of this kind.

Given that IDR-VNS can effectively solve JSS problems, we now investigate how the number of IDRs, k_{max}, influences the performance of the IDR-VNS. Figure 4.7 show the performance of IDR-VNS with $k_{max} = 2, 3$ and 4. It is quite clear that the performance of the IDR-VNS improves significantly when k_{max} increases. This again confirms the importance of using different IDRs to avoid being trapped in a local optimum. Another interesting observation is that while the average relative deviation tends to



(a) Training set/$Jm||\mathrm{C}_{max}$ (b) Test set/$Jm||\mathrm{C}_{max}$

(c) Training set/$Jm||\sum w_j T_j$ (d) Test set/$Jm||\sum w_j T_j$

Figure 4.7: Influence of k_{max} on IDR-VNS.

linearly decrease when k_{max} increases in the training set, the gaps between IDR-VNS with different k_{max} tends to be larger in the test set. This indicates that high k_{max} will also increase the generality of IDR-VNS. A reason is that high k_{max} not only improves the quality of the schedules but also allows the evolved IDR-VNS rules to cover more situations in the training set.

4.5 Look-ahead strategies

In previous experiments, we only considered evolved IDRs which generate non-delay schedules. However, *non-delay* schedules ($\alpha = 0$) may not always provide good results in all cases. For that reason, we need to con-

sider *active* schedules [69] or *hybrid* schedules [24]. Among the three types of schedules, hybrid schedules are the most general since they can be used not only to generate active and non-delay schedules but also schedules with some look-ahead ability. Since the appropriate value for α is quite difficult to choose, as it depends on the status of machines and the type of JSS problems, we also include in our GP individual a component to decide which non-delay values should be used. The representation of this component is shown in Figure 4.8.

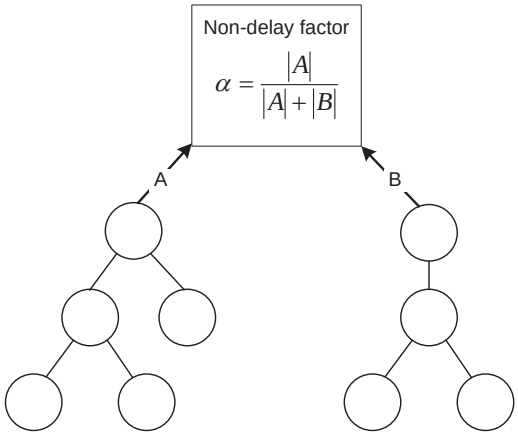
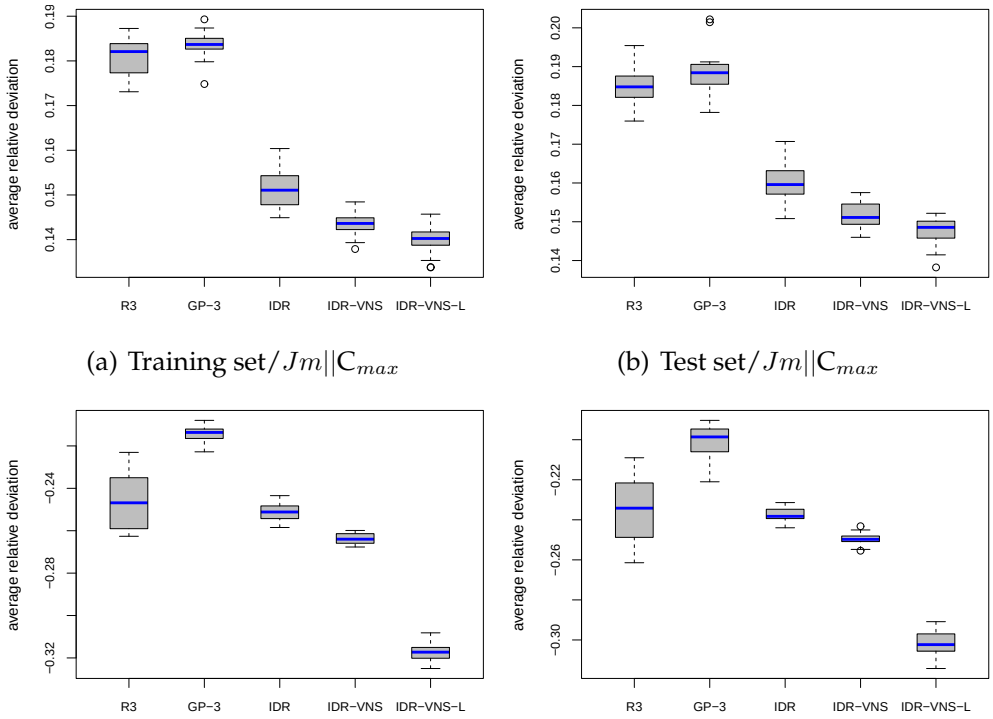


Figure 4.8: Look-ahead component represented in GP.

Basically, the look-ahead component contains two subtrees. The function set used to construct these subtrees is the same as that in the evolved IDRs while the terminal set consist of the six machine attributes (WR, MP, DJ, CMI, CRW, and BWR) used in the R_1 representation. The two subtrees are evaluated and produce two outputs A and B, whose absolute values are used to calculate the non-delay factor α as shown in Figure 4.8. When A and B are both zero, we will use $\alpha = 0$ as default. This calculation ensures that the obtained values will range from 0 to 1.

Figure 4.9 shows the performance of evolved IDR-VNS-L rules (IDR-VNS with a look-ahead component) against the rules evolved by GP with the R_3 representation, GP-3 [86], and the evolved IDR-VNS rules without



(a) Training set/$Jm||\mathrm{C}_{max}$ (b) Test set/$Jm||\mathrm{C}_{max}$

(c) Training set/$Jm||\sum w_j T_j$ (d) Test set/$Jm||\sum w_j T_j$

Figure 4.9: Influence of the evolved look-ahead component.

the look-ahead component. In these experiments, $k_{max} = 2$ is used for both IDR-VNS and IDR-VNS-L. For both $Jm||\mathrm{C}_{max}$ and $Jm||\sum w_j T_j$, the complex GP-3 is significantly worse than R_3 rules in the training and test sets. One explanation is that the individuals in GP-3 are more complicated since they include three GP trees for the three key components (for classification and dispatching rules) and require a much larger population size (10,000 is used in [86]) to obtain good results. It is quite obvious that IDR-VNS and IDR-VNS-L rules show superior performance for both training and test sets in this case. Within these two rules, IDR-VNS-L rules are significantly better than IDR-VNS rules.

4.6 Illustration of an evolved IDR-VNS-L rule

Figure 4.10 shows an example of an IDR-VNS-L rule with $k_{max} = 2$ evolved for $Jm||\sum w_j T_j$. It is easy to see that RFT and RWT appear in all terms within the two functions $\mathcal{F}_1$ and $\mathcal{F}_2$, which suggests that they have a large impact on the entire evolved IDR-VNS-L. In this example, the evolved functions can be considered as an extension of WSPT which is enhanced by incorporating information from the previous schedules since the term W/PR occurs very often in the two functions.

$$(-((\text{PR} * \text{RFT})/((\text{W}/\text{RT})/(\text{RFT} - \text{W}))) + ((\min(\max(\text{RO}, \text{RT}), ((\text{W}/\text{PR}) + \min(0.0269, \text{PR})))$$
$$/(\text{RT} + \text{RFT})) + \min(0.0269, \text{PR}))) + \text{If}(0.443 + \text{W}, \text{RFT} + \text{RFT}, \max(\text{PR}, \text{DD}))$$

(a) *Priority function* $\mathcal{F}_1$

$$\min(\max(\text{RM}, \text{RWT}), ((\text{W}/\text{PR}) + \min(0.026903434, \text{PR})))/\text{If}(0.44284365 + \text{W}, \text{abs}(\text{RFT} - \text{RO})$$
$$/0.19508022, \text{abs}(\text{RT} - \text{RFT})/0.19508022)$$

(b) *Priority function* $\mathcal{F}_2$

$$\text{A} = -0.2272413, \text{B} = \text{DJ} \Longrightarrow \frac{|\text{A}|}{|\text{A}|+|\text{B}|} = \frac{0.2272413}{0.2272413+\text{DJ}}$$

(c) *Look-ahead strategy*

Figure 4.10: Example of an evolved IDR-VNS-L with $k_{max} = 2$

Meanwhile, the look-ahead strategy is a very simple function including only the machine attribute DJ. Figure 4.11 shows the behaviour of the look-ahead strategy. This figure points out two features of the evolved look-head strategy. First, the non-delay factor α should be roughly higher than 0.2. This suggests that a reasonably small waiting time is better than zero waiting time in non-delay schedules ($\alpha = 0$). Second, the non-delay factor should be reduced when DJ increases. This means that when there is no deviation of jobs in the queue, we should allow the machine to process the next job soon. However, when there is a large deviation (low DJ) in the queue, the non-delay factor should be higher so that more potential jobs can be examined by the evolved rules to decide which job is processed next.

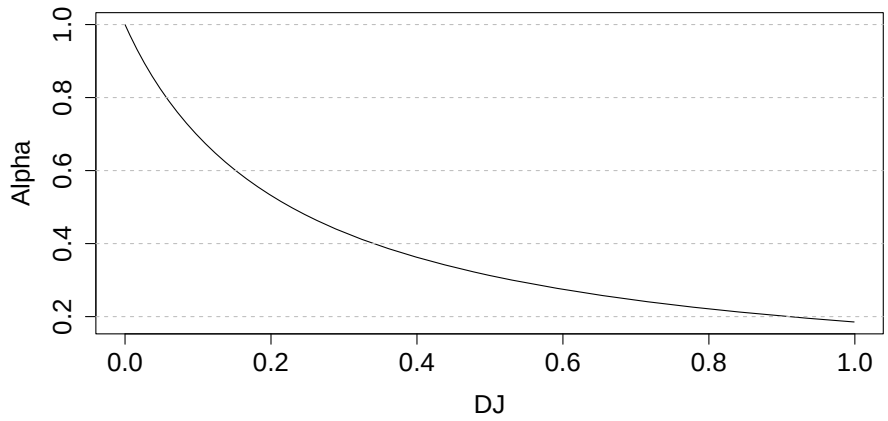


Figure 4.11: The evolved look-ahead strategy.

Table 4.4: Comparison with the multi-pass heuristic

Rules	Training Set			Test Set		
	Min	Mean	Max	Min	Mean	Max
Multi-pass	-0.610	-0.256	0.090	-0.620	-0.253	0.026
IDR-VNS-L	-0.592	-0.342	-0.198	-0.673	-0.339	-0.074

To further assess the effectiveness of this example rule, we also compare it with a multi-pass heuristic, where multiple dispatching rules are used to solve an instance and the best result will be reported. The five existing rules (WSPT, W(CR+SPT), W(S/RPT+SPT), COVERT and ATC) mentioned in Section 3.5 are the candidate rules for the multi-pass heuristic. We also fine-tune the non-delay factor for this heuristic ($\alpha = 0.4$). The results from the example rule and the multi-pass heuristic are shown in Table 4.4. It is obvious that the evolved rule has a much better average relative deviation as compared to the multi-pass heuristic on both training and test sets. This again confirms the superiority of the evolved rules and that the use of the information from previous schedule is very important to enhance the quality of the schedule.

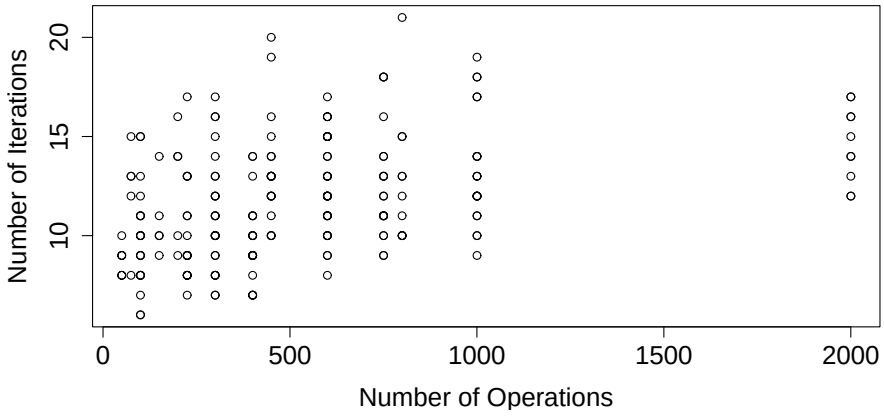


Figure 4.12: Correlation between problem size and the number of iterations from the evolved IDR.

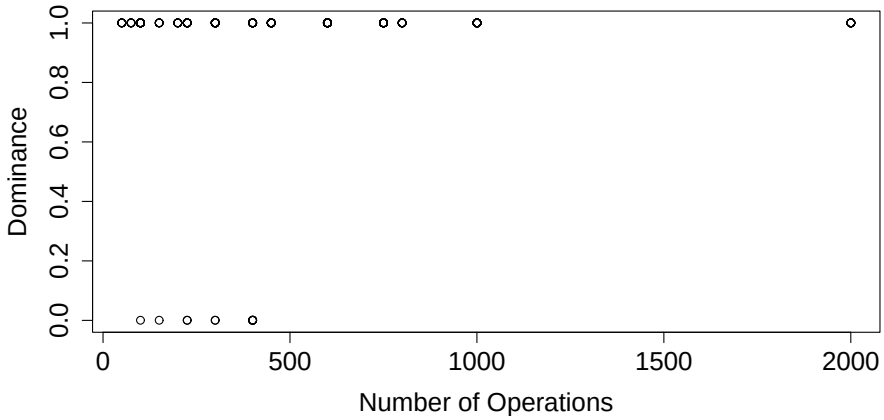


Figure 4.13: Relationship between problem size and the dominance of the evolved IDR.

In Figure 4.12 and Figure 4.13, we show how the problem size (number of operations $= \mathcal{N} \times \mathcal{M}$) influences the number of iterations and the dominance (1 if the evolved rule is better than the multi-pass heuristic and 0 otherwise) of the evolved IDR-VNS-L rule. In general, the number of iterations tends to increase as the problem size increase. This indicates that the evolved rule has more chances to make improvement steps for the larger problem instances. However, it is noted that the increase in the number

of iterations is not large. This suggests that the computational effort for the evolved rules are not increased rapidly as the problem size increases. Figure 4.13 also shows that the evolved rules tend to provide better results when the problem size increases. These observations suggest that the evolved rule is more effective than the multi-pass heuristic, especially for large-scale problems, while still maintaining good efficiency. In our experiments, the example IDR-VNS-L rule can solve all 210 training and test instances in less than a second (coded in Java and run on Intel Core i5-2400 3.10 GHz).

4.7 Chapter Summary

While machine and system attributes in R_1 and R_3 are useful, the sequencing decisions are still myopic. To overcome this drawback, an iterative procedure has been introduced. The key idea is to provide the rules with the ability to take into account the future impact of the sequencing decisions on the quality of the final solutions. The experiments have shown that IDRs are significantly better than other evolved rules. This confirms the importance of estimating the future impact of jobs in the shop for improving the performance of the scheduling system. However, the iterative procedure introduced in this chapter is still limited to the static JSS problem. When applied to a dynamic JSS environment, not only the current jobs in the shop but also the new and future arriving jobs will influence the sequencing decisions. In this case, the future impact needs to be indirectly estimated through due date assignments instead of the iterative procedure. This issue will be investigated in Chapters 6 and Chapters 7.

The GP methods in Chapters 3 and 4 only focused on evolving rules for a specific objective in static JSS environment. However, practical job shops often have to deal with multiple objectives and dynamic changes. Chapter 5 will tackle these issues based on GP.

Chapter 5

Multi-objective GPHH

Handling multiple conflicting objectives in dynamic JSS (DJSS) is challenging because many aspects of the problem need to be considered when designing dispatching rules. Tay and Ho [181] presented the first work on GP that focuses on multi-objective DJSS (MO-DJSS) problems. In their study, they converted the multi-objective problem to a single objective problem by optimising a linearly weighted sum of all the objectives. However, this approach is only effective when there exists good knowledge about the search space of the objectives considered (the scale of each objective, the shape of the true Pareto front), which is not available in most cases. Hildebrandt et al. [78] re-examined the GP system proposed by Tay and Ho [181] in different dynamic job shop scenarios and showed that the rules evolved by Tay and Ho [181] are only slightly better than the earliest release date (ERD) rule and quite far away from the performance of the shortest processing time (SPT) rule with mean flowtime as the objective. This suggested that a linear combination of objectives may not be a suitable approach to deal with MO-DJSS.

This chapter aims to use GP for evolving dispatching rules for multi-objective DJSS (MO-DJSS) problems, which can be used to support the decision makers by providing them with potential trade-offs among different objectives. The objectives of this chapters are:

1. Developing a multi-objective genetic programming based hyper-heuristic (MO-GPHH) method to evolve dispatching rules for DJSS.
2. Performing detailed comparisons between the evolved rules and other existing dispatching rules for DJSS.
3. Assessing the robustness of evolved dispatching rules under different simulation scenarios.

In Section 5.1, a detailed description of the proposed MO-GPHH are presented. Simulation models for DJSS, benchmark dispatching rules in the literature, and statistic analysis procedures to compare different rules are also provided in this section. Section 5.2 examines the performance of the evolved rules through extensive experiments. More insights regarding the evolved Pareto fronts and the robustness of evolved rules are investigated in Section 5.3. Section 5.4 concludes this chapter.

5.1 MO-GPHH for DJSS

This section shows how the proposed MO-GPHH method is used to solve DJSS problems. The first part will show how dispatching rules are represented by GP and how they can be evaluated. Then, the proposed MO-GPHH algorithm is presented. Finally, we describe the simulation model of DJSS used for training/testing purposes and the statistical procedure to analyse the results.

5.1.1 Representation and Evaluation

Similar to previous applications of GP for JSS problems [78, 84, 86, 112, 123, 138, 181], the dispatching rules here are also represented by GP trees [103]. A GP tree in this case will play the role of a priority function which will determine the priorities of jobs waiting in the queue. This chapter only considers the simple GP tree representation (R_2) of dispatching rules

Table 5.1: Terminal and function sets for DR

Symbol	Description
rJ	job release time (arrival time)
RJ	operation ready time
RO	number of remaining operation within the job.
RT	work remaining of the job
PR	operation processing time
DD	due date of the job
RM	machine ready time
SL	slack of the job $=$ DD $-$ (t $+$ RT)
WT	is the current waiting time of the job $= \max(0,$ t $-$ RJ$)$
#	Random number from 0 to 1
NPR	processing time of the next operation
WINQ	work in the next queue
APR	average operation processing time of jobs in the queue
Function set	$+$,$-$,$\times$, $\%$, min, max, abs, and If

*t is the time when the sequencing decision is made.

because GP is capable of evolving effective priority functions that can incorporate the global/local information of the shop for making sequencing decisions. The terminal and function sets of evolved dispatching rules are presented in Table 5.1. In this table, the upper part shows a number of terms that usually appear in the dispatching rules in the literature. The next part in this table shows three terms that reflect the status of the current and downstream machines. It is noted that more global terms, as introduced in Chapter 3, can also be used in this case. However, since dispatching rules work in a dynamic environment where the global information can change rapidly, the use of global information may be outdated very soon after the sequencing decisions are made. So we use the terminals and functions in Table 5.1.

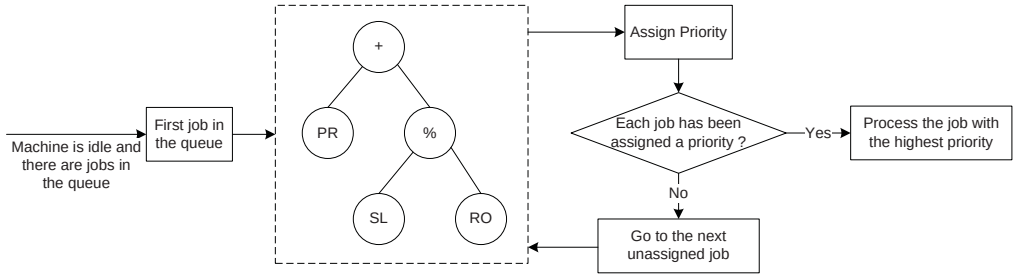


Figure 5.1: Illustration of a dispatching rule in DJSS.

An example of an evolved rule is shown in Figure 5.1. In the job shop, when a machine is idle and a new job arrives at that machine, that job will be processed immediately. In the case that a machine has just completed a job and there are still jobs waiting in the queue to be processed at that machine, the dispatching rule will be applied. To assign a priority to a waiting job, the information about that job will be extracted to be used in the terminals in Table 5.1. Then, the GP tree representing the dispatching rule will be evaluated and the output from this evaluation will be assigned to the considered job as its priority (refer to [103] for detailed discussion on how a GP tree is evaluated). This procedure will be applied until priorities are assigned to all waiting jobs and the job with highest priority will be processed next. This evaluation scheme is similar to that in Section 3.1.2 (on page 71) to generate a non-delay schedule. However, the set Ω here contains not only known jobs in the shop but also new jobs arriving during the simulation. Also, the evaluation continues until the termination condition of the simulation is met instead of when all jobs are scheduled.

5.1.2 The proposed MO-GPHH algorithm

In this work, we want to evolve dispatching rules to minimise five popular objectives in the DJSS literature, which are the mean flowtime (F), maximum flowtime (F_{max}), percentage of tardy jobs (%T), mean tardiness (T), and maximum tardiness (T_{max}) [171, 81, 162] (see Table 2.1 on page 26).

load training simulation scenarios $\mathbb{S} \leftarrow \{S_1, S_2, \ldots, S_T\}$
randomly initialise the population $P \leftarrow \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{popsize}\}$
$\mathcal{P}^e \leftarrow \{\}$ and $generation \leftarrow 0$

1: **while** $generation \leq maxGeneration$ **do**
2: **for all** $\mathcal{R}_i \in P$ **do**
3: $\mathcal{R}_i.objectives \leftarrow$ apply $\mathcal{R}_i$ to each scenario $S_k \in \mathbb{S}$
4: **end for**
5: calculate the Harmonic distance [187] and the ranks for individuals in $P \bigcup \mathcal{P}^e$
6: $\mathcal{P}^e \leftarrow$ select$(P \bigcup \mathcal{P}^e)$
7: $P \leftarrow$ apply crossover, mutation to $\mathcal{P}^e$
8: $generation \leftarrow generation + 1$
9: **end while**
10: return $\mathcal{P}^e$

Figure 5.2: MO-GPHH to evolve dispatching rules for DJSS problems.

The HaD-MOEA algorithm [187] is applied here to explore the Pareto front of non-dominated dispatching rules regarding the five objectives mentioned above. HaD-MOEA can be considered as an extension of NSGA-II [52] and it has been shown to work well on the problems with many objectives. Figure 5.2 shows how the proposed MO-GPHH works. At first, a number of training simulation scenarios (more details will be shown in the next section) are loaded and the initial archive $\mathcal{P}^e$ (parent population) is empty. These scenarios will be used to evaluate the performance of an evolved dispatching rule.

The initial GP population is created using the ramped-half-and-half method [103]. In each generation of MO-GPHH, all individuals in the population will be evaluated by applying them to each simulation scenario. The quality of each individual in the population will be measured by the average value of the objectives across all simulation scenarios. After all individuals have been evaluated, we calculate the Harmonic distance [187]

for each individual. Then, individuals in both archive $\mathcal{P}^e$ and population P are selected to update the archive $\mathcal{P}^e$ based on the Harmonic distance and the non-dominated rank [52]. The new population will be generated by applying subtree crossover and subtree mutation to the current population. Binary tournament selection [52] is used to select the parents for the two genetic operations. The crossover rate and mutation rate used in this method are 90% and 10%, respectively. The maximum depth of GP trees is eight. A population size of 200 is used in this study and the results will be obtained after the proposed method runs for 200 generations. These parameters are selected based on our preliminary experiments to balance between the effectiveness and the (Pareto) diversity of evolved dispatching rules.

5.1.3 Simulation models for dynamic job shop

Simulation is the most popular method to evaluate the performance of dispatching rules in the DJSS literature. Since our goal is to evolve robust dispatching rules, a general job shop would be more suitable than a specific shop. The following factors characterise a dynamic job shop:

- Distribution of processing times ($F1$)
- Utilisation ($F2$)
- Due date tightness ($F3$)

Utilisation is the proportion of time the machine is busy processing an operation of a job. Therefore, it is used to indicate the congestion level of machines (and the shop). The performances of the scheduling decisions under different utilisation levels are of interest in most research in the DJSS literature. Meanwhile, the distribution of processing times and the due date tightness (a factor that controls the time allowance to complete a job) are also very important factors that can influence the performance of a dispatching rule. In this study, we employ a symmetrical (balanced) job shop model in which each operation of a job has equal probability

Table 5.2: Training and testing scenarios

Factor	Training	Testing
$F1$	Discrete Uniform$[1, 49]$	Discrete Uniform$[1, 49]$ and $[1, 99]$
$F2$	$70\%, 80\%$	$85\%, 95\%$
$F3$	c is randomly selected from $(3, 5, 7)$	$c = 4$, $c = 6$, $c = 8$
Summary	$\langle 25, 70, (3, 5, 7)\rangle$, $\langle 25, 80, (3, 5, 7)\rangle$	$\langle 25, 85, 4\rangle$, $\langle 25, 85, 6\rangle$, $\langle 25, 85, 8\rangle$, $\langle 25, 95, 4\rangle$, $\langle 25, 95, 6\rangle$, $\langle 25, 95, 8\rangle$, $\langle 50, 85, 4\rangle$, $\langle 50, 85, 6\rangle$, $\langle 50, 85, 8\rangle$, $\langle 50, 95, 4\rangle$, $\langle 50, 95, 6\rangle$, $\langle 50, 95, 8\rangle$

Total Work Content (TWK) [12] with allowance factor c is used to set the due dates.

to be processed at any machine in the shop. Therefore, machines in the shop expect to have the same level of congestion in long simulation runs. This model has been used very often in the DJSS literature [81, 162, 78] to evaluate performance of dispatching rules. Although it is simple, it reflect many interesting features of real world applications such as multiple (and changing) bottlenecks and complex routings. Therefore, this model is suitable for examining the effectiveness of our proposed method. The scenarios for training and testing of dispatching rules are shown in Table 5.2.

The simulation experiments have been conducted in a job shop with 10 machines. The triplet $\langle m, u, c\rangle$ represents the simulation scenario in which the average processing time is m (m is 25 or 50 when processing times follow discrete uniform distribution [1,49] or [1,99], respectively), the utilisation is $u\%$ and the allowance factor is c (due date = release time + c $\times$ total processing time). In the training stage, two simulation scenarios (corresponding to the two utilisation levels) and five replications will be performed for each scenario. The average value for each objective from $2 \times 5 = 10$ replications will be used to measure the quality of the evolved rules (as described in the previous section). We use the shop with differ-

ent characteristics here in order to evolve rules with good generality. The allowance factors, which decide the due date tightness, are selected randomly from the three values 3, 5, and 7 instead of a fixed allowance factor (for each scenario) in common simulation experiments for DJSS problems. If we train on scenarios with fixed allowance factors, the evolved rules will tend to focus more on the scenarios with small allowance factors to improve the due date performance (mean tardiness, maximum tardiness, etc.) because the values of the due date based objectives are higher in these cases. This may cause an overfitting problem for the evolved dispatching rules. Moreover, training on different scenarios with different fixed allowance factors will also increase the training time of our MO-GPHH method. Simulating multiple utilisation levels in a simulation scenario can also be used to reduce the number of training scenarios but will increase significantly the running time of a replication to obtain the steady state performance of the rules, and indirectly increase the training time of the MO-GPHH method.

In the testing stage, 12 simulation scenarios with 50 replications for each scenario (or shop condition) resulting in $12 \times 50 = 600$ replications will be used to have a comprehensive assessment of the evolved rules. In each replication of a simulation scenario, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the 500^{th} job is considered as the warm-up time and the statistics from the next completed 2000 jobs [81] will be used to calculate the five objective values. The number of operations for each new job is randomly generated from the discrete uniform distribution [2,14] and the routing for each job is randomly generated, with each machine having equal probability to be selected (re-entry is allowed here but consecutive operations are not processed on the same machine). The arrival of jobs will follow a Poisson process with the arrival rate adjusted based on the utilisation level [162].

Table 5.3: Benchmark dispatching rules

SPT	shortest processing time	LPT	longest processing time
EDD	earliest due date	FDD	earliest flow due date
FIFO	first in first out	LIFO	last in first out
LWKR	least work remaining	MWKR	most work remaining
NPT	next processing time	WINQ	work in next queue
CR	critical ratio	AVPRO	average processing time/operation
MOD	modified due date	MOPNR	most operations remaining
SL	negative slack	Slack	slack
PW	process waiting time	RR	Raghu and Rajendran
ATC	apparent tardiness cost	COVERT	cost over time
OPFSLK/PT	operational flow slack per processing time		
LWKR+SPT	least work remaining plus processing time		
CR+SPT	critical ratio plus processing time		
SPT+PW	processing time plus processing wating time		
SPT+PW+FDD	SPT+PW plus earliest flow due date		
Slack/OPN	slack per remaining operations		
Slack/RPT+SPT	slack per remaining processing time plus operation processing time		
PT+WINQ	procesting time plus work in next queue		
2PT+WINQ+NPT	double processing time plus WINQ and NPT		
PT+WINQ+SL	processing time plus WINQ and slack		
PT+WINQ+NPT+WSL	PT+WINQ plus next processing time and waiting slack		

5.1.4 Benchmark dispatching rules

Table 5.3 shows 31 dispatching rules that will be used to compare with the evolved rules in our work. The upper part of this table shows some original dispatching rules and the lower part shows some extensions of the original rules that have been proposed in the literature. The parameters of ATC and COVERT are the same as those used in Vepsalainen and Morton [185] ($k = 3$ for ATC, $k = 2$ for COVERT, and the leadtime estimation parameter $b = 2$). More detailed discussion on these rules can be found in [185, 162, 145, 88, 81].

5.1.5 Statistical Analysis

Since DJSS is a stochastic problem, statistical analysis is required to compare the performance of dispatching rules obtained from simulation. In this work, we use the one-way ANOVA and Duncan's multiple range tests [128] to compare the performance of rules or a set of rules for each objective since this statistical analysis has been used in previous studies [162, 88, 81]. It is noted that the *common random number* technique is used in our experiments for variance reduction.

It is important to note that Pareto-dominance has not been considered before in the dispatching rule literature, even though it is an important concept in the multi-objective optimisation domain. Most studies on dispatching rules have been done mainly based on a single objective even when multiple objectives are investigated. The reason is that the focus of previous studies is on minimising a single objective and the performance on other objectives are not of interest. Also, since DJSS is a stochastic problem, statistical analysis is necessary to examine the Pareto-dominance of rules but there is no standard statistical procedure available for this task. In this work, we describe two procedures to check for the *statistical Pareto-dominance* between two different rules, which can be used to validate each other.

Objective-wise procedure.

In multi-objective optimisation, solution (or dispatching rule in this work) a is said to *Pareto-dominate* solution b if and only if $\forall i \in \{1, 2, \ldots, n\} : f_i(a) \leq f_i(b) \wedge \exists j \in \{1, 2, \ldots, n\} : f_j(a) < f_j(b)$ where n is the number of objectives to be minimised. However, if $f_j(a)$ and $f_j(b)$ are random variables (i.e. solutions a and b produce different outputs in different runs/replications), we cannot use the above definitions to check for the Pareto-dominance. Therefore, we need to redefine the Pareto-dominance for this context. For the objective-wise procedure, solution a *statistically Pareto-dominates* solution b if and only if $\forall i \in \{1, 2, \ldots, n\} : f_i(a) \leq_{\mathcal{T}} f_i(b) \wedge \exists j \in \{1, 2, \ldots, n\} : f_j(a) <_{\mathcal{T}} f_j(b)$, where $f_i(a) \leq_{\mathcal{T}} f_i(b)$ means that a is significantly smaller (better) than or not significantly different from b based on the statistical test $\mathcal{T}$ (e.g. $z\text{-}test$); similarly, $f_j(a) <_{\mathcal{T}} f_j(b)$ means that a is significantly smaller than b based on $\mathcal{T}$. It should be noted that since multiple comparisons (n comparisons for n objectives) need to be done here, we have to adjust the value of the pre-set probability α of a type-1 error [128] in order to control the false positive rate. Many methods have been proposed for this problem such as the Bonferroni method and Scheffe method [159].

Replication-wise procedure.

Different from the above method that examines the Pareto-dominance of two solutions based on the relative performance of each objective, the replication-wise procedure focuses on the Pareto-dominance in each replication/observation to detect the difference between two solutions. This procedure is adapted from the method proposed by Bhowan et al. [23] to compare the performance of different multi-objective GP methods on a run-by-run basis and determine whether a method significantly dominates another over all runs. In this procedure, the traditional Pareto-dominance is used to examine the dominance relation between two solutions in each replication. For instance, $f_j(a) = \{f_j^1(a), f_j^2(a), \ldots, f_j^N(a)\}$

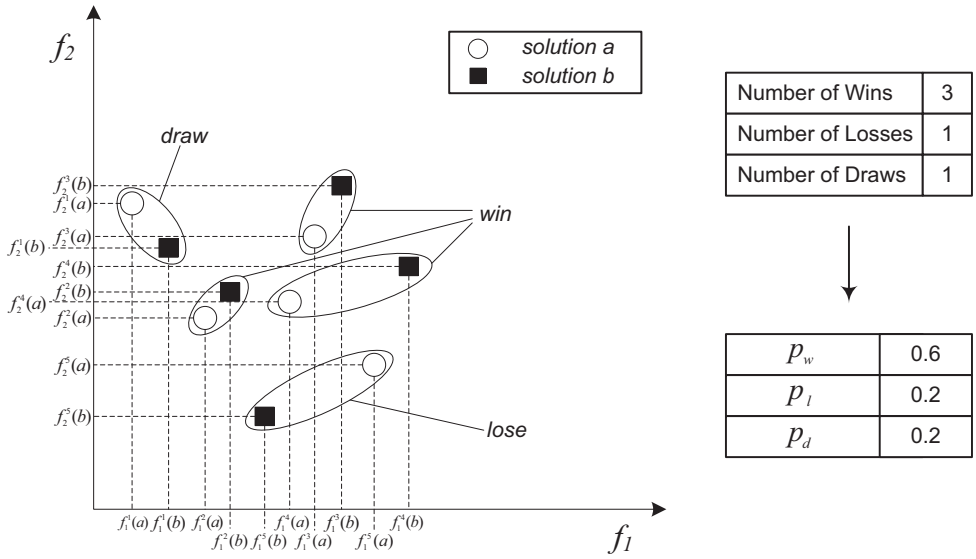


Figure 5.3: Wins, losses and draws in replication-wise procedure.

and $f_j(b) = \{f_j^1(b), f_j^2(b), \ldots, f_j^N(b)\}$ are the values for objective j obtained by solutions a and b from N replications. In replication k, $\{f_1^k(a), f_2^k(a), \ldots, f_n^k(a)\}$ is compared to $\{f_1^k(b), f_2^k(b), \ldots, f_n^k(b)\}$ to determine the Pareto-dominance between a and b in this replication. Three possible outcomes from the comparison are (1) win for a if a dominates b, (2) loss for a if b dominates a, or (3) draw otherwise. The proportions of win (p_w), lose (p_l), and draw (p_d) over N replications is then recorded. Figure 5.3 gives an example to show how p_w, p_l and p_d are calculated in the case with two objectives and $N = 5$. The outcomes here form a *multinominal* distribution since the proportions or probabilities for all outcomes always sum to one. In a multinominal distribution, the $(1 - \alpha)\%$ confidence interval of the difference in the probability of win and lose ($p_w - p_l$) can be calculated as follows:

$$(p_w - p_l) \pm z_{\alpha/2}\sqrt{var(p_w - p_l)} \tag{5.1}$$

where

$$
\begin{aligned}
var(p_w - p_l) &= var(p_w) + var(p_l) - (var(p_w + p_l) - var(p_w) - var(p_l)) \\
&= 2var(p_w) + 2var(p_l) - var(p_w + p_l) \\
var(p_w) &= \frac{p_w(1 - p_w)}{N} \\
var(p_l) &= \frac{p_l(1 - p_l)}{N} \\
var(p_w + p_l) &= \frac{(p_w + p_l)(1 - p_w - p_l)}{N}
\end{aligned}
$$

The confidence interval obtained by the equation (5.1) can be used to determine whether one solution significantly dominates the other. If the lower bound of the confidence interval is positive, solution a significantly dominates solution b. If the upper bound of the confidence interval is negative, solution b significantly dominates solution a. Otherwise, there is no significant dominance between the two solutions.

There are some key differences between these two procedures. While the objective-wise procedure focuses more on the magnitude of the difference between average objectives obtained by the two methods, the replication-wise procedure only cares about the Pareto dominance regardless of the difference between the obtained objective values in each replication. If the variances of the objectives obtained from the simulation are high, the replication-wise procedure may not accurately determine the statistical Pareto dominance between two solutions. For example, when p_w and p_l are very close, it is very likely the replication-wise procedure will conclude that there is no dominance between the two solutions. However, it is intuitively not true if there are some *"big"* wins (there are large difference between pairs of objective values $f_j^k(a)$ and $f_j^k(b)$ for some $j \in \{1, 2, \ldots, n\}$) for a solution in some replications. The advantage of the replication-wise procedure is that one statistical significance test needs to be performed as compared to multiple tests (which make the procedure more complicated) in the objective-wise procedure. In Section 5.2.2, we will apply both

procedures to determine the *statistical Pareto-dominance* between evolved dispatching rules and the dispatching rules reported in the literature.

5.2 Results

Thirty independent runs of the proposed MO-GPHH method are performed and the non-dominated evolved rules from the evolved Pareto front $\mathcal{P}^e$ are recorded. We perform a post-processing step to extract the Pareto front $\mathcal{P}$ from $\mathcal{P}^e$ for each testing scenario based on the average values of five objectives in that scenario. The performance of the evolved rules in $\mathcal{P}$ will be presented in this section. We first examine the quality of these rules for each single objective. Then, we show the Pareto dominance of the evolved dispatching rules as compared to the dispatching rules reported in the literature.

5.2.1 Single Objective

Even though our target is to solve the MO-DJSS problems, it is important to know whether the evolved rules can provide satisfactory results for each single objective. This is also a good opportunity to make a proper comparison of the evolved dispatching rules from a multi-objective GP method and the existing rules which are usually designed for a specific objective. Figures 5.4–5.7 show the performance of the evolved rules for each objective under different shop conditions. For each GP run, the evolved rule within $\mathcal{P}$ that performs best on the objective $\mathcal{O}$ ($\mathcal{O}$ can be F ,F_{max}, %T, T, or T_{max}) is denoted as $\mathcal{R}^*_{\mathcal{O}}$. The left box-plot in each plot in Figures 5.4–5.7 represents the average values of the objective $\mathcal{O}$ obtained by $\mathcal{R}^*_{\mathcal{O}}$ from the 30 GP runs. The right box-plot shows the corresponding values obtained by the top five rules for the objective $\mathcal{O}$ among the 31 existing rules shown in Table 5.3.

A quick observation of Figures 5.4–5.7 shows that the proposed MO-GPHH method can effectively find rules that are better than, or as com-

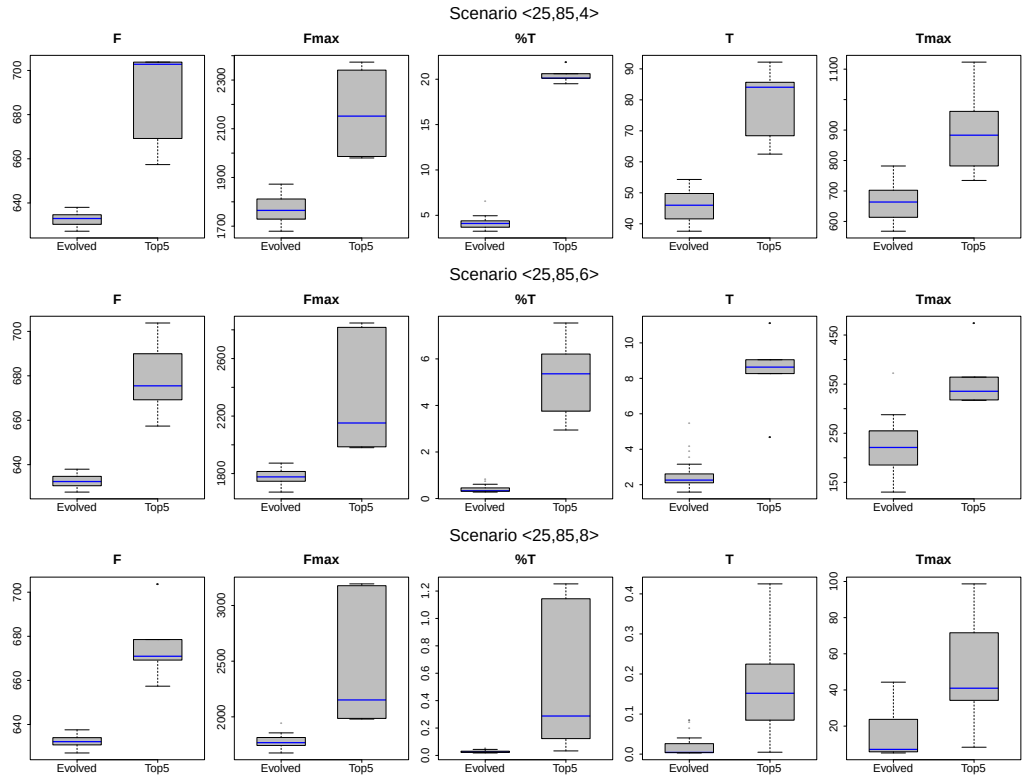


Figure 5.4: Performance of evolved dispatching rules (processing times from [1,49] and utilisation of 85%).

petitive as, the best existing dispatching rules for each objective under different shop conditions. The evolved rule $\mathcal{R}^*_{\mathcal{O}}$ can dominate the existing rules regarding F, F_{max}, %T, and T. For T_{max}, the proposed MO-GPHH can find the rules that dominate the majority of the existing rules and the obtained $\mathcal{R}^*_{\mathcal{O}}$ from some GP runs can also dominate the best existing rule. This suggests that it is totally possible to evolve a superior rule for each single objective by the proposed MO-GPHH. However there are objectives that are more difficult to minimise, e.g., T_{max} in this case. Given that we try to evolve rules to minimise five objectives simultaneously in the general case, the results obtained here for single objective are very competitive.

Further statistical tests are also performed here to confirm the quality of the evolved rules. For a specific objective $\mathcal{O}$ and shop condition $\langle m, u, c \rangle$,

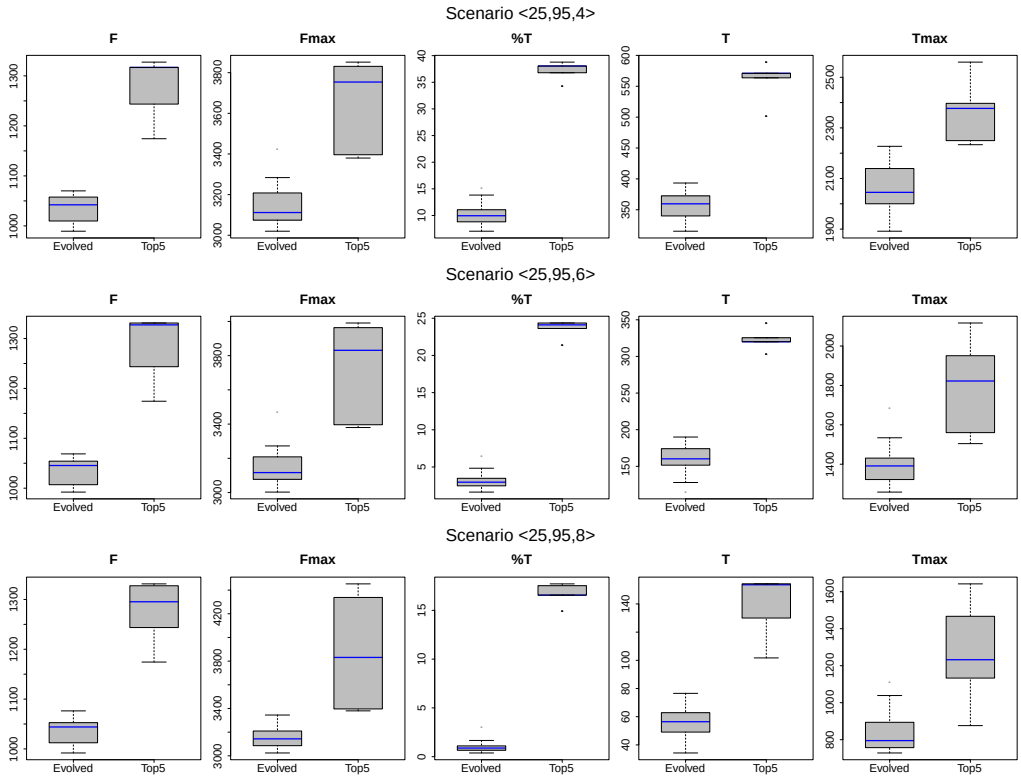


Figure 5.5: Performance of evolved dispatching rules (processing times from [1,49] and utilisation of 95%).

we perform statistical analysis of the $\mathcal{R}^*_{\mathcal{O}}$ rule from each GP run and the best five dispatching rules in the literature (based on the average values of the corresponding objective) using the one-way ANOVA and Duncan's multiple range tests [128] with $\alpha = 0.01$. The summary of all statistical tests is shown in Table 5.4. For each shop condition, the first row shows the number of times the proposed MO-GPHH method is able is find the $\mathcal{R}^*_{\mathcal{O}}$ that is significantly better than the best existing rule for minimising $\mathcal{O}$, which is shown in the second row. In general, the results here are similar to those shown in Figures 5.4–5.7. It is clear that the MO-GPHH method can almost always find a superior rule for minimising F while 2PT+WINQ+NPT is the best existing rule. These observations are consistent with those in [78] and [81]. Similar to [78], when using GP to evolve

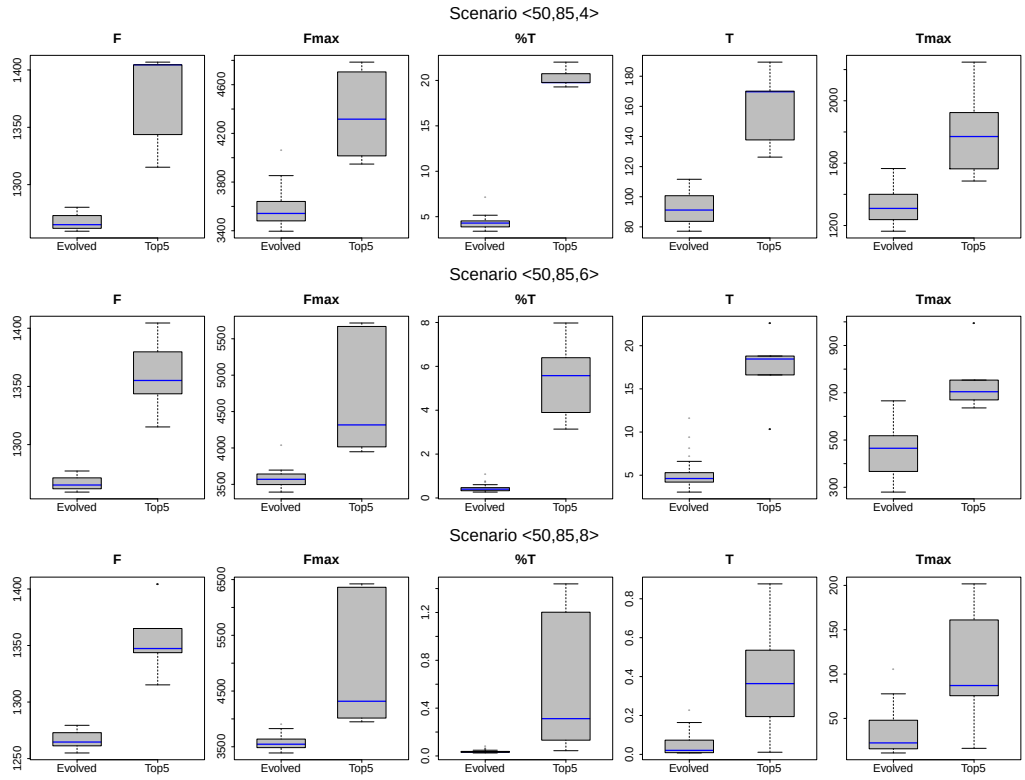


Figure 5.6: Performance of evolved dispatching rules (processing times from [1,99] and utilisation of 85%).

rules for minimising F, the evolved rules can easily beat 2PT+WINQ+NPT across different simulation scenarios. For F_{max}, the evolved rules also dominate the best rule, i.e., SPT+PW+FDD in this case, in the majority of GP runs. It is interesting to note that the 2PT+WINQ+NPT rule and SPT+PW+FDD rule are always the best existing rules for the two objectives (F and F_{max}) under all shop conditions. This suggests that the shop condition does not really have a large impact on the performance of the rules. However, the complexity of the objective may make the design of an effective rule more difficult.

The due date based performance measures such as %T, T, and T_{max} are more sensitive to the shop condition since the best existing rules are different under different shop conditions. %T is an easy objective as it

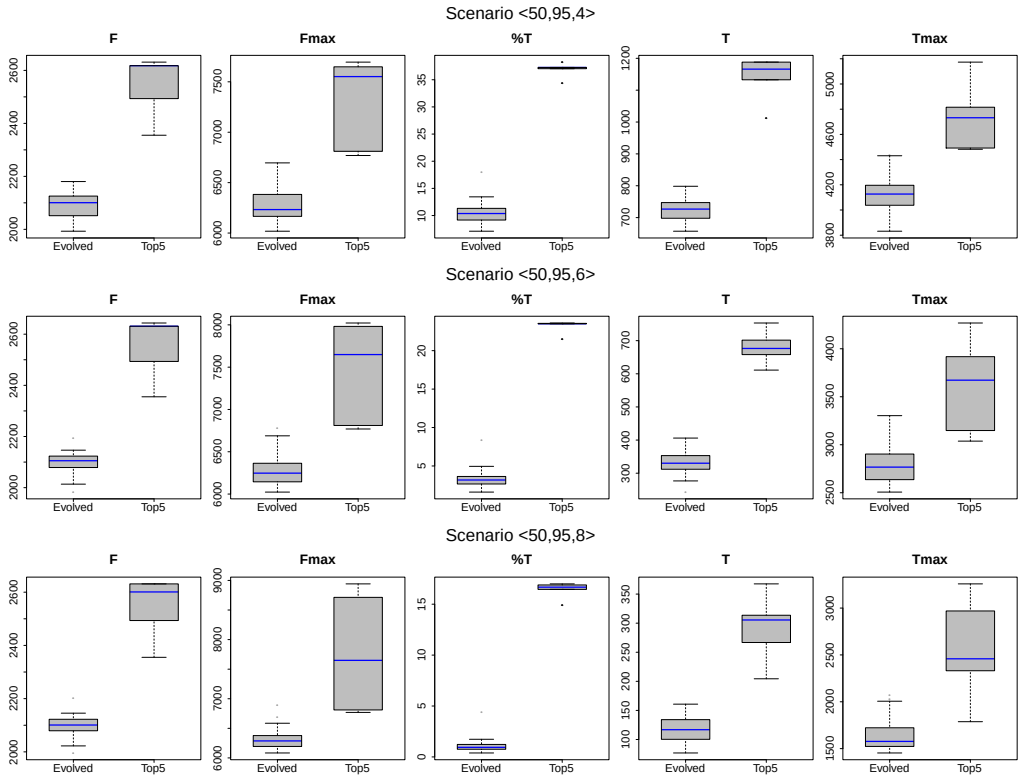


Figure 5.7: Performance of evolved dispatching rules (processing times from [1,99] and utilisation of 95%).

does not take into account the magnitude in which the job misses the due date. Therefore, MO-GPHH is able to find superior rules for this objective in almost all the scenarios. The number of superior evolved rules is not large only in the scenarios with large allowance factor ($c = 8$) and low utilisations (85%). The reason is that the number of tardy jobs is very low (near zero as seen in Figures 5.4–5.7) when due dates are too loose and the shop is not very busy. It is noted that many other existing rules (besides Slack/OPN) can also achieve near zero %T in this case. Therefore, it is very difficult to detect superior evolved rules here. In other cases, the differences between the evolved rules and existing rules for minimising %T are very clear. A similar conclusion can also be applied to T. Perhaps, T_{max} is the most difficult objective among the five objectives that we consider in

Table 5.4: Performance of evolved rules under different shop conditions

		$\mathbf{F}$	$\mathbf{F_{max}}$	$\%\mathbf{T}$	$\mathbf{T}$	$\mathbf{T_{max}}$
$\langle 25, 85, 4\rangle$	*	30/30	30/30	30/30	30/30	26/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	MOD	COVERT	PT+WINQ+NPT+WSL
$\langle 25, 85, 6\rangle$	*	30/30	30/30	30/30	29/30	29/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	RR	RR	Slack/OPN
$\langle 25, 85, 8\rangle$	*	30/30	30/30	25/30	17/30	18/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	Slack/OPN	Slack/OPN	Slack/OPN
$\langle 25, 95, 4\rangle$	*	30/30	29/30	30/30	30/30	30/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	LWKR+SPT	2PT+WINQ+NPT	PT+WINQ+SL
$\langle 25, 95, 6\rangle$	*	30/30	29/30	30/30	30/30	26/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	LWKR+SPT	RR	PT+WINQ+NPT+WSL
$\langle 25, 95, 8\rangle$	*	30/30	30/30	30/30	30/30	22/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	LWKR+SPT	RR	PT+WINQ+NPT+WSL
$\langle 50, 85, 4\rangle$	*	30/30	29/30	30/30	30/30	28/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	MOD	COVERT	PT+WINQ+NPT+WSL
$\langle 50, 85, 6\rangle$	*	30/30	29/30	30/30	29/30	29/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	RR	RR	PT+WINQ+NPT+WSL
$\langle 50, 85, 8\rangle$	*	30/30	30/30	24/30	10/30	8/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	Slack/OPN	Slack/OPN	Slack/OPN
$\langle 50, 95, 4\rangle$	*	30/30	30/30	30/30	30/30	30/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	LWKR+SPT	2PT+WINQ+NPT	PT+WINQ+NPT+WSL
$\langle 50, 95, 6\rangle$	*	30/30	29/30	30/30	30/30	25/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	LWKR+SPT	RR	PT+WINQ+NPT+WSL
$\langle 50, 95, 8\rangle$	*	30/30	29/30	30/30	30/30	24/30
	**	2PT+WINQ+NPT	SPT+PW+FDD	LWKR+SPT	RR	PT+WINQ+NPT+WSL

this study since it is hard to minimise and also quite sensitive to the shop condition. Even though our MO-GPHH method can find superior rules in most runs overall, the number of superior rules is usually lower than those for other objectives.

In general, the experimental results show that the proposed MO-GPHH can effectively find the good rules for each specific objective we consider in this work. It is obvious that the existing rules that are supposed to be

the best for an objective can also be outperformed by the evolved rules. Since we evolved the Pareto front of non-dominated rules for five objectives with a modest population of 200 individuals, the method may not always find the superior rules for some hard objectives. However, as shown in Table 5.4, because the shop condition can affect the performance of dispatching rules and their relative performance, the rules that are superior under one shop condition may not be the superior one under the other shop conditions. Therefore, evolving a set of non-dominated rules in our method is actually more beneficial than evolving a single rule (either for single objective in [78] or aggregate objective of multiple objective in [181]) since it can provide potential rules to deal with different shop conditions.

5.2.2 Multiple Objectives

The comparison above has shown that the proposed MO-GPHH method can simultaneously evolve superior rules for each specific objective. However, these superior performances come with some trade-offs on other objectives. Previous studies have shown that there is no dispatching rule that can minimise all objectives. Therefore, dispatching rules in the literature are designed for minimising a specific objective only. Although it is true that these rules can effectively minimise the objective that it focuses on, it usually deteriorates other objectives significantly. For example, the 2PT+WINQ+NPT rule can successfully reduce the average flowtime but it performs badly on almost all other objectives. Since the existence of multiple conflicting objectives is a natural requirement in real world scheduling applications, it is crucial to include this issue into the design process of dispatching rules as well. In this part, we will examine the *Pareto-dominance* of the evolved rules against other dispatching rules in the literature.

For each MO-GPHH run, the evolved rules in the Pareto front $\mathcal{P}$ are compared to the set $\mathcal{D}$ of 31 benchmark dispatching rules in from Table 5.3. For each shop condition, we will employ the objective-wise (OBJW)

and replications-wise (REPW) procedures discussed in Section 5.1.5 to determine the statistical Pareto dominance between each pair $(\mathcal{R}_i, \mathcal{B}_j)$ for all $\mathcal{R}_i \in \mathcal{P}$ and $\mathcal{B}_j \in \mathcal{D}$. Therefore, there are $|\mathcal{P}| \times |\mathcal{D}|$ comparisons in total for each MO-GPHH run and each statistical procedure. Both OBJW and REPW procedures will be performed with $\alpha = 0.01$. In the OBJW procedure, we use the Bonferroni method [128] to adjust the value of $\alpha^t = \alpha/n$ in each z-test (for each objective). From this point forward, we use *dominate* or *dominance* when mentioning about the statistical Pareto-dominance, unless otherwise indicated. After all the comparisons in each MO-GPHH run were done, an evolved dispatching rule $\mathcal{R}_i$ is classified into three categories:

1. *Non-dominated* if there is no dominance between $\mathcal{R}_i$ and $\mathcal{B}_j$ for $\forall \mathcal{B}_j \in \mathcal{D}$
2. *Dominating* if $\mathcal{R}_i$ is not dominated by any $\mathcal{B}_j \in \mathcal{D}$ and $\exists \mathcal{B}_j \in \mathcal{D}$ such that $\mathcal{R}_i$ dominates $\mathcal{B}_j$
3. *Dominated* if $\exists \mathcal{B}_j \in \mathcal{D}$ such that $\mathcal{R}_i$ is dominated by $\mathcal{B}_j$

The proportions of evolved rules in the three categories for each $\mathcal{P}$ is determined and the average proportions from 30 MO-GPHH runs are shown in Figure 5.8. The triplets in the figure indicate the shop conditions as explained in the previous section. It is clear that the proposed MO-GPHH method can always find rules that can dominate rules reported in the literature across all objectives. In the worst cases $\langle 25, 95, 4\rangle$ and $\langle 50, 95, 4\rangle$, there are still about 20% of the evolved rules that are dominating rules. The number of dominated evolved rules is also very low and the highest proportions (about 10%) of dominated rules are in $\langle 25, 95, 8\rangle$ and $\langle 50, 95, 8\rangle$. There are also some interesting patterns in Figure 5.8. Different from our comparison for single objective where there are fewer superior rules found when the allowance factor increases, it is easy to see that the number of non-dominated rules decreases and the number of dominating

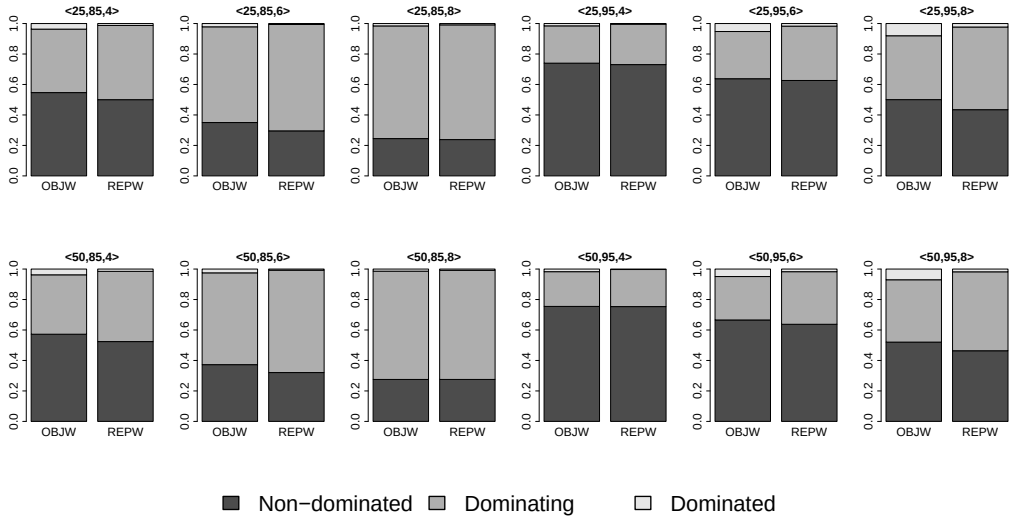


Figure 5.8: Average Pareto dominance proportion of evolved dispatching rules.

rules increases when the allowance factor increases from 4 to 8. This suggests that even when the MO-GPHH method cannot find a superior rule for a specific objective, it can easily find rules that can perform as good as the best existing rule on that objective while significantly improving other objectives. Another interesting pattern in Figure 5.8 is that the number of dominated rules decreases when the allowance factor increases with the shop utilisation of 85%. However, a reverse trend is found with the utilisation of 95% when the number of dominated rules increases when the allowance factor increases. For the cases with utilisation of 85%, the higher allowance made the DJSS problems easier, at least for the due date based performance measures. Therefore, it is difficult for existing rules to dominate the evolved dispatching rules. In the case with utilisation of 95% and low allowance factor, it is very difficult to make a good sequencing decision to satisfy multiple objectives and to find a rule that is superior on all objectives. For that reason, the number of dominating and dominated rules are relatively small compared to the number of non-dominated rules.

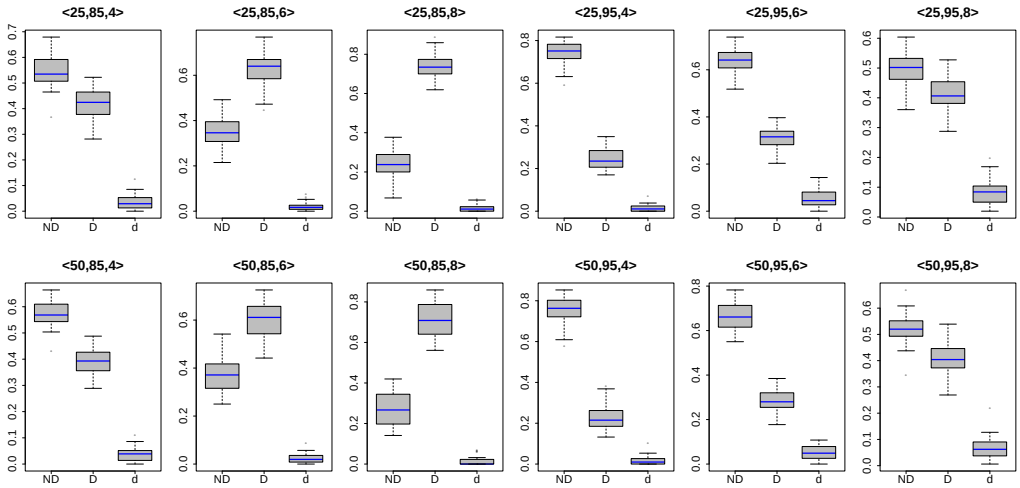


Figure 5.9: Pareto dominance proportions of evolved rules.

When the utilisation is 95% and the allowance factor is high, the number of dominated rules increases because these shop conditions (very busy shop and loose due dates) are quite different from the shop conditions used in the training stage.

It is also noted that the results from OBJW and REPW in Figure 5.8 are very consistent. The REPW procedure results in slightly more dominating rules (fewer non-dominated rules) as compared to the OBJW procedure. Perhaps, this is because the OBJW procedure with the Bonferroni adjustment method is quite conservative, which makes the OBJW procedure more difficult to detect significant differences between two rules. However, the differences between the two procedures in our application is very small. Therefore, both OBJW and REPW are suitable procedures to analyse the results from our experiments. A more detailed Pareto dominance of evolved rules is shown in Figure 5.9. In this figure, the box-plots represent the proportions from the OBJW procedure of non-dominated (ND), dominating (D) and dominated (d) from each run of MO-GPHH. This figure shows that the proposed MO-GPHH is quite stable since the obtained

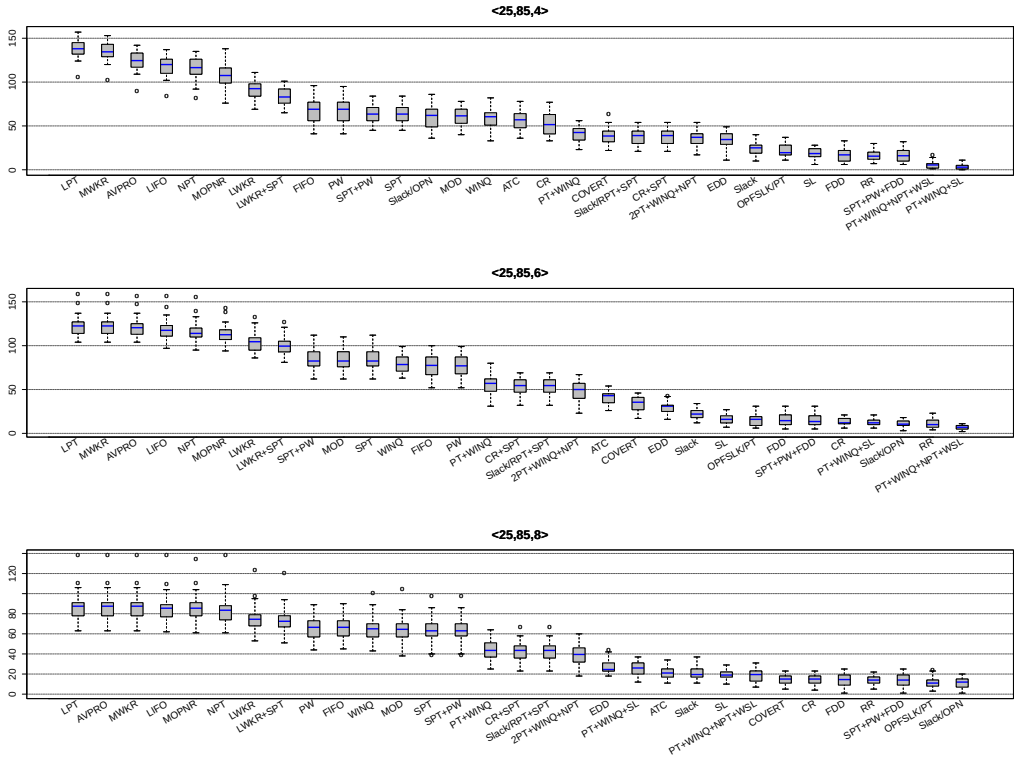


Figure 5.10: NDER for each existing dispatching rules (processing times from [1,49] and utilisation of 85%).

dominance proportions have low variances. Moreover, the proportions of non-dominated and dominating rules are always larger than that of dominated rules. In general, these results suggest that the evolved dispatching rules are significantly better or at least very competitive when compared to the existing dispatching rules.

Through all the comparisons, we also count the number of dominating evolved rules (NDER) in each MO-GPHH run that dominate a specific rule $\mathcal{B}_j$. These values can be used as an indicator for the competitiveness of the existing dispatching rules when multiple objectives are considered. The values of NDER for each rule $\mathcal{B}_j$ shown in Table 5.3 under different shop conditions from 30 MO-GPHH runs are shown in Figures 5.10–5.13.

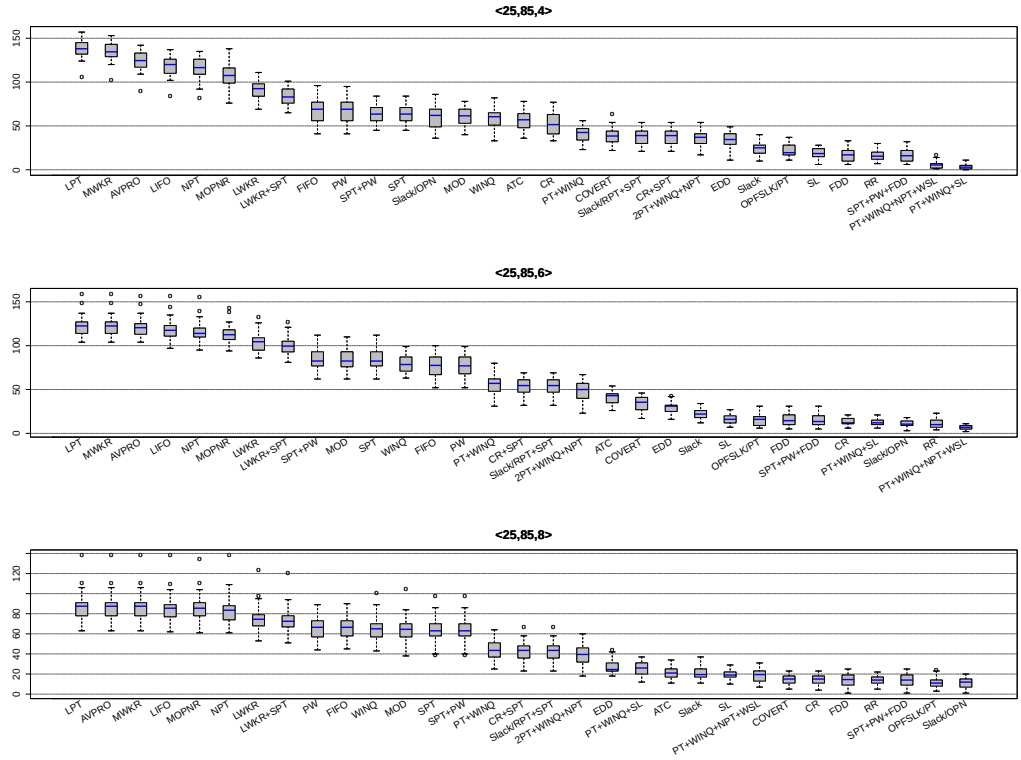


Figure 5.11: NDER for each existing dispatching rules (processing times from [1,49] and utilisation of 95%).

In these figures, the rules are arranged from left to right in the order of decreasing values of the average NDER. It is easy for the MO-GPHH method to evolve rules that dominate the simple rules such as LPT, MWKR, FIFO, etc. It is noted that most rules with low values of NDER are the ones which are designed for minimising due date based performance measures and the ones that achieve the best performance for each objective as shown back in Table 5.4. Since the MO-GPHH method can almost always find superior rules for minimising F and F_{max}, the best existing rules for these two objectives, i.e., 2PT+WINQ+NPT and SPT+PW+FDD, are also easily dominated by the evolved rules (dominating evolved rule for these two rules can be found in all MO-GPHH runs). The most competitive existing rules

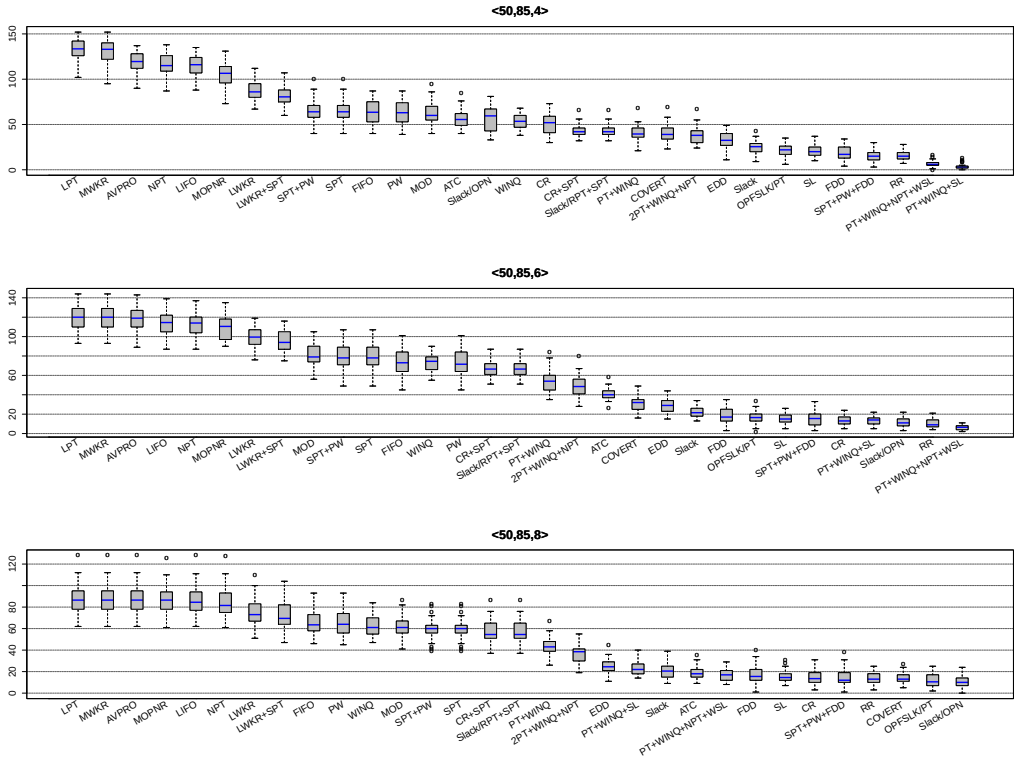


Figure 5.12: NDER for each existing dispatching rules (processing times from [1,99] and utilisation of 85%).

are actually the ones that give reasonably good performance across all objectives such as OPFSLK/PT, which is not the best rule for any particular objective. PT+WINQ+NPT+WSL and PT+WINQ+SL are the most competitive rules overall (with low NDER in most simulation scenarios) and the MO-GPHH method cannot find rules that dominate these two rules in some runs.

Although a lot of efforts have been made in the literature to improve the competitiveness of dispatching rules, it is clear that the search space of potential dispatching rules is very large and there are still many highly competitive rules that have not been explored, especially when different multiple conflicting objectives are simultaneously considered. Manually

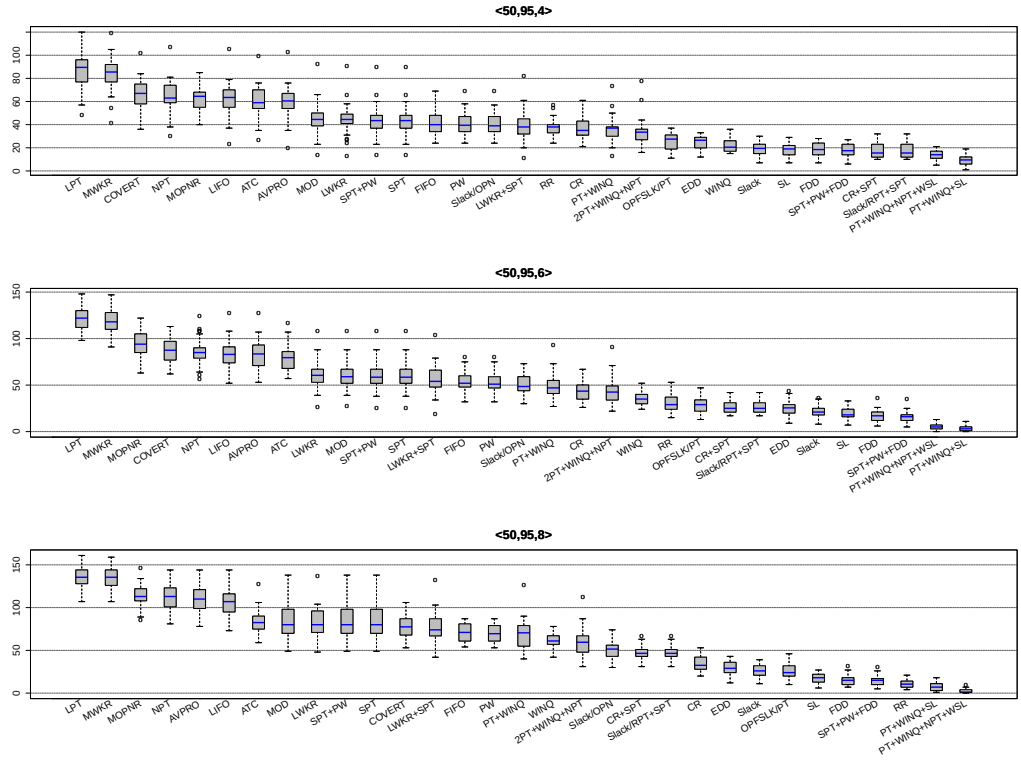


Figure 5.13: NDER for each existing dispatching rules (processing times from [1,99] and utilisation of 95%).

exploring this search space seems to be an impossible task. For that reason, there is a need for automatic design methods such as the MO-GPHH proposed in this work. The extensive experimental results shown here have convincingly confirmed the effectiveness of the proposed MO-GPHH method for evolving dispatching rules for DJSS problems. It is totally possible for the proposed method to evolve rules that are significantly better than rules reported in the literature, not only on a specific objective but also on different objectives of interest.

5.3 Further Analysis

The previous section has shown the performance of the evolved rules when single objective and multiple objectives are considered. In this section, we will provide more insights on the distribution and robustness of the evolved rules on the obtained Pareto front. Some examples of evolved rules are also shown here to demonstrate their robustness as well as how the evolved rules are more effective as compared to the existing rules.

5.3.1 Evolved Pareto front

The comparison results have shown that the proposed MO-GPHH method can evolve very competitive rules. However, we have not fully assessed the advantages of the proposed MO-GPHH methods, more specifically the advantages of the evolved Pareto front of non-dominated evolved rules. In Figure 5.14, we show the *aggregate Pareto front* including the non-dominated evolved rules extracted from Pareto fronts generated by all MO-GPHH runs (based on the traditional Pareto dominance concept) in the scenario with the shop condition $\langle 25, 85, 4 \rangle$. This figure is a scatter plot matrix which contains all the pairwise scatter plots of the five objectives (the two scatter plots which are symmetric with respect to the diagonal are similar except that the two axes are interchanged). The objective values obtained by 31 existing rules are also plotted in this figure (as +).

The first observation is that the Pareto front can cover a much wider range of potential non-dominated rules compared to rules that have been discovered in the literature. The figure not only shows that the evolved rules can dominate the existing rules but the Pareto front of evolved rules also helps with understanding better about the possible trade-offs in this scenario. For example, it can be seen that the percentage of tardy jobs %T can be substantially reduced with only minor deterioration on other objectives. Obviously, this insight cannot be obtained with the available dispatching rules since these rules only suggest that other objectives will be

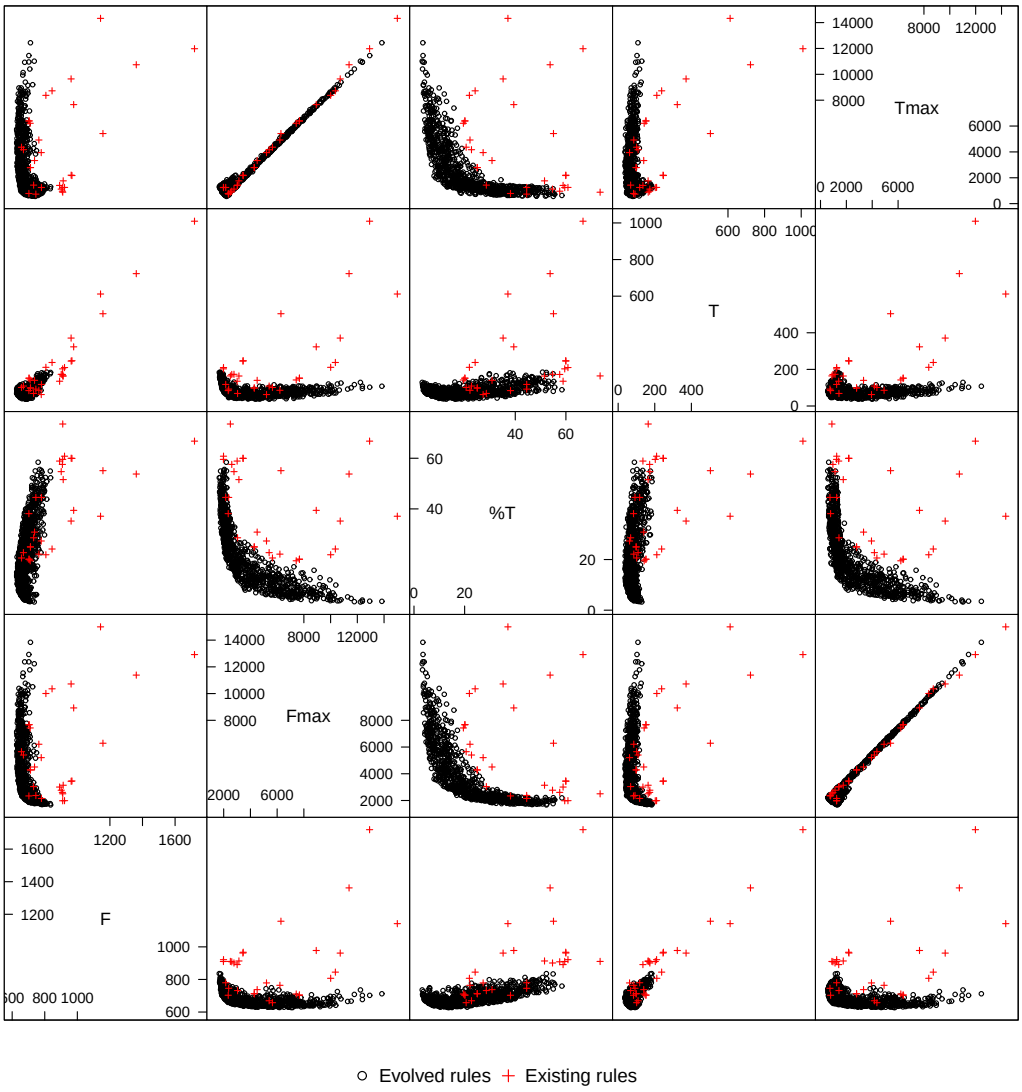


Figure 5.14: Distribution of rules on the evolved Pareto front for the scenario $\langle 25, 85, 4 \rangle$.

deteriorated significantly when we try to reduce %T below 20%. However, we can see from the Pareto front that it is possible to reduce %T further to 10% without major deteriorations in other objectives. In fact, F and T will not be affected when we try to reduce %T to a level above 10%. When

we try to reduce %T below 10%, F_{max} and T_{max} will be greatly deteriorated. In this scenario, we also see that there is a strong correlation between F_{max} and T_{max} when the values are high and the trade-offs between these two objectives are only obvious when they reach their lowest values. This makes sense since high values of F_{max} and T_{max} are caused by some extreme cases. Thus, as long as these extreme cases are handled well, both F_{max} and T_{max} can also be reduced. This observation also suggests that focusing on one of them should be enough if these two objectives are not very important.

This visualisation shows that decision makers can benefit greatly from the Pareto front found by the proposed MO-GPHH method. For DJSS problems, the ability to understand all possible trade-offs is very important since many aspects need to be considered when a decision needs to be made. Without the knowledge from these trade-offs, the decisions will be too extreme (only focus on a specific objective) and they can be practically unreasonable sometimes (e.g. double the maximum tardiness just for reducing %T by 1%). Moreover, the decision makers do not need to decide their preferences on the objectives before the design process, which could be quite subjective in most cases.

5.3.2 Robustness of the evolved dispatching rules

It has been shown that the evolved Pareto fronts contain very competitive rules. In this section, we will investigate the robustness of the evolved rules, which is their ability to maintain their performance across different simulation scenarios. In the single objective problem, the robustness of the evolved rules can be easily examined by measuring and comparing the performance of the rules on different scenarios. However, the assessment of the robustness of the evolved rules are not trivial in the case of multi-objective problems because the robustness of rules depends not only on the values of all the objectives but also on the Pareto dominance relations

of the rules. Unfortunately, there has been no standard method to measure the robustness of the evolved rules for the multi-objective problems. Therefore, we propose a method to help roughly estimate the robustness of the evolved rules. In this work, the robustness of a rule $\mathcal{R}_i$ will be calculated as follows:

$$robustness_i = 1 - \frac{\sum_{s \in \mathbb{S}} Hamming\_Distance(dom_{is}, dom^*_{is})}{|\mathbb{S}| \times |\mathbb{B}|} \tag{5.2}$$

where $dom_{is} = \{d_{is1}, \ldots, d_{isj}, \ldots, d_{is|\mathbb{B}|}\}$ is a binary array which stores the Pareto dominance between $\mathcal{R}_i$ and each rule $\mathcal{B}_j$ in the set $\mathbb{B}$ of reference rules. In a simulation scenario $s \in \mathbb{S}$ (12 test scenarios in our work), d_{isj} is assigned 1 when $\mathcal{R}_i$ statistically dominates $\mathcal{B}_j$, and 0 otherwise. Here, we include in $\mathbb{B}$ ten benchmark rules that are most competitive in Figure 5.10 and Figure 5.11 (FDD, Slack/OPN, OPFSLK/PT, SPT+PW+FDD, PT+WINQ+NPT+WSL, PT+WINQ+SL, RR, 2PT+WINQ+NPT, COVERT, and SL). Meanwhile, dom^*_{is} is also a binary array which contains the most frequent value of d_{isj} across all $s \in \mathbb{S}$. The second term in equation (5.2) measures the average Hamming distance per dimension between dom_{is} and dom^*_{is}. From this calculation, if the Pareto-dominance relations between $\mathcal{R}_i$ and each rule $\mathcal{B}_j$ are consistent across all $s \in \mathbb{S}$, this term will be zero and the robustness is one. In the worst case when the Pareto dominance relations are greatly different for each scenario s, the second term in equation (5.2) will approach 1 and the robustness will be near zero.

A histogram of robustness values of all evolved rules obtained by 30 MO-GPHH runs is shown in Figure 5.15. The density in this figure is the number of rules with robustness within the corresponding range (bin) of the histogram. It is clear that the distribution of robustness values is skewed to the right, which indicates that the evolved rules are reasonably robust. The majority of the rules have robustness values from 0.8 to 0.95 and there is only a small proportion of evolved rules with small robustness. This result is consistent with our observation in Section 5.2.2

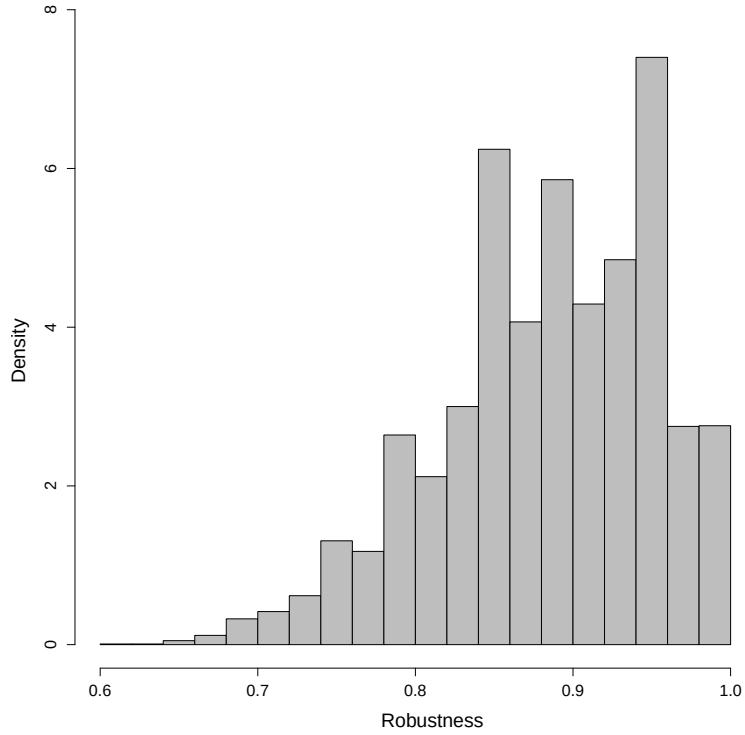


Figure 5.15: Robustness of the evolved rules.

that a small number of evolved rules that do not perform well on unseen scenarios can be dominated by the benchmark rules, in which case their Pareto-dominance relations are changed.

5.3.3 Examples of evolved dispatching rules

This section shows examples of the evolved dispatching rules. Since many rules have been evolved, it is impossible to list all of these rules. We only show here ten typical rules which can achieve balanced performance on all objectives that we considered in this work. These ten rules are shown in Table 5.5 along with their average objective values obtained from the training scenarios. In general, the example rules shown in the table are quite long and include different terminals from Table 5.1. This suggests that different information needs to be considered in order to make good

sequencing decisions that can favour all objectives. Therefore, it seems to be infeasible to design such rules manually, especially when different trade-offs have to be taken into account. Although the rules here are quite long, they are mainly synthesised based on very basic mathematical operations, and therefore it is possible to simplify these rules or to understand how they can effectively solve the DJSS problems.

The performance of the example rules and some benchmark rules on two unseen simulation scenarios $\langle 50, 95, 4\rangle$ and $\langle 50, 95, 6\rangle$ are shown in Table 5.6. It is easy to realise that most benchmark rules are dominated, regarding all objectives, by some example evolved rules. For instance, sophisticated rules such as RR and COVERT are greatly dominated by rules #3 and #4 in the two testing simulation scenarios. PT+WINQ+SL is the only benchmark rule that is not dominated by our example rules, based on the average objective values shown in the table. This is not surprising since PT+WINQ+SL is one of the most competitive rules, but there are still several evolved rules that can dominate PT+WINQ+SL as shown in Figures 5.10–5.13. Rules #1 and #7 are two rules with results quite similar to those from PT+WINQ+SL and only slightly worse than PT+WINQ+SL in some objectives. In $\langle 50, 95, 4\rangle$, rule #1 is only worse than PT+WINQ+SL for T. However, it is noted that rule #1 can achieve much better %T and T_{max}.

5.4 Chapter Summary

Most of the dispatching rules for DJSS problems proposed in the literature are designed for minimising a specific objective. However, the choice of a suitable dispatching rule has to depend on the performance of the rule across multiple conflicting objectives. In this work, we show how we can use GP to handle this issue. The proposed MO-GPHH method aims at exploring the Pareto front of evolved rules which can be used to support the decision making process. Extensive experiments have been performed

Table 5.5: Some typical examples of evolved rules

Rule #1 – Objectives(757.16, 3520.19, 0.17, 164.52, 1811.77)
$(((\mathtt{IF}(\mathtt{SJ},\mathtt{RJ},\max(\mathtt{PR},\mathtt{WT}))+(\max(\mathtt{RO},\mathtt{RT})+(\mathtt{RJ}/\mathtt{IF}(\mathtt{SJ},\mathtt{PR},\mathtt{rJ}))))-\mathtt{WINQ})+(((\max(\mathtt{RO},\mathtt{RT})+\mathtt{IF}(\mathtt{SJ},\mathtt{IF}(\mathtt{SJ},\mathtt{PR},\mathtt{rJ}),\mathtt{rJ}))+(-1\times(\mathtt{IF}(\mathtt{SJ},\mathtt{PR},\mathtt{rJ}))+\mathtt{IF}(\mathtt{SJ},\mathtt{DD}/\mathtt{PR},\mathtt{rJ})))-\min(\mathtt{SJ},(\mathtt{WINQ}\times\min(\mathtt{PR},\mathtt{WINQ})))))-\mathtt{Abs}((\mathtt{rJ}-\mathtt{RT})+\min(\min(\mathtt{SJ},\mathtt{IF}(\mathtt{SJ},\mathtt{PR},\mathtt{rJ})),(\mathtt{rJ}-\mathtt{RT})))$
Rule #2 – Objectives(828.45, 2322.88, 0.19, 165.04, 1931.40)
$(-\mathtt{rJ}-\mathtt{SJ}+\max(\mathtt{RO},\mathtt{RT})+(((((\mathtt{RJ}/\mathtt{PR})+\max(\mathtt{RO},\mathtt{RT}))+\max(\mathtt{PR},\max(\mathtt{RO},\mathtt{RT})))+(-\mathtt{PR}-\mathtt{RT}))-0.8968051))-\mathtt{Abs}(\mathtt{IF}(\min(\mathtt{SJ},\mathtt{WINQ}),\mathtt{WINQ},\mathtt{DD}/\mathtt{PR})+\mathtt{Abs}(\min(\mathtt{SJ},\mathtt{WINQ})))$
Rule #3 – Objectives(720.28, 4383.52, 0.09, 105.67, 2401.59)
$((\max(\mathtt{RM},(\mathtt{Abs}(\min(\mathtt{WT},\mathtt{SJ}))\times(\mathtt{RT}\times\mathtt{PR})))/\mathtt{PR})/\mathtt{Abs}(\mathtt{PR}+\mathtt{RO}))/\mathtt{Abs}(\max(((\mathtt{PR}\times\max(\mathtt{RT},\max(\mathtt{APR},\mathtt{SJ})))\times(\mathtt{PR}+\mathtt{WINQ})),((\mathtt{Abs}(\mathtt{PR})\times(\mathtt{RT}\times\mathtt{PR}))\times\mathtt{DD})\%\mathtt{Abs}(\min(\mathtt{WT},(\mathtt{SJ}/\mathtt{APR})))))$
Rule #4 – Objectives(716.52, 3842.36, 0.11, 82.67, 1714.88)
$((\max((\mathtt{PR}\times\mathtt{APR}),(\mathtt{Abs}(\min(\mathtt{WT},\mathtt{SJ}))\times\mathtt{WINQ}))/\mathtt{PR})\%\mathtt{WINQ})/\mathtt{Abs}(\max(((\mathtt{PR}\times\mathtt{PR})\times(\max(\mathtt{Abs}(\mathtt{RT}),\max(\mathtt{APR},\mathtt{SJ}))+\min(\mathtt{WT},(\mathtt{SJ}/\mathtt{APR})))),((\mathtt{PR}\times\mathtt{WINQ})\times\mathtt{DD})\%\mathtt{Abs}(\min(\mathtt{WT},(\mathtt{SJ}/\mathtt{APR})))))$
Rule #5 – Objectives(708.05, 4141.63, 0.13, 109.08, 1977.63)
$(((((\mathtt{RT}/\mathtt{rJ})+\mathtt{rJ})/\max(\min(\mathtt{DD},\mathtt{SJ}),\mathtt{RT}))-\min(-(\mathtt{IF}(\mathtt{SJ},\mathtt{RJ},\mathtt{NPR})/(\mathtt{SJ}+\mathtt{WINQ})),\mathtt{DD}))+(-\mathtt{WINQ}+(-\mathtt{RO}-\min(\min(\mathtt{SJ},\mathtt{WINQ}),\mathtt{rJ}))))+((\max(\mathtt{SJ},\mathtt{rJ})+((\mathtt{IF}(\mathtt{SJ},\mathtt{RJ},-\mathtt{RO})/\mathtt{PR})-(\mathtt{rJ}+\max((\mathtt{WINQ}+\mathtt{PR}),0.371))))-\mathtt{NPR})$
Rule #6 – Objectives(687.85, 5708.02, 0.16, 134.13, 4046.06)
$\mathtt{Abs}((((\mathtt{RJ}/\mathtt{SJ})/\mathtt{PR})/\mathtt{PR})/\max(\mathtt{APR},\mathtt{WINQ}))\times\mathtt{Abs}(((((\mathtt{SJ}/\mathtt{APR})-\mathtt{SJ})/\min(\mathtt{RT},\mathtt{SJ}))\times\min(\mathtt{RT},\mathtt{SJ}))/\min(((\mathtt{RJ}/\mathtt{SJ})\times(\mathtt{RJ}/\mathtt{SJ})),\mathtt{RT}))$
Rule #7 – Objectives(798.58, 3383.23, 0.15, 73.83, 602.15)
$((\mathtt{SJ}/\mathtt{APR})+(-(\min(\min(\mathtt{RT},\mathtt{PR}),\mathtt{rJ})-(\min(\mathtt{PR},\mathtt{SJ})-\min(\mathtt{WINQ},\mathtt{SJ})))-\min(\mathtt{RT},\mathtt{SJ})))+((\mathtt{RT}+(((\mathtt{RJ}/\mathtt{SJ})/\mathtt{PR})\times\min(\mathtt{RT},\mathtt{SJ})))+(\mathtt{WINQ}-\max(\mathtt{APR},\mathtt{WINQ})))$
Rule #8 – Objectives(697.64, 6306.54, 0.06, 114.89, 4488.11)
$\mathtt{Abs}((((\mathtt{RJ}/\mathtt{SJ})/\mathtt{PR})/\mathtt{PR})/\min((\min(\mathtt{PR},(\mathtt{RJ}/\mathtt{SJ}))-(\mathtt{SJ}/\mathtt{rJ})),(\mathtt{DD}-\mathtt{WINQ})))\times\mathtt{Abs}(((\mathtt{PR}/(\min(\mathtt{PR},\mathtt{RM})+\mathtt{WINQ}))\times\min(\mathtt{RT},\mathtt{SJ}))/\min((\mathtt{RT}\times\min(\mathtt{PR},(\mathtt{RJ}/\mathtt{SJ}))),\mathtt{RT}))$
Rule #9 – Objectives(845.94, 2261.88, 0.27, 150.63, 1074.42)
$((((-\mathtt{rJ}-(-\mathtt{rJ}\%(\mathtt{SJ}-\mathtt{RT})))-(\mathtt{WINQ}+(0.559+\mathtt{PR})))-\max(\mathtt{Abs}(-\mathtt{RT}+\mathtt{rJ}),-\mathtt{rJ}))-((\mathtt{SJ}-\max(\mathtt{Abs}(\mathtt{SJ}-\mathtt{RT}),\mathtt{SJ}))-(-\mathtt{rJ}/\min((\mathtt{RM}+\mathtt{WT}),\mathtt{DD}))))+\mathtt{RT}-1-\mathtt{SJ}+\mathtt{DD}-\mathtt{APR}\times\mathtt{PR}-\mathtt{WINQ}-3\times(0.559+\mathtt{PR})$
Rule #10 – Objectives(737.34, 3790.15, 0.13, 84.86, 1118.69)
$\max((\mathtt{WT}-2\times\mathtt{SJ}+0.563716-\mathtt{APR}\times\mathtt{PR}),((\mathtt{Abs}(\mathtt{IF}(\mathtt{PR}+\mathtt{SJ},\mathtt{RM}/\mathtt{PR},2\times\mathtt{SJ}-\mathtt{WT}))/\max(\mathtt{PR}+0.024362229,\max(\mathtt{SJ},\mathtt{RT})))/\mathtt{Abs}(\max(-\mathtt{APR}+\mathtt{WINQ},\mathtt{APR})/(\mathtt{RM}/\mathtt{PR}))))$

*IF(a, b, c) will return b if a $\geq$ 0; otherwise it will return c.

Table 5.6: Performance of example evolved rules in $\langle \mathbf{50}, \mathbf{95}, \mathbf{4} \rangle$

$\langle \mathbf{50}, \mathbf{95}, \mathbf{4} \rangle$	$\mathbf{F}$	$\mathbf{F_{max}}$	$\%\mathbf{T}$	$\mathbf{T}$	$\mathbf{T_{max}}$
PT+WINQ+SL	2991.03	7551.84	88.47	1431.15	4491.71
Slack/OPN	4532.45	13977.43	97.45	2930.66	11409.89
RR	2943.40	15050.27	85.46	1374.69	12255.53
COVERT	2744.83	37158.13	77.18	1166.01	34645.73
2PT+WINQ+NPT	2355.21	31321.28	45.65	1010.55	28785.17
PT+WINQ+NPT+WSL	3356.03	7695.30	94.06	1771.03	4481.94
SPT+PW+FDD	3710.73	6769.10	96.94	2115.04	5268.29
OPFSLK/PT	3108.31	7648.78	91.88	1530.30	5909.04
FDD	3740.53	6810.86	96.98	2144.74	5322.64
SL	3638.32	7967.81	98.59	2038.43	4732.04
Rule #1	2933.19	7096.85	68.44	1446.81	4193.42
Rule #2	3381.54	6018.19	85.10	1812.70	5486.42
Rule #3	2548.54	9075.00	46.06	1126.13	6047.45
Rule #4	2472.12	9615.42	51.21	1018.41	6983.91
Rule #5	2527.99	8182.37	56.04	1081.19	5192.21
Rule #6	2229.99	13394.98	37.33	928.19	10625.92
Rule #7	3049.49	7492.19	90.03	1462.91	4406.31
Rule #8	2323.42	16446.96	24.61	1021.91	13515.90
Rule #9	3042.21	6800.97	86.39	1477.09	4645.73
Rule #10	2629.42	7945.21	63.78	1124.64	4772.82

Table 5.7: Performance of example evolved rules in $\langle \mathbf{50}, \mathbf{95}, \mathbf{6} \rangle$

$\langle \mathbf{50}, \mathbf{95}, \mathbf{6} \rangle$	$\mathbf{F}$	$\mathbf{F_{max}}$	$\%\mathbf{T}$	$\mathbf{T}$	$\mathbf{T_{max}}$
PT+WINQ+SL	2714.00	7982.00	61.49	657.95	3148.09
Slack/OPN	3941.20	12989.18	77.26	1581.59	8974.86
RR	2811.91	11882.35	53.49	611.18	7107.35
COVERT	2999.75	31445.59	56.98	676.51	27970.88
2PT+WINQ+NPT	2355.21	31321.28	26.12	701.61	27540.38
PT+WINQ+NPT+WSL	2932.42	8021.92	68.06	771.68	3037.79
SPT+PW+FDD	3710.73	6769.10	81.53	1421.97	5064.32
OPFSLK/PT	3108.31	7648.78	68.09	927.30	5441.89
FDD	3740.53	6810.86	82.00	1448.29	5118.74
SL	3524.23	8647.86	91.95	1163.81	3672.67
Rule #1	2768.33	8172.39	40.60	817.59	3702.13
Rule #2	3264.87	6243.41	57.39	1081.99	5428.79
Rule #3	2444.66	9914.98	26.03	576.57	5190.51
Rule #4	2419.77	9898.71	28.31	520.51	5564.72
Rule #5	2426.92	9221.52	34.36	585.90	4464.28
Rule #6	2213.12	14116.06	23.18	651.85	9846.80
Rule #7	2761.50	7893.13	51.89	588.13	3061.45
Rule #8	2280.87	17798.84	14.04	627.54	13237.53
Rule #9	2934.48	7009.52	60.37	798.31	3879.64
Rule #10	2487.80	9041.86	34.35	514.90	4150.00

and the results show that the evolved Pareto front contains superior rules as compared with rules reported in the literature when both single and multiple objectives are . Moreover, it has been shown that the obtained Pareto front can provide valuable insights on how trade-offs should be made.

We have also discussed and implemented different analyses on the experimental results, which help us confirm the effectiveness of the evolved dispatching rules. In these analyses, we focus on two issues. First, we try to define a standard procedure in order to properly compare the performance of rules within the multi-objective stochastic environments. Second, we need to find a way to assess the robustness of the evolved rules under different simulation scenarios. Although they are two very important issues, there have been no existing guidelines on how they should be done. In this work, we proposed different approaches to handle these issues. Even though there are still some limitations with these approaches, they can nevertheless be used as a good way to assess the performance of rules in such a complicated problem. Certainly, these two issues can also be interesting issues for future studies.

In this chapter, we have employed TWK to assign due dates to new arriving jobs. While this due date assignment rule is easy to apply, it may not incorporate effectively with the existing or evolved rules to achieve better scheduling objectives. Therefore, it is important to design suitable due date assignment rules to maximise the effectiveness of dispatching rules. The next chapter will investigate new GP methods to evolve effective and reusable due date assignment rules for job shop scheduling.

Chapter 6

Evolving Due Date Assignment Rules

A large number of studies on JSS have focused on sequencing decisions, which determine the order in which waiting jobs are processed on a set of machines in a manufacturing system. However, sequencing is only one of several steps in the scheduling process [1]. One of the other important activities in JSS is due date assignment (DDA), sometimes referred to as estimation of job flowtimes (EJF). This activity arises when a manager need to *"promise"* a delivery date to a customer. The objective of DDA is to determine the due dates for arriving jobs by estimating the job flowtimes (the time from the arrival until the completion of the job), and therefore DDA strongly influences the delivery performance, i.e., the ability to meet promised delivery dates, of a job shop [40].

Many due date assignment rules (DDARs) have been proposed in the job shop literature. The traditional DDARs focus on exploiting the shop and job information to make a good flowtime estimation. Most of the early DDARs are based on linear combinations of different terms (variables) and the coefficients of the rules are then determined based on simulation results. Regression has been used very often in order to help find the best coefficients for the rules employed [160, 62, 168, 176, 94]. Since the early

1990s, artificial intelligence and statistical methods have also been applied to deal with DDA problems, e.g., gene expression programming [17], neural networks [174, 147], decision trees [144], regression trees [175], and a regression based method with case-based tuning [176].

Even though experimental results with these DDARs are promising, some limitations are still present. First, since a job can include several operations which represent the processing steps of that job at particular machines, the operation-based flowtime estimation (OFE) method [168], which utilises the detailed job, shop and route information for operations of jobs, can help improve the quality of the prediction. However, this OFE method depends strongly on the determination of a large number of coefficients, which is not an easy task. Thus, there is a need to create a dynamic OFE method similar to Dynamic Total Work Content (DTWK), Dynamic Processing Plus Waiting (DPPW) [41], and ADRES [19] to overcome this problem by replacing the coefficients with more general aggregate terms (job characteristics and states of the system). Second, there are no studies on the reusability of the DDARs in the JSS literature, so it is questionable whether the rules can be applied when there are changes in the shop without major revisions. Finally, various relevant factors need to be considered in order to make a good estimation of flowtime, which makes the design of a new DDAR a time-consuming and complicated task.

This chapter aims to develop a new approach that employs GP to evolve *reusable* DDARs for job shop environments. We expect the evolved DDARs to outperform the existing rules in terms of mean absolute percentage error and to be reusable for new (unseen) job shop simulation scenarios. Two types of DDARs considered in this study are Aggregate Due date Assignment Rules (ADDARs) and Operation-based Due date Assignment Rules (ODDARs). The difference between these two rules is that ADDARs employ the aggregate information from jobs, machines and the shop to predict the due date while ODDARs indirectly predict the due date by estimating the flowtime of each operation.

The objectives for this chapter are:

1. Developing GP methods to automatically evolve reusable ADDARs and ODDARs for the job shop environment.

2. Comparing the evolved DDARs obtained from the two GP methods with existing DDARs.

3. Analysing the proposed GP methods and the evolved DDARs to understand how these rules can estimate flowtime well, with good reusability.

The rest of this chapter is organised as follows. The development and descriptions of ADDAR and ODDAR are given in Section 6.1 and the experimental setting is presented in Section 6.2. The experimental results and the comparison of DDARs are provided in Section 6.3. Analysis of the proposed algorithm and evolved DDARs is presented in Section 6.4. Further investigation into sophisticated dispatching rules is shown in Section 6.5.

6.1 GP for evolving DDARs

In this section, we describe two GP methods GP-ADDAR and GP-ODDAR to evolve ADDARs and ODDARs, respectively. First, the representation and evaluation scheme are discussed. Then, a fitness function is provided to measure the performance of the evolved DDARs.

6.1.1 Representation

The purpose of the proposed GP-ADDAR and GP-ODDAR is to evolve dynamic ADDARs and ODDARs that estimate job flowtimes (i.e. due dates by using equation (2.1)) by employing information from jobs and the shop similar to DTWK and DPPW. In this case, we use tree-based GP to

create mathematical combinations of these pieces of information in each GP individual. For this reason, the function set will consist of standard mathematical operators $+$,$-$,$\times$, and protected division $\%$, along with a conditional function `If` to allow GP to evolve sophisticated DDARs. The protected division function $\%$ returns a value of 1 when division by 0 is attempted. Function `If` includes three arguments and if the value from the first argument is greater than or equal to zero, `If` will return the value from the second argument; otherwise `If` will return the value from the third argument. Since ADDARs and ODDARs need different types of information, GP-ADDAR and GP-ODDAR will use different terminal sets as shown in Table 6.1. In this table, the first five terminals are the same for the two proposed GP methods. The next eight terminals are variables that characterise the state of operations/machines for GP-ODDAR and their aggregate counterparts for GP-ADDAR. The last terminal of each method provides extra information to estimate the flowtime. While `SL` provides the information of previous arriving jobs, `PEF` is especially proposed in this study to estimate the changes of the system through the period of time the new job spends in the system (more details are given in Section 6.1.2). `SOTR` and `SAPR` are calculated based on the 20 previous jobs processed at machine δ. On the other hand, `SAR` and `SL` are calculated based on the arrivals of the last 100 jobs and 20 jobs respectively.

6.1.2 Evaluation

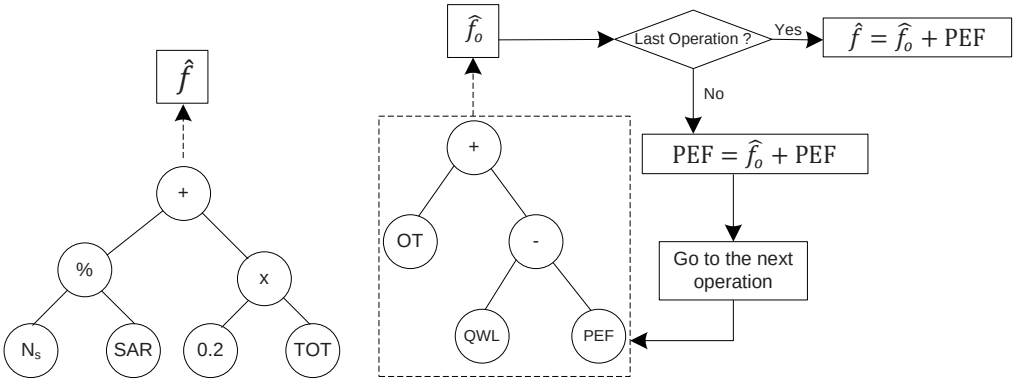
An example of how an individual in GP-ADDAR is evaluated is shown in Figure 6.1(a). In this method, a GP individual represents a mathematical function and the output of this function is the estimated flowtime $\hat{f}$ of the new job. The information used in this function is extracted from the new job and machines in the shop.

The GP individual in GP-ODDAR aims at estimating the flowtime of each operation of the new job. Therefore, instead of using the function obtained from the GP individual to estimate job flowtime $\hat{f}$, the output of

Table 6.1: Terminal sets for GP-ADDAR and GP-ODDAR (ψ is the new job, ϕ is the considered operation in GP-ODDAR, and δ is the machine that process will ϕ)

GP-ADDAR		GP-ODDAR	
N		Number of jobs in the shop	
SAR		Sampled arrival rate	
NO		Number of operations of job ψ	
M		Number of machines	
#		Random number from 0 to 1	
TAPR	total average processing time of job in queues of machines that ψ will visit	APR	average processing times of jobs in the queue of the machine that processes ϕ
TOT	total processing time of ψ	OT	processing time of ϕ
TLOT	average LOT for all machines that ψ will visit	LOT	time for δ to finish the leftover job
AOTR	average OTR for all queues of machines that ψ will visit	OTR	percentage of jobs in queues of δ that require less processing time than OT
ASOTR	average SOTR for all queues of machines that ψ will visit	SOTR	percentage of sampled jobs processed at δ that require less processing time than OT
TQWL	total QWL for all machines that ψ will visit	QWL	total processing time of jobs in the queue of δ
TSAPR	total SAPR for all machines that ψ will visit	SAPR	sampled average processing time of jobs processed at δ
TRWL	total RWL for all machines that ψ will visit	RWL	total processing time of jobs that need to be processed at δ
SL	sampled average number of operations of jobs	PEF	partial estimated flowtime

this function is used to estimate the operation flowtime $\hat{f}_o$ of each operation of the new job, starting from the first operation. When $\hat{f}_o$ is obtained, a condition is checked to see whether the operation being considered is the last operation. If it is not the last operation of the new job, $\hat{f}_o$ will be used to update the partial estimated flowtime (PEF), which will also be used as a terminal in the GP individual. So PEF here is a dynamic terminal. Then, the GP individual is applied to estimate the flowtime for the next operation. In the case that the flowtime of the last operation has been estimated,



(a) GP-ADDAR (b) GP-ODDAR

Figure 6.1: DDAR evaluation scheme

$\hat{f}_o$ will be added to the current PEF to obtain the final estimated flowtime $\hat{f}$. The evaluation scheme for GP-ODDAR is shown in Figure 6.1(b) (noting that only the tree in the figure is evolved by the GP). The use of PEF (initially zero for the first operation) in the terminal set of GP-ODDAR also provides DDARs a chance to predict the changes of the system, assuming that the partial estimated flowtime is predicted well.

6.1.3 Fitness Function

As discussed in Section 2.2.3, the performance of a DDAR can be measured in many different ways, which indicate the delivery accuracy and delivery reliability. In this study, we will use `MAPE` to measure the quality of evolved DDARs because it is a good indicator for both delivery accuracy and delivery reliability. A discrete-event simulation model of a job shop was implemented in order to evaluate the evolved DDARs. In this model, the inter-arrival times of jobs, the processing times and route information of jobs will follow some particular probability distributions (more details are provided in Section 6.2.1). Upon the arrival of a job j, the DDAR will be applied to estimate the flowtime $\hat{f}_j$ of that job. The error e_j of this estimation is recorded when job j leaves the system and the errors of all

recorded jobs will be used to calculate MAPE as shown in Table 2.3 (on page 2.3). Since our objective is to evolve reusable DDARs, the quality of the evolved DDARs will be measured based on their performance on a number of simulation scenarios $\mathbb{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$, which represent different shop characteristics. For a simulation scenario $\mathcal{S}_k$, the quality of a DDAR p_i is indicated by $\texttt{MAPE}_{p_i}^{\mathcal{S}_k}$. The fitness value of p_i is calculated as followed:

$$fitness(p_i) = \frac{1}{K}\sum_{k=1}^{K} \texttt{MAPE}_{p_i}^{\mathcal{S}_k} \tag{6.1}$$

With this design, a smaller $fitness(p_i)$ indicates that the evolved DDAR p_i produces more accurate estimations of jobs across different scenarios.

6.1.4 Evolution of DDARs

The same GP method as shown in Figure 2.8 (page 50) is used to evolve ADDARs and ODDARs. A variety of simulation scenarios will be employed in this algorithm to provide the evolved (trained) DDARs better generality, but it should be noted that a large number of scenarios also increases the computation time of the GP systems. The evolutionary process will be terminated when the maximum generation is reached and the algorithm will return the best found DDAR p^*. Also notice that GP-ADDAR and GP-ODDAR use the same algorithm, but the terminals used by these two methods are different since they have different focus (as mentioned in Sections 6.1.1 and 6.1.2).

6.2 Experimental Setting

This section discusses the simulation environments in which the DDARs are trained or evolved. Then, the details of the training and testing scenarios are provided. Finally, the settings of the GP systems are given.

6.2.1 Job Shop Simulation Environment

Simulation is the most popular method to evaluate the performance of a DDAR in the JSS literature. Since our goal is to design reusable DDARs, a general job shop would be more suitable than a specific shop. The following factors characterise a job shop:

- Number of machines (F1)
- Utilisation (F2)
- Arrival process
- Distribution of processing times (F3)
- Distribution of number of operations (F4)

The number of machines will be the main factor that shows the scale of the shop; this may also influence the complexity of the JSS decisions. Utilisation, on the other hand, is the proportion of time a machine is busy. The performance of the JSS decisions under different utilisation levels are of interest in most research in the JSS literature. The arrival process, distribution of processing times and number of operations are factors that directly influence the difficulty of JSS decisions.

In our experiments, we employ a symmetrical job shop model in which each operation of a job has equal probability to be processed at any machine in the shop (a job visits each machine at most once). Therefore, machines in the shop expect to have the same level of congestion in long simulation runs. This model has been used very often in the JSS literature [35, 41, 168, 108, 78]. This simulation model can be considered as an extension of the simulation applied in Chapter 5 to further explore other key factors in JSS such as number of machines, distributions of processing time and distribution of number of operations. Based on the discussion above, the scenarios for training and testing of DDARs are designed and shown in Table 6.2.

Table 6.2: Training and testing scenarios

Factor	Training	Testing
F1	4,6	4,5,6,10,20
F2	70%,80%,90%	60%,70%,80%,90%,95%
F3	Exponential	Exponential, Erlang-2, Uniform
F4	missing	missing, full

In these experiments, without loss of generality, the processing times are randomly generated based on a specific distribution with mean equal to 1 and the arrival of jobs will follow a Poisson process with the arrival rates adjusted based on the utilisation level [162]. For the distribution of number of operations, the *missing* setting is used to indicate that the number of operations will follow a discrete uniform distribution from 1 to the number of machines. Meanwhile, the *full* setting indicates the case that each job will have its number of operations equal to the number of machines in the shop. In each replication of a simulation scenario, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the 1000^{th} job is considered as the warm-up time and the information collected from the next 5000 completed jobs (set $\mathbb{C}$ in Section 2.2.3) is used to evaluate the performance of DDARs.

In the training stage of a particular GP run, since the simulation is very time-consuming, we only perform one replication for each scenario. There are $(2\times3\times1\times1) = 6$ simulation scenarios used to evaluate the performance of the evolved DDARs. For testing, the best DDAR p^* obtained from a run of GP is applied to $(5 \times 5 \times 3 \times 2) = 150$ simulation scenarios and 30 simulation replications are performed for each scenario; therefore, we need $150 \times 30 = 4500$ simulation replications to test the performance of p^*. The use of a large number of scenarios and replications in the testing stage will help us confirm the quality and reusability of the evolved DDARs. For

Table 6.3: Parameters of the proposed GP systems

Population size	1000	Crossover rate	80%
Mutation rate	15%	Reproduction rate	5%
Generations	50	Maximum depth	10

the shop floor level, First-In-First-Out (FIFO) is used as the dispatching rule to sequence jobs in queues of machines. By using FIFO, the earliest job that joins the queue of the machine will be processed first. We adopt FIFO in this study because it is one of the most popular dispatching rules in the scheduling literature.

6.2.2 GP parameters

The GP system for evolving DDARs is developed based on the ECJ20 library [117]. The parameter settings of the GP system used in the rest of this study are shown in Table 6.3. The initial GP population is created using the ramped-half-and-half method [103]. Tournament selection is used to select individuals for genetic operators. Normally, a tournament selection size from 4 to 7 is used and we use a tournament size of 5 in this study in order to maintain a balance between diversity and the convergence of the proposed GP methods [103]. Parameters in Table 6.3 are similar to those in other applications of GP [103]. Since the terminal set includes many different terminals, the mutation rate is set to 15% to provide sufficient genetic material through the evolution process of the proposed GP methods.

6.3 Results

A comparison of the best evolved DDARs with some existing DDARs is provided to show the effectiveness of the evolved DDARs. Then, we compare the performance of the two proposed GP methods.

6.3.1 Comparison of evolved DDARs with existing DDARs

For each GP method, 30 independent runs are performed and the best ADDARs and ODDARs obtained from each run are recorded and compared with existing dynamic DDARs (DTWK, DPPW, and ADRES). Tables 6.4 and 6.5 show the comparison between evolved DDARs and other DDARs on 150 testing scenarios. In these tables, $\langle \cdot, \cdot, \cdot \rangle$ is the number of evolved DDARs that are significantly better (by t-test with significance level of 0.05) than DTWK, DPPW, and ADRES, respectively. It is easy to see that the evolved DDARs dominate other DDARs in most scenarios. These experimental results indicate the effectiveness of the proposed GP methods for evolving DDARs. It is also interesting to see that the evolved DDARs have very good reusability since the evolved DDARs can provide superior performance even on unseen scenarios (e.g. with *full* setting).

Among the five factors discussed in Section 6.2.1, the distribution of processing times (F3) seems to have less impact on the reusability of the evolved DDARs. The use of processing times drawn from an exponential distribution provides a wide range of jobs that can also reflect the cases when processing times follow other distributions. For some scenarios with high utilisation level, *full* setting and large numbers of machines, the number of evolved DDARs that can beat DPPW is smaller, especially when the processing times follow an Exponential or an Erlang-2 distribution. Most evolved DDARs cannot show superior performance compared to DPPW on these scenarios. This may be because DPPW is based on the steady state performance of the system and can perform better in shops with less diverse jobs when the *full* setting is used and with high levels of utilisation. Another reason is that the evolved DDARs have not been trained on these extreme scenarios (this issue will be investigated in Section 6.4.2). In general, it should be noted that the existing dynamic DDARs can only perform well with less diverse jobs, which is not always the case for job shops in real world applications. On the other hand, the evolved DDARs not only show their superiority compared to the existing dynamic DDARs, but also

Table 6.4: Comparing the evolved ADDAR with existing DDARs

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 1, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 1, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 0, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨29, 1, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨29, 0, 30⟩	⟨29, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨29, 0, 30⟩	⟨29, 0, 30⟩

Table 6.5: Comparing the evolved ODDAR with existing DDARs

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 2, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 2⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 28, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 24, 30⟩	⟨30, 7, 30⟩	⟨30, 2, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 29⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 9, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 4, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 15, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 4, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 25, 30⟩	⟨28, 25, 30⟩	⟨30, 7, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 12, 30⟩	⟨30, 11, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 13, 30⟩	⟨30, 10, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 12, 30⟩	⟨30, 5, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨29, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 10, 30⟩	⟨30, 2, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨28, 26, 30⟩	⟨28, 26, 30⟩	⟨28, 25, 30⟩	⟨27, 9, 30⟩	⟨26, 0, 30⟩

show good reusability, which makes them more robust when applied to complicated job shops in real world situations.

6.3.2 GP-ADDAR vs. GP-ODDAR

Table 6.6 shows the p-values of t-tests of the average $\texttt{MAPE}_{p^*}^{\mathcal{S}_k}$ (from 30 simulation replications) over the testing scenarios between GP-ADDAR and GP-ODDAR. In this table, the highlighted values indicate that GP-ADDAR

is not significantly different from GP-ODDAR and other values show that GP-ODDAR is significantly better than GP-ADDAR (with $p\text{-}value$ smaller than 0.05). The results show that GP-ODDAR is significantly better than GP-ADDAR on most simulation scenarios, especially in the case where the *missing* setting for the number of operations is used. In the case that the *full* setting for the distribution of number of operations is used, GP-ODDAR is also significantly better than GP-ADDAR in most cases except for the scenarios with a high level of utilisation (e.g. 95%) and large numbers of machines (e.g. 20). These results suggest that the aggregate information of jobs used in ADDARs is usually sufficient for estimating the flowtime of jobs when the shop is at a high congestion level and has a large number of machines. Also, ODDARs may have difficulty estimating the operation flowtimes of later operations of jobs with a large number of operations. This observation, together with that observed in Section 6.3.1 for `DPPW`, indicates the importance of aggregate information for flowtime estimation in the cases of high utilisation levels, fewer diverse jobs and large numbers of machines. It suggests that systematic incorporation of information between ADDAR and ODDAR could enhance the accuracy of the flowtime estimation.

6.4 Analysis

The previous section has shown that the evolved DDARs can be used effectively to solve DDA problems in job shops. This section will futher investigate key issues that can influence the performance as well as the computational time of the proposed GP methods.

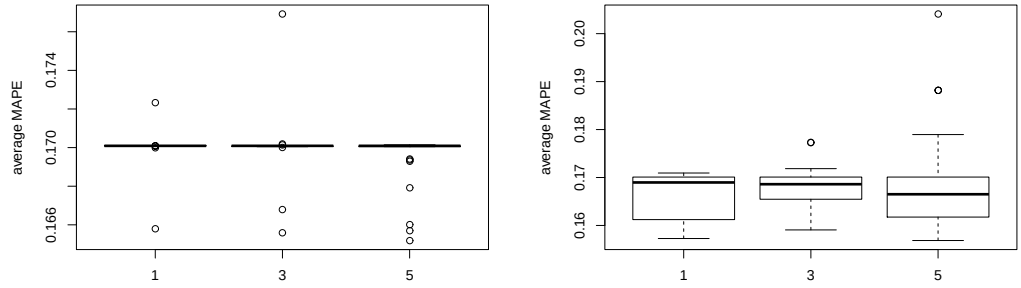
6.4.1 Number of simulation replications for training DDARs

The number of simulation replications is normally an important parameter when we try to measure the performance of a stochastic system through simulation. Even though large numbers of simulation replications are es-

Table 6.6: GP-ADDAR vs. GP-ODDAR (p-values from t-tests)

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	0.0002	0.0002	0.0002	0.0002	0.0006	0.0001	0.0001	0.0001	0.0002	0.0072
	5	0.0002	0.0001	0.0001	0.0002	0.0015	0.0001	0.0001	0.0001	0.0004	0.0273
	6	0.0001	0.0001	0.0001	0.0002	0.0007	0.0001	0.0001	0.0001	0.0016	0.0399
	10	0.0001	0.0001	0.0001	0.0002	0.0725	0.0068	0.0034	0.0087	0.1891	0.8219
	20	0.0020	0.0017	0.0012	0.0982	0.1201	0.1576	0.3126	0.7293	0.4448	0.0567
Erlang-2	4	0.0002	0.0002	0.0002	0.0002	0.0005	0.0001	0.0001	0.0001	0.0001	0.0003
	5	0.0002	0.0001	0.0001	0.0002	0.0004	0.0001	0.0001	0.0001	0.0001	0.0004
	6	0.0001	0.0001	0.0001	0.0001	0.0004	0.0001	0.0001	0.0000	0.0001	0.0006
	10	0.0001	0.0001	0.0001	0.0002	0.0058	0.0391	0.0017	0.0001	0.0065	0.0376
	20	0.0195	0.0038	0.0006	0.0016	0.6183	0.1356	0.2247	0.9441	0.5709	0.6905
Uniform	4	0.0002	0.0002	0.0003	0.0003	0.0005	0.0002	0.0002	0.0002	0.0002	0.0002
	5	0.0002	0.0003	0.0003	0.0003	0.0005	0.0006	0.0003	0.0002	0.0001	0.0001
	6	0.0003	0.0003	0.0003	0.0003	0.0004	0.0044	0.0006	0.0001	0.0001	0.0001
	10	0.0026	0.0007	0.0003	0.0002	0.0007	0.8953	0.3616	0.0156	0.0006	0.0006
	20	0.6324	0.8460	0.0700	0.0052	0.1147	0.1064	0.1263	0.3579	0.5118	0.4639

sential for accurately measuring the performance of the stochastic systems, they will significantly increase the computational time of the proposed GP methods. Figure 6.2 shows the average $\texttt{MAPE}_{p^*}^{\mathcal{S}_k}$ from all testing scenarios obtained by the evolved DDARs when different numbers of simulation replications are used for evaluating the evolved DDARs. This figure shows that the use of more replications for training DDARs does not appear to improve the quality of the evolved DDARs. An explanation for these results is that one replication is quite enough for training DDARs because the performance measure of the evolved DDARs has been applied thousands of times in each simulation replication, which can characterise various situations in the dynamic systems. Also, accurately determining the performance measures of evolved DDARs in the proposed GP methods is not as important as that in common simulation applications because what we need is to find out which evolved DDARs are better than the others (for individual selection in GP) and the performance measures from a single replication appears to be good enough for this purpose.



(a) GP-ADDAR (b) GP-ODDAR

Figure 6.2: Influence of the number of simulation replications on the performance of the proposed GP methods (the horizontal axis shows the number of simulation replications).

6.4.2 The choice of simulation scenarios for training DDARs

As shown in Section 6.3.1, the evolved DDARs have trouble beating DPPW in the scenarios with *full* jobs sometimes. This problem may come from the fact that the evolved DDARs have not been trained on these scenarios; therefore they cannot handle these scenarios as well as the scenarios with *missing* jobs. This raises the issue of how to choose suitable training scenarios in order to maximise the reusability of the evolved DDARs without significantly increasing the computational time of the proposed GP methods.

Tables 6.7 and 6.8 show the comparison between existing dynamic DDARs and the evolved DDARs trained on the same simulation scenarios in Table 6.2 but the *full* setting is used instead of the *missing* setting. The results in these tables suggest that the evolved DDARs can only show their superiority on the scenarios with full setting and processing times following Exponential and Erlang-2 distributions. This indicates that the evolved DDARs do not have as good reusability as those trained with *missing* jobs as shown in Tables 6.4 and 6.5. However, these evolved DDARs can dominate DPPW in some scenarios with a small number of machines, utilisation

Table 6.7: Performance of evolved ADDARs trained with *full* setting

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨4, 6, 6⟩	⟨4, 0, 0⟩	⟨6, 14, 6⟩	⟨17, 26, 15⟩	⟨23, 27, 14⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩
	5	⟨4, 8, 18⟩	⟨4, 8, 16⟩	⟨7, 16, 14⟩	⟨5, 0, 0⟩	⟨23, 26, 14⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 22, 30⟩
	6	⟨5, 8, 18⟩	⟨6, 12, 16⟩	⟨7, 16, 14⟩	⟨17, 22, 15⟩	⟨20, 23, 11⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 2, 30⟩
	10	⟨5, 5, 13⟩	⟨5, 7, 7⟩	⟨12, 21, 17⟩	⟨7, 4, 0⟩	⟨24, 24, 18⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 27, 30⟩	⟨30, 4, 30⟩	⟨30, 0, 29⟩
	20	⟨6, 12, 22⟩	⟨10, 14, 20⟩	⟨11, 18, 14⟩	⟨19, 23, 13⟩	⟨22, 23, 13⟩	⟨29, 12, 30⟩	⟨30, 9, 30⟩	⟨30, 6, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 29⟩
Erlang-2	4	⟨11, 15, 27⟩	⟨17, 18, 27⟩	⟨20, 22, 26⟩	⟨25, 25, 21⟩	⟨25, 25, 21⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 26, 30⟩
	5	⟨13, 16, 27⟩	⟨16, 17, 25⟩	⟨18, 22, 26⟩	⟨25, 25, 23⟩	⟨25, 25, 18⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 22, 30⟩
	6	⟨13, 16, 27⟩	⟨16, 17, 26⟩	⟨17, 21, 25⟩	⟨25, 25, 23⟩	⟨25, 25, 19⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 3, 30⟩
	10	⟨12, 13, 28⟩	⟨15, 17, 25⟩	⟨22, 22, 25⟩	⟨25, 25, 20⟩	⟨25, 25, 20⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 16, 30⟩	⟨30, 0, 30⟩
	20	⟨11, 11, 26⟩	⟨13, 13, 25⟩	⟨19, 19, 23⟩	⟨23, 23, 20⟩	⟨23, 23, 20⟩	⟨30, 15, 30⟩	⟨30, 11, 30⟩	⟨30, 12, 30⟩	⟨30, 0, 30⟩	⟨29, 0, 29⟩
Uniform	4	⟨12, 12, 30⟩	⟨15, 14, 27⟩	⟨19, 19, 24⟩	⟨23, 23, 23⟩	⟨25, 25, 22⟩	⟨30, 28, 30⟩	⟨30, 28, 30⟩	⟨30, 24, 30⟩	⟨30, 24, 30⟩	⟨30, 16, 30⟩
	5	⟨14, 12, 29⟩	⟨15, 14, 28⟩	⟨18, 18, 24⟩	⟨24, 23, 22⟩	⟨25, 25, 21⟩	⟨30, 28, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 25, 30⟩	⟨30, 11, 30⟩
	6	⟨13, 13, 30⟩	⟨15, 15, 27⟩	⟨16, 16, 26⟩	⟨23, 23, 23⟩	⟨24, 24, 21⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 28, 30⟩	⟨30, 25, 30⟩	⟨30, 5, 30⟩
	10	⟨12, 11, 29⟩	⟨15, 13, 26⟩	⟨15, 15, 25⟩	⟨23, 23, 23⟩	⟨24, 24, 21⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨28, 28, 30⟩	⟨29, 5, 30⟩
	20	⟨12, 10, 27⟩	⟨13, 12, 26⟩	⟨13, 13, 23⟩	⟨21, 21, 21⟩	⟨24, 24, 20⟩	⟨30, 26, 30⟩	⟨29, 23, 30⟩	⟨29, 21, 30⟩	⟨28, 18, 30⟩	⟨28, 0, 29⟩

Table 6.8: Performance of evolved ODDARs trained with *full* setting

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨21, 25, 28⟩	⟨21, 0, 0⟩	⟨28, 29, 27⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 28, 30⟩
	5	⟨22, 28, 30⟩	⟨25, 28, 30⟩	⟨28, 30, 30⟩	⟨27, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 13, 30⟩
	6	⟨23, 28, 30⟩	⟨28, 30, 30⟩	⟨28, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 27⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩
	10	⟨23, 26, 30⟩	⟨25, 28, 28⟩	⟨30, 30, 30⟩	⟨28, 22, 1⟩	⟨30, 30, 28⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 29⟩
	20	⟨28, 30, 30⟩	⟨28, 30, 30⟩	⟨29, 30, 30⟩	⟨30, 30, 29⟩	⟨30, 30, 28⟩	⟨29, 27, 30⟩	⟨30, 20, 30⟩	⟨30, 4, 30⟩	⟨30, 0, 29⟩	⟨30, 0, 28⟩
Erlang-2	4	⟨29, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 22, 30⟩
	5	⟨29, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 29⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 9, 30⟩
	6	⟨29, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 28⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩
	10	⟨29, 29, 30⟩	⟨29, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 29⟩	⟨30, 30, 28⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 13, 30⟩	⟨30, 0, 30⟩
	20	⟨28, 28, 30⟩	⟨28, 28, 30⟩	⟨29, 29, 29⟩	⟨29, 29, 29⟩	⟨29, 29, 28⟩	⟨30, 28, 30⟩	⟨30, 27, 30⟩	⟨30, 25, 30⟩	⟨30, 0, 30⟩	⟨29, 0, 29⟩
Uniform	4	⟨27, 25, 30⟩	⟨30, 28, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨27, 26, 30⟩	⟨29, 15, 30⟩
	5	⟨27, 25, 30⟩	⟨29, 28, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 28⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨28, 26, 30⟩	⟨29, 13, 30⟩
	6	⟨27, 27, 30⟩	⟨29, 27, 30⟩	⟨29, 29, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 28⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨28, 25, 30⟩	⟨28, 9, 30⟩
	10	⟨26, 26, 30⟩	⟨27, 26, 30⟩	⟨28, 28, 30⟩	⟨29, 29, 29⟩	⟨29, 29, 28⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨27, 27, 30⟩	⟨28, 23, 30⟩	⟨27, 4, 30⟩
	20	⟨26, 25, 30⟩	⟨26, 26, 30⟩	⟨28, 28, 29⟩	⟨29, 28, 29⟩	⟨28, 28, 28⟩	⟨27, 25, 30⟩	⟨26, 25, 30⟩	⟨26, 25, 30⟩	⟨28, 18, 30⟩	⟨25, 0, 30⟩

of 95% and *full* jobs, which cannot be achieved by the evolved DDARs trained with *missing* jobs. Tables 6.9 and 6.10 show the performance of evolved DDARs trained with both *full* and *missing* jobs (a total of 12 simulation scenarios used for training). These tables show that the evolved DDARs in this case have better reusability than those evolved with *full* or *missing* alone. However, it is noted that these results are obtained by doubling the number of simulation scenarios, and therefore it also doubles the computational effort of the proposed GP methods.

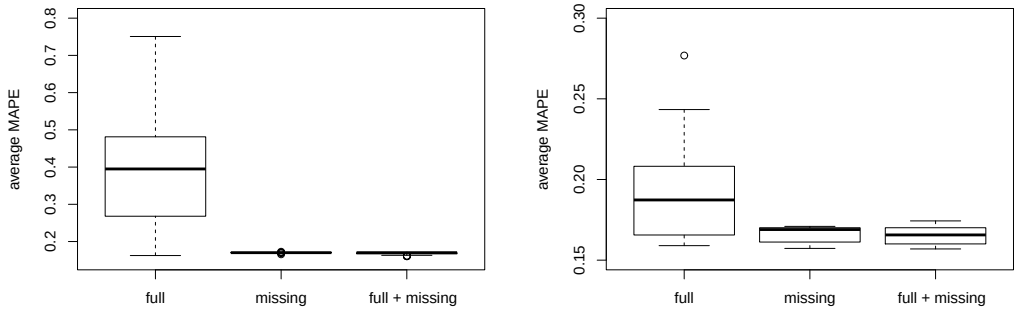
Table 6.9: Performance of evolved ADDARs trained with *full + missing* setting

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 4, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 3, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 7, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 6, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 5, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 8, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 4, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 7, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 7, 30⟩	⟨30, 2, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 7, 30⟩	⟨30, 2, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 7, 30⟩	⟨30, 2, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 7, 30⟩	⟨30, 2, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 6, 30⟩	⟨30, 0, 30⟩

Table 6.10: Performance of evolved ODDARs trained with *full + missing* setting

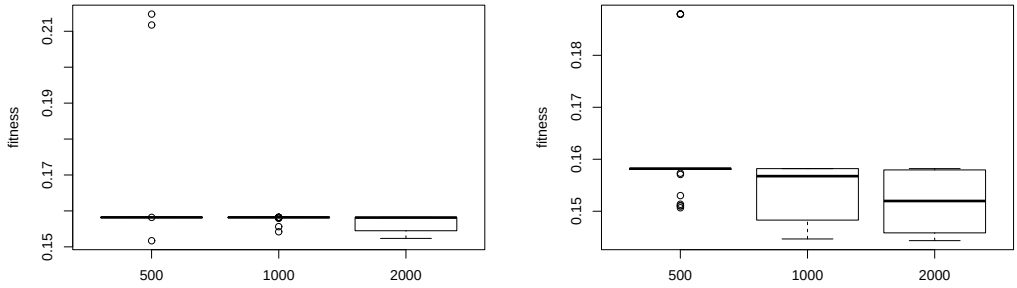
Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 5, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 1⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 28, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨28, 23, 29⟩	⟨29, 13, 30⟩	⟨29, 2, 30⟩	⟨30, 0, 30⟩	⟨30, 0, 29⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 13, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 8, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 21, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 28, 30⟩	⟨30, 4, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨28, 24, 30⟩	⟨28, 25, 30⟩	⟨30, 12, 30⟩	⟨29, 0, 29⟩	⟨30, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 20, 30⟩	⟨30, 13, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 20, 30⟩	⟨30, 13, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 19, 30⟩	⟨30, 8, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 18, 30⟩	⟨30, 4, 30⟩
	20	⟨30, 29, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 26, 30⟩	⟨29, 26, 30⟩	⟨28, 26, 30⟩	⟨30, 13, 30⟩	⟨29, 0, 30⟩

Figure 6.3 shows the average $\texttt{MAPE}_{p^*}^{\mathcal{S}_k}$ from all testing scenarios obtained by the evolved DDARs when different training sets are employed. It is obvious that evolved DDARs trained with full jobs are not as good as those trained with *missing* or *full + missing* jobs. It is also noted that the evolved DDARs trained with *full + missing* jobs are not significantly better than



(a) GP-ADDAR (b) GP-ODDAR

Figure 6.3: Influence of training set on the performance of evolved DDARs.



(a) GP-ADDAR (b) GP-ODDAR

Figure 6.4: Influence of population size on the performance of the proposed GP methods.

those trained with *missing* jobs. The results in this section suggest that using missing jobs may be sufficient for evolving DDARs with good reusability for these problems.

6.4.3 Population size of the proposed GP methods

This section examines the influence of population size on the performance of the proposed GP methods. Figure 6.4 shows that a population size of 1000 would be sufficient for the two proposed GP methods to explore the

search space of DDARs since population sizes more than 1000 do not show significant improvement on the performance of the GP methods. Also, increasing the population size from 500 to 1000 does not show significant improvements for GP-ADDAR but it can significantly improve the performance of GP-ODDAR. This observation suggests that the search space of ODDARs is more complicated than that of ADDARs and requires a larger population size to ensure the diversity in the population.

6.4.4 Performance of evolved DDARs on shops with a bottleneck

Previous experiments have shown that the evolved DDARs produce very good results on symmetrical job shops. In this section, we examine the reusability of the evolved DDARs in unbalanced shops. Table 6.11 and Table 6.12 show the performance of the previous DDARs (trained from symmetrical job shops) on shops with a single bottleneck machine. The utilisation values shown in these tables are the utilisation values of the bottleneck machine. In these testing scenarios, the utilisations of non-bottleneck machines are 5% less than that of the bottleneck machine. In general, these results are similar to those in Table 6.4 and Table 6.5. The evolved DDARs still provide superior performances as compared to `DTWK`, `DPPW`, and `ADRES`. Moreover, the performance of evolved ODDARs on some scenarios with high utilisation level and *full* setting is better than that in Table 6.5 (more superior evolved ODDARs). Perhaps, training DDARs in different simulation scenarios has made the evolved DDARs more robust than the existing DDARs such as `DTWK` and `DPPW`, which depend on the variation of job flowtimes. This also suggests that ODDARs are more effective in these cases because they take into account detailed information of the machine that processes each operation of a job instead of using the aggregate information in ADDARs.

Table 6.11: Performance of evolved ADDARs on shops with a bottleneck

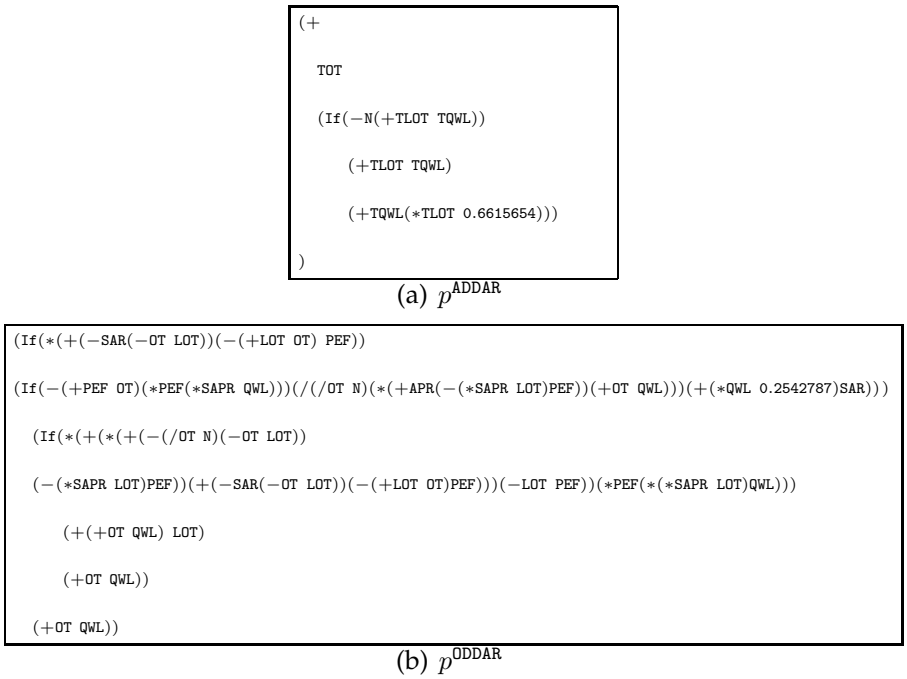
Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 0⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 1, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨29, 1, 30⟩	⟨30, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨29, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨29, 0, 30⟩

Table 6.12: Performance of evolved ODDARs on shops with a bottleneck

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 13, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 0⟩	⟨30, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 11, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 25, 30⟩	⟨30, 23, 30⟩	⟨30, 4, 30⟩	⟨30, 1, 30⟩	⟨30, 0, 30⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 14, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 12, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 12, 30⟩	⟨30, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 25, 30⟩	⟨28, 25, 30⟩	⟨30, 21, 30⟩	⟨30, 3, 30⟩	⟨30, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 12, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 13, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 17, 30⟩	⟨30, 12, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨29, 29, 30⟩	⟨29, 29, 30⟩	⟨30, 16, 30⟩	⟨30, 10, 30⟩
	20	⟨30, 28, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨28, 26, 30⟩	⟨28, 26, 30⟩	⟨27, 24, 30⟩	⟨27, 24, 30⟩	⟨27, 9, 30⟩

6.4.5 Evolved DDARs

In this section, we further examine the evolved DDARs to explore useful patterns for the development of more effective DDARs. The best evolved ADDAR $p^{\texttt{ADDAR}}$ and ODDAR $p^{\texttt{ODDAR}}$ in a set of evolved DDARs obtained from 30 independent GP runs are shown in Figure 6.5. Both evolved DDARs include the total processing times of jobs in queues and the pro-



(a) $p^{\texttt{ADDAR}}$

(b) $p^{\texttt{ODDAR}}$

Figure 6.5: Best evolved DDARs.

cessing time of the new job, i.e., QWL + OT for ODDAR and TQWL + TOT for ADDAR (when expanding the If expression in Figure 6.5(a)). This term is actually a good estimate of flowtime for a job with a small number of operations (for ADDAR) or for the first operation of a new job (for ODDAR). The main difference between these two evolved DDARs is the use of conditional terms to decide which extra terms should be included in the estimation. For ODDAR, PEF, QWL and LOT are used in the conditional term of function If. This DDAR shows that PEF is an important term to provide better flowtime estimations.

Table 6.13 shows $\texttt{MAPE}^{\mathcal{S}_k}_{p^{\texttt{ODDAR}}}$ and the t-test results between the $p^{\texttt{ODDAR}}$ in Figure 6.5(b) and other DDARs over 150 testing scenarios. In this table, the a, b, c, and d are the indices to represent $p^{\texttt{ADDAR}}$, DTWK, DPPW, and ADRES, respectively. A superscript of a result in this table shows the DDARs that are **not** significantly different (by using t-tests) from the $p^{\texttt{ODDAR}}$. Meanwhile, a subscript shows the DDARs that are significantly better than $p^{\texttt{ODDAR}}$, re-

Table 6.13: MAPE values obtained by the best evolved DDARs

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	0.1485	0.1534^{cd}	0.1452	0.1235	0.1025^{a}	0.2209	0.2197	0.2054	0.1673^{a}	0.1301^{a}
	5	0.1636	0.1661	0.1555	0.1294^{acd}	0.1049^{a}	0.2251	0.2202	0.2034	0.1650^{a}	0.1299^{ac}
	6	0.1731	0.1760	0.1668	0.1403	0.1165^{a}	0.2252	0.2202	0.2025^{a}	0.1636^{a}	0.1297^{ac}
	10	0.1921	0.1913	0.1804	0.1498^{ad}	0.1265^{a}	0.2070^{a}	0.2015^{a}	0.1864_{a}	0.1512^{ac}	0.1227^{ac}
	20	0.1935	0.1943	0.1834	0.1543^{a}	0.1369^{a}	0.1720_{a}	0.1715_{a}	0.1618_{ac}	0.1339_{ac}	0.1162_{ac}
Erlang-2	4	0.1376	0.1442	0.1408	0.1204	0.0960^{a}	0.2026	0.2076	0.1997	0.1698	0.1348^{a}
	5	0.1524	0.1567	0.1533	0.1319	0.1068^{a}	0.2038	0.2062	0.1975	0.1661	0.1353^{a}
	6	0.1601	0.1658	0.1608	0.1386	0.1147^{a}	0.2013	0.2030	0.1937	0.1656	0.1364^{ac}
	10	0.1767	0.1800	0.1734	0.1467	0.1197^{a}	0.1818	0.1836	0.1764	0.1508^{a}	0.1237^{ac}
	20	0.1739	0.1787	0.1740	0.1518	0.1337^{a}	0.1468	0.1526^{a}	0.1507_{a}	0.1318_{ac}	0.1140_{ac}
Uniform	4	0.1228	0.1322	0.1341	0.1208	0.1006^{a}	0.1735	0.1833	0.1857	0.1707	0.1462
	5	0.1350	0.1433	0.1443	0.1297	0.1082^{a}	0.1729	0.1820	0.1828	0.1690	0.1461
	6	0.1423	0.1507	0.1510	0.1352	0.1124^{a}	0.1684	0.1774	0.1788	0.1647	0.1421
	10	0.1510	0.1597	0.1618	0.1467	0.1247^{a}	0.1493	0.1572	0.1600	0.1493	0.1282
	20	0.1458	0.1559	0.1586	0.1460	0.1281^{a}	0.1184	0.1276	0.1332	0.1266	0.1122^{c}

spectively. If an index is neither shown in the superscript nor subscript, it means that $p^{\texttt{ODDAR}}$ is significantly better than all other DDARs. The tests are considered significant when the obtained p-value is less than 0.05. For example, in the scenario with 6 machines, full setting, utilisation of 80% and processing times follow Exponential, 0.2025^{a} shows that there is no significant difference between $p^{\texttt{ODDAR}}$ and $p^{\texttt{ADDAR}}$, and $p^{\texttt{ODDAR}}$ is significantly better than DTWK, DPPW, and ADRES. It is easy to see that the DDARs evolved by GP-ODDAR dominate other DDARs in most scenarios. In a few specific cases, $p^{\texttt{ADDAR}}$, DPPW, and ADRES are competitive with the evolved DDARs. $p^{\texttt{ADDAR}}$ and DPPW can also beat the evolved DDARs in some scenarios with high utilisation level (95%), *full* setting and large numbers of machines. These results are quite consistent with what has been observed in Section 6.3.1. It is also noted that $\texttt{MAPE}_{p^*}^{\mathcal{S}_k}$ is better when the utilisation increases, which is similar to that observed in [168]. When the number of machines increases, it is also interesting to see that the performance of the evolved

DDARs deteriorates if the *missing* setting is used, but the performance of these DDARs improves if the *full* setting is used.

Tables 6.14 and 6.15 show the detailed results obtained by the evolved and the existing DDARs for two particular scenarios. The mean and the corresponding standard deviation of each performance measure are shown in this table to provide a general evaluation of each DDAR. The MAPE, MAE and STDL of the evolved DDARs are better (smaller) than those obtained by the existing DDAR. This indicates that the evolved DDARs provide better delivery accuracy and delivery reliability than existing DDARs. It is also interesting that the MPE of p^{ODDAR} is positive while those of other DDARs are negative. This means that the other DDARs tend to overestimate the job flowtimes while p^{ODDAR} tends to underestimate the job flowtimes but the estimations made by p^{ODDAR} are better because its MAPEs in absolute value are

Table 6.14: Performance of DDARs (90% utilisation, *missing* jobs, 6 machines, processing times follow Erlang-2 distribution)

DDAR	MAPE	MAE	MPE	STDL	%T
p^{ADDAR}	0.145 ± 0.007	3.672 ± 0.348	-0.024 ± 0.004	5.622 ± 0.563	42.1599 ± 0.8230
p^{ODDAR}	0.139 ± 0.006	3.667 ± 0.356	0.021 ± 0.003	5.609 ± 0.569	50.4126 ± 0.6114
DTWK	0.579 ± 0.057	8.735 ± 1.522	-0.214 ± 0.063	12.106 ± 2.184	53.7524 ± 1.2697
DPPW	0.618 ± 0.081	6.231 ± 1.093	-0.388 ± 0.081	8.245 ± 1.474	48.7390 ± 1.2112
ADRES	0.427 ± 0.033	6.071 ± 0.485	-0.316 ± 0.036	7.671 ± 0.771	31.5857 ± 1.7690

Table 6.15: Performance of DDARs (90% utilisation, *full* jobs, 6 machines, processing times follow Erlang-2 distribution)

DDAR	MAPE	MAE	MPE	STDL	%T
p^{ADDAR}	0.170 ± 0.008	6.164 ± 0.472	-0.028 ± 0.004	7.990 ± 0.630	47.5206 ± 0.6539
p^{ODDAR}	0.166 ± 0.007	6.253 ± 0.490	0.027 ± 0.002	7.982 ± 0.645	57.8573 ± 0.7259
DTWK	0.269 ± 0.004	10.428 ± 1.223	-0.000 ± 0.015	14.071 ± 1.783	55.1111 ± 1.7751
DPPW	0.178 ± 0.008	6.508 ± 0.547	-0.011 ± 0.012	8.391 ± 0.733	50.3110 ± 2.4945
ADRES	0.281 ± 0.021	9.001 ± 0.503	-0.211 ± 0.023	9.789 ± 0.766	26.6826 ± 1.4803

smaller compared to those of existing DDARs. Since the existing DDARs overestimate flowtimes, some of the $\%\mathtt{T}$ values of those DDARs are smaller than those of the evolved DDARs. However, with the current emphasis on the just-in-time (JIT) [42] production concept where both earliness and tardiness are undesirable and meeting the target job due date would be of significance for the practice of JIT philosophy, smaller MAPE and STDL would be more attractive than smaller $\%\mathtt{T}$.

Figure 6.6 presents a simulation example to show the actual flowtimes and flowtimes estimated by different DDARs. In this figure, it is easy to see that the flowtimes estimated by evolved ADDAR $p^{\mathtt{ADDAR}}$ and ODDAR $p^{\mathtt{ODDAR}}$ are better than those estimated by the existing DDARs which tend to overestimate job flowtimes because the flowtimes estimated by these evolved DDARs are much closer to the actual flowtimes. The estimated flowtimes from the evolved ADDAR and ODDAR are quite similar and the ODDAR is able to make slightly better predictions compared to ADDAR for some jobs.

6.5 Further investigation with due date based dispatching rules

It has been shown that the evolved ODDARs have very good reusability when tested under different shop conditions. However, it is still questionable if these evolved ODDARs can be reused when other dispatching rules, rather than FIFO, are applied. To examine this issue, we test the performance of the evolved DDARs with the popular Critical Ratio plus Shortest Processing Time (CR+SPT) rule, which usually provides good due date related performance [41]. Different from the case when FIFO is used as the dispatching rule, it is noted that estimating job flowtimes here is more complicated. The reason is that FIFO is not influenced by the estimated due dates from the DDAR while the sequencing decisions from

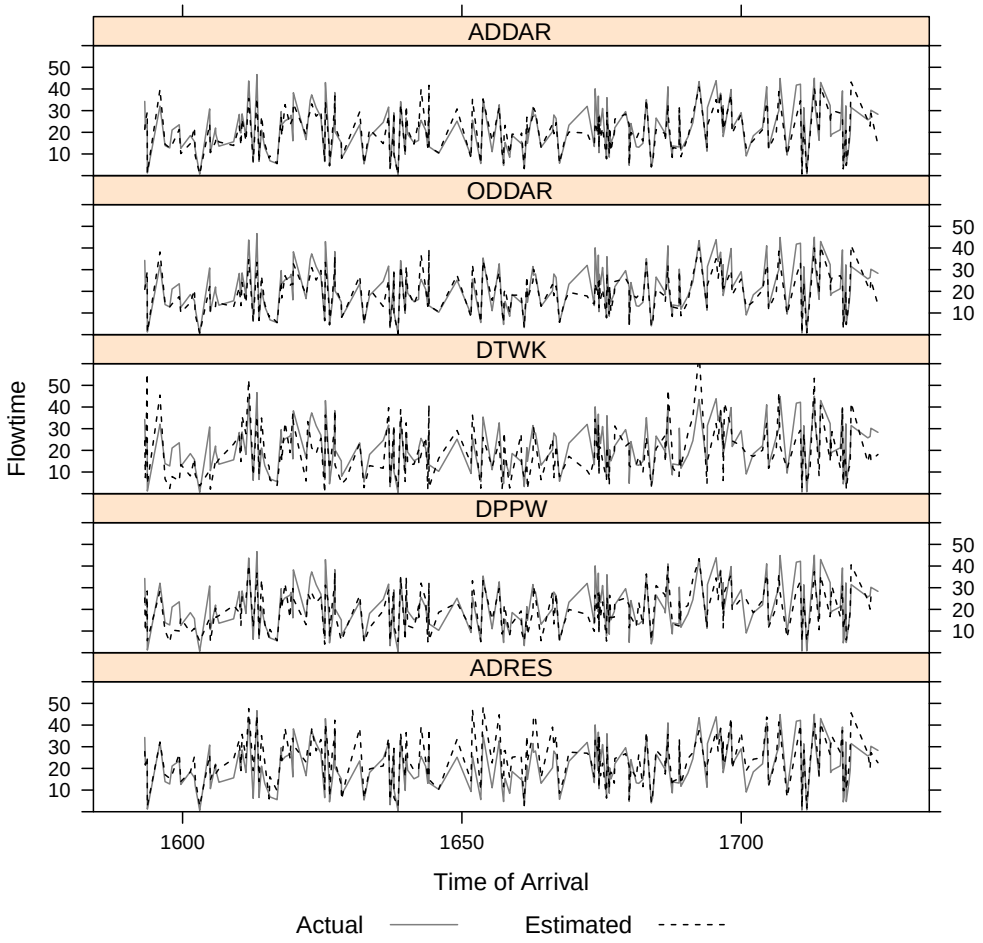


Figure 6.6: Simulation illustration of DDARs (utilisation = 90%, missing jobs, 6 machines, processing times follow Erlang-2 distribution).

CR+SPT are strongly affected by the estimated due dates. For example, the priority (with CR+SPT) of waiting jobs can be calculated as:

$$Z_{ji} = p_{ji} \times \max\left\{\frac{d_j - t}{\sum_{r=i}^{m_j} p_{jr}}, 1\right\} \tag{6.2}$$

where p_{ji} is the processing time of the operation (j, i), which is the i^{th} operation of job j. After all jobs in the queue have been assigned priorities

determined by equation (6.2), the job with the smallest priority value will be processed next. While the value of p_{ji} is available when the job arrives, d_j and t depend on the DDAR and the moment the sequencing decision is made. Therefore, we do not know the exact priorities of the new job at the machines it is planned to visit. Unfortunately, in order to make an accurate due date estimate of a new job, it is important to consider the priorities of that job determined by the dispatching rule (when waiting in the queues) because they will influence the job's waiting times [168]. To tackle this issue with our proposed GP-ODDAR method, we will include into the terminal set a new term called *estimated priority ratio* (EPR), which can be determined by equation (6.3).

$$EPR_{ji} = \frac{\sum_{(l,k)\in Q'_m} p_{lk}}{\sum_{(l,k)\in Q_m} p_{lk}} \tag{6.3}$$

where Q_m is the set of operations (l,k) in the queue at machine m at the time the new job arrives and $Q'_m = \{(l,k) \in Q_m : \hat{Z}_{ji} < \hat{Z}'_{lk}\}$. $\hat{Z}_{ji}$ and $\hat{Z}'_{lk}$ are the estimated priorities of the operation (j,i) of the new job j and other operations (l,k) which are calculated as:

$$\begin{aligned}
\hat{Z}_{ji} &= p_{ji} \times \max\left\{\frac{(r_j + f'_j) - (r_j + PEF)}{\sum_{r=i}^{m_j} p_{jr}}, 1\right\} \\
&= p_{ji} \times \max\left\{\frac{f'_j - PEF}{\sum_{r=i}^{m_j} p_{jr}}, 1\right\} \qquad (6.4) \\
\hat{Z}'_{lk} &= p_{lk} \times \max\left\{\frac{d_l - (r_l + PEF)}{\sum_{r=k}^{m_l} p_{lr}}, 1\right\} \qquad (6.5)
\end{aligned}$$

where $r_j + f'_j$ is a rough estimate of the due date and $r_j + PEF$ is the estimate of the decision moment t. Since DPPW is shown to be a competitive DDAR in our previous experiment, it will be used to calculate $r_j + f'_j$. The role of EPR_{ji} is to represent the dominance relation regarding the priorities between the new job and jobs currently in the shop.

Table 6.16 shows the performance of the evolved DDARs with CR+SPT as the dispatching rule. We use the population size of 2000 in this exper-

Table 6.16: Performance of evolved ODDARs with CR+SPT as the dispatching rule

Setting		missing					full				
		%60	%70	%80	%90	%95	%60	%70	%80	%90	%95
Exponential	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 0, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨28, 1, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨29, 28, 30⟩	⟨27, 0, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨29, 25, 30⟩	⟨18, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 28, 30⟩	⟨28, 24, 30⟩	⟨15, 13, 30⟩	⟨2, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨27, 30, 30⟩	⟨26, 29, 30⟩	⟨28, 18, 30⟩	⟨23, 12, 30⟩	⟨16, 11, 30⟩	⟨7, 2, 30⟩	⟨0, 0, 30⟩
Erlang-2	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨25, 14, 30⟩	⟨2, 0, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 30, 30⟩	⟨29, 28, 30⟩	⟨24, 11, 30⟩	⟨1, 0, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 29, 30⟩	⟨29, 27, 30⟩	⟨21, 3, 30⟩	⟨0, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 30, 30⟩	⟨30, 29, 30⟩	⟨30, 26, 30⟩	⟨26, 20, 30⟩	⟨1, 0, 30⟩	⟨0, 0, 30⟩
	20	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨26, 28, 30⟩	⟨25, 26, 30⟩	⟨21, 25, 30⟩	⟨28, 22, 30⟩	⟨23, 14, 30⟩	⟨14, 10, 30⟩	⟨1, 0, 30⟩	⟨0, 0, 30⟩
Uniform	4	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 30, 30⟩	⟨29, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 28, 30⟩	⟨25, 19, 30⟩	⟨2, 1, 30⟩	⟨0, 0, 30⟩
	5	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨29, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 27, 30⟩	⟨25, 20, 30⟩	⟨4, 1, 30⟩	⟨1, 0, 30⟩
	6	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨29, 29, 30⟩	⟨29, 29, 30⟩	⟨30, 29, 30⟩	⟨30, 26, 30⟩	⟨23, 18, 30⟩	⟨5, 1, 30⟩	⟨0, 0, 30⟩
	10	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨27, 27, 30⟩	⟨23, 25, 30⟩	⟨20, 20, 30⟩	⟨30, 26, 30⟩	⟨28, 20, 30⟩	⟨21, 16, 30⟩	⟨4, 1, 30⟩	⟨0, 0, 30⟩
	20	⟨29, 27, 30⟩	⟨26, 24, 30⟩	⟨20, 20, 30⟩	⟨13, 13, 30⟩	⟨9, 9, 30⟩	⟨27, 18, 30⟩	⟨20, 13, 30⟩	⟨12, 0, 30⟩	⟨1, 1, 30⟩	⟨1, 1, 26⟩

iment in order to ensure enough diversity to create more sophisticated DDARs to deal with this situation. It can be seen that the the evolved ODDARs still perform better than the existing DDARs in most scenarios, especially the cases with *"missing"* setting. For the scenarios with *"full"* setting, there are a fewer number of superior evolved ODDARs that can beat DTWK and DPPW when the utilisation increases. These results suggest that DTWK and DPPW again significantly enhance their performance with the less diverse jobs and high utilisation levels. Meanwhile, the evolved ODDARs still show their superiority in more complicated scenarios when tested with CR+SPT.

6.6 Chapter Summary

In this chapter, two proposed GP methods have been developed for evolving due date assignment rules. This is the first time that GP has been applied to evolve reusable DDARs. The experimental results show that the evolved DDARs can outperform the existing dynamic DDARs with MAPE as the performance measure. Comparisons using other performance mea-

sures also confirm the effectiveness of the evolved DDARs. From the performance of the evolved DDARs on a number of simulation scenarios, it can also be concluded that the evolved DDARs have good reusability and they are able to make good job flowtime estimates for unseen scenarios with different processing time distributions, utilisation, job settings and numbers of machines. When comparing the two proposed GP methods, it has been shown that GP-ODDAR is significantly better than GP-ADDAR in most testing scenarios. Typical examples of the evolved DDARs show that these GP evolved DDARs are (at least partially) understandable.

This study has shown the effectiveness of GP for evolving DDARs in the case where FIFO, or a due date based dispatching rule CR+SPT, is used as the dispatching rule. However, there are always different conflicting objectives to be considered in JSS, which required better customised dispatching rules. Therefore, it is crucial that these two scheduling rules and conflicting objectives of interest are considered simultaneously to design effective comprehensive scheduling system. In the next chapter, we will further investigate these issues by developing a new cooperative coevolution based on GP.

Chapter 7

Automatic Design of Scheduling Policies

Practical JSS applications usually involve different scheduling/planning decisions. However, most studies on JSS only consider one of the many decisions and fix the others in order to reduce the complexity of the scheduling problems. These approaches are valid when there is no interaction among the scheduling decisions, which is often not the case for real world applications. Although JSS has been popular for decades and investigation of the interactions among various decisions is essential for the development of effective and comprehensive scheduling systems, studies on the interactions among different scheduling decisions are rather limited. The studies [160, 12, 124, 39] mainly examined the performance of simple combinations of different existing dispatching rules (DRs) and due-date assignment rules (DDARs). One of the reasons for the lack of research in this direction is that dealing with each scheduling decision is already difficult; and thus considering multiple scheduling decisions simultaneously will be even more complicated. To tackle this problem, there is a need to develop new methodologies for improving the scheduling decisions and their interactions, which should also be able to cope with the dynamic features of JSS problems.

In this chapter, GP is used as a hyper-heuristic method [30] for the automatic design of scheduling policies (SPs) which include sequencing rules and due date assignment rules for dynamic JSS problems. GP is a suitable method for designing SPs because of its flexibility to encode different scheduling rules in the representation. Moreover, as an evolutionary approach, GP can be applied to handle the multiple conflicting objectives of JSS problems (as shown in Chapter 5). Another advantage of GP is that evolved scheduling policies are potentially interpretable, which is important and useful for understanding how the problem is solved by the evolved policies and how the trade-offs among the different objectives of JSS can be obtained.

This chapter presents novel methodologies to design efficient SPs for solving dynamic multi-objective JSS problems via genetic programming based hyper-heuristic (GPHH) [31]. In order to address drawbacks of existing methods, three important aspects are considered in our proposed algorithms: (i) representations of different scheduling rules; (ii) evolutionary optimisation to evolve the trade-offs in SPs; and (iii) reusability (ability to be reused on new unseen problems [31, 30]) of the evolved SPs. The first aspect concentrates on how the scheduling rules can be represented and evaluated in GP. The second applies multi-objective evolutionary algorithms [8, 196, 52] to explore the Pareto front of the evolved SPs. In order to examine how training scenarios may influence reusability of the evolved rules on unseen situations, four multi-objective genetic programming based hyper-heuristic (MO-GPHH) methods are proposed to deal with the JSS problems. Different from other proposed GPHH methods which only evolve scheduling rules to handle a specific scheduling decision, the MO-GPHH methods developed in this chapter will simultaneously evolve two scheduling rules to handle sequencing and due date assignment decisions. Because the two scheduling rules are considered together, they can interact and support each other in order to improve the overall scheduling performance.

The objectives of this chapter are:

1. Developing MO-GPHH methods for automatic design of scheduling policies for dynamic job shop scheduling problems.

2. Comparing the evolved scheduling policies with existing scheduling policies from combinations of existing dispatching rules and due-date assignment rules.

3. Evaluating the reusability of the evolved scheduling policies.

4. Analysing the performance of the proposed MO-GPHH methods and the evolved scheduling policies.

Section 7.1 describes the proposed MO-GPHH methods and settings for our experiments. The experimental results and comparison of the evolved scheduling policies and the existing scheduling policies are presented in Section 7.2. The analysis of the proposed methods and the evolved scheduling policies are shown in Section 7.3. Conclusions are drawn in Section 7.4.

7.1 Proposed MO-GPHH methods

This section describes the new GPHH methods for evolving scheduling policies, which include rules for due date assignment and sequencing decisions in dynamic job shop environments. We will first show how DDARs and DRs are represented by individuals in GP. Then, new MO-GPHH methods based on NSGA-II, SPEA2, HaD-MOEA and a proposed cooperative coevolution method are applied to deal with the dynamic JSS problems. These MO-GPHH methods are different in the way that the Pareto fronts of non-dominated scheduling policies are explored. Lastly, the job shop simulation model used for the training and testing will be described.

7.1.1 Representations

As investigated in previous chapters, there are several ways to represent scheduling rules (either DDARs or DRs). For DDARs in this chapter, we employ the representation and the evaluation scheme of GP-ODDAR (see Section 6.1.1) since it was shown to provide better estimation of job flow-times. The same terminal and function sets of GP-ODDAR are used to construct DDARs in our evolved SPs. Regarding DRs, we use the simple arithmetic representation to construct composite dispatching rules (CDRs) as presented in Chapters 3, 4, and 5. This representation is selected because the experiments in previous chapters showed that rules evolved with this representation have good reusability and can be easily applied to dynamic environments. The terminal set used to evolve CDRs in this chapter is shown in Table 7.1. Because we do not consider non-delay factor α in our experiments for dynamic JSS, the six machine attributes (as described in Chapter 3) in the lower part of Table 7.1 can be employed directly from the terminal set instead of utilising the grammar structure as shown in Section 3.1. Figures 7.1 and 7.2 show the moment at which each rule is activated and illustrate how decisions are made with these evolved rules.

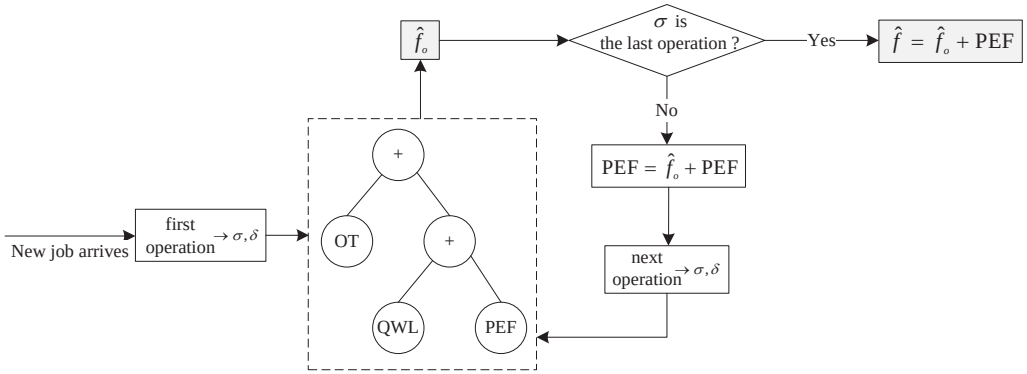


Figure 7.1: Operation-based DDAR.

Table 7.1: Terminal set for CDR

rJ	job release time (arrival time)
RJ	operation ready time
RO	number of remaining operation of the job.
RT	work remaining of the job
PR	operation processing time
W	weight of the job
DD	due date of the job
RM	machine ready time
SJ	slack of the job $=$ DD $-$ (t $+$ RT)
#	Random number from 0 to 1
WR	workload ratio $= \frac{\sum_{\sigma\in\Omega} p(\sigma)}{\sum_{\sigma\in J} p(\sigma)}$
MP	machine progress $= \frac{\sum_{\sigma\in K} p(\sigma)}{\sum_{\sigma\in\Lambda} p(\sigma)}$
DJ	deviation of jobs in queue $= \frac{\min_{\sigma\in\Omega}\{p(\sigma)\}}{\max_{\sigma\in\Omega}\{p(\sigma)\}}$
CWR	critical workload ratio $= \frac{\sum_{\sigma\in\Omega^c} p(\sigma)}{\sum_{\sigma\in\Omega} p(\sigma)}$
CWI	critical machine idleness, WR of the critical machine
BWR	bottleneck workload ratio $= \frac{\sum_{\sigma\in\Omega^b} p(\sigma)}{\sum_{\sigma\in\Omega} p(\sigma)}$

*t is the time when the sequencing decision is made.

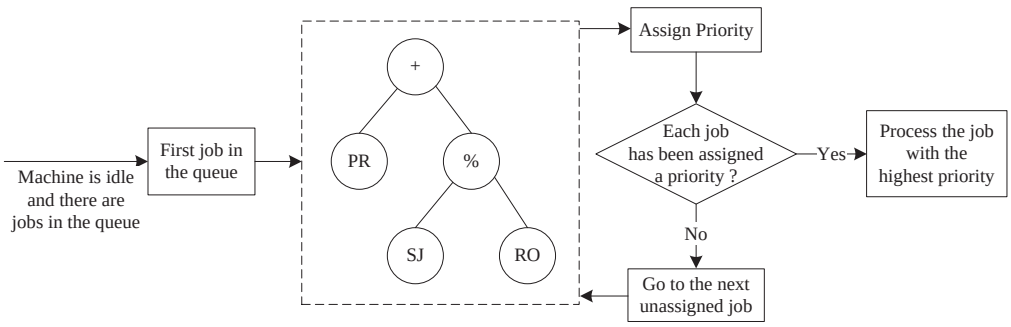


Figure 7.2: Example of GP tree as a CDR.

7.1.2 A Cooperative Coevolution MO-GPHH for DJSS

As discussed earlier, this work aims to evolve scheduling policies that include two key components, i.e., due date assignment rules and dispatching rules. While the representation of the rules has been discussed previously, we need to specify how these rules are evolved in our proposed GPHH methods. In this work, two approaches are examined. In the first approach, a GP individual contains two GP trees for the two rules as presented above. In this case, each individual is equivalent to a scheduling policy. The scheduling policy is evaluated by applying the first tree as a DDAR when a new job arrives at the job shop to assign a due date to that job. Meanwhile, the second tree is applied when a machine becomes idle and there are jobs in the queue of that machine to find the next job to be processed. For this approach, we apply NSGA-II [52], SPEA2 [195], and HaD-MOEA [187] to explore the Pareto front of non-dominated scheduling policies similar to the common applications of these algorithms.

The second approach to evolving scheduling policies is to employ cooperative coevolution [72, 73, 180] to evolve two decision rules in two sub-populations. This approach is similar to the cooperative coevolution framework proposed by Potter and de Jong [156], in which the scheduling policy is the combination of an individual in a sub-population with a *representative* from the other sub-population, and some specialised operations are also employed here to help explore the Pareto front of the scheduling policies. In this work, we propose a new diversified multi-objective cooperative coevolution (DMOCC) method based on the second approach. An overview of the proposed DMOCC is shown in Figure 7.3. Here each sub-population (P_1 for DDARs and P_2 for DRs) represents one rule of the complete scheduling policy. For each individual $p_i^r \in P_r$, the objective values which determine the quality (fitness) of p_i^r are obtained by combining that individual with a representative from the other population to form a complete scheduling policy S. When a complete scheduling policy is applied to the job shop, the quality of that policy is characterised by the

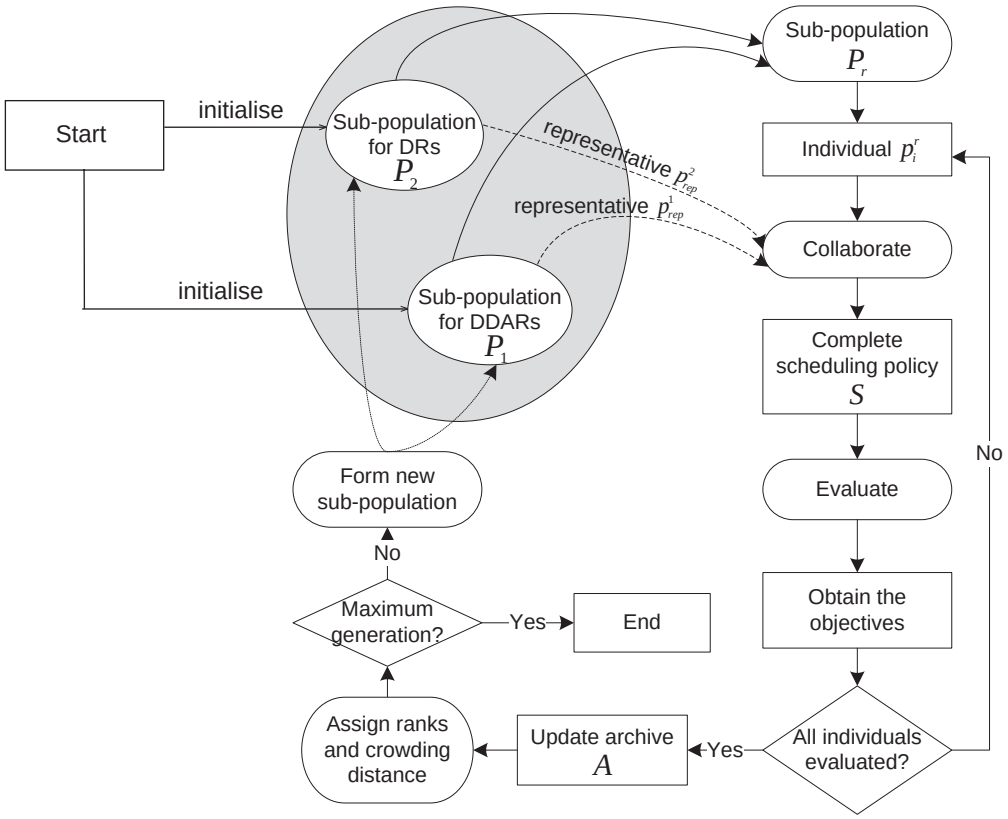


Figure 7.3: Overview of DMOCC.

expected values of three performance measures: (1) makespan (C_{max}) [153]; (2) total weighted tardiness (TWT) [153]; and (3) mean absolute percentage error (MPEA) [19] (see Table 7.4). C_{max} and TWT are two popular performance measures for evaluating dispatching rules or scheduling methods while MPEA is used to indicate the accuracy of the due date assignment rules. In this work, the scheduling policies are evolved such that these three performance measures are minimised.

Within DMOCC, we use the *crowding distance* (individuals with higher crowding distances are located in less crowded areas of the objective space) and *non-dominated rank* [52] (individuals with the same *rank* are not dominated by each other and dominated by at least one individual with a

smaller *rank*) to select GP individuals for genetic operations and for collaboration between the two sub-populations. *Representatives* for collaboration are selected based on a *binary tournament selection* method [52], which randomly selects two individuals and the one with a lower non-dominated rank will be chosen. In the case that two individuals have the same rank, the individual with a higher crowding distance will be selected. The binary tournament selection is employed in DMOCC because it takes into account both the quality of the non-dominated individuals and their spread/distribution.

An external archive A is employed in this method to store the non-dominated scheduling policies. After all individuals have been evaluated, a set of non-dominated scheduling policies are extracted from individuals in the two sub-populations and the current archive to form a new archive. Besides storing the non-dominated scheduling policies, the archive in DMOCC is also used for two other purposes. First, it is used to evaluate the quality (rank and crowding distance) of the evolved scheduling policies in the two sub-populations. Instead of evaluating the quality of the evolved rules independently in each sub-population, it is better to assess their quality based on comparisons with those in the archive and the other sub-population to identify other potential non-dominated scheduling policies. Secondly, the archive can provide genetic materials which are needed for the crossover operation (more details are provided in Section 7.1.3).

Different from NSGA-II, SPEA2 and HaD-MOEA, the size of archive in DMOCC is not fixed, although the number of complete scheduling policies stored in the archive cannot exceed a predefined maximum size. When the number of non-dominated scheduling policies extracted from a generation is more than the maximum size, only individuals with the highest crowding distance will be preserved in the archive. Since new individuals will be created from parents in the archive through crossover, such a dynamic archive will help focus the search towards non-dominated schedul-

ing policies at the early stage of the evolution. When the number of individuals in the archive increases, the shape of the Pareto front will be characterised and the method will focus on distributing the individuals uniformly.

The pseudo code of DMOCC is shown in Figure 7.4. The algorithm starts by populating the two sub-populations P_1 and P_2 with randomly generated individuals. In each generation, each individual p_i^r of the two populations collaborates with the representative $p_{rep}^{r'}$ from the other sub-population to create a complete scheduling policy S. Then, the objective values of p_i^r are obtained by applying S to the simulated job shop. When all individuals have been evaluated, the archive A will be updated. Ranks and crowding distances are then assigned to individuals in A, P_1, and P_2. Here, new sub-populations are generated by genetic operations and the algorithm starts a new generation if the maximum generation is not reached.

7.1.3 Genetic Operators

Traditional genetic operators are employed by the proposed MO-GPHH methods. For crossover, GP uses the subtree crossover [103], which creates new individuals for the next generation by randomly recombining subtrees from two selected parents. SPEA2 uses tournament selection to select parents in the population with the highest fitness value in the tournament. NSGA-II and HaD-MOEA use binary tournament selection based on rank and crowding distance as explained in the previous section. For DMOCC, binary tournament selection is used to select one parent from a sub-population and one parent from the archive. Since individuals in the archive have a rank of zero, the selection is made only based on the crowding distance in order to direct the search to less crowded areas. Here, mutation is performed by the subtree mutation [103], which randomly selects a node of a chosen individual in the population and replaces the subtree rooted at that node by a newly randomly-generated subtree. For NSGA-II,

```
1: initialise each sub-population P_r with r = {1, 2} P_r ← {p_1^r, p_2^r, ..., p_N^r}
2: A ← {}
3: while maxGeneration is not reached do
4:     for r = 1 → 2 do
5:         for i = 1 → N do
6:             S ← collaborate(p_i^r, p_rep^r') where r' ≠ r
7:             p_i^r.objectives ← evaluate(S)
8:         end for
9:     end for
10:    A ← update(A, P_1, P_2)
11:    assign ranks and crowding distance
12:    for r = 1 → 2 do
13:        P_r ← genetic_operations(P_r, A)
14:    end for
15: end while
16: return A
```

Figure 7.4: Pseudo code for DMOCC.

SPEA2 and HaD-MOEA, the genetic operations will first randomly choose which tree (either DR or DDAR) of the parents to perform the operations on since each individual includes two trees for the two scheduling rules. If the crossover is applied, only genetic materials from the selected tree of the same type will be exchanged (e.g. a tree representing DR in one parent will only crossover with a tree representing DR of the other parent).

7.1.4 Parameters

Table 7.2 shows the parameters used by the four proposed MO-GPHH methods. SPEA2 applied tournament selection with a tournament size of 5 to select individuals for the genetic operations. NSGA-II, SPEA2, and HaD-MOEA used a population size of 200 while DMOCC used a popu-

Table 7.2: Parameter settings

Initialisation	ramped-half-and-half [103]
Crossover rate	90%
Mutation rate	10%
Maximum depth	8
Number of generations	100
Population size	200 for NSGA-II, SPEA2, and HaD-MOEA, and 100 for each sub-population of DMOCC

lation size of 100 for each sub-population to ensure that the number of program evaluations remains the same for all methods. The archive size of SPEA2 and *maximum-size* of DMOCC are set to 200. These settings are used so that the proposed methods will give the same number of non-dominated scheduling policies at the end of each run.

7.1.5 Job Shop Simulation Model

The same simulation model used in Chapter 6 is employed here for training and testing purposes. Table 7.3 presents all the training and testing simulation scenarios in our experiments. In each replication of a simulation scenario, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the 1000^{th} job is considered as the warm-up time and the statistics from the next completed 5000 jobs (set $\mathbb{C}$) are recorded to evaluate the performance measures of the scheduling policies as shown in Table 7.4. In this table, `M` is the number of machines in the shop, $\bar{w}$ is the average weight and $\frac{1}{\mu}$ is the average processing time of an operation. The average values of these performance measures across different simulation scenarios/replications are the objectives to be minimised by the proposed MO-GPHH methods.

Table 7.3: Training and testing scenarios

Factor	Training	Testing
Number of machines	4,6	5,10,20
Utilisation	80%,90%	70%,80%,90%,95%
Distribution of processing time	Exponential	Exponential, Uniform
Distribution of # of operations	missing	missing,full

In the training stage, due to the heavy computation time, we only perform one replication for each scenario. In Table 7.3, there are $(2\times2\times1\times1) = 4$ simulation scenarios used for evaluating the performance of the evolved scheduling policies. It should be noted that the performance measures are obtained for each scenario by applying the evolved scheduling policies thousands of times, since there are thousands of due date assignment and sequencing decisions needed to be made during a simulation replication of that scenario. During the testing stage, each of the non-dominated scheduling policies from a GP run is applied to $(3 \times 4 \times 2 \times 2) = 48$ simulation scenarios (see Table 7.3) and 5 simulation replications are performed for each scenario; therefore, we perform $48 \times 5 = 240$ simulation replications for testing the performance of the obtained non-dominated scheduling policies. Limited simulation scenarios are used in these experiments as compared to those in Chapter 6 because the evaluation times of all scheduling policies in the obtained Pareto fronts are very time-consuming. The testing scenarios are selected to help us assess the performance of evolved rules on seen and unseen cases.

7.1.6 Performance Measures for MO-GPHH Methods

Similar to other multi-objective optimisation applications, we are interested in the quality of the obtained Pareto fronts in terms of (1) *conver-*

Table 7.4: Performance measures of scheduling policies

Makespan [153]	$\mathtt{C_{max}} = \max_{j\in\mathbb{C}}\{f_j\}$
Normalised Total Weighted Tardiness [185]	$\mathtt{TWT} = \frac{\sum_{j\in\mathbb{C}} w_j T_j}{\lvert\mathbb{C}\rvert\times\mathtt{M}\times\frac{1}{\mu}\times\bar{w}}$
Mean Absolute Percentage Error [19]	$\mathtt{MAPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{\lvert e_j\rvert}{f_j}$

gence to the trade-off solutions and (2) the *spread* or *distribution* of the solutions on the obtained Pareto front. Three popular performance metrics for multi-objective optimisation are used here: hypervolume ratio (HVR) [196, 183]; SPREAD [52]; and generational distance (GD) [8].

Hypervolume (HV) and Hypervolume Ratio (HVR)

Hypervolume is used to measure the "volume" in the objective space covered by the obtained non-dominated solutions for minimisation problems,

$$HV = volume(\bigcup_{i=1}^{n_{PF}} \nu_i) \tag{7.1}$$

where n_{PF} is the number of members in the obtained Pareto front PF_{known}, ν_i is the hypercube constructed with a reference point and the member i as the diagonal of the hypercube [196]. van Veldhuizen and Lamont [183] normalised HV by using the hypervolume ratio which is the ratio of the hypervolume of PF_{known} and the hypervolume of the reference Pareto front PF_{ref},

$$HVR = \frac{HV(PF_{known})}{HV(PF_{ref})} \tag{7.2}$$

The Pareto fronts found with larger HVRs are preferred as they contain quality and widespread solutions.

SPREAD

This metric measures the non-uniformity of PF_{known} [52]. A widely and uniformly spread out set of non-dominated solutions in the PF_{known} will result in a small $SPREAD$.

$$SPREAD = \frac{d_f + d_l + \sum_{i=1}^{n_{PF}-1} |d'_i - \bar{d}|}{d_f + d_l + (n_{PF} - 1)\bar{d}} \tag{7.3}$$

where d'_i is the Euclidean distance between member i and its nearest member in PF_{known}, $\bar{d}$ is the average of all d'_i, and d_f and d_l are the Euclidean distances between the extreme solutions and the boundary solutions of PF_{known}.

Generational Distance (GD)

This metric is used to measure the distance between the obtained Pareto front (PF_{known}) and the reference Pareto front (PF_{ref}) [8],

$$GD = \left(\frac{1}{n_{PF}} \sum_{i=1}^{n_{PF}} d_i^2 \right)^{1/2} \tag{7.4}$$

where d_i is the Euclidean distance between the member i in PF_{known} and its nearest member in PF_{ref}.

PF_{ref} is normally the true Pareto front, which is unknown in the simulation here. Therefore, we need to adopt a reference Pareto front PF_{ref} in the calculation of these performance metrics. In this work, PF_{ref} includes the non-dominated scheduling policies extracted from all scheduling policies found by the four MO-GPHH methods (NSGA-II, SPEA2, HaD-MOEA, and DMOCC) in all independent runs. Basically, the evolved scheduling policies from 4 (methods) $\times$ 30 (runs) = 120 Pareto fronts are combined into a common pool, and the non-dominated sorting technique (as employed in NSGA-II) is used to find the non-dominated scheduling policies from this pool.

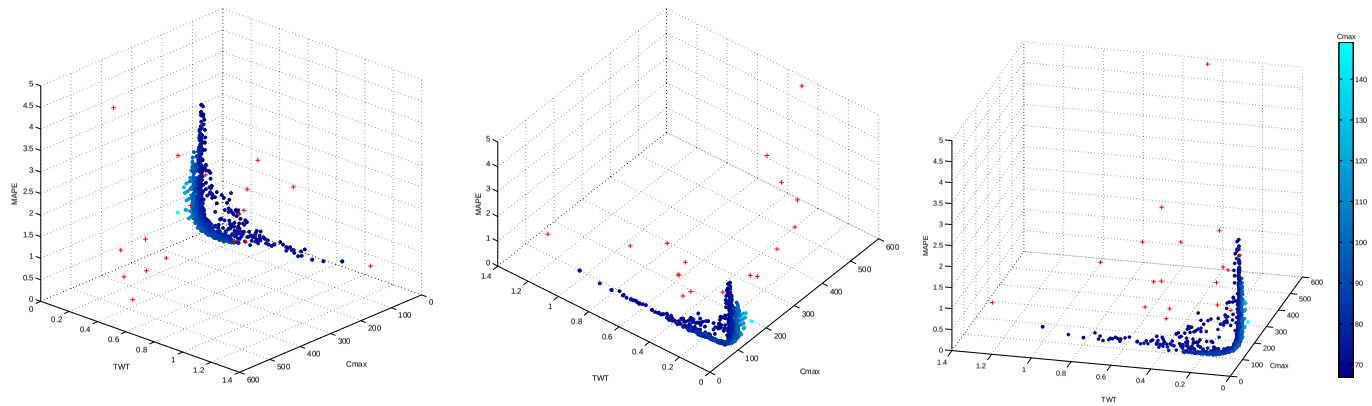
7.2 Results

In order to evaluate the effectiveness of the proposed methods, 30 independent runs of each MO-GPHH method are performed and the evolved Pareto fronts obtained from each run are recorded. The evolved non-dominated scheduling policies (SPs) are then compared with the existing SPs based on combinations of well-known dispatching rules with dynamic and regression-based due date assignment rules.

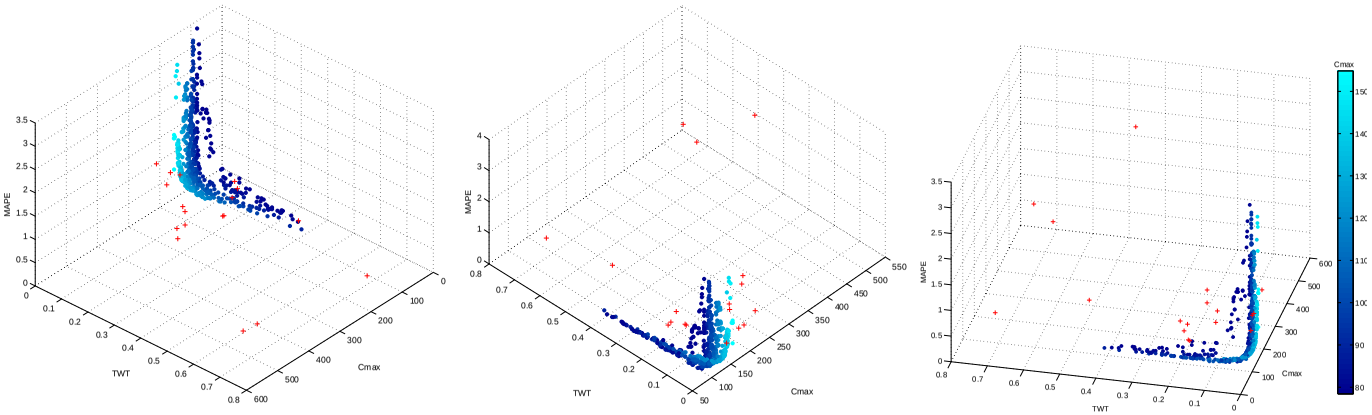
7.2.1 Pareto Front of the Evolved Scheduling Policies

Figure 7.5(a) and Figure 7.5(b) show the *aggregate Pareto fronts* extracted from all the evolved scheduling policies, which were obtained by the four proposed MO-GPHH methods in 30 independent runs for both the training and testing scenarios. It can be observed that the three objectives $\mathtt{C}_{\max}$, TWT and MAPE are conflicting objectives. When tracing along the Pareto front to find SPs that are able to minimise $\mathtt{C}_{\max}$ and TWT, it can be seen that the value of MAPE tends to be increased. This suggests that scheduling policies that provide better shop performance (small $\mathtt{C}_{\max}$ and TWT) will result in flowtimes that are hard to predict accurately (large MAPE). Given a similar value of MAPE, trade-off between $\mathtt{C}_{\max}$ and TWT can also be observed, which suggests that there is no evolved dispatching rule that can simultaneously optimise these objectives. Such an observation is consistent with those discussed in the literature [88, 181].

In both the testing and training scenarios, it is observed that $\mathtt{C}_{\max}$ and TWT can be significantly reduced by using SPs with MAPE smaller than 0.5. The use of more sophisticated SPs can provide slightly better $\mathtt{C}_{\max}$ and TWT but they also make the job flowtimes much more difficult to be estimated. These results show that there are many trade-offs to be considered when selecting an appropriate SP for a scheduling system and the knowledge about these trade-offs is useful in making a better decision. For example, the obtained Pareto fronts suggest that much better delivery reliability (a



(a) Training Scenarios



(b) Testing Scenarios

Figure 7.5: Pareto front of non-dominated scheduling policies. The plots show three different views of the aggreate Pareto fronts obtained for training and testing, • and + respectively represent evolved and existing scheduling policies as shown in Section 7.2.2.

small MAPE) can be achieved with a reasonable sacrifice in C_{max} or TWT. However, if a single objective such as C_{max} or TWT is to be minimised in this case, the evolved SPs will lead to very poor delivery reliability (a high MAPE) and thus reduce the customer satisfaction. Also, given the shape of the Pareto fronts in Figure 7.5, it would be difficult to apply a traditional linear combination of objective values for fitness assessment to find desirable rules due to the difficulty in identifying suitable weights for each objective. These observations show that handling multiple objectives with knowledge about their Pareto front is crucial for the design of effective SPs.

7.2.2 Comparison to Existing DRs and Dynamic DDARs

The combination of six popular DRs and three dynamic DDARs are evaluated on both the training and testing scenarios. The results (red +) are compared with the evolved non-dominated SPs as shown in Figure 7.5. The six DRs used in this comparison are First-In-First-Out (FIFO), Critical Ratio (CR), Slack-per-Operation (S/OPN), Shortest Processing Time (SPT) [145], weighted Apparent Tardiness Cost (ATC) and weighted Cost Over Time (COVERT) [185]. The parameters of ATC and COVERT are the same as those used in Vepsalainen and Morton [185] ($k = 3$ for ATC, $k = 2$ for COVER, and the leadtime estimation parameter $b = 2$). The three dynamic DDARs are DTWK, DPPW [41] and ADRES [19]. These DDARs are selected for comparison because they are well-known in the scheduling literature and the application of these rules does not require predetermination of any parameter or coefficient for each simulation scenario. The objective values obtained by these 18 combinations for the training and testing scenarios are shown in Table 7.5 and Table 7.6 and are visualised as crosses in Figure 7.5(a) and (b).

Among these existing scheduling policies, the ones given with FIFO provide the best C_{max}. The scheduling policies with DTWK provide the best

Table 7.5: Performance of existing scheduling policies for *training* scenarios (C_{max}, TWT, MAPE)

	DTWK	DPPW	ADRES
FIFO	(101.5, 1.25, 0.81)	(101.5, 0.71, 2.00)	(101.5, 0.36, 1.05)
CR	(174.6, 0.53, 0.73)	(127.4, 0.52, 2.47)	(178.0, 0.49, 1.33)
S/OPN	(156.9, 0.42, 0.57)	(114.0, 0.42, 1.63)	(123.9, 0.14, 1.68)
SPT	(575.8, 0.60, 0.67)	(575.8, 0.69, 1.45)	(575.8, 0.46, 4.96)
ATC	(476.9, 0.32, 0.58	(504.3, 0.36, 1.32)	(173.8, 0.06, 2.17)
COVERT	(301.6, 0.25, 0.40)	(362.9, 0.23, 1.00)	(145.4, 0.09, 0.96)

Table 7.6: Performance of existing scheduling policies for *testing* scenarios (C_{max}, TWT, MAPE)

	DTWK	DPPW	ADRES
FIFO	(149.7, 0.73, 0.35)	(149.7, 0.47, 0.80)	(149.7, 0.21, 0.60)
CR	(160.5, 0.19, 0.18)	(115.2, 0.18, 0.71)	(206.1, 0.03, 0.60)
S/OPN	(159.9, 0.20, 0.20)	(114.2, 0.19, 0.58)	(184.6, 0.02, 0.74)
SPT	(510.2, 0.68, 0.56)	(510.2, 0.74, 0.86)	(510.2, 0.45, 2.59)
ATC	(302.8, 0.19, 0.29)	(306.4, 0.19, 0.53)	(220.4, 0.01, 1.05)
COVERT	(242.8, 0.15, 0.21)	(237.9, 0.14, 0.46)	(152.2, 0.02, 0.55)

MAPE, and the combination of ATC and ADRES achieves the best TWT. However, these existing scheduling policies are easily dominated by the evolved scheduling policies from the *aggregate Pareto fronts* as shown in Figure 7.5. Moreover, when compared with the non-dominated scheduling policies obtained by *each independent run* of NSGA-II, SPEA2, HaD-MOEA and DMOCC, it can be observed from our experiments that these scheduling policies are dominated by at least one of the evolved scheduling policies using the proposed methods in both the training and testing scenarios. These results show that the non-dominated scheduling policies evolved

by the proposed MO-GPHH methods not only show good performance on the training scenarios, but can also be effectively reused for unseen scenarios.

7.2.3 Comparison to Existing DRs and Regression-based DDARs

We further examine the effectiveness of the evolved SPs by comparing them with existing DRs and regression-based DDARs. The four due date assignment models used here are TWK, NOP, JIQ and JIS in combination with the six dispatching rules reported in the previous section.

Different from the dynamic DDARs, the coefficients of the employed models have to be determined by regression methods for each job shop setting. Figure 7.6 and Table 7.7 show the performance of these $(6 \times 4) = 24$ combinations and the *aggregate Pareto front* of the non-dominated scheduling policies for the case with utilisation of 90%, 5 machines, *full* setting and processing times following an exponential distribution. In this case, the coefficients of the due date assignment models TWK, NOP, JIQ and JIS were determined by using Iterative Multiple Regression (IMR) [66]. The values shown in the figure are the average values of the three objectives obtained from 30 independent simulation replications.

Since this work deals with a dynamic JSS environment with stochastic factors (such as arrival process, processing time), we also examine the Pareto dominance of SPs under uncertainty. We will utilise the concept of *statistical Pareto dominance* based on OBJW introduced in Section 5.1.5 to help determine the Pareto dominance relation between two scheduling policies in this case.

The results show that each of the 24 existing SPs considered here is statistically dominated (with a significant level $\alpha = 0.05$ and using the Bonferroni method [128] used to adjust the value of each individual statistical test) by

Figure 7.6: DRs and Regression-based DDARs vs. evolved scheduling policies. (● and ▲ are respectively the non-dominated and dominating evolved scheduling policies; and + represents existing scheduling policies).

Table 7.7: Performance of DRs and regression-based DDARs

	C_{max}	TWT	MAPE
FIFO + TWK	129.381 ± 27.26	3.231 ± 1.38	0.495 ± 0.05
FIFO + NOP	129.381 ± 27.26	1.933 ± 1.31	0.451 ± 0.09
FIFO + JIQ	129.381 ± 27.26	0.924 ± 0.09	0.178 ± 0.01
FIFO + JIS	129.381 ± 27.26	0.871 ± 0.13	0.178 ± 0.01
CR + TWK	183.169 ± 44.14	1.239 ± 1.06	0.291 ± 0.07
CR + NOP	139.275 ± 40.01	1.183 ± 0.94	0.310 ± 0.09
CR + JIQ	103.784 ± 15.15	0.406 ± 0.04	0.084 ± 0.01
CR + JIS	102.935 ± 16.73	0.384 ± 0.04	0.085 ± 0.01
S/OPN + TWK	156.279 ± 27.73	1.650 ± 1.36	0.440 ± 0.10
S/OPN + NOP	126.500 ± 29.73	1.698 ± 1.44	0.375 ± 0.10
S/OPN + JIQ	102.718 ± 15.44	0.383 ± 0.05	0.088 ± 0.08
S/OPN + JIS	101.404 ± 15.74	0.390 ± 0.07	0.089 ± 0.09
SPT + TWK	603.661 ± 235.69	0.996 ± 0.30	0.608 ± 0.03
SPT + NOP	603.661 ± 235.69	1.370 ± 0.33	0.844 ± 0.03
SPT + JIQ	603.661 ± 235.69	0.979 ± 0.29	0.632 ± 0.02
SPT + JIS	603.661 ± 235.69	0.990 ± 0.30	0.616 ± 0.02
ATC + TWK	600.539 ± 213.47	0.507 ± 0.19	0.481 ± 0.04
ATC + NOP	609.517 ± 200.21	0.702 ± 0.21	0.441 ± 0.03
ATC + JIQ	570.168 ± 201.18	0.409 ± 0.10	0.395 ± 0.02
ATC + JIS	548.237 ± 190.89	0.381 ± 0.08	0.362 ± 0.02
COVERT + TWK	511.948 ± 204.55	0.424 ± 0.18	0.225 ± 0.03
COVERT + NOP	540.604 ± 210.92	0.483 ± 0.19	0.233 ± 0.04
COVERT + JIQ	379.399 ± 141.83	0.258 ± 0.03	0.141 ± 0.02
COVERT + JIS	336.875 ± 116.26	0.237 ± 0.02	0.135 ± 0.02

at least one evolved SP in the aggregate Pareto front, which is indicated as a *dominating evolved scheduling policy* in Figure 7.6 and Table 7.8. This further shows the high-quality of the evolved SPs even when they are

Table 7.8: Performance of dominating *evolved* scheduling policies

SP/Objective	C_{max}	TWT	MAPE
#1	91.507 ± 15.32	0.204 ± 0.04	0.114 ± 0.01
#2	91.824 ± 15.47	0.194 ± 0.04	0.114 ± 0.01
#3	95.497 ± 16.46	0.135 ± 0.02	0.116 ± 0.01
#4	95.590 ± 16.61	0.182 ± 0.03	0.093 ± 0.01
#5	96.075 ± 16.85	0.134 ± 0.02	0.117 ± 0.01
#6	96.996 ± 16.67	0.182 ± 0.03	0.093 ± 0.01
#7	101.597 ± 15.45	0.180 ± 0.02	0.097 ± 0.01
#8	102.104 ± 17.60	0.171 ± 0.02	0.111 ± 0.01
#9	102.430 ± 19.77	0.133 ± 0.01	0.127 ± 0.01
#10	102.562 ± 18.31	0.132 ± 0.01	0.122 ± 0.01
#11	102.567 ± 19.25	0.122 ± 0.01	0.125 ± 0.01
#12	102.885 ± 18.65	0.118 ± 0.01	0.135 ± 0.02
#13	103.970 ± 18.12	0.106 ± 0.02	0.138 ± 0.02
#14	106.151 ± 17.82	0.096 ± 0.01	0.084 ± 0.01
#15	106.892 ± 17.40	0.085 ± 0.01	0.136 ± 0.02
#16	107.526 ± 21.07	0.086 ± 0.01	0.122 ± 0.01
#17	109.083 ± 18.78	0.165 ± 0.01	0.066 ± 0.01
#18	110.068 ± 22.30	0.126 ± 0.01	0.071 ± 0.01
#19	110.518 ± 19.41	0.163 ± 0.01	0.069 ± 0.01
#20	110.667 ± 19.84	0.154 ± 0.01	0.068 ± 0.01
#21	110.754 ± 19.52	0.080 ± 0.01	0.136 ± 0.02
#22	111.922 ± 19.59	0.040 ± 0.01	0.134 ± 0.01
#23	112.373 ± 17.29	0.040 ± 0.01	0.122 ± 0.01
#24	114.717 ± 23.73	0.072 ± 0.01	0.097 ± 0.01
#25	115.209 ± 24.68	0.057 ± 0.01	0.101 ± 0.01
#26	116.613 ± 22.75	0.069 ± 0.01	0.091 ± 0.01
#27	120.045 ± 27.26	0.053 ± 0.01	0.107 ± 0.01
#28	120.625 ± 25.72	0.052 ± 0.01	0.107 ± 0.01
#29	124.931 ± 24.49	0.065 ± 0.01	0.095 ± 0.01
#30	125.122 ± 26.18	0.065 ± 0.01	0.096 ± 0.01
#31	132.354 ± 26.58	0.105 ± 0.01	0.080 ± 0.01

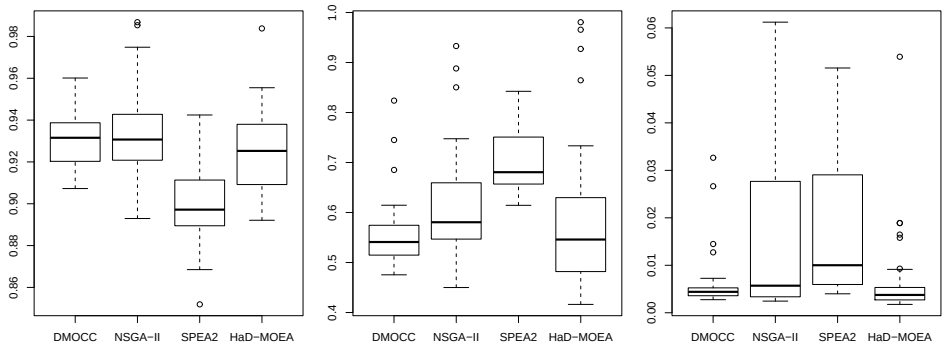
compared with customised SPs. Figure 7.6 also reveals that the combinations of existing DRs and DDARs do not cover all promising regions in the objective space. This observation suggests that automatic design methods like the proposed MO-GPHH methods are essential in order to provide informed knowledge about any potential SPs. Moreover, these results suggest that the evolved SPs are robust to uncertain JSS environments even though they are trained/evolved based on the mean values of the objectives across different simulation scenarios.

7.3 Further Analysis

The comparison results in the previous section have shown the effectiveness of the proposed MO-GPHH methods for evolving efficient SPs. In this section, we will compare the ability of the proposed MO-GPHH methods in exploring the Pareto front of non-dominated SPs.

7.3.1 Performance of MO-GPHH Methods

The performance indicators of the four MO-GPHH methods are shown in Figure 7.7 and Figure 7.8 (better methods have higher HVR and smaller SPREAD and GD). With the training scenarios, *Wilcoxon signed-rank tests* (with significance level of 0.05) show that the HVRs obtained by DMOCC, NSGA-II, and HaD-MOEA are significantly better (higher) than that of SPEA2. This means that the SPs obtained by these methods can significantly dominate those obtained by SPEA2. In terms of HVR, there is no significant difference between DMOCC, NSGA-II, and HaD-MOEA but the standard deviations of HVRs obtained by DMOCC and HaD-MOEA are slightly smaller than those obtained by NSGA-II. For the distribution of the obtained SPs on the Pareto fronts, the SPREAD values obtained by DMOCC and HaD-MOEA are significantly better than those obtained by NSGA-II and SPEA2. Although DMOCC uses *crowding distance* (like



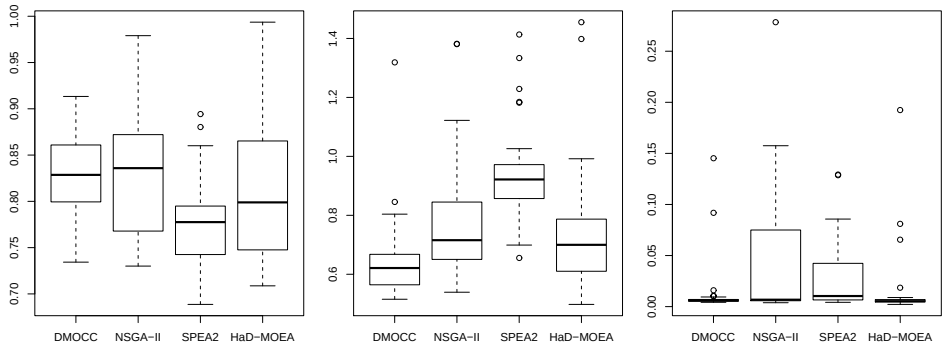
(a) HVR (b) SPREAD (c) GD

Figure 7.7: Performance of MO-GPHH methods on the *training* scenarios. (HVR to be maximised; and SPREAD and GD to be minimised).

NSGA-II) as the indicator for individuals in less crowded areas, the selection method for choosing representative individuals as well as individuals for crossover has significantly improved the uniformity of the Pareto fronts obtained by DMOCC. Given a better distribution of scheduling policies, GD of DMOCC is significantly smaller than NSGA-II although there is no significant difference in HVR. Overall, DMOCC and HaD-MOEA are the two most competitive methods for the problems studied in this chapter. It should be noted that performances of the obtained non-dominated SPs on the testing scenarios are rather consistent with those obtained in the training scenarios. However, the SPREAD of DMOCC is significantly better than all the other methods. These experimental results show that the proposed DMOCC is a very promising approach for evolving highly efficient SPs.

7.3.2 Complexity of DMOCC

The complexity of DMOCC depends on the operations performed at each generation. Similar to NSGA-II, the three basic operations of DMOCC are (1) non-dominated sorting, (2) crowding-distance assignment, and (3) sorting for genetic and representative selection. For non-dominated sort-



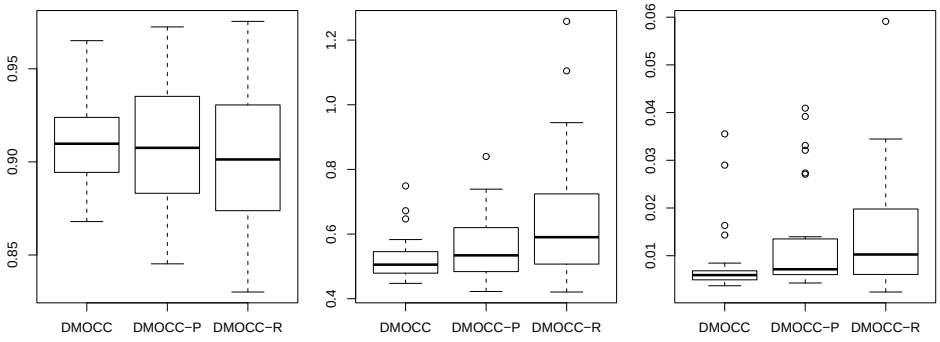
(a) HVR (b) SPREAD (c) GD

Figure 7.8: Performance of MO-GPHH methods on the *testing* scenarios.

ing, we adopt the procedure proposed by Deb et al. [52], which results in the worst-case complexity of $O(MR^2)$ where M is the number of objective functions to be minimised and R is the size of the joined population. Assuming that the size of each sub-population N is the same and the maximum size of the archive is A, the size of the joined population is $R = 2N + A$. The worst-case complexity of the crowding-distance assignment and sorting for genetic operators and representative selection are $O(MR\log(R))$ and $O(R\log(R))$, respectively. It is obvious that the complexity of the algorithm is $O(MR^2)$, governed by the non-dominated sorting procedure. Therefore, both the sub-population size N and archive size A will influence the complexity of DMOCC. For complex problems where GP needs a large population size in order to maintain the diversity of the population, the complexity of NSGA-II ($O(MN^2)$) will increase since its complexity depends mainly on the population size. Because the number of final non-dominated solutions is not necessarily as large as the population, the complexity of DMOCC can be smaller than that of NSGA-II by maintaining a small archive.

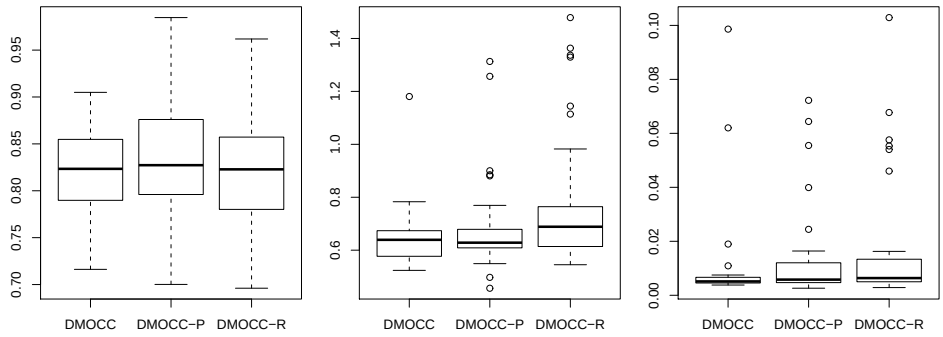
7.3.3 Representative Selection

As mentioned earlier, representative selection is an important factor in the proposed cooperative coevolution method. Here we will examine the influence of representative selection methods on the performance of the proposed DMOCC. Apart from the representative selection method discussed above, two other methods are also examined here. The first is a problem-based method which applies two different representative selection strategies for each sub-population. In this method, the representatives of the sub-population of DRs are selected by using a similar method to that in DMOCC (based on the non-dominated rank and crowding distance). On the other hand, the representatives of DDARs are selected based on the values of `MAPE`. This method assumes that good DDARs (with small values of `MAPE`) are able to cope with a wide range of DRs, and thus it will only focus on `MAPE` when selecting representatives to form complete SPs with the evolved DRs. The second method simply selects random representatives from each sub-population. The performances of the three representative selection methods are shown in Figure 7.9 and Figure 7.10. The DMOCC-P and DMOCC-R are similar to DMOCC, except that they employ problem-based and random representative selection methods, respectively.



(a) HVR (b) SPREAD (c) GD

Figure 7.9: Influence of representative selection methods on the *training* scenarios.



(a) HVR (b) SPREAD (c) GD

Figure 7.10: Influence of representative selection methods on the *testing* scenarios.

The results from these figures show that the HVR values of the three selection methods are not significantly different. However, DMOCC gives significantly better SPREAD and GD performances as compared to DMOCC-P and DMOCC-R. Also, the DMOCC-P is better than DMOCC-R according to these two performance metrics. The results show that it is important to include the representative selection method based on the non-dominated rank and crowding distance. Although individuals selected for genetic operations for each sub-population also employ the non-dominated rank and crowding distance, the features of non-dominated rank and crowding distance still have a strong impact on the performance of the representative selection method. Also, the evolved DDARs with good `MAPE` are not necessarily suitable for a wide range of DRs, since DMOCC-P does not produce Pareto fronts with the performance of SPREAD as good as DMOCC.

7.3.4 Choice of Training Scenarios

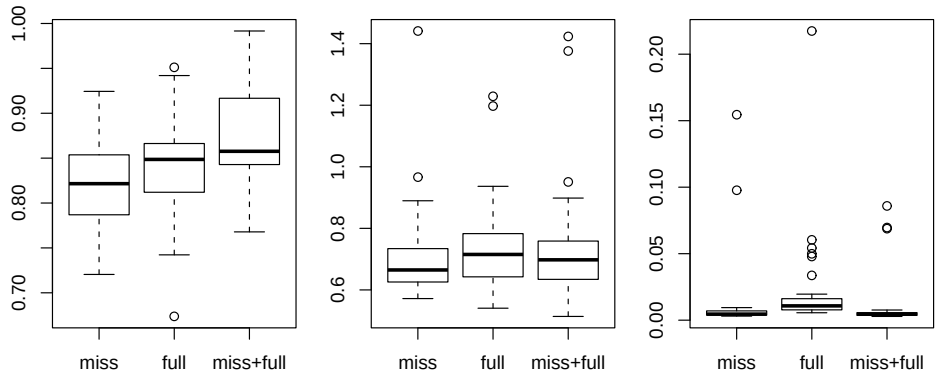
Like other machine learning methods, it would be interesting to examine how the choices of training sets or training scenarios may influence the ability of the proposed MO-GPHH in exploring effective scheduling policies. Previously, we have trained the proposed MO-GPHH on scenarios

with the *missing* setting of arriving jobs. This section will further examine the cases where *full* and *missing+full* settings are used. The first case used 4 simulation scenarios and the second case used 8 simulation scenarios for training. Figure 7.11 shows the performance of DMOCC on the testing scenarios when different training scenarios have been used.

The results show that there is no significant difference between the cases where either *missing* or *full* setting is used. When both *missing* and *full* settings are used for training, the obtained HVRs are significantly better than those obtained in the cases where either *missing* or *full* setting is used and there is no significant differences in SPREAD and GD. This indicates that more general training scenarios are necessary in order to improve the quality of the evolved scheduling policies. Although the simulation scenarios with jobs following the *missing* setting also include jobs following the *full* setting, it is still unable cover all situations that happened in the simulation scenarios with jobs following the *full* setting. The major problem is that the use of a large number of simulation scenarios will increase the computation cost of the proposed methods. Thus, there is a trade-off between the computational effort and the reusability of the evolved scheduling policies. Depending on the available computational resources and the environments where the evolved scheduling policies will be applied, the training simulation scenarios should be logically selected.

7.3.5 The Evolved Scheduling Policies

This section investigates how the evolved scheduling policies can effectively solve the problem and how trade-offs can be made among different objectives. Since many SPs have been obtained from our experiments, we will present some examples of the SPs. Figure 7.12 is the same as Figure 7.5(a) with a different view and the points surrounded by rectangles are the example SPs, which are also presented in Table 7.9.



(a) HVR (b) SPREAD (c) GD

Figure 7.11: Influence of training scenarios on testing performance of DMOCC.

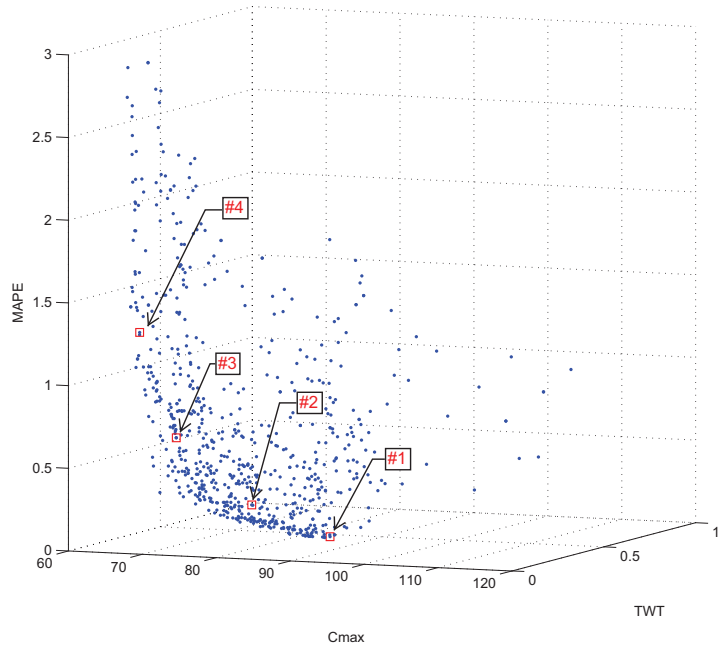


Figure 7.12: Pareto front and selected evolved scheduling policies.

In general, the evolved rules are not very complicated and are in forms that are explainable, especially for the DDARs, which are simply linear

Table 7.9: Examples of the evolved scheduling policies

Scheduling Policy #1 ($C_{max} = 90.774, TWT = 0.170, MAPE = 0.098$)	
DR :	$\frac{2RO^2MP}{(0.9087407-RJ)} + IF(\max(BWR, WR) - 2SJ, -PR, -RJ)$
DDAR:	$OT + LOT + QWL$
Scheduling Policy #2 ($C_{max} = 83.597, TWT = 0.048, MAPE = 0.312$)	
DR :	$\max(-0.043989424, \frac{1}{PR}(\frac{2PR+RT}{PR} + \max(RT, PR) - SJ))$
DDAR:	$OT + LOT + QWL$
Scheduling Policy #3 ($C_{max} = 66.844, TWT = 0.311, MAPE = 0.608$)	
DR :	$RT\frac{RO}{PR} + IF(-CWR\frac{SJ}{PR}, DJ, RT(\frac{RO}{PR})^2) - 2rJ + W$
DDAR:	$OT + 2LOT + QWL$
Scheduling Policy #4 ($C_{max} = 68.162, TWT = 0.059, MAPE = 1.321$)	
DR :	$RT(2\frac{RO}{PR} - CWR) - 3rJ + W$
DDAR:	$OT + 2LOT + 2QWL + 2SOTR$

*These rules have been simplified for better presentation but still ensure to achieve the same objective values obtained by the original evolved rules.
*IF(a, b, c) will return b if a $\geq$ 0; otherwise it will return c.

combinations of different terms. Scheduling Policy #1 is the one that achieves the best MAPE among the four SPs. Since Scheduling Policy #2 also employs the same DDAR, the better MAPE obtained by the first SP is strongly influenced by its DR. The first component of the DR in Scheduling Policy #1 will be negligible at the latter stage of the simulation since it has RJ in its denominator, which increases with the time. Therefore, this component has little impact on the performance of the scheduling policy and the performance of the rules will be governed by the second component. At the first glance, the second component is a combination of both SPT and FIFO because the priorities of jobs are either $-$PR (higher priority for jobs with smaller processing time) or $-$RJ (higher priority for jobs arriving at the machine earlier). The switch between FIFO and SPT is controlled by $\max(BWR, WR) - 2SJ$. In the case that the slack of jobs SJ is positive (not late) and larger than $\frac{1}{2}\max(BWR, WR)$, FIFO will be applied; oth-

erwise SPT will be used. The purpose of this rule is to maintain a more predictable flow (by FIFO) of jobs when the jobs are not late, and to finish jobs with a smaller processing time first so as to reduce the number of jobs in the shop. These features make the flowtime prediction by $\mathtt{OT} + \mathtt{LOT} + \mathtt{QWL}$ more accurate because it is important for a shop to have a smooth flow of jobs.

Scheduling Policy #2 provides a better $\mathtt{C}_{\mathtt{max}}$ and TWT as compared to the first SP. Different from the DR in the first SP, the DR in this SP emphasises more on reducing the time jobs stay in the shop as well as on reducing the lateness of jobs. In this case, the rule will give a higher priority to jobs with a higher RT in order to reduce its flowtime (and the makespan in general). In the case that jobs have large negative slacks, the rule will give a higher priority to jobs with larger negative slacks and reduce the lateness of jobs. However, because this rule will disturb the flow of jobs, the prediction will become less reliable.

Scheduling Policies #3 and #4 are the two SPs that provide the smallest makespans among the four. In general, the DRs in these two SPs give a higher priority for jobs with a larger RT and RO and a smaller release time rJ in order to reduce the makespan. However, when these values are similar for the considered jobs, W is used to break the ties and gives a higher priority to jobs with higher weights. The focus on makespan has made the delivery performance of Scheduling Policy #3 worse as compared to the first and the second SPs. Scheduling Policy #4 tries to minimise TWT by over-estimating the flowtimes for reducing the tardiness of jobs. The consequence is that the reliability of the due date is deteriorated significantly.

From Scheduling Policies #1 to #4, MAPE tends to be increased and either $\mathtt{C}_{\mathtt{max}}$ or TWT is reduced, especially for $\mathtt{C}_{\mathtt{max}}$. Tracking the evolved SPs along this direction helps explain how the trade-offs can be achieved, mainly between $\mathtt{C}_{\mathtt{max}}$ and MAPE. When observing the evolved SPs on the Pareto fronts along other directions, we are also able to explore other types of trade-offs, e.g., accepting a higher makespan for a better TWT. Since the

evolved SPs are constructed based on the genetic materials of other SPs or individuals, we can easily examine the connection among these SPs and understand what creates the trade-offs. In other words, the evolved DRs and DDARs are interpretable in this case.

7.4 Chapter Summary

Designing an effective scheduling policy is challenging and time-consuming because it needs to take into account multiple scheduling decisions and conflicting objectives in a manufacturing system. The original contributions of this chapter can be summarised as follows. First, we developed genetic programming based hyper-heuristics for automatic design of scheduling policies. The novelty here is on the representations and evolutionary search approaches to handling multiple scheduling rules and conflicting objectives in the evolution of scheduling policies. Four genetic programming based hyper-heuristics have been proposed in this chapter. The performances of the proposed methods were examined by training and testing the evolved scheduling policies on various simulation scenarios. The results show that the evolved scheduling policies outperform the existing scheduling policies created from combinations of popular dispatching and dynamic or regression-based due-date assignment rules on both the training and testing scenarios. Moreover, the Pareto fronts obtained also provide much better knowledge about the space of potential scheduling rules, which cannot be achieved by simple combinations of existing scheduling rules or by methods using an aggregate objective function to handle multiple objectives. Another advantage of the proposed methods is that they perform well on unseen situations, which makes the evolved scheduling policies more robust when they are employed in stochastic and dynamic job shops. Analysis of the evolved scheduling policies also shows that the proposed methods can evolve not only effective but also very meaningful scheduling policies. In addition, it is easy to apply the proposed method

to track the evolved scheduling policies along the Pareto front for better understanding of the trade-off among different objectives.

The second contribution is the development of a diversified multi-objective cooperative coevolution method, which shows favourable performance as compared to other popular evolutionary multi-objective search strategies. The experimental results show that the proposed DMOCC method can evolve Pareto fronts that are better than NSGA-II and SPEA2. It is also very competitive on all performance metrics for the training scenarios and provides a better spread of the evolved scheduling policies for the testing scenarios. Further analysis also shows that the representative selection strategies based on non-dominated rank and crowding distance play a very important role in the proposed cooperative coevolution for evolving well-distributed Pareto fronts. Another advantage of the coevolution approach is that multiple scheduling decisions can be evolved in different sub-populations in order to reduce the complexity of evolving the sophisticated scheduling policies. This method can be modified to take advantage of parallelisation techniques so as to reduce the computational time.

For future studies, we will enhance the performance of the proposed cooperative coevolution method by improving representation of the scheduling rules, genetic operations and strategies to explore optimal Pareto fronts. When examining the impact of the training scenarios, it has been observed that general training scenarios helped improve the quality of the evolved scheduling policies. However, this also increased the computational time, and thus we will also study decomposition approaches for better learning/evolving in the algorithm. The incorporation of other scheduling rules such as order review/release and order acceptance/rejection will also be considered. In addition, we want to extend the proposed methods to evolve heterogeneous scheduling rules for shops with specialised machines or groups of machines (batch processing, machines with sequence-dependent setup, etc.). Besides, we will further examine the performance of DMOCC on other multi-objective optimisation problems.

Chapter 8

Conclusions

This thesis focuses on developing new GPHHs for job shop scheduling problems. The overall goal is to enhance the performance of GPHHs for JSS by investigating different key aspects of GP and JSS such as representations, evaluation schemes for evaluating dispatching rules, and evolutionary search mechanisms. In order to achieve this goal, several new GP methods are developed to evolve effective dispatching rules which can cope with complex characteristics of JSS. The reusability and the effectiveness of the evolved dispatching rules are measured by using popular benchmark JSS instances and simulation models in the literature.

The rest of this chapter provides conclusions for each of the research objectives of this thesis and gives main conclusions and highlights from each chapter. Then, insightful discussions are provided to help understand key issues in this research direction. Finally, the chapter presents potential research areas for future works.

8.1 Achieved Objectives

The following research objectives have been fulfilled by this thesis:

- Through this thesis, three new GP methods were developed to evolve dispatching rules with adaptive behaviours. The first two GP methods corresponding to representations R_1 and R_3 (Chapter 3) focus on embedding machines attributes and the non-delay factor into the evolved rules to help them adapt their decisions based on changes in the shop. The third method aims to design iterative dispatching rules (Chapter 4) which can learn from the mistakes made by previous completed schedules to construct (and fine-tune) better scheduling decisions. The experimental results show that the new proposed methods can evolve rules that are significantly better than rules evolved by other GP methods and well-known dispatching rules in the literature. Within the three proposed methods, the GP method to evolve iterative dispatching rules shows the best performance when tested on a set of benchmark static JSS instances.

- This thesis developed a new multi-objective GPHH (Chapter 5) to evolve a Pareto front of non-dominated dispatching rules for JSS. Instead of evolving a single rule with assumed preferences between different objectives, the new methods simultaneously evolve non-dominated rules to help decision makers select appropriate rules based on further knowledge of possible trade-offs. The results show that the evolved Pareto front contains rules that are significantly better than the best existing rules in the literature for a specific objective. Moreover, we find that GP can help explore rules with much better trade-offs as compared to dispatching rules designed by human experts when multiple conflicting objectives are considered.

- This thesis developed new GPHHs to design scheduling policies for JSS by simultaneously evolving multiple scheduling rules for multiple scheduling decisions. Besides the sequencing decisions, this thesis also focuses on due date assignment decisions (Chapter 6) to improve the delivery performance of job shops. The cooperative co-

evolution approach (Chapter 7) is employed to evolve rules for handling a specific scheduling decision from a specific sup-population to reduce the complexity of evolving sophisticated scheduling policies. In order to explore non-dominated scheduling policies, a new evolutionary search mechanism and customised genetic operations are proposed. The experimental results show that the evolved scheduling policies are significantly better than scheduling policies in the literature. Detailed analysis also shows that the evolved scheduling policies are interpretable and we can understand how trade-offs between different objectives can be made.

8.2 Main Conclusions

This section presents the main conclusions for the three research objectives drawn from the five major contribution chapters (Chapter 3 to Chapter 7).

8.2.1 Representations and Evaluation Schemes

In order to evolve dispatching rules for scheduling problems, two key important issues must be handled. First, we need to decide how rules can be represented by GP programs. Second, given a specific representation of rules, we need to determine how the outputs from the rules are used to construct schedules.

Representations

Similar to any application of evolutionary computation, representations play a very important role in the effectiveness of GP for JSS problems. Previous works have only focused on simple arithmetic representation since it is the most straightforward way to represent priority functions. However, our work has shown that arithmetic representation is not the unique way

to deal with scheduling problems. Assuming that good attributes and effective rules are available, we can apply a decision-tree like representation to create adaptive behaviours for our evolved rules (as shown in Chapter 3). In the case that scheduling problems are too complicated and existing dispatching rules are not sufficient to handle all possible situations, a hybrid (or mixed) representation between decision-tree like representation and arithmetic representation can be used. The main advantage of the hybrid representation is that we do not need to rely on existing rules (which can cause certain biases because these rules are usually designed by a human expert for some specific situations). Therefore, the hybrid representation can help improve the generality of evolved rules by preventing overfitting problems.

In order to allow GP to evolve more effective dispatching rules, different aspects have been considered through this thesis. One of the key remarks on this issue is that more effective rules are usually evolved by using sophisticated representations. The reason is that simple representations are not powerful enough to handle all possible situations (e.g. changes of due date tightness). Our studies on iterative dispatching rules in Chapter 4 show that the multiple-tree representation can help evolve better dispatching rules as compared to the single tree representation. Also, evolving an independent tree for the look-ahead strategy is useful to enhance the competitiveness of evolved rules.

Evaluation Schemes

For JSS, our studies show that evaluation schemes are very important. Given a specific representation, different evaluation schemes can be applied to solve JSS. While representations decide how the rules should look, the role of evaluation schemes is not that straightforward. In this thesis, evaluation schemes define how schedules or solutions are constructed (or reconstructed); and therefore, define the nature of evolved rules of heuristics.

In Chapter 4, given the arithmetic representation, outcomes of evolved rules can be very different. By using an evaluation scheme that iteratively constructs new schedules, GP can significantly enhance the performance of evolved rules. As a result, although the rules obtained by GP with the simple evaluation scheme and the "iterative" evaluation scheme are simply priority functions, they have different ways to find a final schedule. For GP with a simple evaluation scheme, the evolved priority functions act like *construction heuristics* that construct the schedule step by step (by adding operations into the partial schedule). On the other hand, the priority functions representing iterative dispatching rules evolved by GP play the role of both *construction heuristics* and *improvement heuristics*. As analysed in Chapter 4, these iterative dispatching rules have the characteristics of local search heuristics. Understanding the characteristics of the evolved rules has allowed us to enhance their performance, e.g., by applying the concept of variable neighbourhood search.

Chapter 6 also emphasises the importance of evaluation schemes when GP is employed to evolve due date assignment rules. In Chapter 6, extensive experimental results indicate that operation-based due date assignment rules are more effective than aggregate due date assignment rules. The difference here is also caused by different evaluation schemes. Instead of directly estimating job flowtimes, evaluating the GP tree to gradually estimate operation flowtimes and partial job flowtimes can lead to better predictions. In this case, aggregate due date assignment rules are just simple functions while operation-based due date assignment rules can be considered as *recursive* functions.

8.2.2 Multi-objective GPHH

Handling multiple objectives is important for practical applications in JSS. In order to tackle multiple objectives, different aspects have to be considered.

Evolutionary Search Mechanisms

The aim of MO-GPHH is to find the Pareto front of non-dominated scheduling rules. In our work, the average measures from each objective obtained from all training simulation scenarios are used to determine the Pareto dominance of evolved rules.

In Chapter 5, a new MO-GPHH method is proposed. In this method, we apply the harmonic crowding distances from HaD-MOEA and non-dominated ranking to select evolved rules for genetic operations. The experimental results have shown that the evolved rules are competitive when tested on dynamic JSS simulation models with five objectives. The evolved rules are not only better than existing rules designed for specific objectives but also outperform existing rules in terms of Pareto dominance. When comparing the Pareto front with a large set of existing rules, it can be concluded that there are many promising trade-off dispatching rules that have not been explored by human experts. This confirms the need to have such an automatic design method as MO-GPHH to ensure that potential non-dominated rules will not be ignored.

More challenges occur in Chapter 7 where two scheduling rules have to be evolved simultaneously to satisfy multiple objectives. Methods similar to MO-GPHH in Chapter 5 can still be applied. However, because of the complexity of evolved schedule policies, that method (in Chapter 5) have trouble finding effective and diverse Pareto fronts. A new evolutionary search mechanism based on cooperative coevolution has been proposed. This method allows specific rules to be evolved in sub-populations and the quality of each scheduling rule is measured by combining the rule with representative rules from the other sub-population to form a complete scheduling policy. A new crossover operation and a new representative selection strategy are also developed to help diversify non-dominated rules on the obtained Pareto front. The results show that the new evolutionary search mechanism is very competitive and able to evolve Pareto fronts with good uniformity.

Statistical Pareto Dominance

When dispatching rules are applied to dynamic and stochastic environments, assessing Pareto dominance becomes a challenging issue. To overcome this problem, we have proposed two new statistical procedures in Chapter 5 to help determine whether a dispatching rule statistically Pareto dominates the another. The first procedure tries to determine the Pareto dominance based on the overall dominance of each objective. The second procedure emphasises more on the Pareto dominance in each simulation replication. While the replication-wise procedure is simpler as compared to the objective-wise procedure (only one statistical test is needed), it has trouble detecting the dominance between two solutions when variances of obtained objective values are large. The results show that the two procedures provide consistent conclusions in our experiments.

Robustness

For a single objective, it is reasonably straightforward to evaluate the reusability of evolved rules on unseen scenarios or instances. However, it is not a trivial task in the case of multiple objectives. In this case, the reusability or the robustness of evolved rules is not only decided by any specific objective but also based on the Pareto dominance relations. Chapter 5 presents the first systematic approach to measuring the robustness of evolved rules when multiple objectives are considered. The key idea of this approach is to measure the robustness based on the difference in Pareto dominance relations of evolved rules and benchmark rules across all simulation scenarios. The experimental results indicate that evolved rules are quite robust when tested on unseen simulation scenarios with five different objectives.

8.2.3 Evolving Comprehensive Scheduling Systems

In order to develop a comprehensive scheduling system, we have to ensure that all related decisions are considered and well coordinated. We

deal with this problem by first developing GP methods for due date assignment (Chapter 6). Then, more sophisticated GP methods are proposed to simultaneously deal with multiple decisions (Chapter 7).

Chapter 7 investigates two approaches to evolving scheduling policies consisting of dispatching rules and due date assignment rules. The first approach tries to evolve GP individuals with two trees representing two scheduling rules (due date assignment and dispatching rules). The second approach aims to reduce the complexity of evolving sophisticated scheduling policies by co-evolving scheduling rules from specific sub-populations. By applying this approach, we can easily adjust the size of each specific sub-population depending on the complexity of each scheduling rules (i.e. simple rules can be evolved from a smaller sub-population). Also, a common external archive is used to store the non-dominated scheduling policies from the two sub-populations to better assess the quality of evolved scheduling rules and provide genetic materials for the specialised crossover operation. The experimental results show that the cooperative coevolution approach can effectively find the Pareto front of non-dominated scheduling policies and have lower complexity as compared to methods using the first approach.

8.3 Discussion

The previous section provided a summary of key findings in this thesis. In this section, we provide further discussions on general issues covered by the thesis and related to this field of research.

8.3.1 Job Shop Scheduling: Static vs Dynamic

Although job shop scheduling has been extensively studied in the literature, there are still missing links between research on static and dynamic scheduling problems. Dispatching rules are an approach that can be adopted

to handle both static and dynamic problems. However, because of the difference in the assumptions between the two problems, dispatching rules that are effective for static problems do not necessarily perform well on dynamic problems, and vice versa. Even though our ultimate goal is to apply the rules to dynamic environments, studying rules under static environments is useful to understand how optimisation approaches can be adapted to improve their scheduling performances.

In Chapters 3 and 4, several GP methods have been proposed to evolve advanced dispatching rules. However, the evolved rules in these chapters are still restricted to static problems either in terms of effectiveness or applicability. For example, as shown in the analysis section of Chapter 3, the rapid changes of dynamic job shops make these evolved rules less effective. In Chapter 4, the iterative dispatching rules are very promising but they cannot directly be applied to dynamic environments without considering other scheduling/planning factors such as planning periods (e.g. new schedules are constructed every three working days) or job release methods (which jobs are to be released at the beginning of a planning period). The problems of iterative dispatching rules are similar to those of other optimisation methods. However, these rules are much more efficient than traditional optimisation methods, which makes this method attractive for complex and large-scale manufacturing systems.

The missing links between static and dynamic problems are not only concerns in this thesis but also an interesting research topic in the research community. This thesis has successfully reduced the gap between the two problems by developing GP methods to evolve effective and flexible dispatching rules. We believe that it is a very promising research direction that should attract more attention in the future studies.

8.3.2 Need "Better" Representations

Dispatching rules in the literature are usually in the form of arithmetic functions. Most methods for evolving dispatching rules in this thesis also employ the arithmetic representation. This is because arithmetic operators provide a convenient way to represent the relationship between different scheduling terms such as processing times and due dates. However, there are still two main drawbacks with this representation:

- The sophisticated rules evolved by GP are sometimes not very intuitive. In order to understand these rules, we still need to simplify the rules and utilise expert knowledge to explain evolved rules as shown in the analysis throughout this thesis. This also causes difficulty for the shop operators and production managers to understand why jobs should be scheduled in a certain way. Therefore, we still need to develop new representations with better interpretability.

- The quality of evolved rules also depends on the selected function sets. Although basic functions such as $+$, $-$, $\times$ and $\%$ in combination with conditional functions can help evolve very sophisticated functions, finding these effective rules is still challenging because of local optima and the large search space. For this reason, it is important to develop new representations that can evolve good dispatching rules and can be explored effectively by GP.

Moreover, it is important that the new representations can help avoid overfitting issues of evolved rules and will not be too sensitive to the choice of GP parameters. This thesis successfully made an attempt by developing R_1 and R_3 representations, but more efforts are needed to improve both quality and interpretability of evolved rules.

8.3.3 GPHHs and Other Optimisation Techniques

While treated as machine learning techniques throughout this thesis, our proposed GPHHs have different relationships with other optimisation techniques for JSS. GPHHs can also be considered as an optimisation approach similar to applications of other meta-heuristics (e.g. genetic algorithms and simulated annealing) for scheduling. The key difference is that GPHHs explore the heuristic search space instead of the solution (schedule) search space. From the implementation viewpoint, the output of GPHHs is an instruction (rule or heuristic) on how a schedule is generated. Meanwhile meta-heuristics produce a complete schedule of all jobs or at least a permutation of jobs, which can easily be transformed into a detailed schedule. Because of this difference, the obtained rules from GPHHs can be reused for further problem instances while the obtained solutions from meta-heuristics typically can only be used for the corresponding instances.

From the optimisation viewpoint, GPHHs still work on the same heuristic search space regardless of the number of jobs or job information. On the other hand, meta-heuristics work on different solution search spaces when dealing with different problem instances. Therefore, the search space of meta-heuristics increase as the number of jobs increase while the search space of GPHHs remain the same. As the result, searching for good solutions with meta-heuristics is more difficult than search for good dispatching rules as the number of jobs increases.

8.3.4 Multiple Scheduling Decisions

Handling multiple scheduling decisions is not a trivial task. Using GPHHs to evolve multiple scheduling decisions can be challenging and time consuming. In this thesis, we apply the cooperative coevolution approach to reducing the complexity of evolving multiple rules. This approach is quite promising as shown in our experiments. Moreover, as we aim to use GP for automatically designing the whole scheduling system involv-

ing many related scheduling/planning decisions, cooperative coevolution would be very useful. However, if we need to deal with multiple conflicting objectives, representative selection strategies will play a very important role, either to control the selection pressure or to maintain the diversity of evolved non-dominated scheduling policies. This is still an open topic for future research.

8.4 Future work

Applying GP for JSS in particular, and scheduling problems in general, is still at a very early stage and there are many things to be done to make GP a powerful method for scheduling problems. This section highlights some future work, motivated by our studies in this thesis.

8.4.1 Feature Selection

Feature selection in GP is an important step to enhance the performance of GP for evolving dispatching rules by (1) reducing the search space of evolved rules, (2) improving the generality of the evolved rules, (3) dealing with specific objectives (each objective may require a specific set of features), and (4) creating specialised rules for complex processes (bottleneck machine, batching, assembly, etc.). Because there are only a few studies that have considered feature selection when learning scheduling rules in the literature, there are many issues that need to be investigated.

Besides selecting low-level features, future studies need to focus on constructing high-level features evolved by GP. We can develop two-stage learning systems in which GP is employed at the first level to evolve good high-level features; then the second stage will employ GP or other machine learning approaches (neural network, decision tree, etc.) to learn effective rules based on the evolved features.

8.4.2 Advanced Dispatching Rules for Dynamic JSS

Evolving dispatching rules with GP for dynamic JSS environments is shown to be very promising. Currently, we have only inspected the use of GP to evolve composite dispatching rules (priority functions) and many key aspects have not been considered yet. Some potential approaches to enhancing the quality of evolved rules are: (1) predicting the future impact of a sequencing decision, (2) evolving self-adapted dispatching rules which can improve performance by adjusting their behaviour based on historical data, and (3) evolving periodic scheduling/rescheduling rules by applying the rolling horizon concept.

8.4.3 Hybrid Hyper-heuristic Methods

Dealing with multiple scheduling decisions and multiple conflicting objectives creates great challenges for hyper-heuristics. In this thesis, all scheduling rules are automatically designed by the tree-based GP. However, tree-based GP is not necessarily the best method to evolve each scheduling rule. Future work needs to explore other machine learning methods such as linear GP and learning classifier systems. As a result, effective hyper-heuristics could involve combinations of different machine learning methods.

8.4.4 Benchmarks of Dynamic JSS

Most experiments on dynamic JSS are restricted to simple balanced job shops. In order to have a good assessment of hyper-heuristics, a more comprehensive set of benchmarks are needed. The benchmarks should be categorised into different classes based on the sets of features, the flow of jobs, the flexibility of machines, specialised operations (batching, assembly, etc.). These factors strongly influence the complexity of the shops and help create a standard way to evaluate the effectiveness of hyper-heuristics and GP methods.

8.4.5 Two-Stage Learning/Optimising Systems

For static scheduling problems, a two-stage learning/optimising system can be developed. The objective of such a system is to use GPHHs for offline learning of reusable scheduling rules which can be reused to generate initial solutions for meta-heuristics. By applying this system, we can reduce the computational time of meta-heuristics by providing them with good initial solutions. We expect that this system can utilise the advantages of GPHHs and meta-heuristics to compensate their weakness. Specially, GPHHs can help quickly produce good solutions for complex and large scheduling problems and reduce the total time of the system to find near optimal solutions. Meanwhile, meta-heuristics help refine the solutions provided by GPHH to improve the quality of solutions found by the system. This combination forms an effective and efficient way to handle scheduling problems.

Glossary of Terms

This glossary provides a list of common terminology used across this thesis and along with their definitions.

active schedule A schedule in which no operation can start earlier without delaying other operations.

arrival rate The rate that jobs arrive at the manufacturing system.

cooperative coevolution A evolutionary computation framework in which solutions are decomposed to be evolved by different sub-populations.

crossover A genetic operation used to combine the genetic materials from two parents to create a new individual.

crowdedness A measure of how crowded the area surrounding a solution is in multi-objective optimisation.

dispatching rule A simple heuristic which is used for sequencing tasks in a scheduling problem. At the moments when a sequencing decision needs to be made, dispatching rules will prioritise the jobs in the queue of the machine. Then, the job with the highest priority is the next one to be processed at the corresponding machine.

due date assignment A scheduling decision that determines the due date for a new arriving job (new order received from a customer).

dynamic job shop scheduling A scheduling problem in job shops where new jobs arrival over time, usually according to some stochastic process.

evaluation In evolutionary computation, evaluation is an important step to determine the fitness of an individual. It also strongly influences the computational times of the considered algorithm.

evolutionary algorithm A subfield of evolutionary computation that focuses on algorithms inspired by Darwinian biological evolution.

evolutionary computation A research area in artificial intelligence that focuses on developing nature-inspired algorithms.

evolutionary multi-objective optimisation A subfield of evolutionary computation that focuses on find solutions for optimisation problems with multiple conflicting objective.

evolutionary search mechanism The process that an evolutionary computation technique applies to guide the search towards better solutions. Evolutionary search mechanism includes the procedure to select individuals, genetic operators, and decomposition of solutions.

fitness The quality of evolved programs/solutions.

flowtime The time that a job spends in the manufacturing system from its release time until the completion time of its last operation.

genetic programming An evolutionary computation technique that is employed to evolve computer programs for solving a specific computational task.

heuristic An experience-based technique to find good solutions for a specific computational problem.

hyper-heuristic A framework that employs heuristics or meta-heuristics to explore the "heuristic search space" for heuristic selection or heuristic generation.

initialisation The first step of a heuristic search technique (e.g. tabu search, genetic algorithm). The purpose of this step is to generate the initial solution(or population of solutions) to start the search.

job A job which is sometimes referred to as a customer order (in a simple manufacturing environment) is a components that need to be processed at at least one specific machine in the manufacturing system. A job may include one more processing steps or operations.

job shop Job shop is used to indicate companies that produce customer-specific components in small batches. One feature that sets job shop apart from other production environments is the large variety of routings with different operation processing times through a set of machines (work centre).

job shop scheduling, (JSS) The objective of JSS is to find the best schedule of jobs in a job shop environment. JSS is a popular research topic in the field of operations research and computer science.

machine A type of resource in a manufacturing system used to process a job.

meta-heuristic Optimisation methods designed to deal with hard optimisation problems. Meta-heuristics are search methods containing general low level heuristics that help explore the solution search space to find near-optimal solutions.

multi-pass dispatching rule A heuristic that uses multiple dispatching rules to generate schedules for a scheduling problem instance. Then, the best obtained schedule will be selected.

mutation A genetic operation which is used to modify a parent to create a new individual.

non-delay factor A parameter that is used to determine the time that a machine can wait while there are jobs available in the queue.

non-delay schedule A schedule in which no machine is allowed to be idle when there are jobs in its queue.

non-dominated Solution that cannot be totally dominated by any other solutions regarding all considered objectives in a multi-objective optimisation problems.

non-dominated ranking A procedure used to find the sets of non-dominated solutions from a pool of solutions.

objective A performance measure or a function to be optimised.

operation A processing step of a job,characterised by its required machine and processing time. In practical situations an operation may also include setup time, technological requirements, etc.

order A request from the customer that usually needs to be delivered by a due date. Otherwise, based on the tardiness of the order, a fine will be applied.

Pareto front A set of non-dominated solutions in a multi-objective optimisation problems.

processing time Time that a machine needs to process an operation or a job.

queue The set of jobs waiting to be processed at a specific machine. If a job arrives at a machine to be processed and the machine is busy, the job will placed in the corresponding queue.

release time The time when a job can start being processed (from its first operation).

representation The form in which a solution or heuristic/rule/computer program is represented for computational purposes.

reusability The ability of rules or heuristics to be reused in unseen (new) scenarios.

schedule A detailed plan produced by production managers or computer programs to indicate the start and finish time of each operation and job.

scheduling policies A set of scheduling rules or heuristics to make different scheduling decisions in a manufacturing system.

scheduling Process by which a schedule is generated.

selection In evolutionary computation, selection is a step to decide which individuals to be selected for genetic operators (e.g. crossover, mutation).

sequencing Sequencing is a process to determine the order in which jobs are processed at a certain machine.

shop The place where manufacturing activities take place.

static job shop scheduling Scheduling problems where all information about jobs and machines are available in advance.

swarm intelligence A research area that focus on developing systems that imitate collective intelligence of groups of simple agents.

tardiness The time from the completion of a job until its due date if the job is completed after its due date. If the job completes before its due date, tardiness is zero.

utilisation An indicator to show how busy the shop or machines are during a simulation.

weight (in multi-objective) Importance of a certain objective/criteria.

weight (in tardiness) The penalty applied to tardy jobs or orders (delivered after their due date).

Bibliography

[1] AHMED, I., AND FISHER, W. W. Due date assignment, job order release, and sequencing interaction in job shop scheduling. *Decision Sciences 23*, 3 (1992), 633–647.

[2] AI, T. J., AND KACHITVICHYANUKUL, V. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research 36*, 5 (May 2009), 1693–1702.

[3] ALPAYDIN, E. *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[4] ANDERSON, E.J., G. C., AND POTTS, C. Machine scheduling. In *Local Search in Combinatiorial Optimization* (1997), Aarts and Lenstra, Eds., Wiley, pp. 361–414.

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

[6] ASANO, M., AND OHTA, H. A heuristic for job shop scheduling to minimize total weighted tardiness. *Computers and Industrial Engineering 42* (2002), 137–147.

[7] ATLAN, L., BONNET, J., AND NAILLON, M. Learning distributed reactive strategies by genetic programming for the general job shop

problem. In *Proceedings of the 7th Annual Florida Artificial Intelligence Research Symposium* (1994).

[8] A.VAN VELDHUIZEN, D., AND LAMONT, G. B. Multiobjective evolutionary algorithm research: a history and analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Ohio, 1998.

[9] AZARIA, Y., AND SIPPER, M. GP-gammon: Genetically programming backgammon players. *Genetic Programming and Evolvable Machines 6*, 3 (2005), 283–300.

[10] BADER-EL-DEN, M. B., POLI, R., AND FATIMA, S. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing 1*, 3 (2009), 205–219.

[11] BADRAN, K. M. S., AND ROCKETT, P. I. The roles of diversity preservation and mutation in preventing population collapse in multiobjective genetic programming. In *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (2007), vol. 2, ACM Press, pp. 1551–1558.

[12] BAKER, K. R. Sequencing rules and due-date assignments in a job shop. *Management Science 30* (1984), 1093–1104.

[13] BAKER, K. R., AND TRIETSCH, D. *Principles of Sequencing and Scheduling*. Wiley Publishing, 2009.

[14] BALAS, E., AND VAZACOPOULOS, A. Guided local search with shifting bottleneck for job shop scheduling. *Management Science 44* (1998), 262–275.

[15] BANZHAF, W., NORDIN, P., KELLER, R., AND FRANCONE, F. *Genetic Programming: An Introduction*. Morgan Kaufmann, San Francisco, 1998.

[16] BARLOW, G. J. Design of autonomous navigation controllers for unmanned aerial vehicles using multi-objective genetic programming. Master's thesis, North Carolina State University, 2004.

[17] BAYKASOGLU, A., AND GOCKEN, M. Gene expression programming based due date assignment in a simulated job shop. *Expert Systems with Applications 36* (December 2009), 12143–12150.

[18] BAYKASOGLU, A., AND GOCKEN, M. A simulation based approach to analyse the effects of job release on the performance of a multi-stage job-shop with processing flexibility. *International Journal of Production Research 49*, 2 (2011), 585–610.

[19] BAYKASOGLU, A., GOCKEN, M., AND UNUTMAZ, Z. D. New approaches to due date assignment in job shops. *European Journal of Operational Research 187* (2008), 31–45.

[20] BELL, J. E., AND MCMULLEN, P. R. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics 18*, 1 (2004), 41 – 48.

[21] BENI, G., AND WANG, J. Swarm intelligence in cellular robotic systems. In *Robots and Biological Systems: Towards a New Bionics?*, P. Dario, G. Sandini, and P. Aebischer, Eds., vol. 102 of *NATO ASI Series*. Springer Berlin Heidelberg, 1993, pp. 703–712.

[22] BEYER, H.-G., AND SCHWEFEL, H. P. Evolution strategies - a comprehensive introduction. 3–52.

[23] 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 (To Appear)* (2012).

[24] BIERWIRTH, C., AND MATTFELD, D. C. Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation 7*, 1 (Mar. 1999), 1–17.

[25] BOLTE, A., AND THONEMANN, U. W. Optimizing simulated annealing schedules with genetic programming. *European Journal of Operational Research 92*, 2 (1996), 402–416.

[26] BONABEAU, E., DORIGO, M., AND THERAULAZ, G. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc., New York, NY, USA, 1999.

[27] BRAMEIER, M., AND BANZHAF, W. *Linear Genetic Programming*. Springer, 2007.

[28] BURKE, E., KENDALL, G., NEWALL, J., HART, E., ROSS, P., AND SCHULENBURG, S. Hyper-heuristics: An emerging direction in modern search technology. In *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds., vol. 57 of *International Series in Operations Research & Management Science*. Springer US, 2003, pp. 457–474.

[29] BURKE, E. K., HYDE, M. R., AND KENDALL, G. Evolving bin packing heuristics with genetic programming. In *Parallel Problem Solving from Nature (PPSN)* (2006), pp. 860–869.

[30] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND QU, R. Hyper-heuristics: A survey of the state of the art. Tech. Rep. Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham, 2010.

[31] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. Exploring hyper-heuristic methodologies with genetic programming. In *Computational Intelligence*,

C. Mumford and L. Jain, Eds., vol. 1 of *Intelligent Systems Reference Library*. Springer Berlin Heidelberg, 2009, pp. 177–201.

[32] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (2007), pp. 1559–1565.

[33] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. R. The scalability of evolved online bin packing heuristics. In *CEC'07: IEEE Congress on Evolutionary Computation* (2007), pp. 2530–2537.

[34] BURKE, E. K., KENDALL, G., AND SOUBEIGA, E. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics 9*, 6 (2003), 451–470.

[35] CHANG, F.-C. R. A study of due-date assignment rules with constrained tightness in a dynamic job shop. *Computers & Industrial Engineering 31* (1996), 205–208.

[36] CHEN, X. S., ONG, Y. S., LIM, M. H., AND TAN, K. C. A multi-facet survey on memetic computation. *IEEE Transactions on Evolutionary Computation* (to appear).

[37] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part I: representation. *Computers & Industrial Engineering 30*, 4 (1996), 983–997.

[38] 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 and Industrial Engineering 36*, 2 (1999), 343–364.

[39] CHENG, T. C. E. Integration of priority dispatching and due-date assignment in a job shop. *International Journal of Systems Science 19*, 9 (1988), 1813–1825.

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

[41] CHENG, T. C. E., AND JIANG, J. Job shop scheduling for missed due-date performance. *Computers & Industrial Engineering 34* (1998), 297–307.

[42] CHENG, T. C. E., AND PODOLSKY, S. *Just-in-Time Manufacturing: An Introduction*. Chapman and Hall, London, 1993.

[43] CHIANG, T. C., SHEN, Y. S., AND FU, L. C. A new paradigm for rule-based scheduling in the wafer probe centre. *International Journal of Production Research 46*, 15 (2008), 4111–4133.

[44] COELLO COELLO, C. A., LAMONT, G. B., AND VELDHUIZEN, D. A. V. *Evolutionary Algorithms for Solving Multi-Objective Problems (Genetic and Evolutionary Computation)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[45] CORDON, O., HERRERA-VIEDMA, E., AND LUQUE, M. Evolutionary learning of boolean queries by multiobjective genetic programming. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature* (2002), no. 2439 in Lecture Notes in Computer Science, LNCS, Springer-Verlag, pp. 710–719.

[46] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III* (2000), pp. 176–190.

[47] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference, MIC 2001* (2001), pp. 127–131.

[48] DASGUPTA, D. An overview of artificial immune systems and their applications. In *Artificial Immune Systems and Their Applications*, D. Dasgupta, Ed. Springer Berlin Heidelberg, 1999, pp. 3–21.

[49] DE JONG, E. D., AND POLLACK, J. B. Multi-objective methods for tree size control. *Genetic Programming and Evolvable Machines 4*, 3 (2003), 211–233.

[50] DE JONG, E. D., WATSON, R. A., AND POLLACK, J. B. Reducing bloat and promoting diversity using multi-objective methods. In *GECCO'01: Proceedings of the Genetic and Evolutionary Computation Conference* (2001), Morgan Kaufmann, pp. 11–18.

[51] DEB, K., MOHAN, M., AND MISHRA, S. Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. *Evolutionary Computation 13* (2005), 501–525.

[52] 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.

[53] DEMIRKOL, E., MEHTA, S., AND UZSOY, R. Benchmarks for shop scheduling problems. *European Journal of Operational Research 109*, 1 (1998), 137–141.

[54] 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.

[55] DORIGO, M., AND STÜTZLE, T. *Ant Colony Optimization*. Bradford Company, Scituate, MA, USA, 2004.

[56] DORNDORF, U., AND PESCH, E. Evolution based learning in a job shop scheduling environment. *Computers and Operations Research 22*, 1 (1995), 25–40.

[57] DOWSLAND, K. A., SOUBEIGA, E., AND BURKE, E. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. *European Journal of Operational Research 179*, 3 (2007), 759–774.

[58] EKART, A., AND NEMETH, S. Z. Selection based on the Pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines 2*, 1 (2001), 61–73.

[59] ESSAFI, I., MATI, Y., AND DAUZÈRE-PÉRÈS, S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computer & Operations Research 35*, 8 (2008), 2599–2616.

[60] FOGEL, L. J. *Intelligence through simulated evolution: forty years of evolutionary programming*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[61] FRENCH, S. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Ellis Horwood, 1986.

[62] FRY, T. D., PHILIPOOM, P. R., AND MARKLAND, R. E. Due date assignment in a multistage job shop. *IIE Transactions 21*, 2 (1989), 153–161.

[63] FUKUNAGA, A. Automated discovery of composite SAT variable-selection heuristics. In *Eighteenth National Conference on Artificial Intelligence* (2002), pp. 641–648.

[64] GAGNÉ, C., AND PARIZEAU, M. Co-evolution of nearest neighbor classifiers. *International Journal of Pattern Recognition and Artificial Intelligence 21*, 5 (2007), 921–946.

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

[66] GEE, E. S., AND SMITH, C. H. Selecting allowance policies for improved job shop performance. *International Journal of Production Research 31*, 8 (1993), 1839–1852.

[67] GEIGER, C. D., AND UZSOY, R. Learning effective dispatching rules for batch processor scheduling. *International Journal of Production Research 46* (2008), 1431–1454.

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

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

[70] GLOVER, F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research 13* (May 1986), 533–549.

[71] GLOVER, F., AND LAGUNA, M. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[72] GOH, C. K., AND TAN, K. C. An investigation on noisy environments in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation 11*, 3 (2007), 354–381.

[73] GOH, C. K., AND TAN, K. C. A competitive-cooperative coevolutionary paradigm for dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation 13*, 1 (2009), 103–127.

[74] GONCALVES, J. F., DE MAGALHAES MENDES, J. J., AND RESENDE, M. G. C. A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research 167*, 1 (2005), 77–95.

[75] GRUAU, F. On using syntactic constraints with genetic programming. In *Advances in Genetic Programming 2*, P. J. Angeline and K. E. Kinnear, Jr., Eds. MIT Press, Cambridge, MA, USA, 1996, ch. 19, pp. 377–394.

[76] HANSEN, P., AND MLADENOVIC, N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research 130*, 3 (2001), 449 – 467.

[77] HART, E., AND ROSS, P. A heuristic combination method for solving job-shop scheduling problems. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature* (1998), PPSN V, pp. 845–854.

[78] HILDEBRANDT, T., HEGER, J., AND SCHOLZ-REITER, B. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation* (New York, USA, 2010), pp. 257–264.

[79] HOAI, N. X., MCKAY, R. I., AND ABBASS, H. A. Tree adjoining grammars, language bias, and genetic programming. In *Genetic Programming, Proceedings of EuroGP'2003* (Essex, 2003), vol. 2610 of *LNCS*, Springer-Verlag, pp. 335–344.

[80] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA, 1992.

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

[82] HOPP, W. J., AND SPEARMAN, M. L. *Factory Physics: Foundations of Manufacturing Management*. McGraw-Hill, 2000.

[83] HORNBY, G. S., AND POLLACK, J. B. Body-brain co-evolution using L-systems as a generative encoding. In *GECCO'01: Proceedings of the Genetic and Evolutionary Computation Conference* (2001), Morgan Kaufmann, pp. 868–875.

[84] INGIMUNDARDOTTIR, H., AND RUNARSSON, T. Supervised learning linear priority dispatch rules for job-shop scheduling. In *Learning and Intelligent OptimizatioN (LION 5)* (2011), pp. 263–277.

[85] ISHIBUCHI, H., YOSHIDA, T., AND MURATA, T. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation 7* (2002), 204–223.

[86] JAKOBOVIC, D., AND BUDIN, L. Dynamic scheduling with genetic programming. In *EuroGP'06: Proceedings of the 9th European Conference on Genetic Programming* (2006), pp. 73–84.

[87] JAKOBOVIC, D., JELENKOVIC, L., AND BUDIN, L. Genetic programming heuristics for multiple machine scheduling. In *EuroGP'07: Proceedings of the 10th European Conference on Genetic programming* (2007), pp. 321–330.

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

[89] 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.

[90] JIN, N. Equilibrium selection by co-evolution for bargaining problems under incomplete information about time preferences. In *CEC'05: Proceedings of the 2005 IEEE Congress on Evolutionary Computation* (2005), pp. 2661–2668.

[91] JOHN, D. J. Co-evolution with the Bierwirth-Mattfeld hybrid scheduler. In *GECCO '02: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation* (2002), p. 259.

[92] JONES, A., AND RABELO, L. C. Survey of job shop scheduling techniques. Tech. rep., NISTIR, National Institute of Standards and Technology, Gaithersburg, USA, 1998.

[93] JONG, K. A. D. *Evolutionary computation - a unified approach*. MIT Press, 2006.

[94] JOSEPH, O., AND SRIDHARAN, R. Analysis of dynamic due-date assignment models in a flexible manufacturing system. *Journal of Manufacturing Systems 30*, 1 (2011), 28–40.

[95] KACEM, I., HAMMADI, S., AND BORNE, P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation 60*, 3-5 (2002), 245–276.

[96] KARABOGA, D., AND BASTURK, B. On the performance of artificial bee colony (abc) algorithm. *Applied Soft Computing 8*, 1 (2008), 687 – 697.

[97] KELLER, R., AND POLI, R. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Proceedings of the 8th International Conference on Artifcial Evolution* (2007), pp. 13–24.

[98] KELLER, R., AND POLI, R. Linear genetic programming of parsimonious metaheuristics. In *CEC'07: IEEE Congress on Evolutionary Computation* (2007), pp. 4508–4515.

[99] KENNEDY, J., AND EBERHART, R. C. *Swarm intelligence*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[100] KIM, J., AND MILLER, J. Building the value factory: a progress report for U.S. manufacturing. Tech. rep., Boston University Manufacturing Roundtable, 1992.

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

[102] KNOWLES, J. D., AND CORNE, D. W. Approximating the nondominated front using the pareto archived evolution strategy. *Evolutionary Computation 8* (2000), 149–172.

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

[104] KRASNOGOR, N., AND SMITH., J. Emergence of profitable search strategies based on a simple inheritance mechanism. In *GECCO '01: Proceedings of the Genetic and Evolutionary Computation Conference* (2001), pp. 432–439.

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

[106] LAM N.V., KACHITVICHYANUKUL, V., AND LUONG. A multistage parallel genetic algorithm for multi-objective job shop scheduling. In *The 6th Asia Pacific Industrial Engineering and Management Systems Conference (APIEMS 2005), Manila, Philippines* (2005).

[107] LAND, M., AND GAALMAN, G. Workload control concepts in job shops a critical assessment. *International Journal of Production Economics 46*, 1 (December 1996), 535–548.

[108] LAND, M. J. *Workload Control in Job Shops, Grasping the Tap*. PhD thesis, University of Groningen, The Netherlands, 2004.

[109] LANGDON, W. B. Size fair and homologous tree genetic programming crossovers. *Genetic Programming and Evolvable Machines 1*, 1/2 (2000), 95–119.

[110] 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*, S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, Eds., vol. 4 of *Handbooks in Operations Research and Management Science*. Elsevier, 1993, ch. 9, pp. 445–522.

[111] LAWRENCE, S. *Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques*. PhD thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

[112] LI, X., AND OLAFSSON, S. Discovering dispatching rules using data mining. *Journal of Scheduling 8* (2005), 515–527.

[113] LICHODZIJEWSKI, P., AND HEYWOOD, M. I. Coevolutionary bid-based genetic programming for problem decomposition in classification. *Genetic Programming and Evolvable Machines 9* (2008), 331–365.

[114] LITTLE, J. D. C. A proof for the queuing formula: $L = \lambda W$. *Operations Research 9*, 3 (1961), 383–387.

[115] LOURENCO, H. R. Job-shop scheduling: Computational study of local search and large-step optimization methods. *European Journal of Operational Research 83*, 2 (1995), 347–364.

[116] LU, H., HUANG, G. Q., AND YANG, H. Integrating order review/release and dispatching rules for assembly job shop scheduling using a simulation approach. *International Journal of Production Research 49*, 3 (2011), 647–669.

[117] LUKE, S. *Essentials of Metaheuristics*. Lulu, 2009.

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

[119] MCKAY, R. I., HOAI, N. X., WHIGHAM, P. A., SHAN, Y., AND O'NEILL, M. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines 11*, 3-4 (Sept. 2010), 365–396.

[120] MENDES, R. R. F., VOZNIKA, F. D. B., FREITAS, A. A., AND NIEVOLA, J. C. Discovering fuzzy classification rules with genetic programming and co-evolution. In *PKDD '01: Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery* (London, UK, 2001), Springer-Verlag, pp. 314–325.

[121] MICHALEWICZ, Z., AND FOGEL, D. B. *How to Solve It: Modern Heuristics*. Springer, 2004.

[122] MILLER, J. F., AND HARDING, S. L. Cartesian genetic programming. In *GECCO'08: Proceedings of Genetic and Evolutionary Computation Conference* (New York, NY, USA, 2008), GECCO '08, ACM, pp. 2701–2726.

[123] MIYASHITA, K. Job-shop scheduling with GP. In *GECCO'00: Proceedings of the Genetic and Evolutionary Computation Conference* (2000), pp. 505–512.

[124] MIYAZAKI, S. Combined scheduling system for reducing job tardiness in a job shop. *International Journal of Production Research 19*, 2 (1981), 201–211.

[125] MIZRAK, P., AND BAYHAN, G. M. Comparative study of dispatching rules in a real-life job shop environment. *Applied Artificial Intelligence 20* (2006), 585–607.

[126] MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. *Foundations of Machine Learning*. The MIT Press, 2012.

[127] MONTANA, D. J. Strongly typed genetic programming. *Evolutionary Computation 3*, 2 (1995), 199–230.

[128] MONTGOMERY, D. C. *Design and Analysis of Experiments*. John Wiley & Sons, 2001.

[129] MOREIRA, M., AND ALVES, R. A methodology for planning and controlling workload in a job-shop: a four-way decision-making problem. *International Journal of Production Research 47*, 10 (2009), 2805–2821.

[130] NGUYEN, S., ZHANG, M., AND JOHNSTON, M. A genetic programming based hyper-heuristic approach for combinatorial optimisation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation* (New York, NY, USA, 2011), GECCO '11, ACM, pp. 1299–1306.

[131] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. *Automated Scheduling*. Studies in Computational Intelligence. Springer, ch. Dynamic Job Shop Scheduling Problems: A MO-GPHH Approach. (accepted).

[132] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Genetic programming for evolving reusable due-date assignment models in job shop environments. *Evolutionary Computation*. (accepted).

[133] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Automatic design of scheduling policies for dynamic multi-objective

job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation* (2012). DOI:10.1109/TEVC.2013.2248159.

[134] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In *CEC '12: Proceedings of the IEEE Congress on Evolutionary Computation* (2012), pp. 3261–3268.

[135] 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* (2012). DOI:10.1109/TEVC.2012.2227326.

[136] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming. In *EuroGP'12: Proceedings of the 15th European Conference on Genetic Programming* (2012), pp. 121–133.

[137] 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* (2013). DOI:10.1007/s00170-013-4756-9.

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

[139] NOWICKI, E., AND SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. *Management Science 42* (1996), 797–813.

[140] OCHOA, G., HYDE, M., CURTOIS, T., VAZQUEZ-RODRIGUEZ, J., WALKER, J., GENDREAU, M., KENDALL, G., MCCOLLUM, B., PARKES, A., PETROVIC, S., AND BURKE, E. Hyflex: A benchmark framework for cross-domain heuristic search. In *European Conference on Evolutionary Computation in Combinatorial Optimisation(EvoCOP 2012)* (2012), pp. 136–147.

[141] O'NEILL, M., AND RYAN, C. *Grammatical Evolution: Evolutionary Automatic Programming in a Arbitrary Language*. Kluwer Academic Publishers, 2003.

[142] ONG, Y. S., AND KEANE, A. Meta-Lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation 8*, 2 (2004), 99–110.

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

[144] OZTURK, A., KAYALIGIL, S., AND OZDEMIREL, N. E. Manufacturing lead time estimation using data mining. *European Journal of Operational Research 173*, 2 (2006), 683–700.

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

[146] PARDALOS, P., AND SHYLO, O. An algorithm for the job shop scheduling problem based on global equilibrium search techniques. *Computational Management Science 3* (2006), 331–348.

[147] PATIL, R. J. Using ensemble and metaheuristics learning principles with artificial neural networks to improve due date prediction performance. *International Journal of Production Research 46*, 21 (2008), 6009–6027.

[148] PETROVIC, S., AND CASTRO, E. A genetic algorithm for radiotherapy pre-treatment scheduling. In *Applications of Evolutionary Computation*. 2011, pp. 454–463.

[149] PETROVIC, S., FAYAD, C., PETROVIC, D., BURKE, E., AND KENDALL, G. Fuzzy job shop scheduling with lot-sizing. *Annals of Operations Research 159* (2008), 275–292.

[150] PFUND, M., MASON, S., AND FOWLER, J. Semiconductor manufacturing scheduling and dispatching. In *Handbook of Production Scheduling*, J. Herrmann, Ed., vol. 89 of *International Series in Operations Research & Management Science*. Springer US, 2006, pp. 213–241.

[151] PHILIPOOM, P. R., REES, L. P., AND WIEGMANN, L. Using neural networks to determine internally-set due-date assignments for shop scheduling. *Decision Sciences 25*, 5–6 (1994), 825–851.

[152] PINEDO, M., AND SINGER, M. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics 46*, 1 (1999), 1–17.

[153] PINEDO, M. L. *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer, 2008.

[154] POLI, R., LANGDON, W. B., AND MCPHEE, N. F. *A Field Guide to Genetic Programming*. Published via `http://lulu.com` and freely available at `http://www.gp-field-guide.org.uk`, 2008.

[155] PONNAMBALAM, S. G., RAMKUMAR, V., AND JAWAHAR, N. A multiobjective genetic algorithm for job shop scheduling. *Production Planning and Control 12*, 8 (2001).

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

[157] POTTER, M. A., AND JONG, K. A. D. A cooperative coevolutionary approach to function optimization. In *Proceedings of the 3rd International Conference on Parallel Problem Solving from Nature* (1994), Springer-Verlag, pp. 249–257.

[158] POTTS, C. N., AND STRUSEVICH, V. A. Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society 60*, Supplement 1 (2009), 41–68.

[159] RAFTER, J. A., ABELL, M. L., AND BRASELTON, J. P. Multiple comparison methods for means. *SIAM Review 44*, 2 (2002), 259–278.

[160] RAGATZ, G. L., AND MABERT, V. A. A simulation analysis of due date assignment rules. *Journal of Operations Management 5*, 1 (1984), 27–39.

[161] RAGATZ, G. L., AND MABERT, V. A. An evaluation of order release mechanisms in a job-shop evironment. *Decision Sciences 19*, 1 (1988), 167–189.

[162] RAJENDRAN, C., AND HOLTHAUS, O. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research 116*, 1 (1999), 156–170.

[163] RAMASESH, R. Dynamic job shop scheduling: a survey of simulation research. *Omega 18*, 1 (1990), 43–57.

[164] RAMUDHIN, A., AND MARIER, P. The generalized shifting bottleneck procedure. *European Journal of Operational Research 93*, 1 (1996), 34 – 48.

[165] RIEZEBOS, J., HOC, J. M., MEBARKI, N., DIMOPOULOS, C., WEZEL, W., AND PINOT, G. Design of scheduling algorithms: Applications. In *Behavioral Operations in Planning and Scheduling*, J. C. Fransoo, T. Waefler, and J. R. Wilson, Eds. 2011, pp. 371–412.

[166] ROHLEDER, T. R., AND SCUDDER, G. A comparison of order-release and dispatching rules for the dynamic weighted early/tardy problem. *Production and Operations Management 2*, 3 (1993), 221–238.

[167] RUSSELL, S. J., NORVIG, P., CANDY, J. F., MALIK, J. M., AND EDWARDS, D. D. *Artificial intelligence: a modern approach*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[168] SABUNCUOGLU, I., AND COMLEKCI, A. Operation-based flowtime estimation in a dynamic job shop. *Omega 30*, 6 (2002), 423–442.

[169] SARIN, S. C., VARADARAJAN, A., AND WANG, L. A survey of dispatching rules for operational control in wafer fabrication. *Production Planning & Control 22*, 1 (2011), 4–24.

[170] SCHWEFEL, H.-P. *Numerical Optimization of Computer Models*. John Wiley & Sons, Inc., New York, NY, USA, 1981.

[171] SELS, V., GHEYSEN, N., AND VANHOUCKE, M. A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research* (2011).

[172] SERAFIN, M. J., TSANG, E. P. K., AND MARKOSE, S. Coevolution of genetic programming based agents in an artificial stock market. Computing in Economics and Finance 2006 398, Society for Computational Economics, July 2006.

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

[174] SHA, D., AND HSU, S. Due-date assignment in wafer fabrication using artificial neural networks. *The International Journal of Advanced Manufacturing Technology 23* (2004), 768–775.

[175] SHA, D., AND LIU, C.-H. Using data mining for due date assignment in a dynamic job shop environment. *The International Journal of Advanced Manufacturing Technology 25* (2005), 1164–1174.

[176] SHA, D. Y., STORCH, R. L., AND LIU, C. H. Development of a regression-based method with case-based tuning to solve the due date assignment problem. *International Journal of Production Research 45*, 1 (2007), 65–82.

[177] SLOAN, T. Shop-floor scheduling of semiconductor wafer fabs: exploring the influence of technology, market, and performance objectives. *IEEE Transactions on Semiconductor Manufacturing 16*, 2 (2003), 281–289.

[178] STORER, R. H., WU, S. D., AND VACCARI, R. New search spaces for sequencing problems with application to job shop scheduling. *Management Science 38*, 10 (1992), 1495–1509.

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

[180] TAN, K. C., YANG, Y. J., AND GOH, C. K. A distributed cooperative coevolutionary algorithm for multiobjective optimization. *IEEE Transactions on Evolutionary Computation 10*, 5 (2006), 527–549.

[181] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computer & Industrial Engineering 54* (2008), 453–473.

[182] 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.

[183] VAN VELDHUIZEN, D. A., AND LAMONT, G. B. Multiobjective evolutionary algorithm test suites. In *Proceedings of the 1999 ACM Symposium on Applied Computing (SAC'99)* (1999), pp. 351–357.

[184] VÁZQUEZ-RODRÍGUEZ, J. A., AND PETROVIC, S. A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics 16*, 6 (2010), 771–793.

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

[186] VINOD, V., AND SRIDHARAN, R. Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system. *International Journal of Production Economics 129*, 1 (2011), 127–146.

[187] WANG, Z., TANG, K., AND YAO, X. Multi-objective approaches to optimal testing resource allocation in modular software systems. *IEEE Transactions on Reliability 59*, 3 (2010), 563–575.

[188] XIA, W., AND WU, Z. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering 48*, 2 (2005), 409–425.

[189] XING, L.-N., CHEN, Y.-W., WANG, P., ZHAO, Q.-S., AND XIONG, J. A knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing 10*, 3 (2010), 888 – 896.

[190] YAMADA, T., AND NAKANO, R. A genetic algorithm with multi-step crossover for job-shop scheduling problems. In *GALESIA: First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications* (1995), pp. 146–151.

[191] YIN, W. J., LIU, M., AND WU, C. Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In *CEC'03: IEEE Congress on Evolutionary Computation* (2003), pp. 1050–1055.

[192] ZHANG, B.-T., AND CHO, D.-Y. Coevolutionary fitness switching: Learning complex collective behaviors using genetic programming. In *Advances in Genetic Programming 3*, L. Spector, W. B. Langdon, U.-M. O'Reilly, and P. J. Angeline, Eds. MIT Press, 1999, ch. 18, pp. 425–445.

[193] ZHOU, A., JIN, Y., ZHANG, Q., SENDHOFF, B., AND TSANG, E. Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. In *CEC'06: IEEE Congress on Evolutionary Computation* (2006), pp. 892–899.

[194] 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.

[195] ZITZLER, E., LAUMANNS, M., AND THIELE, L. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001)* (2002), pp. 95–100.

[196] ZITZLER, E., AND THIELE, L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation 3*, 4 (1999), 257–271.

[197] ZITZLER, E., THIELE, L., LAUMANNS, M., FONSECA, C., AND DA FONSECA, V. Performance assessment of multiobjective optimizers: an analysis and review. *CEC'03: IEEE Transactions on Evolutionary Computation 7*, 2 (2003), 117–132.