Genetic Programming Hyper-heuristics for Dynamic Flexible Job Shop Scheduling

by

Fangfang Zhang

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
2021

Abstract

Dynamic flexible job shop scheduling (DFJSS) has received widespread attention from academia and industry due to its reflection in real-world scheduling applications such as order picking in the warehouse and the manufacturing industry. It requires complex routing and sequencing decisions under unpredicted dynamic events. Genetic programming, as a hyper-heuristic approach (GPHH), has been successfully applied to evolve scheduling heuristics for DFJSS automatically due to its flexible representation. Although GPHH has achieved certain success in solving the DFJSS problems, there are still some limitations for applying GPHH to DFJSS, particularly in terms of its training efficiency, large search space, search mechanism, and multitask solving ability.

The overall goal of this thesis is to develop effective GPHH algorithms to evolve scheduling heuristics for DFJSS efficiently. Different machine learning techniques, i.e., surrogate, feature selection, specialised genetic operator, and multitask learning, are incorporated in this thesis to tackle the limitations.

First, this thesis develops a novel multi-fidelity based surrogate-assisted GPHH for DFJSS to improve the training efficiency of GPHH. Specifically, multi-fidelity based surrogate models are first designed by simplifying the problem to be solved. Then, an effective collaboration mechanism with knowledge transfer is proposed for utilising the advantages of the multi-fidelity based surrogate models to solve the problem. The results show that the proposed algorithm can dramatically reduce the computational cost of GPHH without sacrificing the performance in all the test scenarios. With the same training time, the proposed algorithm can achieve signifi-

cantly better performance than its counterparts in most scenarios while no worse in others.

Second, this thesis designs a novel two-stage GPHH framework with feature selection to evolve scheduling heuristics for DFJSS automatically. Based on this framework, this thesis further proposes to evolve scheduling heuristics with only the selected features by eliminating the unselected features properly. Specifically, individual adaptation strategies are proposed to generate individuals with only the selected features by utilising the information of both the selected features and the investigated individuals during the feature selection process. The results show that the proposed algorithm can successfully achieve scheduling heuristics with fewer unique features and smaller sizes, which tends to be more interpretable. In addition, the proposed algorithm can evolve comparable scheduling heuristic with that obtained by the traditional GPHH within a much shorter training time.

Third, this thesis proposes a novel recombinative mechanism to provide guidance for GPHH based on the importance of subtrees to realise effective and adaptive recombination for parents to produce offspring. Two measures are proposed to measure the importance of all the subtrees of an individual. The first one is based on the frequency of features, and the second is based on the correlation between the behaviour of subtrees and the whole tree (i.e., an individual). The importance information is utilised to decide the crossover points for the parents. The proposed recombinative guidance mechanism attempts to improve the quality of offspring by preserving the promising building-blocks of one parent and incorporating good building-blocks from the other. The results show that the proposed algorithm based on the correlation importance measure performs better than the proposed algorithm based on the feature frequency importance measure. In addition, the proposed algorithm based on the correlation importance measure between the behaviour of subtrees significantly also outperforms the state-of-the-art algorithms on most tested scenarios.

Last, this thesis proposes a multitask GPHH approach and a surrogate-assisted multitask GPHH approach to solving multiple DFJSS tasks simultaneously. First, an effective hyper-heuristic multitask algorithm is proposed by adapting the traditional evolutionary multitask algorithms based on the characteristics of GPHH. Second, this thesis develops a novel surrogate-assisted multitask GPHH approach to solving multiple DFJSS tasks by sharing useful knowledge between different DFJSS scheduling tasks. Specifically, the surrogate-assisted multitask GPHH algorithm employs the phenotypic characterisation technique to measure the behaviours of scheduling rules to build a surrogate for each task accordingly. The built surrogates are not only used to improve the efficiency of solving each single DFJSS task but also utilised for knowledge sharing between multiple DFJSS tasks in multitask learning. The results show that the proposed algorithm can significantly improve the quality of scheduling heuristics for all the test scenarios. The results also observe that the proposed algorithms manage to solve multiple tasks collaboratively in terms of the evolved scheduling heuristics for different tasks in a multitask scenario.

List of Publications

[1] **Fangfang Zhang**, Yi Mei, Su Nguyen, and Mengjie Zhang. "Correlation Coefficient based Recombinative Guidance for Genetic Programming Hyper-heuristics in Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552-566, 2021.

[2] **Fangfang Zhang**, Yi Mei, Su Nguyen, Mengjie Zhang, and Kay Chen Tan. "Surrogate-Assisted Evolutionary Multitask Genetic Programming for Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651-665, 2021.

[3] **Fangfang Zhang**, Yi Mei, Su Nguyen, and Mengjie Zhang. "Evolving Scheduling Heuristics via Genetic Programming with Feature Selection in Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 1797-1811, 2021.

[4] **Fangfang Zhang**, Yi Mei, Su Nguyen, and Mengjie Zhang. "Collaborative Multifidelity-Based Surrogate Models for Genetic Programming in Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Cybernetics*, 2021, pp. 1-15. (Doi: 10.1109/TCYB.2021.3050141)

[5] **Fangfang Zhang**, Yi Mei, Su Nguyen, Kay Chen Tan, and Mengjie Zhang. "Multitask Genetic Programming Based Generative Hyper-heuristics: A Case Study in Dynamic Scheduling". Submitted to *IEEE Transactions on Cybernetics*, Doi: 10.1109/TCYB.2021.3065340.

[6] **Fangfang Zhang**, Yi Mei, and Mengjie Zhang. "A Two-stage Genetic Programming Hyper-heuristic Approach with Feature Selection for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, 2019, pp. 347-355.

[7] **Fangfang Zhang**, Yi Mei, Su Nguyen, and Mengjie Zhang. "A Preliminary Approach to Evolutionary Multitasking for Dynamic Flexible Job Shop Scheduling via Genetic Programming". *in Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, 2020, pp. 107-108.

[8] **Fangfang Zhang**, Yi Mei, Su Nguyen, and Mengjie Zhang. "Guided Subtree Selection for Genetic Operators in Genetic Programming for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the European Conference on Genetic Programming*, Springer, 2020, pp. 262-278.

[9] **Fangfang Zhang**, Yi Mei, Su Nguyen, and Mengjie Zhang. "Genetic Programming with Adaptive Search Based on the Frequency of Features for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimisation*, Springer, 2020, pp. 214-230.

[10] **Fangfang Zhang**, Yi Mei, and Mengjie Zhang. "Can Stochastic Dispatching Rules Evolved by Genetic Programming Hyperheuristics Help in Dynamic Flexible Job Shop Scheduling?". *in Proceedings of the Congress on Evolutionary Computation*, IEEE, 2019, pp. 41-48.

[11] **Fangfang Zhang**, Yi Mei, and Mengjie Zhang. "Evolving Dispatching Rules for Multi-objective Dynamic Flexible Job Shop Scheduling via Genetic Programming Hyper-heuristics". *in Proceedings of the Congress on Evolutionary Computation*, IEEE, 2019, pp. 1343-1350.

[12] **Fangfang Zhang**, Yi Mei, and Mengjie Zhang. "A New Representation in Genetic Programming for Evolving Dispatching Rules for Dy-

namic Flexible Job Shop Scheduling". *in Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization*, IEEE, 2019, pp. 33-49.

[13] **Fangfang Zhang**, Yi Mei, and Mengjie Zhang. "Genetic Programming with Multi-tree Representation for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the Australasian Joint Conference on Artificial Intelligence*, Springer, 2018, pp. 472-484. (**Best Paper Runner-Up Award**)

[14] **Fangfang Zhang**, Yi Mei, and Mengjie Zhang. "Surrogate-Assisted Genetic Programming for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the Australasian Joint Conference on Artificial Intelligence*, Springer, 2018, pp. 766-772.

[15] **Fangfang Zhang**, Yi Mei, Su Nguyen, Kay Chen Tan and Mengjie Zhang. "Adaptive Multitask Genetic Programming for Dynamic Job Shop Scheduling". Submitted to *IEEE Transactions on Evolutionary Computation* (Under Review)

[16] **Fangfang Zhang**, Su Nguyen, Yi Mei and Mengjie Zhang. "Genetic Programming for Production Scheduling: An Evolutionary Learning Approach". Book draft has been submitted, June 2021.

Acknowledgments

I would like to express my very appreciation to my supervisors, Dr. Yi Mei, Prof. Mengjie Zhang, Dr. Su Nguyen and Prof. Kay Chen Tan, for their guidance and support during my PhD study. Dr. Mei has spent lots of hours for my research and provided very useful feedback to improve my research skills. Prof. Zhang is always positive, and has encouraged me a lot to achieve my research goals. Dr. Nguyen is very nice to talk to, and the discussions with him are quite useful and enjoyable. Prof. Tan has given constructive comments for my studies, despite the time zone. In addition, I would like to thank Prof. Bing Xue for guiding me to this excellent research group.

I am grateful to the China Scholarship Council / Victoria University of Wellington Scholarship for their financial support for my PhD study over the past more than three years. Thanks also go to the Marsden Fund of New Zealand Government under Contract VUW 1509 for support in part of my PhD study, conference attendance, and other aspects.

I would like to thank my friends in the Evolutionary Computation Research Group for the inspiring research environment. Many thanks to my friend Ke Chen for sharing delicious food and supporting my research. Thank you to Bach Hoai Nguyen and Qurrat Ul Ain for their kind help, especially the hard time when I just started my PhD. Thanks Mazhar Ansari Ardeh, Yanan Sun, Andrew Lensen, Qi Chen, Binzi Xu, Jiabin Lin, Baolei Li, Mahdi Abdollahi, Boxiong Tan, Trung Nguyen, Tao Shi, Yahui Jia, Ying Bi, Shaolin Wang, Peng Wang, Joao Costa, Kosisochukwu Madukwe, Ba-

ligh Al-Helali, Harisu Abdullahi Shehu and so many unlisted for their jokes and discussions. Many thanks to Jing Li, Baobao Wang, Xiaohan Bai, Pengfei Liu, Zhaojun Ding, Yuye Zhang and Kun Huang for your warm friendship.

Last but not least, I wish to thank my parents, my younger brother, and sister in law for great support and understanding. You have always been the source of love that helps me complete this journey.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter begins by introducing job shop scheduling with a focus on dynamic flexible job shop scheduling. Then, the existing approaches for solving the job shop scheduling problems, including exact approaches, heuristic approaches, and hyper-heuristic approaches, especially genetic programming hyper-heuristics are discussed. Details of the motivations and research goals of this thesis are also presented. Last, the major contributions are provided, followed by the organisation of this thesis.

1.1 Job Shop Scheduling

Job shop scheduling (JSS) [150] is an important combinatorial optimisation problem, which captures practical issues in real-world applications such as grid/cloud computing [14], order picking in the warehouse [111], and manufacturing industry [80, 219]. JSS has received widespread attention in both academia and industry due to its practical applications [137, 242, 251]. The job shop contains a number of jobs need to be processed by a set of machines. The goal of JSS is to optimise the use of machine resources to achieve production efficiency such as minimising makespan [141] to reduce the total processing time and tardiness [200] to reduce the production delays.

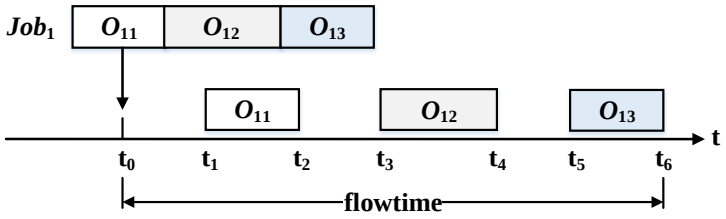

Figure 1.1: An example of the flowtime of a job (Job_1).

Depending on whether the information of jobs is known in advance or not, JSS can be classified as *static (classical) JSS* or *dynamic JSS* [175]. Depending on whether a job can be processed on more than one machine, JSS can be categorised into *flexible JSS* and *non-flexible JSS* [242].

Static (Classical) Job Shop Scheduling: In the typical version of JSS [150], a number of jobs need to be processed on a set of machines. Each job has a sequence of operations. The operations of a job need to be executed in a predefined order, and each operation can *only* be processed at a specified machine. In addition, information about jobs is available when making a schedule. The flowtime of a job is the duration from the job arrives at the shop floor to its completion time. An example of the flowtime of Job_1 with three operations can be found in Figure 1.1. We can see that Job_1 arrives at time t_0, and the starting processing time of its first operation O_{11} is t_1. The last operation O_{13} of Job_1 is finished at time t_6. Therefore, the flowtime of Job_1 is $t_6 - t_0$.

Dynamic Job Shop Scheduling: Static JSS implies that the information of jobs is known when we make a schedule for the production in a job shop. However, in practice, the *environment* of a job shop floor is usually dynamic, e.g., new jobs can arrive dynamically [70, 239, 240], and the machine may break down unexpectedly [187, 229]. Dynamic JSS [159, 175] is used for considering the scheduling situations with dynamic events. This thesis considers the dynamic events that job can arrive over time, since it is the most common dynamic factor in a job shop for real-world applications [102]. For example, it is quite common for a company to receive

orders from customers dynamically. However, it is not easy for a company to predict the information of orders before the orders arrive. This means that in dynamic JSS, information about jobs is not known until they arrive at the job shop floor.

Flexible Job Shop Scheduling: Flexible JSS [23, 74] is an extension of the classical JSS in which *each operation can be processed by a set of candidate machines* rather than a specific machine. Each operation will be processed on one of its candidate machines, and its processing time depends on the machine that processes it. There are two decisions that need to be made simultaneously in flexible JSS. One is *machine assignment* for allocating a ready operation to a machine, and the other is *operation sequencing* for choosing an operation to be processed next when a machine is idle and there are operations in its queue. Given a number of jobs and a set of machines, flexible JSS aims to determine which machine to process a particular job and which job will be chosen to be processed next by a particular machine.

Dynamic Flexible Job Shop Scheduling: Dynamic flexible JSS (DFJSS) [244] needs to do machine assignment and operation sequencing simultaneously under a dynamic environment with unpredicted events such as new jobs arriving dynamically [230, 249] and machine breakdown unexpectedly [229]. DFJSS is more challenging than static (classical) JSS, dynamic JSS and flexible JSS, since DFJSS involves more than one decision and dynamic environment compared with other variations of JSS.

In general, dynamic JSS and flexible JSS are variants of classical JSS. Furthermore, DFJSS is a combination of dynamic JSS and flexible JSS. The details about the availability of job information and the required decisions of different types of JSS are shown in Table 1.1. Table 1.1 shows that DFJSS is the most complex one among the four types of JSS with two decisions and unknown job information in advance.

Table 1.1: The availability of job information and decision requirement in different types of JSS.

	Problem	Classical JSS	Dynamic JSS	Flexible JSS	DFJSS
Job Information	known	✓		✓	
	unknown		✓		✓
Decision	routing			✓	✓
	sequencing	✓	✓	✓	✓

1.2 Existing Approaches

Over the years, a number of approaches, including exact approaches, heuristic approaches, and hyper-heuristic approaches along with GPHH, have been adapted for solving the JSS problems. Different types of these approaches are discussed as follows.

Exact Approaches: Many techniques search for optimal solutions, which are known as exact approaches such as dynamic programming [37], branch-and-bound [135] and integer linear programming [213], have been investigated for static JSS [32, 34, 37]. However, DFJSS is an NP-hard problem [216] and cannot be handled efficiently with exact optimisation approaches. Normally, exact approaches are only limited to solve *small scale static JSS* problems. Exact approaches are too time-consuming when the problems are getting large. It is also hard for exact approaches to handle dynamic problems where a lot of real-time decisions are needed to be made quickly.

Heuristic Approaches: A heuristic approach is designed for solving a problem more quickly when the exact approaches are too slow, or for finding an approximate solution when the exact approaches fail to find the optimal solution [192]. A heuristic approach aims to produce a good enough solution for the problem at hand in a reasonable time. There are two different types of heuristics, i.e., *improvement heuristic* and *construction heuristic*, depending on how they build solutions. An improvement

heuristic starts with a complete initial solution, and improves the solution iteratively until a stopping criterion is met [224]. A construction heuristic starts with an empty solution and repeatedly updates the partial solutions until a complete solution is obtained [127].

Improvement heuristics such as bee colony algorithm [48, 140], tabu search [180], simulated annealing [225], and genetic algorithms [46, 196] have been proposed to find "good enough" solutions for solving JSS problems [3, 83, 193, 221] in a reasonable time. However, they are often not suitable for solving the dynamic JSS problems due to their lack of ability to react in time (i.e., face with rescheduling issue, which is time-consuming). Construction heuristics such as dispatching rules, have been popularly used for dynamic JSS due to its efficiency to make real-time decisions (i.e., dispatching rules are used as priority functions to prioritise jobs and machines at the decision points). Comprehensive comparison among a large number of dispatching rules can be found in [212]. It is noted that a scheduling heuristic in DFJSS consists of a routing rule (i.e., for machine assignment) and a sequencing rule (i.e., for operation sequencing) [242, 244]. Scheduling heuristics make decisions according to the *priority values* of machines or operations *only at the decision points*. This leads to two main reasons for the success of scheduling heuristics in DFJSS [233, 244]. One is its ability to handle large scale problems efficiently. The other is its efficiency to make real-time decisions with dynamic events. However, the scheduling heuristics, such as SPT (i.e., shortest processing time) and some composite rules [114], are often manually designed by experts [47, 65, 114, 115, 226]. The designing process is time-consuming, and the designed rules are typically too specific to be applied to different scenarios [18, 82, 122].

Hyper-Heuristic Approaches: A hyper-heuristic [21, 25, 27] is an automated methodology for selecting or generating heuristics to solve hard computational search problems. The unique characteristic is that hyper-heuristic approaches work on the heuristic space instead of the solution

space. In other words, a hyper-heuristic approach outputs a heuristic rather than a solution.

There are two types of hyper-heuristic approaches [26]. One is *heuristic selection* which aims to choose existing heuristics for different problem scenarios. The goal of a heuristic selection approach is to find a sequence of heuristics. The other is *heuristic generation* which aims to generate new high-level heuristics using existing low-level heuristics. The goal of a heuristic generation approach is to generate an informative high-level heuristic rather than finding the solution directly [26]. For DFJSS, heuristic generation is commonly used to generate high-level scheduling heuristics with low-level heuristics. Hyper-heuristic approaches have two main advantages. First, the learned high-level heuristics have good reusability, and can easily be applied to a range of problem scenarios. Second, generative hyper-heuristic approaches can handle the dynamic problems efficiently, because the evolved heuristics are typically used as priority functions and can make real-time decisions. A recent state-of-the-art survey of existing studies on hyper-heuristics and its applications can be found in [21].

Genetic Programming Hyper-heuristic: Genetic programming (GP) [131] is a genetic-based machine learning [85, 119] algorithm with many successful applications in regression [42], classification [39, 109], and computer vision [161]. To solve a task, GP initializes and maintains a population of programs which compete for survival and reproduction based on how well they solve the task, i.e., their fitness. Variable-length representations such as trees or graphs are used to compose the programs, and genetic operators are applied to produce child programs. By evolving its population through many generations, GP can explore the program search space to discover the program that performs best for a given task. The flexible representations and search mechanisms make GP a suitable approach to designing scheduling heuristics, since the structure and size of optimal heuristics are not known in advance. A general GP based hyper-heuristic

(GPHH) framework was presented in [27], and it has been successfully used in different problems such as packing [28, 107], timetabling [8, 198], arc routing [5, 227] and JSS [106, 169, 170, 172, 241]. In the last decade, GPHH works as heuristic generation approach, has been the dominating technique to evolve scheduling heuristics for DFJSS automatically. The heuristics obtained by GP can be interpreted due to its function characteristic, which is a very important for real-world applications.

Summary: *Exact optimisation approaches* are usually not applicable to dynamic and large scale problems. *Improvement heuristic approaches* can obtain high-quality solutions in a reasonable time. However, most of them can hardly handle dynamic environments efficiently because the schedule re-optimisation process is too time-consuming to react in real-time. *Construction heuristic approaches* such as scheduling heuristics are the most popularly used heuristics for DFJSS. However, the scheduling heuristics are usually designed by experts *manually*. The design highly relies on domain knowledge, especially for complex scenarios, which is often unavailable to the end-users. In addition, the designed heuristics are usually too specific to be reused in other scenarios. GPHH is naturally a good candidate to evolve scheduling heuristics *automatically* for DFJSS [158, 173, 242, 249] due to its flexible representation.

1.3 Motivations

Although GPHH has shown its superiority for DFJSS, there are still limitations of the existing GPHH approaches in terms of training efficiency, search ability, search mechanism, and multitask solving ability.

First, the GP individual evaluation is **time-consuming** in DFJSS, since a lot of priority calculations with the GP individual are involved for making decisions during simulation runs. Simulation [57] is a promising technique to investigate the complex real-world problems such as health care [133] and manufacturing [176]. DFJSS is implemented with a complex sim-

ulation environment to simulate the real-world production situations.

Surrogate-assisted evolutionary computation [116] has been widely used for reducing computational time in evolutionary optimisation of expensive problems such as drug design [30, 67] and surgical training optimisation [19, 147], where complex computational simulations are involved. Surrogate approaches are also used with pre-selection strategy [73] to improve the effectiveness of the evolutionary algorithms. However, the studies related to surrogates for JSS is very limited. To the best of our knowledge, the performance of the existing two surrogate approaches on JSS, either based on K nearest neighbour [99, 174] or simplified model [176, 245], are sensitive to the accuracies of the built surrogates, since there is only a single surrogate model for the investigated problem.

Generally, high-fidelity surrogate models can provide more reliable and accurate results but with higher computational cost than low-fidelity surrogate models [116]. A promising way to achieve a trade-off between prediction accuracy and computational cost in modelling is to integrate the information from both high-fidelity and low-fidelity simulations by constructing a multi-fidelity surrogate model [258]. Introducing multiple models can reduce the dependence on one single surrogate model. In addition, if the surrogate models can help each other effectively, they may enhance each other, thus benefiting the overall algorithm. However, the multi-fidelity surrogate model approach has not to be studied in DFJSS. This motivates this thesis to propose an effective multi-fidelity based surrogate-assisted GPHH algorithm for DFJSS.

Second, GP has **large search space** due to its variable-length representation, which can limit its search effectiveness and efficiency. Reducing the search space can not only enhance the effectiveness of GP but also improve the interpretability of evolved scheduling heuristics with fewer features. Feature selection [93] is an important task to select relevant and complementary features to reduce the search space. Feature selection has been successfully used for different tasks such as classification [56, 223, 231],

clustering [136], and regression [42].

However, there are some challenges which make traditional feature selection approaches not directly applicable in DFJSS. First, the task (i.e., prioritising operations or machines) in DFJSS and the training instances are different from the traditional machine learning tasks and instances. The training data are generated with the simulation execution in DFJSS while the training data already exist in the traditional machine learning tasks. In this case, filter-based feature selection approaches for traditional machine tasks such as classification cannot be applied for DFJSS, since it is impossible to measure the importance of each feature based on the filter measures such as entropy [120] and Pearson's correlation [210]. Second, it is much more computationally expensive to apply the wrapper-based feature selection approaches in DFJSS than in traditional machine tasks. Specifically, running a GP process to obtain a reliable estimation of the best objective value of a terminal set is much slower than training a classifier (e.g., decision tree) in traditional machine learning tasks. Besides, in most embedded approaches, GP has been successfully used to handle both the feature selection and the supervised machine learning tasks such as the regression [42, 44] and classification [40, 162] simultaneously. However, feature selection is rarely used in GP for JSS which is an unsupervised machine learning task.

Existing works [152, 155] about feature selection for dynamic JSS were mainly presented in an offline way. This means that feature selection is always applied as a pre-processing step to get a promising subset of terminals first, and then the selected terminals are used in another independent GP run to solve the problems. It is noted that this pre-processing step is actually running GP for more than one time, and detects important features based on the outputs of GP runs. There are some drawbacks of the offline way. First, this will lose some generated good structures of individuals which have been generated in the process of feature selection (i.e., GP runs). Second, GP evaluation is very time-consuming, and GP

with feature selection requires more individual evaluations to investigate the feature importance, and thus making it even time-consuming. In addition, there is no work about using feature selection to solve the DFJSS problem. All these unexplored research questions motivate this thesis to propose an effective GPHH algorithm with feature selection for DFJSS.

Third, the **search mechanism** for generating the offspring of the traditional GP is not effective, since there is no guidance for choosing the genetic materials from the parent(s). A GP individual (i.e., a tree) contains a number of subtrees. Taking the crossover operator as an example, in a typical GP crossover, subtrees are randomly chosen from the two parents for swapping to produce two offspring. However, the importance of subtrees of a GP individual can be different. Some subtrees play important roles for an individual, and removing them may worsen the fitness of an individual. Some other subtrees may be redundant (unimportant) [64], and removing them does not affect the fitness of an individual. In general, it is more likely to get offspring with good fitness, if the important subtrees of one parent can be reserved and the unimportant subtrees of one parent can be replaced by the important subtrees from the other parent.

Theoretical studies in [203, 204] suggest that introducing biases of GP operators can be beneficial for different purposes, such as improving the quality of offspring and controlling the size of offspring. To improve the quality of offspring, this thesis aims at incorporating biases for the crossover operator by measuring the importance of subtrees, since crossover is the most important genetic operators for GP [129]. However, there are no existing studies about measuring the importance of subtrees for a GP individual in DFJSS. In addition, it is not clear how to apply the expected "biases" to GP for DFJSS. All these open research questions motivate this thesis to propose effective specialised genetic operators to improve the search mechanism of GP.

Last, the multitask GP algorithm in the hyper-heuristics domain for **solving multiple JSS tasks simultaneously** is still an unexplored area.

Multitask learning is a paradigm in the optimisation that aims at solving multiple self-contained tasks simultaneously. The paradigm of multifactorial optimisation toward evolutionary multitask (MFEA) was developed in [90, 91] for solving multiple tasks simultaneously by evolutionary algorithms. The success of MFEA relies on the knowledge sharing mechanism [92] between tasks by assortative mating and vertical cultural transmission during the evolutionary process. Specifically, the candidate solutions in a population for different tasks enhance each other by harnessing the hidden relationships between them via continuous genetic sharing in a unified search space. Based on the assumption that many problems in real-world are interconnected, MFEA has been widely and successfully applied to solve different problems such as continuous numeric optimisation [91, 138, 256], symbolic regression [255], and JSS [234, 239]. A recent multitask selective hyper-heuristic has been investigated in [96] on exam timetabling and graph colouring problems. However, to the best of our knowledge, there is no study on multitask generative hyper-heuristic.

In addition, surrogate techniques have been incorporated into multitask learning [62, 104, 146, 157]. However, the surrogate techniques were used to improve the performance of a single task rather than enhancing the core mechanism of multitask learning, such as the knowledge sharing between tasks. All these unexplored research questions motivate this thesis to propose an effective multitask GPHH algorithm to solve multiple DFJSS tasks simultaneously, and develop an effective surrogate-assisted multitask GPHH algorithm by enhancing the knowledge sharing between tasks with surrogate techniques.

1.4 Research Goals

The overall goal of this thesis is to develop an effective GPHH approach by considering the training efficiency, search space, search mechanism, and multitask solving ability, to evolving scheduling heuristics automatically

for DFJSS from different aspects. Specifically, this thesis focuses on improving training efficiency of GPHH via surrogate techniques, reducing the search space of GPHH with feature selection techniques, improving the search mechanism of GPHH by developing specialised genetic operators, and improving the multiple task solving ability of GPHH with multitask learning and surrogate techniques. The details of each objective are shown as follows.

The first objective is to **develop an effective GPHH algorithm with multi-fidelity based surrogate models to automatically evolve scheduling heuristics for the DFJSS problems more efficiently.** The proposed algorithm is expected to both speed up the convergence and reduce the training time of GPHH for DFJSS. Specifically, multi-fidelity surrogates are built by simplifying the DFJSS problem to be solved to different degrees. Involving multi-fidelity surrogates aims to reduce the dependence of the algorithm performance on specific model accuracy. In addition, an effective collaboration framework with knowledge transfer is proposed to utilise the advantages of surrogate models with different fidelities to enhance the final performance of the overall algorithm.

The second objective is to **develop effective GPHH algorithms with feature selection to reduce the search space for evolving scheduling heuristics for DFJSS efficiently.** The proposed algorithms are expected to help GPHH find smaller rules only with selected features without sacrificing the performance. The rules with smaller sizes and fewer unique features tend to be more interpretable. First, this thesis develops an effective two-stage GPHH feature selection framework for DFJSS with surrogate and niching techniques to detect important features effectively and efficiently. Second, this thesis proposes GPHH feature selection algorithms with novel individual adaptation strategies to generate new individuals only with the selected features but with similar performance. Individual adaptation strategies are proposed to utilise the information of both the selected features and examined individuals during the feature selection process.

The third objective is to **develop effective GPHH algorithms by improving the quality of generated offspring with new search mechanisms based on the importance of subtrees for DFJSS.** The proposed algorithm is expected to help GPHH find better scheduling heuristics by improving the quality of the produced offspring. This thesis proposes two subtree importance measures. One is based on the frequency of features, and the other is based on the correlation between subtrees and the whole tree (i.e., an individual). The developed correlation importance measure reflects the degree of relationship between the behaviour of the subtree and the entire tree. An effective recombinative guidance strategy is developed to improve the quality of offspring in GPHH for DFJSS via the crossover operator. Specifically, the probability of a subtree to be chosen is set based on its importance. An offspring is generated by replacing an unimportant subtree from one parent with an important subtree from the other.

The last objective is to **develop effective multitask GPHH algorithms for evolving scheduling heuristics for multiple DFJSS tasks automatically.** The proposed algorithm is expected to both speed up the convergence of GPHH and improve the quality of obtained scheduling heuristics for solving multiple DFJSS tasks simultaneously. A multitask GPHH framework is firstly developed by adapting the traditional evolutionary multitask algorithm based on the characteristics of GPHH. In addition, based on the proposed multitask GPHH framework, this thesis introduces to use the surrogate technique to assist multitask GPHH for DFJSS. The built surrogates are used to improve the effectiveness of solving a single task and the effectiveness of sharing knowledge between tasks.

1.5 Major Contributions

This thesis makes the following contributions:

1. This thesis has shown how to employ multi-fidelity based surrogate models to improve the **efficiency** of GPHH to evolve schedul-

ing heuristics automatically for DFJSS. This is successfully achieved by proposing an effective collaboration framework in GPHH that allows multi-fidelity based surrogates to learn from each other with an effective knowledge transfer mechanism.

The results show that the proposed algorithm can dramatically reduce the computational time of GPHH for DFJSS without losing its performance. Within the same training time, the proposed algorithm can achieve significantly better performance in most test scenarios, while no worse than its counterparts in all the scenarios. The efficiency of the proposed algorithm is verified by comparing the training time. The effectiveness of the proposed algorithm is verified with two stopping criteria, i.e., the same number of generations and the same training time. The knowledge transfer mechanism is also further analysed. In summary, the proposed algorithm can successfully improve the efficiency of GP, and achieve effective scheduling heuristics for DFJSS. The proposed algorithm shows its superiority compared with the state-of-the-art algorithms related to surrogate for the JSS problems.

Part of this contribution has been published in:

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. "Collaborative Multifidelity-Based Surrogate Models for Genetic Programming in Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Cybernetics*, 2021, pp. 1-15. (Doi: 10.1109/TCYB.2021.3050141)

Fangfang Zhang, Yi Mei, and Mengjie Zhang. "Surrogate-Assisted Genetic Programming for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the Australasian Joint Conference on Artificial Intelligence*, 2018, pp. 766-772.

2. This thesis has shown how feature selection can be applied to GPHH to **reduce the search space** for evolving scheduling rules with fewer unique features and smaller sizes for DFJSS without compromising

the performance. This is achieved by proposing a two-stage GPHH feature selection framework with surrogate and niching techniques for DFJSS. In addition, the GPHH feature selection algorithm is further improved with novel individual adaptation strategies to eliminate the unselected features for evolving scheduling rules without sacrificing the performance. The effective individual adaptation strategy utilises the information of the selected features and the investigated individuals in the feature selection process.

The results show that the proposed two-stage GPHH algorithm with feature selection can effectively and efficiently detect important features for routing and sequencing rules. In addition, the evolved rules by the proposed GPHH feature selection algorithm with individual adaptation strategies have fewer unique features and smaller rule sizes. The proposed algorithm is less time-consuming than the baseline algorithm, since the average rule size over population is reduced. Overall, the proposed algorithm can efficiently evolve scheduling heuristics with only the selected features without comprising the performance.

Part of this contribution has been published in:

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. "Evolving Scheduling Heuristics via Genetic Programming with Feature Selection in Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 1797-1811, 2021.

Fangfang Zhang, Yi Mei, and Mengjie Zhang. "A Two-stage Genetic Programming Hyper-heuristic Approach with Feature Selection for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 347-355.

3. This thesis has shown how effective recombinative guidance strategies in GPHH can improve its **search mechanism** to evolve effective scheduling heuristics by improving the quality of produced off-

spring for DFJSS. This is achieved by proposing effective ways to measure the importance of subtrees of an individual. This thesis proposes two measures for the importance of subtrees, one based on the frequency of features, and the other based on the correlation between the behaviours of the subtree and the entire tree. These two measures are compared with each other to show their effectiveness. A recombinative guidance mechanism is carefully designed to utilise the subtree importance information to replace unimportant subtrees from one parent with important subtrees from the other parent in crossover to improve the quality of generated offspring in GPHH.

The results show that the evolved rules by the proposed algorithm with correlation coefficient based recombinative guidance have better performance in most scenarios while no worse in all other scenarios due to its effectiveness for producing offspring. The correlation based importance measure is better than the feature frequency based importance measure for detecting the importance of subtrees. The analyses also verify this in terms of the depth ratios of selected subtrees, the correlations of selected subtrees, and the probability difference between subtrees during the evolutionary process. In addition, the proposed algorithm does not need extra computational time compared with its counterparts. This verifies the advantages of utilising the information produced by GPHH during the evolutionary process and the efficient information calculation techniques such as correlation coefficient.

Part of this contribution has been published in:

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. "Correlation Coefficient based Recombinative Guidance for Genetic Programming Hyper-heuristics in Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 3, pp. 552-566, 2021.

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. "Guided Subtree Selection for Genetic Operators in Genetic Programming for Dynamic Flexible Job Shop Scheduling". *in Proceedings of the European Conference on Genetic Programming*, Springer, 2020, pp. 262-278.

4. This thesis has shown how multitask learning can be used to handle multiple DFJSS problems simulataneously for the first time. First, this thesis proposes a new multitask GPHH algorithm by adapting the traditional evolutionary multitask algorithms based on the characteristics of GPHH for **solving multiple DFJSS tasks simultaneously**. Second, based on the proposed multitask GPHH algorithm, this thesis proposes a novel surrogate-assisted multitask GPHH for DFJSS. The built surrogate mechanism can not only improve the effectiveness of solving a single task, but also help share knowledge between tasks in the multitask learning framework. The proposed surrogate-assisted multitask algorithm has three main features as compared to the traditional multitask framework. First, a large number of new offspring are generated for providing useful materials for solving the tasks. Second, the newly generated individuals are evaluated with the surrogate rather than actual simulation evaluations. Third, the individuals are assigned to optimise tasks based on the estimated fitness by surrogates directly rather than the computationally expensive simulation evaluations.

   The results show that the proposed multitask GPHH framework can solve multiple related DFJSS tasks effectively. In addition, the proposed surrogate-assisted multitask GPHH algorithm can evolve effective scheduling heuristics for DFJSS with high convergence speed for all the examined multitask scenarios. The performance of the proposed algorithm is examined by comparing the convergence speed, the quality of evolved scheduling heuristics, the analyses of the diversity of individuals for tasks and the structures of the evolved

scheduling heuristics. It has also been observed that the individual allocations for the tasks are highly related to the utilisation level of the job shop. This implies that the complexities of tasks (i.e., a task with a higher utilisation level is more complex) significantly impact knowledge sharing between tasks in a multitask DFJSS scenario. In addition, we found that the sizes of the evolved rules over generations are highly related to the objective examined rather than the utilisation level.

Part of this contribution has been published in:

Fangfang Zhang, Yi Mei, Su Nguyen, Mengjie Zhang, and Kay Chen Tan. "Surrogate-Assisted Evolutionary Multitask Genetic Programming for Dynamic Flexible Job Shop Scheduling". *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651-665, 2021.

Fangfang Zhang, Yi Mei, Su Nguyen, Kay Chen Tan, and Mengjie Zhang. "Multitask Genetic Programming Based Generative Hyper-heuristics: A Case Study in Dynamic Scheduling". *IEEE Transactions on Cybernetics*, 2021, pp. 1-15. (Doi: 10.1109/TCYB.2021.3065340)

Fangfang Zhang, Yi Mei, Su Nguyen, and Mengjie Zhang. "A Preliminary Approach to Evolutionary Multitask for Dynamic Flexible Job Shop Scheduling via Genetic Programming". *in Proceedings of the Genetic and Evolutionary Computation Conference*, ACM, 2020, pp. 107-108.

Fangfang Zhang, Yi Mei, Su Nguyen, Kay Chen Tan and Mengjie Zhang. "Adaptive Multitask Genetic Programming for Dynamic Job Shop Scheduling". Submitted to *IEEE Transactions on Evolutionary Computation* (Under Review)

1.6 Terminology

To avoid confusion due to ambiguity, below are the definitions of the terms commonly used in this thesis:

- A **problem** is a high-level proposition we aim at solving, such as JSS.
- A **simulation** is the process to represent the environment of a problem.
- An **instance** is a specific simulation with a fixed random seed.
- A **scenario** represents a specific problem to be solved with the instances generated by the same problem configuration, e.g., the same objective and utilisation level. An instance is an example of a scenario.
- A **task** is a specific problem to be solved which is represented by a scenario. Solving the problems in different scenarios simultaneously is a multitask learning problem.

1.7 Organisation of Thesis

Figure 1.2 shows the outline of this thesis, including the main goals (listed with •) and involved techniques (listed with ■) in each chapter, and the connection between the chapters in this thesis. The remainder of this thesis is organised as follows. Chapter 2 presents the literature review. Then, our achievements on the four research objectives are presented in four chapters from Chapter 3 to Chapter 7. Chapter 8 concludes this thesis. An overview of each chapter is shown as follows.

Chapter 2 presents the literature review, including scheduling, evolutionary computation, GP, and heuristic and hyper-heuristics. The detailed descriptions of the JSS problem with a focus on DFJSS are given. How to

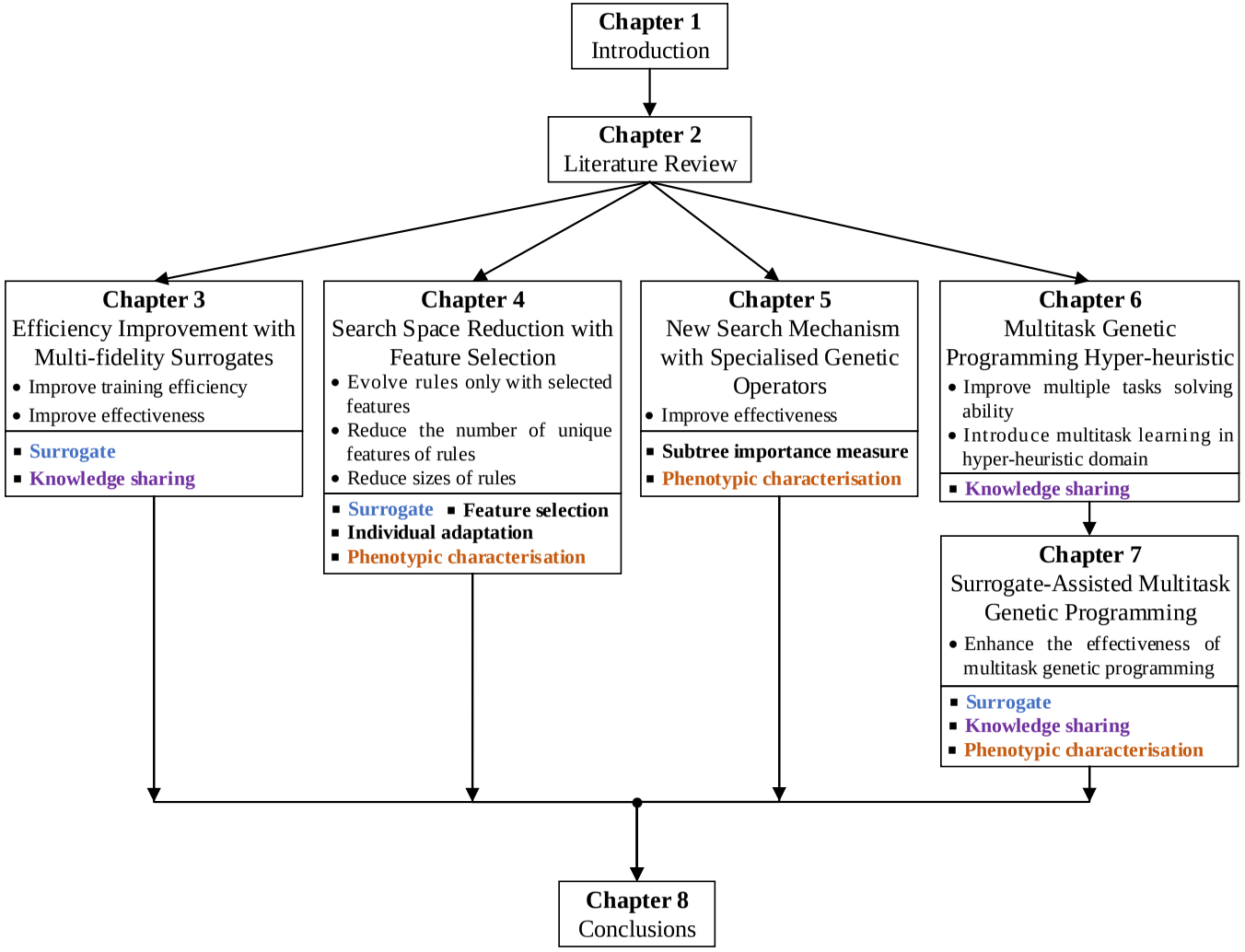

Figure 1.2: The outline of this thesis, including the main goals and involved techniques of each chapter, and the connection between the chapters in this thesis.

use scheduling heuristics for DFJSS is also described. This chapter also provides details on how to use GPHH to evolve scheduling heuristics for DFJSS automatically. In addition, existing studies related to the four research objectives of this thesis are discussed in details.

Chapter 3 describes the proposed GPHH algorithm with multi-fidelity surrogate models to improve the training efficiency and effectiveness for DFJSS. Details of the proposed GPHH algorithm with multi-fidelity surrogates are given in this chapter. Specifically, the framework of the proposed algorithm is described firstly. Then, the main component of the proposed algorithm, i.e., the knowledge transfer mechanism for collaborating the multi-fidelity surrogates is provided. The efficiency and effectiveness of the proposed algorithm are verified along with further analyses. The surrogate and knowledge sharing mechanism are the main techniques in this chapter.

Chapter 4 presents the proposed novel two-stage GPHH feature selection framework to reduce the search space of GPHH by selecting important features for routing and sequencing rules. Based on the proposed framework, this chapter further describes a new GPHH feature selection algorithm with individual adaptation strategies. The performance of the proposed algorithms, the sizes of the evolved scheduling heuristics, the number of unique features in the evolved scheduling heuristics, and the semantics of the evolved routing and sequencing rules (i.e., how they make decisions) are analysed. The surrogate, feature selection, individual adaptation, and phenotypic characterisation are the main techniques in this chapter.

Chapter 5 introduces the newly proposed recombinative guidance mechanism for generating offspring with good quality to improve the search mechanism of GPHH based on the characteristics of DFJSS. Two strategies of measuring the importance of subtrees of an individual are given in details. Different aspects such as the performance, the depth ratios and correlations between the selected subtrees of the proposed algorithm are

studied. In addition, the sizes and the semantics of the evolved scheduling heuristics are analysed to show how the proposed algorithm can effectively solve the DFJSS problems. The subtree importance measure and phenotypic characterisation are the main techniques in this chapter.

Chapter 6 gives the details of the proposed multitask GPHH approach in hyper-heuristic domain, focusing on the proposed knowledge sharing mechanism. The comparison between the multitask GPHH and traditional multitask evolutionary algorithms is discussed in details with homogeneous and heterogeneous multitask scenarios. In addition, the quality of the evolved scheduling heuristics, and the evolved routing and sequencing rules are analysed. The knowledge sharing is the main technique in this chapter.

Chapter 7 illustrates the proposed surrogate-assisted multitask GPHH algorithm for solving multiple DFJSS tasks simultaneously. The quality of the evolved scheduling heuristics and the effectiveness of the built surrogates are verified. The diversity of individuals and individual allocation by the surrogate technique for tasks are analysed. Last, the sizes and semantic insight of the evolved scheduling heuristics are further studied. The surrogate, knowledge sharing, and phenotypic characterisation are the main techniques in this chapter.

Chapter 8 summaries the achieved objectives and the main conclusions of this thesis. Some discussions and future research directions are also presented in this chapter.

Figure 1.2 shows that this thesis aims to enhance the performance of GPHH for DFJSS from different aspects. However, different chapters in this thesis share the same techniques from chapter to chapter, and they are highly connected. It is noted that the ways to use the same technique in a different chapter can be different. Each chapter also has its unique characteristic for the goal in that chapter.

Chapter 2

Literature Review

This chapter starts by introducing basic concepts in scheduling, the machine learning basics and the related methodologies used in this thesis such as evolutionary computation, GP, heuristics and hyper-heuristics. Second, the details of DFJSS are presented, followed by introducing on how to use scheduling heuristics for DFJSS. Third, this chapter illustrates how to use GPHH to evolve scheduling heuristics for DFJSS. Forth, the JSS approaches, including exact optimisation approach, heuristic approach, and hyper-heuristic approach, are described. Last, the related studies for the research goals in this thesis are discussed in details.

2.1 Basic Concepts

This section provides basic concepts of scheduling, machine learning basics, evolutionary computation, GP, heuristic and hyper-heuristics.

2.1.1 Scheduling

Scheduling [87, 200] is a decision-making process to optimise the resources, which is commonly used in many applications, especially in the manufacturing and services industries [149, 199]. The resources can be machines in

a job shop, gates at an airport, the staff at the hospital, and so on. In other words, scheduling is the activity of planning the times at which particular tasks will be done, or events will happen. The optimised scheduling objectives can be different, either minimising the total cost to get more benefit or minimising the total operating time to deliver service to customers on time.

Scheduling problems in this thesis are characterised by three types of sets, i.e., the set $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ of n jobs, the set $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ of m machines, and the set $O_{J_j} = (O_{j1}, O_{j2}, ..., O_{jl_j})$ for l_j operations for job J_j. A scheduling problem aims to use the set of machines to complete the given set of jobs. According to [17], there are three common models of scheduling, i.e., flow shop, open shop, and job shop.

A job in **flow shop scheduling** consists of *an ordered list of operations*. The number of operations of each job equals the number of machines. All operations *have the same processing order* through the machines. This means that the processing order for each operation in flow shop scheduling is the same, i.e., the i^{th} operation of the job must be executed on the i^{th} machine. Flow shop scheduling can occur in a manufacturing site with 100% standardization operated in an assembly production line. **Open job scheduling** is the same as the flow shop scheduling except that the *order of processing operations comprising one job may be arbitrary*. This means that there are no ordering constraints on operations. A job in **job shop scheduling** is composed of *an ordered list of operations*. Each job can *have different processing order* through the machines. A job shop can occur in a business with 100% customisation with a typical batch size of 1, which implies that every finished product is unique. The job shop is complex because of the different production processes for jobs. Job shop scheduling provides flexible form of manufacturing, and thus it is the focus of this thesis.

It is challenging to make an effective schedule efficiently for scheduling problems, especially when there are any dynamic factors (e.g., jobs arrives continuously) or a large scale scheduling problem. Manually designing

heuristics is only suitable for static (i.e., static scheduling, for example, the information of all jobs is available when making the schedule) and small scale problems. However, in practice, the scheduling environment is normally under dynamic situations with unpredicted events (e.g., in dynamic scheduling, the jobs can arrive in real-time, and the information of an arrived job is unknown until it arrives at the shop floor). Advanced and automated approaches are worth investigating.

2.1.2 Machine Learning Basics

Machine learning can be defined as computational methods using experience (i.e., reflected in the form of data) to improve performance or make accurate predictions [160]. Machine learning is an area of artificial intelligence [209] that can learn from experience automatically and improve from experience without being explicitly programmed.

There are three main types of learning problem in machine learning: (1) **supervised learning**, (2) **unsupervised learning**, and (3) **reinforcement learning**. In supervised learning problems (e.g., classification and regression), the target outputs for a problem are known (i.e., labelled data) for learning from the experience. On the other hand, in unsupervised learning problems (e.g., clustering), the target outputs for a problem are unknown (i.e., unlabelled data), and the models need to discover information by themselves. In reinforcement learning problems (e.g., game), the learner interacts with the environment and receives an immediate reward for each action. Reinforcement learning differs from supervised learning in not needing labelled data instances to be presented, and there is no long-term reward feedback provided by the environment.

In machine learning, the data instances are normally divided into two subsets, i.e., **training set** and **test set**. The training set is a subset of the data to derive a model. The test data is unseen to the trained models, and used to measure the performance of the trained models. **Generalisation**

is a term used to describe a model's ability to react to test data, which is central to the success of a trained model. **Overfitting** is to describe the issue that a model is trained too well on training data to generalise on test data. Overfitting is a potential problem mainly with supervised learning. A good model is expected to have good generalisation ability.

Three popularly used machine learning techniques (i.e., surrogate, feature selection/construction, transfer learning especially multitask learning) used in this thesis, so they are introduced in this subsection. **Surrogate techniques** aim to build computationally cheap models that can estimate the outcome of interest as closely as possible [116]. Surrogate models have attracted increasing attention to assist evolutionary algorithms to reduce the computationally expensive problems [117]. **Feature selection** [93] is the process of selecting the features which contribute most to a particular task. Removing weakly relevant features or redundant features with feature selection techniques can bring a number of benefits such as reducing the search space and improve the training efficiency, since there are few opportunities to make decisions based on noise. Feature construction [148] is the application of a set of constructive operators to generate new features from a set of existing features. Transfer learning [185] focuses on gaining knowledge for solving one problem and applying the knowledge to a different but related problem. **Multitask learning** [35] is a typical transfer learning technique that addresses multiple related tasks simultaneously. The success of multitask learning lies in effective knowledge sharing between the tasks. Multitask learning may be particularly helpful if the tasks share significant commonalities and are generally slightly under sampled [95].

2.1.3 Evolutionary Computation

Evolutionary computation [7] is a computational intelligence technique inspired from natural evolution based on population, which is a sub-field of

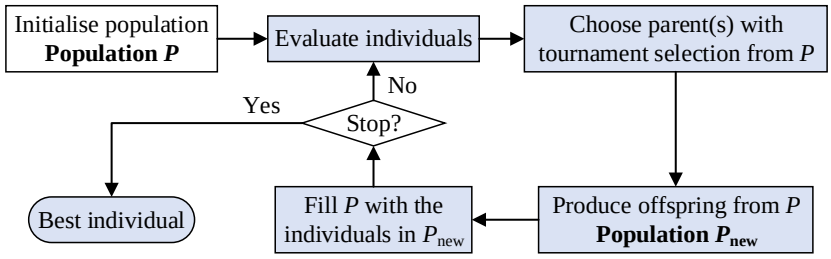

Figure 2.1: The flowchart of a typical evolutionary computation algorithm.

artificial intelligence [36]. The success of evolutionary computation relies on the improvement of individuals generation by generation. There are two main categories in EC, which are *evolutionary algorithms* such as genetic algorithms [50], GP [202], evolution strategies [16] and evolutionary programming [77], and *swarm intelligence* such as particle swarm optimisation [125, 179] and ant colony optimisation [66]. Evolutionary algorithms are the focus in this thesis.

Figure 2.1 shows the flowchart of a typical evolutionary algorithm. An evolutionary algorithm starts with generating a population with many individuals that represent solutions to the problem. The initial population could be created randomly. The individuals are evaluated with a fitness function, and the output of the function shows how well the individuals solve the investigated problem. Then, genetic operators, such as crossover, mutation, and reproduction, are applied to the selected parents to generate new individuals. The individuals with higher quality have higher chances to be selected as the parent(s) to produce offspring. This process continues until the termination criterion (e.g., reaching a certain number of generations) is met. The best individual is selected as the output of the evolutionary algorithm.

Since GP is the main technique used in this thesis, we will provide details of GP in the next subsection.

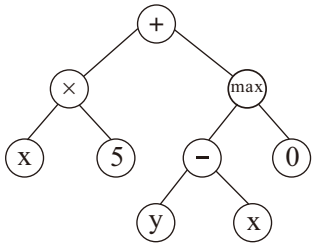

Figure 2.2: An example of tree-based GP program.

2.1.4 Genetic Programming

GP [129], is one of the most popular used evolutionary algorithms, can automatically generate computer programs to solve problems. The distinguishing feature of GP from other evolutionary algorithms is its variable-length representation. In the rest of this section, the main concepts of GP are presented to show how GP works.

Representation/Program Generation

Representation [72] is a way of representing individuals in evolutionary algorithms. A typical GP uses the tree structure to represent individuals. Specifically, GP generates tree-based individuals based on a terminal set (for leaf nodes) and a function set (for non-leaf nodes). Figure 2.2 shows an example of a GP program $5x+max(y-x,0)$. In this program, the terminals consist of the variables $\{x, y\}$ and two constants $\{5, 0\}$, and the functions are composed of $\{\times, +, -, max\}$. The program is the combination of the components in the terminal set and the function set. Except for tree-based GP, there are also some other popular representations such as linear GP [12, 181], graph-based GP [201], grammar-guided GP [130] and cartesian GP [156].

It is noted that the selection of the terminals and functions is critical for GP to succeed. The terminal set and the function set should be selected to satisfy the requirements of *sufficiency* and *closure*. Sufficiency means that there must be some combinations of terminals and functions that can solve

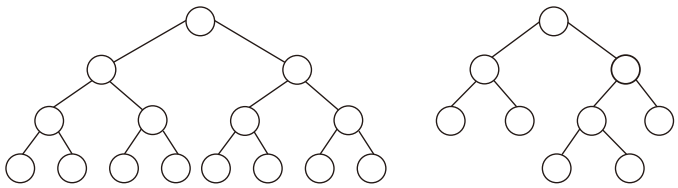

Figure 2.3: An example of programs generated by *full* method and *grow* method.

the problem, while closure means that any function can accept any input value returned by any function and terminal.

Initialisation/Program Generation

Initialisation [72] is the first step of GP to generate a population with a number of individuals randomly. For GP, *full* and *grow* [129] methods are commonly used to initialise population. A maximum depth can be determined for each GP individual to restrict the size of one program. For the full method, the terminals can only be sampled at the maximum depth of the trees. On the other hand, in the grow method, the terminals can be sampled at any position of the tree to early prune some branches. As a result, the full method always generates full trees at a given depth, while the grow method can generate trees that are much smaller and unbalanced than the full trees. In order to improve the diversity of the initial population, these two methods are commonly combined to initialise the population, which is known as *ramped half-and-half*. Specifically, this hybrid method is to generate half of the population by the full method, and the other half by the grow method. An example of generated individual by full method and grow method are shown in Figure 2.3 (a) and 2.3 (b).

Evaluation

Evaluation is an important step to measure the quality of the individuals in the GP population according to the fitness function. The fitness function plays a significant rule in GP to guide the search to find good programs. Individuals with bad quality are eliminated generation by generation. In general, the fitness function is defined based on the objectives of the problem. For example, fitness can be defined as the classification accuracy in the classification problem, and the total flowtime in a JSS problem.

Selection

Selection is the stage for evolutionary algorithms to decide which individual to chosen from a population (i.e., parent) for later breeding. There are a number of methods for selection such as roulette wheel selection [145] and tournament selection [128]. For roulette wheel selection, the probability of choosing an individual for breeding of the next generation is proportional to its fitness. The better the fitness is, the higher chance for that individual to be chosen. Tournament selection also chooses parent(s) based on fitness. It randomly selects a number of individuals first, and then selects the one with the best fitness as the parent. Tournament selection is widely used in GP [42, 99, 166].

Evolution

Evolution is the main process of GP to generate offspring for the next generation. There are three genetic operators for GP, which are *crossover*, *mutation* and *reproduction*. These operators aim at generating a new population by inheriting good materials from the old population.

- For crossover, two individuals are selected as the parents with the selection method. First, a subtree will be selected randomly from each parent. Then, these two subtrees from the parents are swapped

to produce two new individuals, and they will be put into the new population.

- For mutation, one individual is selected as the parent with the selection method. First, a subtree of the parent will be selected randomly. Then, the chosen subtree will be replaced by a newly generated subtree.
- For reproduction, an individual will be firstly chosen by the selection method, then the selected individual is copied into the new population directly. It is noted that the elitism mechanism (i.e., a special case of reproduction) picks up the top individuals from the current population. The selected individuals are inserted into the population of the next generation. This aims to ensure that the best individuals will not be lost when generating new populations.

2.1.5 Heuristics and Hyper-heuristics

In computer science, heuristic [192] is a technique designed for solving a problem more quickly when classic approaches are too slow, or for finding an approximate solution when classic approaches fail to find any exact solution. Commonly used evolutionary heuristic algorithms are genetic algorithms and particle swarm optimisation algorithm [124, 179]. A simple rule, such as the dispatching rule in scheduling that can rank alternatives in a search process based on the available information, is also a heuristic.

A hyper-heuristic [24, 25] is a heuristic search approach to selecting and generating heuristics to solve hard computational search problems. A hyper-heuristic is a learning algorithm when it uses some feedback from the search process. The term hyper-heuristics was first used to describe heuristics to choose heuristics in combinatorial optimisation [51]. The hyper-heuristic approaches can be classified into two types according to different considerations. In terms of the nature of the search space of the heuristics, there are two kinds of hyper-heuristic approach [26]. One is

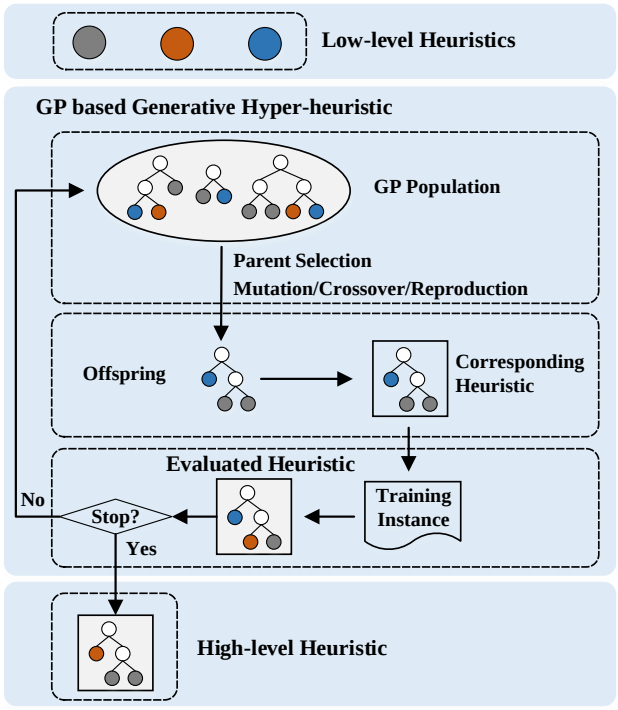

Figure 2.4: An example of generating high-level heuristic from simple low-level heuristics with GP.

heuristic selection, which aims to select from existing heuristics for different situations. The other is *heuristic generation*, which generates new high-level heuristics from existing low-level heuristics. In terms of the sources of feedback information, the hyper-heuristics approaches can be grouped into *online learning hyper-heuristics* and *offline learning hyper-heuristics*. For the online learning hyper-heuristics, the learning algorithm aims at solving an instance of a problem. For the offline learning hyper-heuristics, the learning algorithm learns knowledge in the form of rules or programs from a set of training instances, and expects the learned rules or programs to have good generalisation to unseen instances.

Figure 2.4 shows how to generate high-level heuristic from low-level heuristics with GP. The circles with different colours indicate three different low-level heuristics. The boxes with light grey background mean the

corresponding heuristics are evaluated with fitness. The low-level heuristics are set as the terminal set of GP and combined with the function set to form a GP individual. The quality of the individuals in the population is improved generation by generation with genetic operators (i.e., mutation, crossover, and reproduction). The offspring is actually a GP individual, and it is a function/rule that can be used as a heuristic to optimise problems by assigning priority values for machines and operations in production scheduling for example. Taking a generated offspring as an example, its corresponding heuristic is obtained based on the GP individual. The heuristic is then executed on training instances to get its fitness value. If the algorithm meets the stopping criterion, the current best heuristic will be selected as the final obtained high-level heuristic. Otherwise, the search continues. The final high-level heuristic is the best generated heuristic in the population.

Overall, the fundamental difference between the heuristic and hyper-heuristic approaches is that heuristic approaches work on solution space, while hyper-heuristic approaches work on heuristic space. In this thesis, we use GP as a hyper-heuristic approach, i.e., heuristic generation, and an offline learning hyper-heuristic approach. The goal is to generate effective heuristics that have good generalisation for solving DFJSS problems.

Evolutionary computation methods are powerful artificial intelligence algorithms that have been used for applications in many areas, including machine learning. GPHH, as one of the popular used evolutionary computation methods, have been successfully used to solve different machine learning tasks, including classification [109] and regression [42]. GPHH has also been successfully used to evolve scheduling heuristics for JSS.

2.2 Dynamic Flexible Job Shop Scheduling

In JSS, n jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ need to be processed by m machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each job has a sequence of operations $O_{J_j} =$

$(O_{j1}, O_{j2}, ..., O_{jl_j})$. The completion of the last operation for a job means the job has been finished. In DFJSS, each operation O_{ji} can only be processed by one of its optional machines $\pi(O_{ji})$ and its processing time $\delta(O_{ji}, M(O_{ji}))$ depends on the machine that processes it. The routing and sequencing decisions need to be made simultaneously in DFJSS. In DFJSS, dynamic events are necessary to be taken into account when constructing schedules. This thesis focuses on one type of dynamic event, i.e., stochastically and continuously arriving new job arrivals. This indicates that the information about a job is unknown until it arrives at the shop floor.

The objective of DFJSS is the optimised performance criterion for a problem while satisfying all the constraints. In this thesis, we consider various commonly used objective functions such as flowtime and tardiness. The parameters, variables, constraints and how to use scheduling heuristics for DFJSS in this thesis are described as follows.

Parameters:

- $\mathcal{J}$: the set of jobs in the job shop
- n: the number of jobs in the job shop
- $\mathcal{M}$: the set of machines in the job shop
- m: the number of machines in the job shop
- j: the index of job
- i: the index of operation
- l_j: the number of operations for job J_j, $l_j <= m$
- $O_{J_j} = (O_{j1}, O_{j2}, ..., O_{jl_j})$: the set of operations of job J_j
- w_{J_j}: the weight of job J_j (i.e., jobs with larger weights are more important)
- d_{J_j}: the due date of job J_j

- $M(O_{ji})$: the machine that processes operation O_{ji}
- $\delta(O_{ji}, M(O_{ji}))$: the processing time of operation O_{ji}
- $\pi(O_{ji})$: the set of candidate machines of O_{ji}, $\pi(O_{ji}) \subseteq \mathcal{M}$. This parameter is used in flexible JSS which will be described later.

Variables:

- C_{J_j}: the completion time of job J_j
- $r(O_{ji})$: the release time of operation O_{ji}. That is, the time that ith operation of job J_j is allowed to be processed. For the first operation of each job, it is set to zero. Otherwise, it is set to the completion time of its preceding operation.
- $r(J_j)$: the release time of job J_j. It is the release time of the first operation of a job.

Constraints:

- The $(i+1)^{th}$ operation of job J_j (i.e., $O_{j(i+1)}$) can only be processed after its preceding operation O_{ji} has been processed (i.e., precedence constraint).
- Each machine can only process at most one operation at a time.
- The scheduling is non-preemptive, i.e. once started, the processing of an operation cannot be stopped or paused until it is completed.
- Each operation O_{ji} can be processed on one of the corresponding set of machines $\pi(O_{ji}) \subseteq \mathcal{M}$ with processing time $\delta(O_{ji}, M(O_{ji}))$.

Scheduling Heuristics for DFJSS:

Due to the precedence constraint, only the *ready operations* are allowed to be allocated to machines. Two kinds of operations will become ready operations. One is the first operation of a job that has arrived at the shop

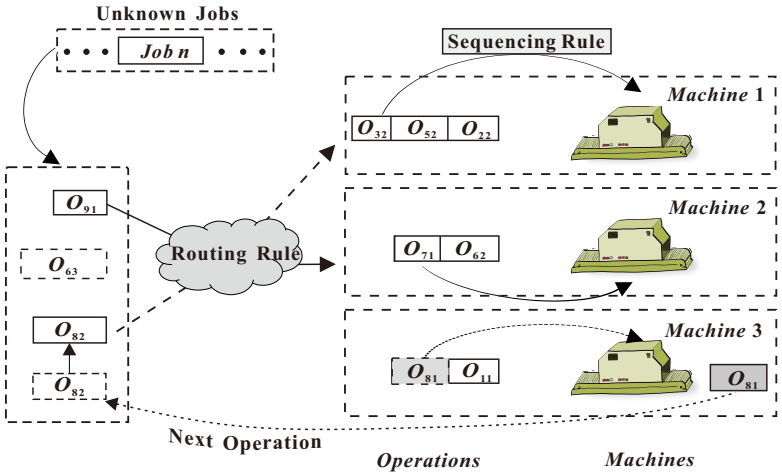

Figure 2.5: An example of decision making processes of DFJSS with scheduling heuristics (i.e., routing rule and sequencing rule).

floor. The other is the subsequent operation whose preceding operation has been finished. As scheduling heuristics, routing rule and sequencing rule work together to do machine assignment and operation sequencing in DFJSS. Both routing and sequencing rules are numerical priority functions, which are used to prioritise the machines or operations in different decision situations. When a new operation becomes ready, the routing rule will be applied to prioritise its candidate machines, and the operation will be assigned to the machine with the best priority (e.g., has the least workload under the least-work-in-queue rule). When a machine becomes idle, the sequencing rule will be triggered to prioritise the operations in its queue, and the operation with the best priority (e.g., the one with the shortest processing time if the shortest-processing-time rule is used) will be chosen to be processed next.

Figure 2.5 shows an example of the decision making processes of DFJSS with scheduling heuristics. There are three machines, each with several operations waiting in its queue. The operation O_{81} is being processed on *Machine 3*. A routing (sequencing) *decision situation* includes a temporal

job shop state, the given operation (machine) and the candidate machines (operations). The routing decision and sequencing decision are made with the routing and sequencing decision situation, respectively.

- **Routing Decision:** Once an operation becomes a ready operation (*a routing decision situation is encountered*), it will be allocated to the machine with the highest priority according to the *routing rule*. For example, when a new job (J_9) arrives at the job shop, its first operation O_{91} is allocated to *Machine 2* which has the highest priority value among the three machines according to the routing rule. In addition, when O_{81} is finished, its next operation (O_{82}) becomes a ready operation and is allocated to *Machine 1* by the routing rule.
- **Sequencing Decision:** When a machine (e.g., *Machine 1*) becomes idle, and its queue is not empty (*a sequencing decision situation is encountered*), the *sequencing rule* will be used to calculate the priority value of each operation in its queue. The operation with the highest priority is then chosen as the next operation to be processed (e.g., in this case, O_{32} is selected to be processed on *Machine 1*).

2.3 GPHH for DFJSS

2.3.1 Overall Process of GPHH for DFJSS

GP, as a hyper-heuristic approach [27], has been successfully applied to evolve scheduling heuristics for combinatorial optimisation problems such as packing [28, 107], timetabling [8, 198], and JSS [69, 175, 243, 248]. The optimal structures of heuristics are normally not known in real-world applications, which makes the heuristic learning process challenging. Tree-based GP [21, 131] is a good candidate to learn heuristic for DFJSS due to its flexible representation. This implies that the structures of heuristics do not need to be defined in advance. In addition, tree-based programs ob-

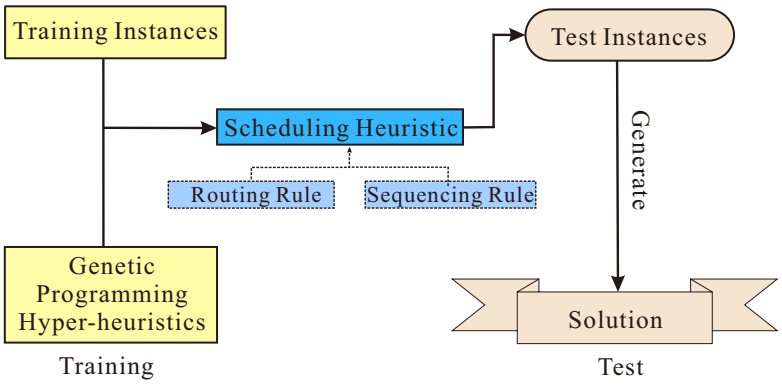

Figure 2.6: The overall process of GPHH for DFJSS.

tained by GP (i.e., can be considered as functions) provide us with opportunities to understand the behaviour of the evolved rules, which is very important for real-world applications.

Figure 2.6 shows the overall process of GPHH for DFJSS in this thesis. In the training phase, GPHH is used to train heuristics based on a set of training instances. The outputs of the training process are heuristics (routing and sequencing rules) rather than solutions (schedules). In the test phase, the evolved heuristics obtained in the training phase are tested on unseen instances. Specifically, to calculate the test performance of a trained heuristic, it is applied to each test instance to construct a schedule. The test performance is then defined as the (normalised) average objective value, such as the mean flowtime, of the constructed schedules on the test instances.

The pseudo-code of GPHH to learn heuristics for DFJSS is shown in Algorithm 1. The input of the proposed algorithm is a task that is expected to be solved, and the output is the learned heuristic h^* with a routing heuristic r^* and sequencing heuristic s^*. As a population-based algorithm, GP starts with a randomly initialised population (line 1). It is noted that each GP individual contains two trees [244]. The first tree represents the routing rule, and the second tree represents the sequencing rule. The fitness

Algorithm 1: Pseudo-code of GPHH to learn routing and sequencing heuristics for DFJSS

Input : A task
Output: The learned scheduling heuristics h^* with r^* and s^*

1: **Initialisation**: Randomly initialise the population
2: set $r^* \leftarrow null$ and $fitness(r^*) \leftarrow +\infty$
3: set $s^* \leftarrow null$ and $fitness(s^*) \leftarrow +\infty$
4: set $h^* \leftarrow r^* \cup s^*$
5: $gen \leftarrow 0$
6: **while** $gen < maxGen$ **do**
7: // **Evaluation**: Evaluate the individuals in the population
8: **for** *i = 1 to popsize* **do**
9: Run a DFJSS simulation with h_i to get the schedule $Schedule_i$
10: $fitness_{h_i} \leftarrow Obj(Schedule_i)$
11: **end**
12: **for** *i = 1 to popsize* **do**
13: **if** $fitness_{h_i} < fitness_{h^*}$ **then**
14: $h^* \leftarrow h_i$
15: **end**
16: **end**
17: **if** $gen < maxGen - 1$ **then**
18: **Evolution**: Generate a new population by crossover, mutation, and reproduction
19: **end**
20: $gen \leftarrow gen + 1$
21: **end**
22: **return** h^* with r^* and s^*

of heuristics are evaluated based on the objective functions (from line 7 to line 11). Specifically, a simulation is run with the heuristic h_i to get a schedule $Schedule_i$ (line 9). The fitness of the heuristic h_i is assigned by calculating the objective value of its obtained schedule $Schedule_i$ (line 10). A new population is generated by recombining the heuristics (crossover), mutating the heuristics (mutation), or copying the heuristics with good fitness directly (reproduction and elitism) (line 18) to the next generation.

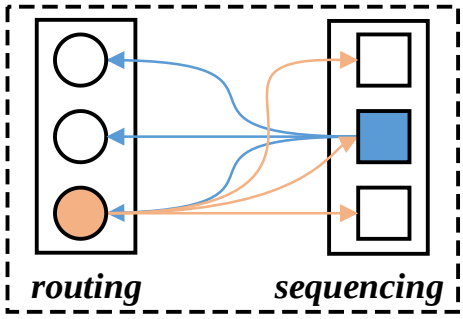

Figure 2.7: An example of the representation with cooperative coevolution for DFJSS.

2.3.2 Representation

According to the characteristics of DFJSS, a routing rule and a sequencing rule are needed to make two decisions simultaneously. To date, there have been three main frameworks to handle DFJSS. Tay et al. [219] proposed to use GP to only evolve the sequencing rule by fixing the routing rule as a manually designed rule for flexible JSS. However, only the sequencing rule is evolved, which may not be effective. Yska et al. [233] introduced a cooperative coevolution framework with GP for the first time to evolve routing and sequencing rules simultaneously by applying two subpopulations. The proposed approach showed its superiority due to the coevolution mechanism. Zhang et al. [244] introduced GP with multi-tree representation for evolving two rules within an individual in one population. The approach is promising in terms of the effectiveness, efficiency, and sizes of evolved rules.

Figure 2.7 shows an example of the representation of evolving routing and sequencing rule simultaneously for DFJSS with cooperative coevolution technique. Routing and sequencing rules are evolved in two different subpopulations (i.e., indicated by rectangles). The best routing rule (in orange) in one subpopulation makes a number of scheduling heuristics pairs with all the sequencing rules for DFJSS, while the best sequencing

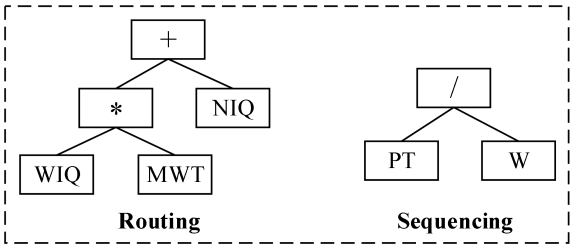

Figure 2.8: An example of the multi-tree representation with a routing rule and a sequencing rule for DFJSS.

rule (in blue) in the other subpopulation makes various scheduling heuristics pairs for DFJSS with all the routing rules for DFJSS. All these pairs are evaluated when evaluating the individuals in these two subpopulations. The fitness of an individual is calculated by the fitness function with the best individual in the other subpopulation.

Figure 2.8 shows an example of the multi-tree representation with a routing rule and a sequencing rule for DFJSS. The routing rule can be considered as a priority function to give priority values for candidate machines. The corresponding priority function of the routing rule in Figure 2.8 is WIQ * MWT + NIQ, where WIQ is the needed total processing time of operations in the queue of a machine, MWT is the waiting time for a machine to become idle, and NIQ is the number of operations in the queue of a machine. The machine with the smallest priority value assigned by the routing rule will be selected to allocate the operation. Similarly, the operation with the smallest priority value assigned by the sequencing rule (i.e., PT / W, where PT is the processing time, and W is the weight of an operation) will be selected as the next operation to be processed on the idle machine.

Both the representations with cooperative coevolution and multi-tree can evolve the routing rule and sequencing rule simultaneously for DFJSS. The advantage of the representation with cooperative coevolution is that it can separate the routing and sequencing decisions properly, which makes

it easy to implement new genetic operators. The representation with multi-tree makes it easy to manage the population, since the routing and sequencing rules are considered as a whole. In this thesis, we choose to use the representation with cooperative coevolution or multi-tree based on the characteristics of the algorithms.

2.3.3 Evaluation

In this thesis, there are mainly four measures for the performance of the algorithms. Different measures are chosen based on the characteristics of the proposed algorithms.

Quality of the Evolved Best Scheduling Heuristics: The objectives considered in this thesis are shown as follows. It is noted that we do not consider all these objectives simultaneously in a many-objective case. We consider them separately as a single objective in different scenarios.

- Max-flowtime: $max_{j=1}^{n}\{C_{J_j} - r(J_j)\}$
- Mean-flowtime: $\frac{\sum_{j=1}^{n}\{C_{J_j} - r(J_j)\}}{n}$
- Mean-weighted-flowtime: $\frac{\sum_{j=1}^{n} w_{J_j} * \{C_{J_j} - r(J_j)\}}{n}$
- Max-tardiness: $\max_{j=1}^{n} max\{0, C_{J_j} - d_{J_j}\}$
- Mean-tardiness: $\frac{\sum_{j=1}^{n} max\{0, C_{J_j} - d_{J_j}\}}{n}$
- Mean-weighted-tardiness: $\frac{\sum_{j=1}^{n} w_{J_j} * max\{0, C_{J_j} - d_{J_j}\}}{n}$

where C_{J_j} is the completion time of a job J_j, $r(J_j)$ is the release time of J_j, d_{J_j} is the due date of J_j, w_{J_j} is the weight (importance) of job J_j, and n is the number of jobs to be processed.

For the sake of convenience, Fmax, Fmean, WFmean, Tmax, Tmean, and WTmean are used to indicate max-flowtime, mean-flowtime, mean-weighted flowtime, max-tardiness, mean-tardiness and mean-weighted-tardiness respectively. The utilisation level is a commonly used parameter [20, 197] to represent different job shop scenarios for measuring the effectiveness of the algorithms. It is noted that a higher utilisation level will lead to a more complex task. This thesis focuses on using the objective function and the utilisation level to construct multiple test scenarios because our preliminary studies have shown that the performance of the evolved scheduling heuristics is influenced significantly by these two factors.

Training Efficiency: Training time is a common way to measure the efficiency of algorithms. If an algorithm can achieve comparable scheduling heuristics with a shorter time, the algorithm is efficient. The CPU time is reported as the training time to measure the efficiency of the algorithms.

Sizes of the Evolved Scheduling Heuristics: The rule size is defined as the number of nodes in this thesis, and the rule with a smaller size is preferred. There are a number of advantages of evolving smaller rules. First, smaller rules tend to be more interpretable by decision makers, which is particularly important for the floor operators of the job shop. Second, smaller rules are easier to implement real-world applications, which are more efficient to make real-time decisions with dynamic events compared with larger rules.

Number of Unique Features of Evolved Scheduling Heuristics: The number of unique features is the number of different features needed to construct the rules. The number of unique features in the scheduling heuristics is one indicator of the complexity of evolved rules [242]. The smaller the number of unique features, the easier the scheduling heuris-

tics to potentially interpret the rules.

It is noted that except for the above measures, the mechanism of the effectiveness of the algorithm (i.e., the measures are defined especially in each chapter) is also evaluated based on the characteristics of the investigated algorithm.

2.4 Job Shop Scheduling Approaches

A large number of approaches have been developed for JSS. This section discusses the exact approaches, heuristic approaches and hyper-heuristic approaches to JSS which have been proposed in the literature.

2.4.1 Exact Optimisation Approaches

The job shop problem is an NP-hard problem for which it is extremely hard to find optimal solutions [216]. Exact mathematical programming such as branch-and-bound [135] and dynamic programming [15], has been applied extensively for JSS. **A branch-and-bound algorithm** carries out a depth-first search to find feasible solutions to the JSS problems [200]. After a feasible solution is found, the algorithm eliminates any branches where the lower bound on the problem is worse than the best solution so far. The algorithm then searches for new solutions exhaustively until all branches have been explored to ignored, and the best solution found is returned. Heuristics have been incorporated to enhance the search of branch-and-bound algorithm [2, 4, 31]. A fast branch and bound algorithm for the JSS problem has been developed in [22] by combining a branching scheme [86] and a method to fix disjunctions before each branching step [33]. This algorithm has been successfully found an optimal solution for a JSS problem with 10 jobs, 10 machines, and 10 operations per job. The details of branch-and-bound techniques can be found in [205]. **Dynamic programming** approaches divide a problem into constituent sub-problems and aim

to solve the problem by solving all the sub-problems. A dynamic programming algorithm was proposed to solve one machine scheduling problem in [134]. The results showed that the proposed algorithm can find the optimal solution for the JSS with one machine. A new dynamic programming algorithm has been proposed to find optimal solutions for solving the JSS problems with up to 10 jobs and 5 machines in [88]. This algorithm was adapted from the proposed algorithm for the travelling salesman problem in [98].

Although exact mathematical programming technique can provide an optimal solution for JSS, they are generally time-consuming. Normally, they are limited to static and small scale JSS. They are not suitable for dynamic JSS, since the dynamic events may lead to the current optimal solution not be optimal any more. Searching for an optimal solution again makes the algorithm time-consuming.

2.4.2 Heuristic Approaches

Heuristics approaches aim to find good enough solutions. Unlike exact optimisation techniques, heuristic approaches do not guarantee to find an optimal solution. The heuristics techniques can be grouped into two categorisations, i.e., heuristic search approach and scheduling heuristic approach.

Heuristic Search

Heuristic search approaches aim to find a good enough solution gradually. The most popularly used heuristic search approaches are evolutionary algorithms. Genetic algorithm is the most popular evolutionary algorithm for JSS [46]. A genetic algorithm with different strategies for generating the initial population, selecting the individuals for reproduction and producing new individuals was proposed for flexible JSS [196]. The result showed that the integration of more strategies in a genetic framework

leads to better results than other compared algorithms. An improved genetic algorithm (i.e., improve the quality and diversity of the initial population, improve crossover to preserve good solutions, adapt mutation probability) was developed to solve JSS problems in [250]. The results demonstrated the good performance of the proposed algorithm.

Swarm intelligence approaches have also been successfully applied to JSS. A new hybrid swarm intelligence algorithm consists of particle swarm optimization, simulated annealing technique and multi-type individual enhancement scheme, was presented to solve the job-shop scheduling problem in [144]. The experimental resulted show that the new proposed job-shop scheduling algorithm is more robust and efficient than the compared algorithms. A hybrid discrete particle swarm algorithm based on maximum fitness function was proposed for a dual-resources constrained flexible JSS problem with multiple optimisation objectives [252]. The simulation results demonstrated that the proposed algorithm effectively decreases both production time and production cost. A two-level particle swarm optimization algorithm was developed for the flexible JSS problem [235]. The upper level handles the operations-to-machines mapping, while the lower level handles the ordering of operations on machines. The algorithm showed its superiority on a significant number of diverse benchmark flexible JSS problems.

Overall, the heuristic search approaches can find good solutions in a reasonable time, and can handle large scale problems well. However, it is not suitable for dynamic JSS, since the rescheduling process for heuristic search approaches is time-consuming. It is not efficient to react to dynamic events quickly.

Scheduling Heuristics

Dispatching rules, as scheduling heuristics, have been popularly used to dynamic JSS due to its ability to cope with the dynamic changes of the job shop. There are a large number of scheduling heuristics in the literature,

and they can be grouped into three categories [118]: (1) Simple priority rules, which are mainly based on the information related to jobs or machines. A commonly used simple routing rule is WIQ (i.e., the workload in the queue of machines), where a ready operation will be allocated to the machine with the least workload. A commonly used sequencing rule is SPT (i.e., shortest processing time), where the idle machine will choose the operation with the shortest processing time as the next operation to be processed. (2) Combination of priority rules, which consists of combinations of well-known rules from the literature. (3) Weighted priority rules, which is a weighted sum of a number of commonly used priority rules.

The comparison of different scheduling heuristics have been studied in many studies [100, 103, 114, 121, 206, 214]. However, they are normally manually designed based on specific scenarios, which is not easy to be applied to other scenarios. In addition, the designing process highly relies on experts, which are time-consuming and not always available. In addition, it is still challenging for scheduling researchers to develop scheduling heuristics that can perform well on multiple objectives.

2.4.3 Hyper-heuristic Approaches

Hyper-heuristics are a relatively new research area that focuses on exploring the heuristic search space rather than solution space [25]. This is different from heuristic search approaches. Two hyper-heuristics research directions are heuristic selection and heuristic generation. The heuristic selection has been successfully used to adaptively select heuristic based on the problem solving states in many JSS studies [9, 78, 79]. A number of hyper-heuristic frameworks have been developed for heuristic selection such as the simulated annealing based hyper-heuristic [68], tabu search based hyper-heuristic [29], and choice function [51, 52]. The output of heuristic selection approaches is normally a sequence of heuristics. Heuristic generation has been widely used to generate comprehensive scheduling heuris-

tic for the JSS problem [97, 188, 215, 242, 259]. The evolved scheduling heuristics by heuristic generation approaches are expected to have good generalisation ability to perform well on a set of unseen problems. GPHH based algorithms are currently the most popularly approaches for heuristic generation. Different machine learning techniques such as surrogate [240, 245], feature selection [59, 60, 152, 155, 242, 249], transfer learning and multitask learning [13, 239], have been incorporated into GPHH to improve its effectiveness to evolve scheduling heuristics for JSS.

2.5 Related Work

As mentioned above, scheduling heuristics are more effective and efficient for JSS, especially for dynamic JSS. In this section, the studies that aim to solve the JSS problems by using GPHH approaches to evolve scheduling heuristics will be presented. This is because GPHH is a dominating approach to automatically evolve scheduling heuristics for dynamic JSS. In addition, the existing studies related to our four research objectives (i.e., surrogate technique, feature selection strategy, specialising genetic operator and multitask learning) are discussed in details.

2.5.1 GPHH to Evolve Scheduling Heuristics for JSS

In this section, the previous literature related to JSS are classified into two categorisations according to the machine environment (i.e., the number of machines in the job shop). The reason is that the machine environment (e.g., single machine environment and multi-machine environment) is a key factor in determining the type of the JSS problems. In a single machine environment, there is only one machine in the job shop. While, in a multi-machine environment, more than one machine is available. The goal here is not to enumerate all the literature, but to give a big picture of the research about GPHH for JSS.

Single Machine Environment

In 2001, Dimopoulos and Zalzala [61] firstly investigated the GP framework to evolve dispatching rules for solving static scheduling problems with one machine. The results showed that the evolved dispatching rules are better than traditional rules in different scenarios with different levels of tardiness and tightness of due dates. Geiger et al. [80] examined a GP learning system for scheduling in a single machine environment. In this work, both the static and dynamic problems were considered, and the results showed GP can handle well these two situations. This paper also investigated a two-machine flow shop environment, however, this work tried to evolve the dispatching rule for each machine, which increased the complexity of the scheduling processes in the job shop system. Nie et al. [177] presented a gene expression programming based scheduling rules constructor to construct effective scheduling rules for the dynamic single-machine scheduling problems with one or more objectives. Yin et al. [232] proposed a bi-tree structured representation scheme for GP to make it possible to search sequencing and idle time in different uncertain environments. In this paper, the use of GP to evolve single-machine predictive scheduling heuristics with stochastic breakdowns was investigated, where both tardiness and stability objectives in the face of machine failures were considered. Jakobović et al. [112] proposed a multiple tree adaptive heuristic for dynamic single machine scheduling problem, where the decision tree was used to distinguish between resources based on their load characteristics. It works as a GP-3 system that evolves three components, a discriminant function and two dispatching rules. Among them, the discriminant serves as a monitor. Choosing which rule to use depends on the decision made by discriminant function, which was designed to identify the bottleneck machine.

In conclusion, in the single machine environment, the job shop scheduling problems can be static or dynamic, but cannot be flexible because flexible problems are generated in the multi-machine environment. The deci-

sion in a single machine environment is operation sequencing only.

Multi-machine Environment

In the multi-machine environment, JSS can be investigated in either static or dynamic conditions. It can be flexible or non-flexible. The type of JSS can also be any combinations of the four factors (i.e., static, dynamic, flexible and non-flexible). This section will give a high-level survey of these studies with focusing on dynamic JSS, flexible JSS and DFSS.

Dynamic Job Shop Scheduling. Jakobović et al. [113] applied GP to build scheduling algorithms for multiple machine environment and also considered the dynamic variants (i.e., job arrival continuously) of the problem. The work showed the effectiveness of GP over existing scheduling approaches. Mei et al. [152] presented a feature selection, in which a niching-based search framework was used for extracting a diverse set of good rules, for job shop scheduling. In addition, a surrogate model was applied to reduce the training time. The experimental studies showed that it took less than about 10% of the training time of the standard GP training process, and can obtain much better feature subsets than the entire feature set. Nguyen et al. [174] adopted the surrogate model in [99] for fitness approximation and proposed a select scheme to investigate the influence of surrogate models in dynamic job shop scheduling. This paper also analysed the advantages and disadvantages of different selection schemes in surrogate-assisted GP. Mei et al. [154] defined the concept of time-invariance and developed a new terminal selection scheme to guarantee the time-invariance throughout the GP process in a dynamic environment. Mei et al. [155] aimed at selecting a small set of useful features according to the contribution of the features and investigated the feature selection mechanism in the static and dynamic environment. The results showed that using only the selected features can achieve significantly better GP-evolved rules on both training and unseen test instances. Nguyen

et al. [169] proposed a diversified multi-objective cooperative coevolution algorithm based on GP to evolve dispatching rules and due-date assignment rules in dynamic job shop scheduling. The results showed that the proposed algorithm can effectively evolve Pareto fronts of scheduling policy compared to NSGA-II [58] and SPEA2 [260] while the uniformity of scheduling policy is better than those evolved by NSGA-II and SPEA2. Nguyen et al. [170] investigated different representations of the dispatching rules based on the previous literature. The results showed that the representation that integrates system and machine attributes can improve the quality of the evolved rules.

All of these studies show the superiority of using GP to evolve rules for dynamic JSS in the multi-machine environment. However, they are still non-flexible JSS.

Flexible Job Shop Scheduling. Ho et al. [101] applied GP to evolve composite dispatching rules for solving the FJSS problem, and the results showed that the obtained rules outperform the selected benchmark rules in 74% to 85% of problem instances. Tay et al. [219] tried to evolve dispatching rules for solving multi-objective flexible job shop scheduling problems using genetic programming. However, the multi-objective problem was converted into a single objective problem by linearly combining all objective functions. Hildebrandt et al. [100] re-examined this work in different dynamic job shop scenarios and showed that the rules evolved in [219] are only slightly better than the earliest release date rule and quite far away from the performance of the SPT rule. They explained that the poor performance of these rules was caused by the use of 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. Thus, Hildebrandt et al. [100] aimed at only minimizing mean flow time by evolving dispatching rules which were trained on four simulation scenarios. The experimental results indicated that the evolved rules were

quite complicated but effective when compared to other existing rules.

The investigations among these studies belong to flexible JSS, which are more practical. However, they are conducted in static environments, which are not normal cases in the real-world. Moreover, the routing rules in these work are fixed, and actually only sequencing rules are evolved.

Dynamic Flexible Job Shop Scheduling. Yska et al. [233] proposed a new GPHH algorithm with cooperative coevolution to explore the possibility of evolving both routing and sequencing rules together. The results showed that co-evolving the two rules together can lead to much more promising results than evolving the sequencing rule only. This is the first work that considered to evolve routing rule and sequence rule simultaneously using GP. Multi-tree representation was also introduced to evolve routing and sequencing rules for DFJSS in [244]. The results showed its effectiveness for evolving the routing rule and sequencing rule simultaneously. Stochastic dispatching rules were studied in [246] to investigate how to use scheduling heuristics for DFJSS. The results showed that always choosing the machines or operations with the highest priority value is an effective way to make scheduling decisions in DFJSS. The surrogate technique was successfully used for DFJSS to reduce GP training time in [245]. However, the effectiveness of the proposed algorithm was not improved. A new representation was developed for GP to evolve effective scheduling heuristics for DFJSS in [248]. The results showed that the proposed algorithm can achieve good performance in a range of scenarios. In addition, multi-objective GP was firstly proposed to solve conflicting objectives in multi-objective DFJSS [247]. The results showed that the proposed multi-objective GP algorithm successfully incorporated the strategy of NSGA-II [58] into GP for DFJSS. An adaptive search was proposed to guide GP explore more promising areas with important features which were detected based on the frequency of features [237]. The results showed that the feature importance varied in DFJSS, and using the feature

importance information can help improve the effectiveness of GP. Feature construction with different strategies was firstly developed in [233] to improve the efficiency of GP. The experiment results showed that although the proposed feature construction algorithms did not manage to improve the results, they improved the stability of the evolutionary process. A preliminary evolutionary multitask approach was firstly developed to solve multiple DFJSS tasks simultaneously [239]. The results showed that the proposed multitask GP algorithm can improve training efficiency dramatically. However, the effectiveness of the proposed algorithm was not significant. In addition, the visualisation technique was introduced in [167] to provide a better understanding of computer programs in GP for DFJSS.

These studies consider to evolve the routing and sequencing rules simultaneously, which have real merits. Different machine learning techniques with GP such as representation, surrogate, feature selection and multitask learning have been studied. However, the research in this field is still in a very early stage, and little work has been reported on these significant topics.

2.5.2 Surrogate Models in GP for JSS

In the past decades, surrogate-assisted evolutionary computation [116, 217] has been widely studied to reduce the computational cost of evolutionary algorithms. The basic idea is to use computationally cheap surrogate models to replace part of the computationally expensive evaluations of individuals. The commonly used techniques for surrogate models include radial basis function networks [189] and kriging model [49].

However, there are some challenges that make these kinds of surrogate techniques not directly applicable in DFJSS. The task (i.e., prioritising operations or machines) and the training instances in DFJSS are different from the traditional machine learning tasks such as regression [44] and numerical optimisation [38]. The training data in DFJSS are dynam-

ically generated along with simulation execution, while the training data are available in advance in traditional machine learning tasks. Although we can collect data during the simulation process, it is not trivial to decide what kinds of information are useful for training the surrogate model, and how to represent the collected data.

There is little research about surrogate-assisted GP for solving the JSS problems. Hildebrandt and Branke [99] proposed to approximately estimate the fitness of individuals by finding the most similar rule in the previous generation. However, the best value in the last generation is an upper bound of estimated fitness, which cannot totally reflect the quality of a new individual. The proposed surrogate approach was applied in a pre-selection way in which a larger number of individuals were generated to form an intermediate population, and only the top individuals were selected into the next generation for real fitness evaluations. The influence of surrogate models and the use of simulation replications on the performance of GP was investigated in [174]. This work showed the advantages and disadvantages of different selection schemes in surrogate-assisted GP. Nguyen et al. [176] also used the pre-selection way for the surrogate with a simplified model based surrogate-assisted GP for dynamic JSS. The novelty of this work lied on the surrogate models based on simplified simulation models of the job shop. It showed the effectiveness of using simplified simulation models. Zhang et al. [245] further proposed to use an adaptive surrogate strategy with dynamic fidelities of simulation models over generation to estimate the fitness of individuals in the population directly rather than in the intermediate population for DFJSS.

The studies mentioned above show the superiority of using surrogate-assisted GP for JSS. However, there are still some limitations. From the perspective of the way to incorporate surrogate into GP, pre-selection technique is commonly used to speed up the convergence of GP by increasing the number of evaluations (i.e., cheap evaluation) of extra individuals which is conducted in an intermediate population [99, 176]. Then, only

the individuals that achieve good fitness which are estimated by the surrogate model, are re-evaluated to get real fitness. The other way is to use the surrogate model to evaluate the fitness of individuals directly [245]. Both ways are sensitive to the accuracy of surrogate models. The surrogate models greatly affect which individuals can be used to generate offspring for the next generation, and thus will further affect the quality of individuals in the population. More advanced surrogate models are worthing to be studied to improve the effectiveness and efficiency of GP for JSS.

2.5.3 Feature Selection in GP for JSS

The performance of GP heavily relies on a proper selection of the terminal set [220]. In the terminal set, features (terminals) are not equally important. Besides, some features may be irrelevant, redundant or noisy, and the original features are typically not informative enough. All of these factors may lead to various performance limitations. Feature selection is an effective process for selecting a subset of relevant and complementary features [41, 231]. Feature selection algorithms are generally classified into three categories [231]: filter approaches, wrapper approaches, and embedded approaches.

However, to the best of our knowledge, little is yet known about using feature selection in JSS. Feature selection based on the frequency of terminals was introduced to help GP evolve dispatching rules for dynamic JSS in [105]. A novel feature importance measure instead of frequency was firstly introduced in [155] to select features for the dynamic JSS problem. Then, an efficient feature selection was proposed in [152] based on the feature importance measure in [155]. However, these approaches are only related to dynamic JSS. The feature selection technique was firstly used for DFJSS in [249]. In [249], a GPHH approach with feature selection was proposed for DFJSS, which involves two feature sets. However, the main drawback in [249] is that the selected features are only used to guide

the behaviour of GP by mutation operator. It does not change the evolution sufficiently, which greatly limits the influence of the feature selection. This points to an important question of applying the selected features effectively after obtaining a great feature set. It is still an important but unexplored research topic in DFJSS.

2.5.4 Genetic Operators in GP

The flexibility of GP makes it stand out among lots of evolutionary computation algorithms. However, GP still has some limitations. For example, an individual is likely to behave very differently and become much worse even after small changes. It is not fully clear what kinds of genetic operators can make the performance of GP better. In terms of the way to enhance the effectiveness of the genetic operators, we group the related studies into three categories. In this section, we review the related studies on genetic operators of GP with a focus on the crossover.

Adaptive Rate for Genetic Operators: Changing the rates of genetic operators is a simple way to improve the effectiveness of producing offspring. Adaptive operator selection rates with designed reward policies were proposed in [126] for GP. The results show that adaptive rate selection is an effective way to improve the performance of GP. Different algorithms of adapting the probabilities of genetic operators were proposed in [178] based on population-level, fitness, or individual-level information during the evolutionary process of GP. In [6], an adaptive decreasing mutation rate was proposed for GP to solve the truss structure optimisation problem. These algorithms succeed by balancing exploration and exploitation during the evolutionary process.

Depth-dependent Crossover: Intuitively, the depth of crossover point is an important factor for the quality of offspring because the performance of subtree is related to the depth to some extent. A general heuristic

that can be used to guide the development of the most effective depth-control strategy for any given problem was discussed in [228]. A "height-fair" crossover operator that only allowed to swap subtrees with the same depth was proposed in [184]. A depth selection probability was defined in [110] to ensure the node towards the root of an individual has a higher probability of being chosen as a crossover point than the ones towards leaves. These approaches aim to bias the crossover depth to improve the performance of GP. However, it is not straightforward to apply them to DFJSS, since the optimal depth is not known.

Semantic Crossover: The information of GP individuals can be used to produce offspring that bias to some semantics. Two new geometric search operators were developed in [43] to fulfil precise semantic requirements for symbolic regression. A novel crossover operator was proposed in [164] to address the exponential growth in the size of the individuals. The constrained dimensionally aware GP was designed based on the types of features in [153] to ensure only semantically correct individuals can be generated to improve the interpretability of evolved rules for JSS. The crossover bias for having the more fit parent as the root parent was presented in [151]. These approaches tend to achieve the goal by utilising the semantics of GP individuals during the evolutionary process.

Although there are some studies [108, 191, 253] on genetic operators of GP, little research has been conducted on the crossover to improve the quality of offspring by investigating the importance of subtrees directly. To this end, this thesis aims to improve the effectiveness of crossover by proposing an effective and adaptive recombinative guidance mechanism based on the importance of subtrees.

2.5.5 Multitask Learning

Multitask learning aims at solving multiple related tasks simultaneously [91], which is an important type of transfer optimisation [92]. Although

evolutionary multitask learning [90, 91] is a relatively new paradigm, it has recently received much research interests in optimising multiple tasks simultaneously. The paradigm of evolutionary multitask learning was given in [90, 91] for solving multiple tasks simultaneously. The success of evolutionary multitask learning relies on the knowledge sharing mechanism between tasks during the evolutionary process. Evolutionary multitask learning has been successfully applied to solve different problems such as continuous numeric optimisation [91, 138, 256], feature selection [39], symbolic regression [255] and job shop scheduling [234, 239]. However, most of the existing approaches are only applied to continuous, numeric optimisation problems rather than discrete, combinatorial problems such as DFJSS. In real-world applications, each product has different demands at different times, which is a typical DFJSS problem. The complexities of job shops such as the frequencies of production orders for producing a specific product, may vary [76]. For example, in the clothing industry, the orders for the down jacket in winter are usually higher than in summer. For the down jacket job shop, the frequencies of orders for producing down jacket vary between seasons. It is thus beneficial to have various kinds of scheduling heuristics for a company to handle different cases. Intuitively, giving multiple solutions simultaneously for a company is an effective way to improve problem-solving capability.

Multitask Learning in Hyper-heuristic Domain: Most existing multitask approaches aim at improving the qualities of solutions for all the tasks directly [62, 143], thus ignoring the hyper-heuristic research area. A unified framework of graph-based evolutionary multitasking hyper-heuristic approach was proposed and examined on timetabling and graph-colouring problems [96]. However, the proposed approach was a heuristic selection approach rather than a heuristic generation approach. In addition, it was only compared with the simple single-tasking hyper-heuristics, and there is no further analysis of the heuristic structure. From the perspec-

tive of involved research fields, evolutionary multitasking has been successfully applied to continuous numeric optimisation with benchmarks [62, 91, 138, 139], and regression problems [255]. However, the studies on discrete, combinatorial problems which have more complex situations are still very limited. This limits its applications in practice. In terms of solution representation, most studies are conducted with the vector-based search space [75, 84, 139, 257] rather than tree-based search space.

Multitask GP has been investigated for combinatorial optimisation problems, such as team orienteering [123] and dynamic job shop scheduling [186, 239]. The problem investigated in [123] is static. The training instances are clustered for each island, and several individuals with better fitness are transferred between islands. However, the approach cannot be applied for dynamic problems with simulation, since the training instances are not available for clustering. In addition, in [186], a niching approach was proposed for dynamic job shop scheduling. However, the main drawback is that the niched individuals need further evaluations, which is inefficient. In [239], multitask GP was applied to DFJSS, and the efficiency of solving multiple DFJSS problems was dramatically improved. However, the quality of the evolved scheduling heuristics were not improved.

Surrogate-assisted Multitask Learning: A surrogate-assisted multitask learning algorithm was proposed in [146] for the memetic algorithm with benchmark problems. However, the surrogate with Gaussian Process [208] was only used to assist the search process in the designed component of global search. In [104], surrogate models are built based on historical search information for each task to reduce the number of fitness evaluations in multitask problems. These works [104, 146] use the surrogate solely to improve the search efficiency for each task independently or for a single component in a multitask problem rather than enhancing the core multitask mechanism such as knowledge sharing. In addition, the mecha-

nism of individual allocation for tasks based on the original evaluations is computationally expensive if applied to DFJSS, since reallocated individuals would need to be re-evaluated with the simulation in the DFJSS.

In summary, the research about multitask learning on combinatorial optimisation, especially with hyper-heuristic approaches, is still very limited. In addition, surrogate-assisted multitask learning is also in its early stage. This thesis focuses on multitask learning in the hyper-heuristic domain with GPHH, and works on combinatorial optimisation, i.e., DFJSS. In addition, this thesis works on the surrogate-assisted multitask GPHH that can benefit the knowledge sharing between DFJSS tasks by the surrogate models directly.

Knowledge Transfer Schemes in Genetic Programming: In the field of transfer learning in GP, according to "what to transfer", there are two main schemes [63]. One is the "*FullTree*" that migrates a number of individuals with good quality from the source domain to the target domain. The other is the "*SubTree*" that is extracted from individuals in the source domain and adapted to the target domain.

Different from the traditional transfer learning in GP, there are no source and target domains in this work. The knowledge is transferred between different tasks directly without the knowledge extraction process from the source domain. In [255], assortative mating and vertical cultural transmission [91] were introduced to transfer information between different tasks in GP, which can be seen as a "*FullTree*" transfer. In [239], the knowledge transfer was realised by the crossover operator between the GP individuals for different tasks, which belongs to the "*Subtree*" transfer scheme. Compared with transferring "*SubTree*", the advantage of transferring "*FullTree*" is that the knowledge extraction process is not necessary. The key to transferring "*FullTree*" is that the chosen individuals must be of good quality for the problem that is expected to be solved. Otherwise, the transferred individuals will be eliminated subsequently during the evolution-

ary process, and lose the role of knowledge transfer [63]. On the contrary, compared with transferring "*FullTree*", the advantage of transferring "*SubTree*" is that the transferred knowledge might be more precise. However, the knowledge extraction of "*SubTree*" is complex and time-consuming.

2.6 Chapter Summary

This chapter introduces the basic concepts of scheduling, evolutionary computation, GP, and heuristics and hyper-heuristics which are fundamental knowledge of this thesis. The details of the investigated problem, i.e., DFJSS, are provided, and how to use scheduling heuristics for DFJSS is given with examples. Meanwhile, the approaches for JSS, including exact optimisation approaches, heuristic approaches, and hyper-heuristics approaches, are discussed. The technical details of using GPHH for DFJSS are described. The existing studies that use GPHH to evolve scheduling heuristics for JSS are discussed by categorising them into different groups. In addition, based on the research objectives of this thesis, the related studies are summarised. The limitations of the existing studies are highlighted as follows.

- The training process of GPHH for DFJSS is time-consuming.
- GPHH has a large search space which limits its search effectiveness and efficiency.
- For GPHH, the method to generate offspring for the next generation is not effective, since there is no guidance for choosing the genetic materials from the parents.
- The multiple task solving ability of GPHH is still an unexplored area.

The contents in this chapter help prepare the reading of this thesis by giving details of related concepts and techniques, and guiding the research

in the following chapters. The follow five contribution chapters will address these limitations to improve the performance of DFJSS problems.

Chapter 3

Efficiency Improvement with Multi-fidelity Surrogates

The literature has identified a number of key gaps and limitations of existing work in GPHH for DFJSS. From this chapter, we will develop new methods to address them one after another, from training efficiency improvement, search space reduction, search mechanism improvement, to multitask learning development. This chapter will focus on using multi-fidelity surrogate techniques to improve the training efficiency of GPHH for DFJSS.

3.1 Introduction

Simulation [57] is a promising technique to investigate the complex real-world problems such as health care [133] and quay crane scheduling [171]. This thesis conducts test beds based on dynamic flexible simulation model for JSS [233] to measure the effectiveness of the proposed GPHH algorithms. The simulation-based individual evaluations increases the training time of GPHH, since there are many priority calculations with GP individuals for making decisions during DFJSS simulation execution. Surrogate models [116, 183, 217, 240] have been widely used to reduce the com-

putational cost in evolutionary computation. The success lies in building computationally cheap models to approximate the fitness of individuals without requiring the original computationally expensive evaluations.

To the best of our knowledge, little is yet known to improve the efficiency of GP in JSS. The existing works of surrogate-assisted GP for JSS can be grouped into two categories according to the way of using the surrogate model to estimate the fitness of a GP individual. One is to use existing models such as applying KNN (i.e., K nearest neighbour) [194] to estimate the fitness of GP individuals with the most similar individual that has been evaluated in the previous generation [99, 174]. The other is to use a simplified simulation model as a surrogate model, which is a problem approximation technique to estimate the fitness of GP individuals [176, 245]. In [99], [174] and [176], the surrogates are used in a pre-selection way. Brief speaking, a large number of offspring are generated, and only the top individuals are selected (i.e., based on the estimated fitness by the surrogates) to be re-evaluated with real fitness evaluations. In [245], surrogates with different fidelities are used to evaluate the individuals directly by increasing the fidelity of applied surrogate model gradually along with the search process.

The studies mentioned above all show the superiority of using the surrogate technique in JSS. However, the performance of the algorithms highly relies on the accuracy of used surrogate models, either based on nearest neighbour [99, 174] or simplified simulation models [176, 245]. In addition, each surrogate in the multi-fidelity surrogate models was used independently at different generations [245]. On one hand, the performance at a specific generation is sensitive to the surrogate used in that generation. On the other hand, it may not be effective since there is no communication between the surrogates with different fidelities. More advanced techniques are worth investigating.

To address the above issues, this chapter proposes to use multiple surrogate models with different fidelities collaboratively to improve the train-

ing efficiency of GPHH to evolve effective scheduling heuristics for DFJSS automatically. Multi-fidelity surrogates are developed to achieve a trade-off between the accuracy and computational cost of surrogate models. Specifically, multi-fidelity surrogates are built by simplifying the DFJSS problem to be solved. In addition, an effective collaboration framework with knowledge transfer is proposed to utilise the information of surrogate models with different fidelities. It is noted that using the multi-fidelity surrogates can reduce the dependence of the algorithm performance on specific model accuracy.

3.1.1 Chapter Goals

The goal of this chapter is to *develop an effective GPHH with multi-fidelity based surrogate models to evolve scheduling heuristics for the DFJSS problems efficiently.* The proposed algorithm is expected to both speed up the convergence and reduce the training time of GPHH for DFJSS. Specifically, this chapter has the following research objectives:

1. Develop multiple surrogate models with different fidelities by simplifying the problem to be solved according to its characteristics.

2. Propose an effective collaboration framework with knowledge transfer for the designed multi-fidelity based surrogate models to learn from each other for solving the desired problem.

3. Analyse the efficiency of the proposed algorithm in terms of the training time and the convergence speed.

4. Analyse the effectiveness of the proposed algorithm in terms of the quality of the evolved rules.

5. Analyse the behaviour of the proposed GPHH algorithm in terms of the effect of the knowledge transfer mechanism.

3.1.2 Chapter Organisation

The rest of this chapter is organised as follows. Detailed descriptions of the proposed algorithm are given in Section 3.2. The experiment design is shown in Section 3.3, followed by results and discussions in Section 3.4. Further analyses are conducted in Section 3.5. Finally, Section 3.6 concludes this chapter.

3.2 Proposed Algorithm

This section describes the framework of the proposed algorithm first. Then, the key components of the proposed algorithm are given.

3.2.1 Framework of the Proposed Algorithm

The main framework of the proposed algorithm is presented in Algorithm 2. The input is k designed surrogate models with different fidelities. The surrogate models are developed using shorter DFJSS simulations. One of them is the original simulation which can be considered as a surrogate model of the original problem with an accuracy of 100%. The output of the proposed algorithm is a set of best evolved rules obtained from subpopulations with different surrogate models. The problems solved in different subpopulations can be considered as similar problems but with different problem scales (i.e., different simulation lengths). The proposed algorithm has three main differences compared with the traditional GPHH for JSS. *At the initialisation stage*, the population is formed with multiple subpopulations to incorporate multi-fidelity surrogate models into GP (line 1). Each subpopulation is associated with a surrogate model, respectively. *During the evaluation process*, the individuals in different subpopulations are evaluated with the surrogate model associated with it (from line 6 to line 18). *During the evolution stage*, the offspring of each subpopulation are generated for the next generation according to the proposed knowledge transfer

Algorithm 2: Framework of the Proposed Algorithm

Input : k multi-fidelity surrogate models S_1, S_2, ... , S_k
Output: The best evolved heuristics with each surrogate model ind_1^*, ind_2^*, ... , ind_k^*

1: **Initialisation**: Randomly initialise the population with k subpopulations
2: set ind_1^*, ind_2^*, ... , $ind_k^* \leftarrow null$
3: set $fitness(ind_1^*)$, $fitness(ind_2^*)$, ... , $fitness(ind_k^*) \leftarrow +\infty$
4: $gen \leftarrow 0$
5: **while** $gen < maxGen$ **do**
6: // **Evaluation**: Evaluate the individuals in the population
7: **for** *i = 1 to* k **do**
8: **for** *j = 1 to* $|subpopsize|$ **do**
9: Run a DFJSS simulation with ind_j to get the schedule $Schedule_j$
10: $fitness(ind_j) \leftarrow Obj(Schedule_j)$
11: **end**
12: **for** *j = 1 to* $|subpopsize|$ **do**
13: **if** $fitness(ind_j) < fitness(ind_i^*)$ **then**
14: $ind_i^* \leftarrow ind_j$
15: $fitness(ind_i^*) \leftarrow fitness(ind_j)$
16: **end**
17: **end**
18: **end**
19: **Evolution**: Generate offspring for each subpopulation by genetic operators with the proposed *knowledge transfer* mechanism — refer to Algorithm 3
20: $gen \leftarrow gen + 1$
21: **end**
22: **return** ind_1^*, ind_2^*, ... , ind_k^*

mechanism (line 19).

The key idea of this chapter is to collaborate multi-fidelity based surrogates for GPHH to evolve scheduling heuristics for DFJSS. Introducing multi-fidelity surrogate models implies that some individuals are evaluated with a simpler surrogate, and some individuals are evaluated with a more complex surrogate. This chapter uses multiple subpopulations to solve multiple tasks simultaneously, each subpopulation for one task. The

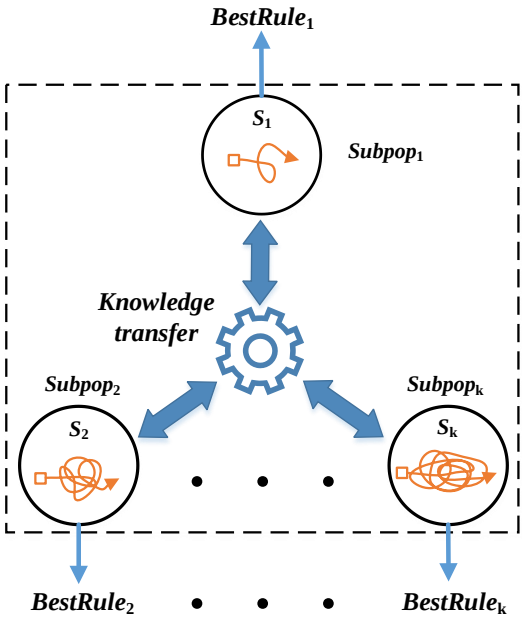

Figure 3.1: The evolutionary framework of the proposed algorithm.

individuals in the same or different subpopulation are evaluated with the surrogate with the same or different fidelity. From the perspective of the evolutionary process, the subpopulations can be considered as several independent evolutionary processes that are evolved simultaneously.

Figure 3.1 shows the evolutionary framework of the proposed algorithm. Assuming k surrogate models (S_1, S_2, ... , S_k) with different fidelities from simple to complex are used to improve the efficiency of GPHH to evolve effective scheduling heuristics, where the problem with S_k is the desired problem to be solved. The population of GP is divided into k subpopulations (e.g., $Subpop_1$, $Subpop_2$, ... , $Subpop_k$) and each subpopulation will evolve scheduling heuristics based on the corresponding surrogate model. In addition, during the evolutionary process, different subpopulations assist each other by sharing their knowledge. It is noted that all the subpopulations are evolved in parallel. Therefore, the output of a GP run consists of k best evolved rules (i.e., $BestRule_1$ (ind_1^*), $BestRule_1$ (ind_2^*), ... , $BestRule_k$ (ind_k^*)). However, we only focus on the best evolved rule

$BestRule_k$ obtained from $Subpop_k$ with S_k, since it is the problem we aim to solve.

There are some advantages of using the proposed evolutionary framework to realise collaborative multi-fidelity based surrogate-assisted GPHH for DFJSS. First, the evolutionary framework facilitates the collaboration between the surrogate models with different fidelities. In addition, the proposed algorithm is not sensitive to the accuracy of any single surrogate model, since multiple surrogate models with different fidelities are involved at each generation. Finally, as a by-product, problems with different scales (e.g., number of different jobs) are solved simultaneously. It is an efficient way to utilise computational resources, if one prefers to solve multiple problems with different scales simultaneously.

3.2.2 Knowledge Transfer

The key element of the proposed algorithm is to introduce a collaborative mechanism to utilise the information of surrogate models with multiple fidelities. "How" and "when" to transfer knowledge, and "what" to transfer are important research questions in this section [109, 185]. Different from the traditional transfer learning in GP, this chapter does not involve the source problem and the target problem. It implies that there is no knowledge extraction process from the source problem. This chapter proposes to conduct the knowledge transfer implicitly via the crossover operator [92].

How and when to transfer. Crossover is an important genetic operator in GP to produce offspring, which can be an effective carrier for knowledge transfer. Crossover can occur between individuals from the same subpopulation or different subpopulations. The proposed knowledge transfer mechanism is shown in Algorithm 3. We define a transfer ratio tr to control when to transfer knowledge from other subpopulations in each generation. A larger (smaller) tr value indicates the knowl-

Algorithm 3: Generating Offspring with Knowledge Transfer

Input : A population with k subpopulations
$P = \{Subpop_1, Subpop_2, ..., Subpop_k\}$
Output: A new population with k subpopulations
$P^{'} = \{Subpop_1^{'}, Subpop_2^{'}, ..., Subpop_k^{'}\}$

set P, $P^{'} \leftarrow null$
$gen \leftarrow 0$
while $gen < maxGen$ **do**
  // **Evaluation**: Evaluate the individuals in each subpopulation, respectively
  // **Evolution**
  for *i = 1 to* k **do**
    if $rand <= tr$ **then**
      $parent_1 \leftarrow$ Select the first parent from $Subpops_i$ — Algorithm 4
      $parent_2 \leftarrow$ Select the second parent from $Subpops_{\neg i}$ — Algorithm 4
      $point_1$: the crossover point of $parent_1$
      $point_2$: the crossover point of $parent_2$
      $offspring$: replace $point_1$ of $parent_1$ by $point_2$
      $Subpop_i^{'} \leftarrow Subpop_i^{'} \cup offspring$
    else
      $parent_1 \leftarrow$ Select the first parent from $Subpops_i$ — Algorithm 4
      $parent_2 \leftarrow$ Select the second parent from $Subpops_i$ — Algorithm 4
      $point_1$: the crossover point of $parent_1$
      $point_2$: the crossover point of $parent_2$
      $offspring_1$: replace $point_1$ of $parent_1$ by $point_2$
      $offspring_2$: replace $point_2$ of $parent_2$ by $point_1$
      $Subpop_i^{'} \leftarrow Subpop_i^{'} \cup offspring_1 \cup offspring_2$
    end
    $P^{'} \leftarrow P^{'} \cup Subpop_i^{'}$
  end
  $gen \leftarrow gen + 1$
end
return $P^{'} = \{Subpop_1^{'}, Subpop_2^{'}, ..., Subpop_k^{'}\}$

edge transfer between different subpopulations is (not) encouraged. If the knowledge transfer mechanism is triggered, the first parent $parent_1$ will be

selected from the current subpopulation (line 8). The other parent $parent_2$ will be selected from one of the other subpopulations (line 9). Only the offspring derived from $parent_1$ is kept in the new generation (line 10 to line 13). If the knowledge mechanism is not triggered, both parents will be selected from the current subpopulation (i.e., the same subpopulation) to produce two offspring for generating the new subpopulation (from line 14 to line 22).

It is noted that the knowledge can be transferred from a subpopulation with a lower-fidelity surrogate model to that with a higher-fidelity surrogate model, and vice versa. From the knowledge transfer perspective, the subpopulation with a lower fidelity surrogate (simpler problem) can find promising individuals faster than the subpopulation with a higher fidelity surrogate (more complex problem). For a subpopulation with a high fidelity surrogate model, introducing knowledge from a subpopulation with a lower fidelity surrogate model can speed up its convergence. For a subpopulation with a lower fidelity surrogate model, learning knowledge from a subpopulation with a higher fidelity surrogate model can help increase the quality of individuals, since the evolved rules with a higher fidelity surrogate model are more reliable. In general, the collaboration with knowledge transfer is expected to benefit all of the involved problems.

What to transfer. It is critical to decide what kinds of knowledge are useful to be transferred. Intuitively, the knowledge carried by promising individuals is beneficial. In this chapter, we use the knee point technique [254] to decide the set of promising individuals. The individuals with smaller fitness values (i.e., our problem is a minimisation problem) than the fitness of the knee point individual, are selected as promising individuals. It is noted that the number of promising individuals varies in different generations, and the knee point technique can capture promising individuals efficiently. In addition, the number of selected promising individuals is not required to be defined in advance (i.e., the knowledge transfer mechanism is *parameter-free*). With the knee point technique, the

Algorithm 4: Pseudo-code of selecting promising individuals for knowledge transfer

Input : The current subpopulation with a set of individuals Ind
Output: Promising individuals Ind^* for knowledge transfer

1: $sort(subpopulation)$
2: minPoint(0, fitness(ind_0)
3: maxPoint($subpopsize$ -1, fitness($ind_{subpopsize-1}$))
4: set $maxDistance \leftarrow 0$ and $kneePointIdx \leftarrow 0$
5: get a line L based on minPoint and maxPoint points
6: **for** *i = 0 to* $subpopsize - 1$ **do**
7: calculate the distance (d) from $Point(i, fitness(ind_i))$ to line L
8: **if** $d > maxDistance$ **then**
9: $maxDistance \leftarrow d$
10: $kneePointIdx \leftarrow i$
11: **end**
12: **end**
13: **for** *i = 0 to* $subpopsize - 1$ **do**
14: **if** $i <= kneePointIdx$ **then**
15: $Ind^* \leftarrow Ind^* \cup Ind[i]$
16: **end**
17: **end**
18: return Ind^*

promising individuals can be decided adaptively. Only the knowledge carried by the promising individuals is allowed to be transferred to other subpopulations.

Algorithm 4 shows the pseudo-code of selecting promising individuals for knowledge transfer. First, the individuals in the subpopulation are sorted in ascending order of their fitness values (line 1). Second, one line (L) is generated between the two points with maximal and minimal fitness. Then, the distance between each individual and L is calculated (from line 6 to line 12), and the knee point which has the highest distance to L is detected. Finally, the individuals whose fitness values are smaller than the fitness of knee point, are chosen as the *promising individuals* for

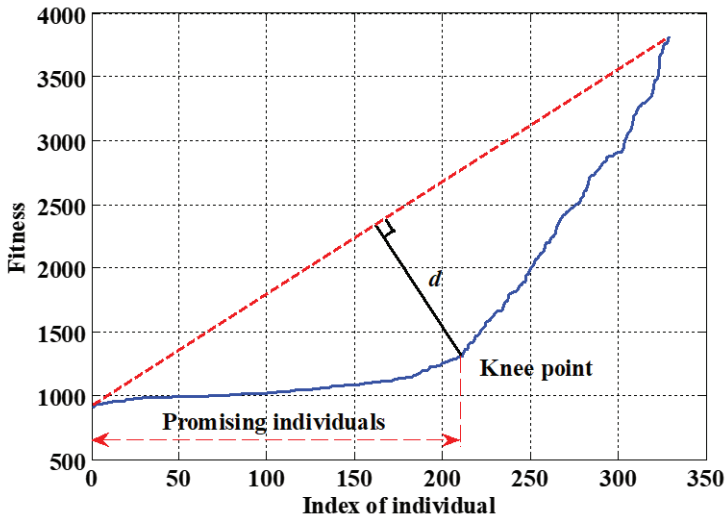

Figure 3.2: The selected promising individuals based on knee-point.

knowledge transfer (line 13 to line 17).

An example of choosing promising individuals from a number of individuals with knee point technique can be found in Figure 3.2. First, the individuals in the population are sorted based on fitness values in ascending order, and a curve related to fitness values is obtained. Second, a line is generated by connecting the points with the smallest and the largest fitness value. Then, the distance between each point on the curve and the line is calculated. The point that has the largest distance to the line is selected as the knee point, and the individuals whose fitness value is smaller than that of the knee point are selected to be adapted.

3.2.3 Algorithm Summary

This chapter develops multi-fidelity surrogate models by shorting DFJSS simulations. Involving lower fidelity surrogate models with simplified DFJSS will reduce the computational cost of GP. However, the evaluations of individuals with lower fidelity surrogate models are often inaccurate. The proposed algorithm deftly addresses this conflicting issue by collabo-

rating the surrogate models with different fidelities. An effective knowledge transfer strategy realises the collaboration mechanism.

3.3 Experiment Design

3.3.1 Simulation Model

The simulation model contains 5000 jobs that need to be processed by 10 machines. Each job has a different number of operations that are randomly generated from a discrete uniform distribution between 1 and 10. The importance of jobs might be different, which are indicated by weights. The weights of 20%, 60%, and 20% jobs are set as 1, 2, and 4 following the setting in [99]. The processing time of each operation is sampled from a uniform discrete distribution with the range [1, 99]. The number of candidate machines for an operation follows a uniform discrete distribution between 1 and 10.

A problem instance is an instantiation of the problem with a particular pseudo-random number generator seed [21]. Multiple different instances will be used to train and test the scheduling heuristics. At each generation, we only use one instance to evaluate the quality of evolved rules. However, the instance will be changed at each generation during the training process by assigning a different random seed to improve the generalisation of the GP algorithm. This strategy has been shown to be useful to improve the effectiveness and generalisation of evolved rules of GP [100, 165].

To verify the effectiveness and efficiency of the proposed algorithm, scenarios with different settings (i.e., different objectives and utilisation levels) are examined. New jobs will arrive over time according to a Poisson process with rate λ. The *utilisation level* (p) is an essential factor to simulate different scenarios. It indicates the proportion of time that a machine is expected to be busy. The expression is shown in Eq. 3.1, where

μ is the average processing time of the machines. P_M is the probability of a job visiting a machine. For example, P_M is 2/10 if each job has two operations. A larger utilisation level leads to a busier and more complex job shop scenario.

$$\lambda = \mu * P_M/p \tag{3.1}$$

The first 1000 jobs are treated as warm-up jobs to get typical situations occurring in a long-term simulation of a dynamic job shop system, and jobs arrive as a continuous arrival process. We collect data from the next 5000 jobs. The simulation stops when the 6000^{th} job is finished.

3.3.2 Comparison Design

The goal of this chapter is to improve the training efficiency of GPHH with collaborative multi-fidelity surrogate models for DFJSS. Two algorithms are involved in this chapter. The GP with multi-tree representation [244] (MTGP) algorithm is selected as the baseline algorithm because it can evolve two rules simultaneously, and its framework is suitable for applying collaborative multi-fidelity surrogate models. The proposed algorithm with collaborative multi-fidelity surrogate models, is named as $\mathrm{M^3GP}$, since it involves multi-population framework, multi-tree representation, and multi-fidelity surrogate models. The algorithms are verified with two surrogate models. It is noted that MTGP works with one population with 1024 individuals while $\mathrm{M^3GP}$ operates with two subpopulations with 1024 individuals (i.e., 512 individuals for each subpopulation). $\mathrm{M^3GP_1}$ and $\mathrm{M^3GP_2}$ can be considered as the algorithms to measure the performance of the evolved rules with the lower surrogate model S_1 and the original model S_2 with the problem to be solved.

The performance of the proposed algorithm is first measured by the comparison between MTGP and $\mathrm{M^3GP_2}$, since we mainly focus on solving the desired problem. The state-of-the-art algorithms in [99] and [176], which is named as $\mathrm{SGP\_K}$ and $\mathrm{SGP\_H}$ in this chapter, are further com-

pared with the proposed algorithm. In addition, the proposed algorithm with more than two surrogates with different fidelities is also studied. In order to verify the effectiveness of the proposed knowledge transfer mechanism for GPHH, the performance of $\mathrm{M^3GP_1}$ with and without the knowledge transfer are compared with each other. Similarly, the effectiveness of knowledge transfer on $\mathrm{M^3GP_2}$ is also further analysed.

Max-flowtime, mean-flowtime and mean-weighted-flowtime are three commonly used objectives. To verify the effectiveness of the proposed algorithm, we expect to test the proposed algorithm on complex scenarios. We set the utilisation level as 0.85 and 0.95, since they can lead to complex job shop scenarios, and are commonly used to evaluate the performance of the proposed algorithm Overall, the proposed algorithms are tested on six different scenarios with the three objectives (i.e., max flowtime, mean flowtime, and mean weighted flowtime) and the two utilisation levels (i.e., 0.85 and 0.95).

3.3.3 Parameter Setting

This section introduces the common parameter settings of GPHH, which are also used for the parameter settings in other chapters if they are not redefined. It is noted that each chapter also has its own specialised parameter settings for GPHH according to the characteristics of the proposed algorithm in the corresponding chapters.

Common Parameter Settings of GPHH

Some common parameter settings of GPHH are shown in Table 3.1 [128]. The individuals are initialised with the ramped-half-and-half method with a minimum depth of 2 and a maximum depth of 6. When generating a GP individual, the rates to the terminal and non-terminal to use are 10% and 90%, respectively. The maximal of the depth of a GP individual is 8. To maintain the quality of the obtained individuals in the evolutionary pro-

Table 3.1: The common parameter settings of GPHH.

Parameter	Value
Method for initialising population	ramped-half-and-half
Initial minimum/maximum depth	2 / 6
Terminal / non-terminal selection rate	10% / 90%
Maximal depth of programs	8
The number of elites for a task	10
Parent selection	Tournament selection with size 7
Crossover / Mutation / Reproduction rate	80% / 15% / 5%
The number of generations	51

cess of GP, the best 10 individuals (i.e., elites) from the last generation are moved to the next generation directly. The rest individuals for the next generation are produced by crossover, mutation, and reproduction (i.e., genetic operators) with a rate of 80%, 15%, and 5%, respectively. Tournament selection with size 7 is used to select the parent(s) for the genetic operators. The algorithm is stopped after 51 generations. In this thesis, the settings of the GPHH parameters in Table 3.1 are the same in all chapters, if they are not redefined in the following chapters. For other parameters, they are specialised in each chapter based on the investigated problems.

The terminals of GP serve as features of the problem to capture sufficient information about the problem. The terminal set of GP in this thesis consists of a number of basic features of machines, jobs and operations in the job shop following the suggestions in [21, 170, 249]. The routing terminal set is set the same as the sequencing terminal set in this thesis.

Machine-related features: the states of machines such as workload are key factors for allocating operations to machines. A good schedule should not overload or underload a machine.

- NIQ is the number of operations in the machine's queue. It is designed to capture the workload of a machine by counting the number of operations in its queue.

- WIQ is the total processing time of the operations in the machine's queue. It is used to capture the workload of a machine by calculating the total processing time required for a machine to finish all the operations in its queue without any delay.

- MWT indicates the waiting time for the machine to become idle again, i.e., the completion time of the current processing on the machine minus the current time.

Job-related features: the states of jobs have a significant effect on deciding which job has a better priority to be processed earlier. A good schedule is expected to process important jobs earlier, and take the current and look-ahead job information into account.

- W is the weight of a job. A job with a larger weight is more important.

- NOR is the number of remaining operations for a job. It reflects the current processing stage of the job.

- WKR is the median processing time needed for the remaining operations. The median time is an estimation of the processing time, since the exact processing time of the operation in DFJSS depends on the machine, and is unknown in advance as the machine is not decided yet. This feature estimates the processing stage of the job in terms of processing time.

- TIS is the time that the job has stayed in the job shop since its arrival.

Operation-related features: the characteristics and states of operations are important factors for choosing the next operation to be processed. A good schedule is supposed to consider the time cost of processing the operation and its waiting time properly.

- PT is the processing time of the operation on the candidate machine.

- NPT is the median processing time of the next operation of the candidate operation (0 if the candidate operation is the last one of the job)

- OWT is the time that the operation has waited in the machine's queue.

GPHH can automatically select proper simple features from the terminals and construct high-level features that are appropriate for a particular problem. The function set is set to $\{+, -, *, /, Max, Min\}$ [170, 242]. The arithmetic operators take two arguments. The "$/$" operator is a protected division, returning one if divided by zero. The Max and Min functions take two arguments and return the maximum and minimum of their arguments, respectively.

In this thesis, all experiments are run on an Arch Linux OS with an Intel (R) Core (TM) i7-4770 CPU at 3.40GHz, with 8-GB RAM. The algorithms are encoded with the java programming language.

Specialised Parameter Settings of GPHH

The specialised parameter settings of GPHH in this chapter are shown in Table 3.2. For simplicity, only two models with different fidelities are mainly considered firstly. Therefore, GP population consists of two subpopulations and the number of individuals is set to 512 for each subpopulation. Borrowing the idea in [176], the designed surrogate model (S_1) is generated by creating a "half shop" job shop with 2500 (i.e., 5000 * 0.5 = 2500) jobs, which reduces the number of jobs to half of the original model but without reducing the number of machines to keep the characteristics of DFJSS. The other model (S_2) is designed by the job shop scenario with 5000 jobs, which is the original model (i.e., can be considered as a surrogate model with 100% accuracy). The only difference between S_1 and S_2 is the number of jobs, and the original model S_2 is more accurate than the surrogate model S_1, since S_2 reflects the problem itself. We set the transfer

Table 3.2: The parameter settings of GPHH.

Parameter	Value
Number of subpopulations	2
Subpopulation size	512
Transfer Ratio tr	0.6
*The number of jobs in surrogate model S_1	2500
*The number of jobs in original model S_2	5000

* for $\mathrm{M^3GP}$ only

ratio tr to 0.6, and the details of the sensitivity analysis of the knowledge transfer ratio are provided in Section 3.5.2.

To make fair comparison, the sizes of intermediate population of $\mathrm{SGP\_H}$, $\mathrm{SGP\_K}$ are set as two times of the population, as suggested in [99]. The half shop surrogate model based on problem approximation is set the same as in [176] but with the maximum number of operations for a job as five for applying the idea properly in the investigated problem. In addition, the number of neighbours for KNN in $\mathrm{SGP\_K}$ is set to 1.

3.4 Results and Discussions

The evolved rule is tested on 50 unseen instances, and the average objective value across the 50 test instances is reported as the test performance of the rule, which can be a good approximation of the true performance of the rule. This thesis works on the minimisation problem, and a smaller objective value indicates a better performance. Friedman's test with a significance level of 0.05 is applied to compare all the algorithms based on their performance. If Friedman's test gives significance results, we will further conduct Wilcoxon rank-sum test with Bonferroni correction between the proposed algorithm and other algorithms with a significance level of 0.05 for pairwise comparisons with at least 30 independent runs (i.e., 30 runs

Table 3.3: The mean (standard deviation) of the **training time** (in minutes) of MTGP and $\mathrm{M^3GP}$ according to 30 independent runs in six DFJSS scenarios.

Scenario	MTGP	$\mathrm{M^3GP}$
<Fmax, 0.85>	64(9)	51(9)(–)
<Fmax, 0.95>	67(12)	53(9)(–)
<Fmean, 0.85>	61(11)	48(8)(–)
<Fmean, 0.95>	64(13)	49(6)(–)
<WFmean, 0.85>	62(13)	49(7)(–)
<WFmean, 0.95>	63(11)	49(6)(–)

and 50 runs). For all the results in this thesis, "–", "+", and "≈" indicate the corresponding result is significantly better than, worse than or similar to its counterpart. An algorithm is compared with the algorithm(s) before it one by one, as shown in the tables in the Results and Discussions section. The results in other chapters are also compared in the same way.

3.4.1 Training Time

Table 3.3 shows the mean and standard deviation of the training time of MTGP and $\mathrm{M^3GP}$ based on 30 independent runs in six DFJSS scenarios. The training time of the proposed algorithm $\mathrm{M^3GP}$ is significantly shorter than that of MTGP in all the examined scenarios. For example, the training time of $\mathrm{M^3GP}$ is reduced by 23.40% in the scenario <Fmean, 0.95>. In general, $\mathrm{M^3GP}$ is more efficient than MTGP, and the training time of $\mathrm{M^3GP}$ is roughly 78.5% of that of MTGP in all the tested scenarios.

Figure 3.3 shows the curve of the training time of MTGP and $\mathrm{M^3GP}$ during the training process in six different scenarios. It shows that the training time of $\mathrm{M^3GP}$ is shorter than MTGP at all generations in all the scenarios. It indicates that $\mathrm{M^3GP}$ can successfully save more computational cost throughout the entire evolutionary process. In addition, the

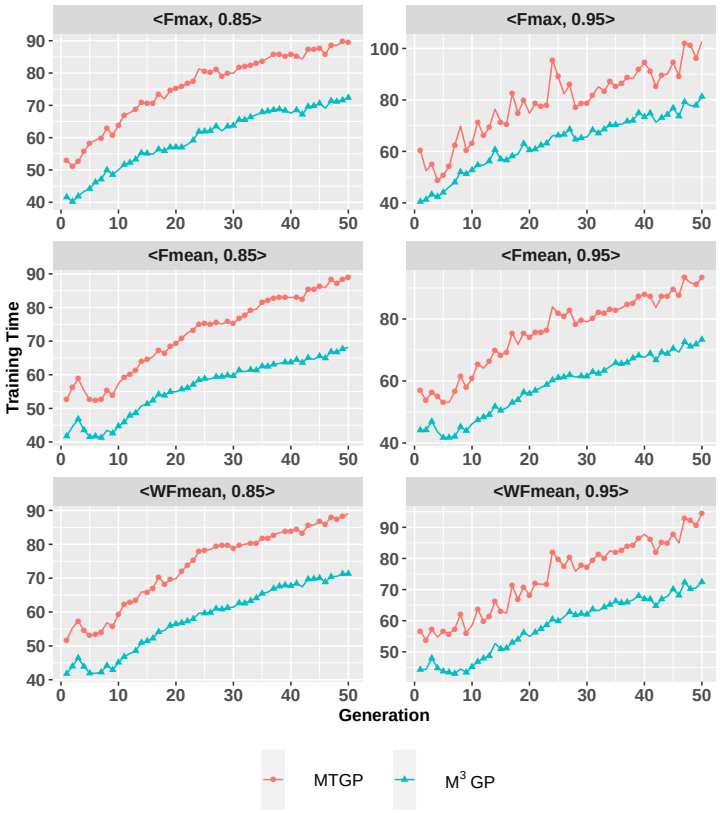

Figure 3.3: The curve of the **training time** (in seconds) of MTGP and M^3GP during the training process according to 30 independent runs in six DFJSS scenarios.

training time of M^3GP is increasing more slowly than that of MTGP. As the number of generation increases, the training time of M^3GP grows from 40 to 70 seconds roughly in all scenarios. However, the training time of MTGP increases from about 50 to 90 seconds in the scenarios with utilisation level as 0.85 (i.e., <Fmax, 0.85>, <Fmean, 0.85> and <WFmean, 0.85>) while rises from 55 to 95 roughly in the scenarios with utilisation level as 0.95 (i.e., <Fmax, 0.95>, <Fmean, 0.95> and <WFmean, 0.95>).

More training time is normally needed in the scenarios with higher utilisation levels than the scenarios with lower utilisation levels, since the

corresponding job shop environments in the scenarios with higher utilisation levels are more complicated. Specifically, compared with the training time needed in scenarios with utilisation level of 0.85, for MTGP, more training time is needed in the scenarios with utilisation level of 0.95. However, this is not the case for $\mathrm{M^3GP}$. This indicates that the training time of $\mathrm{M^3GP}$ is not sensitive to the utilisation level of the job shop. One possible reason is that the surrogate models with lower fidelities are less time-consuming, since the corresponding simulations are shorter. The simpler problem with even a higher utilisation level does not significantly impact the training time.

3.4.2 Quality of the Evolved Scheduling Heuristics

Table 3.4 shows the mean and standard deviation of the objective values on unseen instances of MTGP and $\mathrm{M^3GP_2}$ with the same number of generations over 30 independent runs in six different scenarios. It shows that there is no statistical difference in the performance between MTGP and $\mathrm{M^3GP_2}$ in five out of the six scenarios. In addition, $\mathrm{M^3GP_2}$ performs significantly better than its counterpart in scenario <Fmax, 0.95>. This shows that $\mathrm{M^3GP_2}$ can achieve similar or even better performance than MTGP with a less computational cost.

It is also interesting to know whether the performance of $\mathrm{M^3GP_2}$ can be better than MTGP if the same training time is given. To answer this question, we set a fixed training time for all the algorithms. In this case, the number of generations in each run for one algorithm can be different, and we cannot compare with the algorithms based on generations any more. To make a fair comparison, we process the results by choosing the compared data properly before comparison. We aim to choose data at the same or similar time points from each run of the compared algorithms. In addition, the number of chosen data in each run for all the compared algorithms is expected to be equal. We use $time$ to indicate the training

Table 3.4: The mean (standard deviation) of the objective values on test instances of MTGP and $\mathrm{M^3GP_2}$ **with the same number of generations** over 30 independent runs in six DFJSS scenarios.

Scenario	MTGP	$\mathrm{M^3GP_2}$
<Fmax, 0.85>	1235.73(41.27)	1232.39(31.70)(≈)
<Fmax, 0.95>	1967.24(65.18)	1932.22(42.22)**(–)**
<Fmean, 0.85>	384.55(1.04)	385.98(2.98) (≈)
<Fmean, 0.95>	555.32(9.91)	552.50(5.07)(≈)
<WFmean, 0.85>	831.30(7.32)	831.24(5.22)(≈)
<WFmean, 0.95>	1114.93(13.80)	1114.36(8.88)(≈)

time budget, and the results are equally divided into g groups. The average period time in each group is $\frac{time}{g}$, and the demarcation points of training time of g groups are $\frac{time}{g} * 1$, $\frac{time}{g} * 2$, ..., $\frac{time}{g} * g$. According to the demarcation points, the closest recorded time is identified to map the compared data. It is noted that there are inevitable errors to measure the performance of the algorithm in this way, since the compared data is not obtained with exactly the same number of evaluations. Fortunately, these data are still representative to measure the performance of the algorithms since the sampling time is similar for different runs of one algorithm and different algorithms.

We limit the training time to 80 minutes for MTGP and $\mathrm{M^3GP}$, since the training time of the baseline MTGP with 51 generations is about 80 minutes. We set the number of groups g to 20 that can get enough data for investigating the objective values along with training time of MTGP, $\mathrm{SGP\_H}$, $\mathrm{SGP\_K}$, and $\mathrm{M^3GP_2}$. The objective values around 80 minutes are used to measure the performance of the involved algorithms. Table 3.5 shows the mean and standard deviation of the objective values on unseen instances of MTGP, $\mathrm{SGP\_H}$, $\mathrm{SGP\_K}$, and $\mathrm{M^3GP_2}$ with the same training time over 30 independent runs in the six test scenarios. First, $\mathrm{SGP\_H}$, $\mathrm{SGP\_K}$, and $\mathrm{M^3GP_2}$ are compared with MTGP, respectively. Second, $\mathrm{M^3GP_2}$ is com-

Table 3.5: The mean (standard deviation) of the objective values on test instances of MTGP, SGP_H, SGP_K, and M^3GP_2 **with the same training time** of 80 minutes over 30 independent runs in six DFJSS scenarios.

Scenario	MTGP	SGP_H	SGP_K	M^3GP_2
<Fmax, 0.85>	1225.58(44.21)	1267.49(40.76)**(+)**	1239.48(40.73)(≈)	1212.25(28.60)**(–)(–)(–)**
<Fmax, 0.95>	1963.85(61.53)	1981.81(52.60)(≈)	1956.72(29.60)(≈)	1925.87(28.98)**(–)(–)(–)**
<Fmean, 0.85>	384.20(0.93)	387.24(4.22)**(+)**	386.74(3.32)**(+)**	384.61(1.25)(≈)**(–)(–)**
<Fmean, 0.95>	554.62(9.79)	554.75(6.71)(≈)	554.07(8.18)(≈)	550.56(3.32)**(–)(–)(–)**
<WFmean, 0.85>	830.36(7.09)	834.79(8.13)**(+)**	830.40(5.52)(≈)	829.25(3.37)**(–)(–)**(≈)
<WFmean, 0.95>	1112.40(11.34)	1115.75(13.46)(≈)	1110.21(11.15)(≈)	1109.14(5.47)(≈)**(–)**(≈)

pared with SGP_H and SGP_K, respectively. With the same training time, compared with MTGP, SGP_H and SGP_K perform significantly worse in three and two scenarios, respectively. However, M^3GP_2 can achieve significantly better performance than MTGP in four out of six scenarios (i.e., <Fmax, 0.85>, <Fmax, 0.95>, <Fmean, 0.95> and <WFmean, 0.85>). M^3GP_2 is no worse than MTGP in all the other scenarios. This verifies the effectiveness of the proposed algorithm.

Figure 3.4 shows the curve of average objective values according to 30 independent runs on unseen instances of MTGP, SGP_H, SGP_K, and M^3GP_2 with the same training time in the six DFJSS scenarios. With the same training time, the proposed algorithm M^3GP_2 can converge faster than the compared algorithms in all the scenarios. The performance of M^3GP_2 becomes better than the compared algorithms after about 10 minutes in scenario <Fmean, 0.85> and <Fmean, 0.95>. In addition, M^3GP_2 performs better than its counterparts after 20 minutes roughly in scenario <Fmax, 0.95> and <WFmean, 0.85>. It is noted that SGP_H performs worse than SGP_K, this is consistent with our expectations since the evaluations with half shop surrogate in [176] is more time consuming than the KNN surrogate in [99].

Overall, the proposed algorithm can improve the efficiency of GPHH

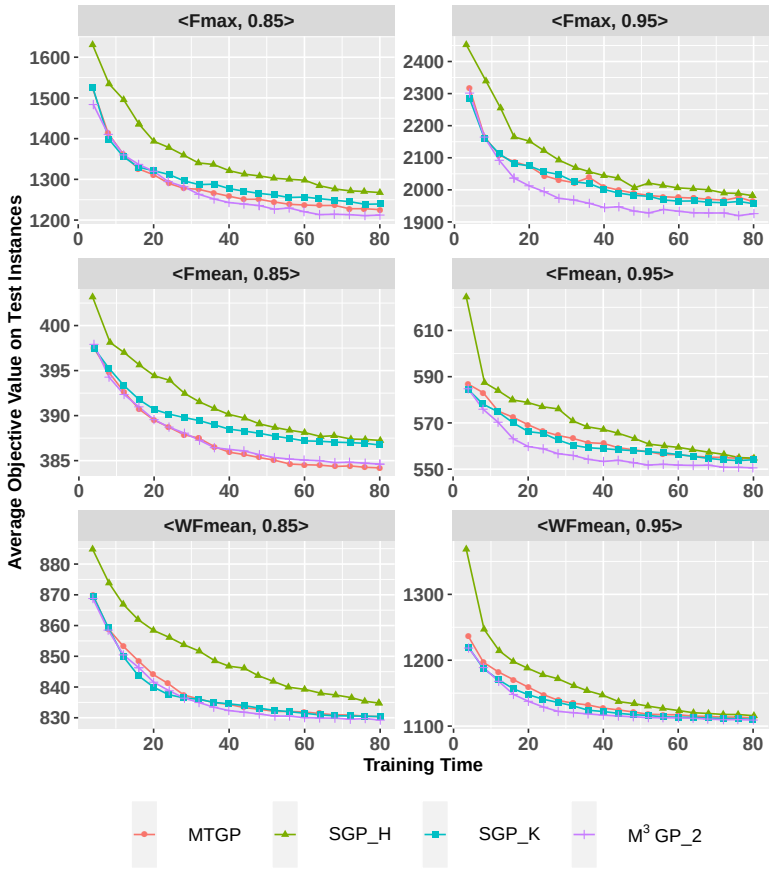

Figure 3.4: The curve of average objective values according to 30 independent runs on test instances of MTGP, $\mathrm{SGP\_H}$, $\mathrm{SGP\_K}$, and $\mathrm{M^3GP_2}$ **with the same training time** (in minutes) in six DFJSS scenarios.

for DFJSS by reducing the computational cost without losing its performance, and thus speeding up its convergence. Given the same training time, the proposed algorithm can achieve significantly better performance than the compared state-of-the-art GPHH algorithms with surrogate in most scenarios while no worse in all the scenarios.

Table 3.6: The mean (standard deviation) of the objective values of $\mathrm{M^3GP_1}$ and $\mathrm{M^3GP_2}$ **with and without knowledge transfer** with the same number of generations on test instances according to 30 independent runs in six DFJSS scenarios.

Scenario	$\mathrm{M^3GP_1(without)}$	$\mathrm{M^3GP_1(with)}$	$\mathrm{M^3GP_2(without)}$	$\mathrm{M^3GP_2(with)}$
<Fmax, 0.85>	1201.44(60.42)	1170.31(29.48)**(–)**	1261.85(65.40)	1232.39(31.70)**(–)**
<Fmax, 0.95>	1815.72(76.82)	1758.22(34.88)**(–)**	2001.56(80.43)	1932.22(42.22)**(–)**
<Fmean, 0.85>	390.24(4.21)	388.14(2.98)**(–)**	387.98(4.10)	385.98(2.98)**(–)**
<Fmean, 0.95>	566.98(7.81)	561.28(5.30)**(–)**	558.10(7.38)	552.50(5.07)**(–)**
<WFmean, 0.85>	840.19(9.50)	835.37(5.21)**(–)**	836.31(9.29)	831.24(5.22)**(–)**
<WFmean, 0.95>	1149.64(25.43)	1135.75(9.23)**(–)**	1130.06(25.95)	1114.36(8.88)**(–)**

3.4.3 Effectiveness of Knowledge Transfer Mechanism

In order to examine the effectiveness of the proposed knowledge transfer mechanism, several experiments are conducted. We set the transfer ratio tr to zero in $\mathrm{M^3GP}$ to disable the knowledge transfer between different subpopulations. $\mathrm{M^3GP_1(without)}$ and $\mathrm{M^3GP_2(without)}$ indicate that there is no knowledge transfer between subpopulations while $\mathrm{M^3GP_1(with)}$ and $\mathrm{M^3GP_2(with)}$ indicate that there is knowledge transfer between subpopulations with a tr of 0.6.

Table 3.6 shows the mean and standard deviation of the objective values of $\mathrm{M^3GP}$ with and without knowledge transfer with the same number of generations on test instances according to 30 independent runs in the six scenarios. With knowledge transfer, the performance of the evolved rules with multi-fidelity surrogate models is significantly better than its counterpart without knowledge transfer. To be specific, the performance of $\mathrm{M^3GP_1(with)}$ is better than $\mathrm{M^3GP_1(without)}$ while the performance of $\mathrm{M^3GP_2(with)}$ is better than $\mathrm{M^3GP_2(without)}$ in all the scenarios. This indicates that the knowledge transfer can benefit both involved problems with different complexities. The knowledge obtained with the simpler surro-

Table 3.7: The mean (standard deviation) of the objective values on test instances of MTGP and $\mathrm{M^3GP_2(without)}$ according to 30 independent runs in six DFJSS scenarios.

Scenario	MTGP	$\mathrm{M^3GP_2(without)}$
<Fmax, 0.85>	1235.73(41.27)	1261.85(65.40)(+)
<Fmax, 0.95>	1967.24(65.18)	2001.56(80.43)(+)
<Fmean, 0.85>	384.55(1.04)	387.98(4.10)(+)
<Fmean, 0.95>	555.32(9.91)	558.10(7.38)(+)
<WFmean, 0.85>	831.30(7.32)	836.31(9.29)(+)
<WFmean, 0.95>	1114.93(13.80)	1130.06(25.95)(+)

gate model is useful for more complex surrogate models. The knowledge derived from complex surrogate model is also beneficial to the simpler surrogate model. This confirms the effectiveness of the proposed knowledge transfer mechanism.

Table 3.7 shows the mean and standard deviation of the objective values on unseen instances of MTGP and $\mathrm{M^3GP_2}$ without knowledge transfer in the six scenarios. It shows that the performance of $\mathrm{M^3GP_2}$ without knowledge transfer is significantly worse than that of MTGP. It verifies the effectiveness of the proposed knowledge transfer mechanism. In addition, it is in line with our expectation, since more computational resources are used for solving the desired problem in MTGP. To be specific, without knowledge transfer, the number of individuals for solving the desired problem in $\mathrm{M^3GP_2(without)}$ is 512, which is only half of the number of individuals in MTGP.

Figure 3.5 shows the curve of average objective values on unseen instances of $\mathrm{M^3GP_1(without)}$ and $\mathrm{M^3GP_1(with)}$ based on 30 independent runs in the six different DFJSS scenarios. It is obvious that the performance of $\mathrm{M^3GP_1(with)}$ is better than that of $\mathrm{M^3GP_1(without)}$ after generation 5 roughly in the max-flowtime related scenarios (i.e., <Fmax, 0.85> and <Fmax, 0.95>) and after about 10 generations in mean-flowtime and

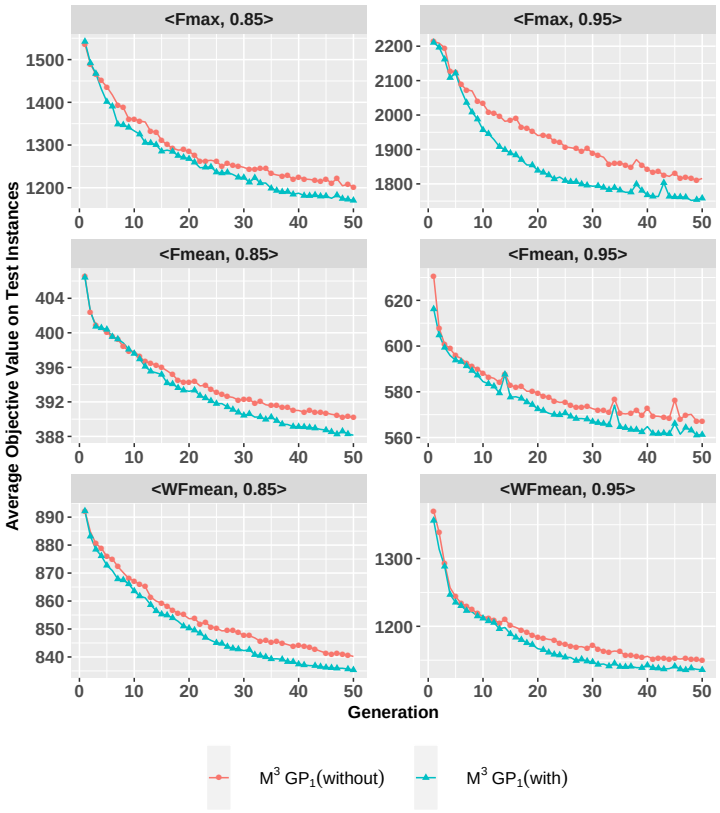

Figure 3.5: The curve of average objective values on test instances of $\mathrm{M^3GP_1(without)}$ and $\mathrm{M^3GP_1(with)}$ according to 30 independent runs in six DFJSS scenarios.

weighted mean-flowtime related scenarios (i.e., <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>). It may be because the individuals in the population before generation 5 or generation 10 have not reached good quality yet, and the transferred knowledge does not have sufficient contribution to the other subpopulation. Fortunately, the transferred knowledge before generation 5 or 10 does not have a negative effect on the other subpopulation. The same pattern is also found between $\mathrm{M^3GP_2(without)}$ and $\mathrm{M^3GP_2(with)}$, which is shown in Figure 3.6.

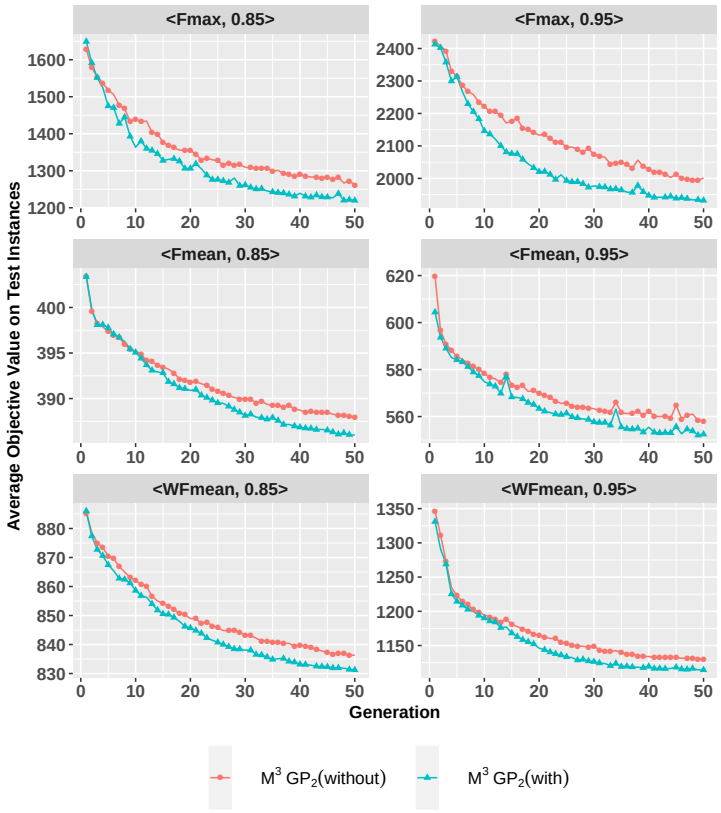

Figure 3.6: The curve of average objective values on test instances of $\mathrm{M^3GP_2(without)}$ and $\mathrm{M^3GP_2(with)}$ according to 30 independent runs in six DFJSS scenarios.

3.5 Further Analyses

To deeply understand the effectiveness of the proposed algorithm, we have conducted in-depth analyses. The proposed algorithm with more surrogate models with different fidelities and the sensitivity analysis of knowledge transfer ratio, are further analysed in this section.

Table 3.8: The settings of the number of individuals/jobs of the proposed algorithm with two, three, four, and five surrogates.

Algorithm	$Subpop1$	$Subpop2$	$Subpop3$	$Subpop4$	$Subpop5$
$\mathrm{M^3GP_2}$	512/2500	512/5000	–	–	–
$\mathrm{M^3GP_3}$	256/1667	256/3333	512/5000	–	–
$\mathrm{M^3GP_4}$	170/1250	171/2500	171/3750	512/5000	–
$\mathrm{M^3GP_5}$	128/1000	128/2000	128/3000	128/4000	512/5000

3.5.1 Number Analysis of Multi-fidelity Surrogate Models

It is interesting to investigate whether the collaboration between more models with different fidelities can benefit problem-solving or not. To ensure the performance of the algorithm for the problem to be solved, half population individuals in the population are kept for optimising the desired problem. The other half population individuals are divided equally for solving simplified problems with simpler surrogate models. The number of jobs between surrogate models with different fidelities follows an arithmetic sequence with an upper bound as 5000. $\mathrm{M^3GP_2}$, $\mathrm{M^3GP_3}$, $\mathrm{M^3GP_4}$, and $\mathrm{M^3GP_5}$ denote the corresponding algorithms on the desired problem, respectively. Table 3.8 shows the settings of the number of individuals and jobs of the proposed algorithm with one, two, three, and four surrogates.

Table 3.9 shows the mean and standard deviation of the training time of the involved algorithms with the same number of generations according to 30 independent runs in six different scenarios. Compared with $\mathrm{M^3GP_2}$, as the number of surrogate models increases, the training time of $\mathrm{M^3GP_3}$, $\mathrm{M^3GP_4}$, and $\mathrm{M^3GP_5}$ has no significant difference in most scenarios.

Table 3.10 shows the mean and standard deviation of the objective values of $\mathrm{M^3GP_2}$, $\mathrm{M^3GP_3}$, $\mathrm{M^3GP_4}$, and $\mathrm{M^3GP_5}$ on unseen instances with the same number of generations according to 30 independent runs in six different scenarios. In terms of the objective values on the unseen data, there is no significant difference among the compared algorithms in most sce-

Table 3.9: The mean (standard deviation) of the training time (in minutes) of the involved algorithms with the same number of generations based on 30 independent runs in six DFJSS scenarios.

Scenario	M^3GP_2	M^3GP_3	M^3GP_4	M^3GP_5
<Fmax, 0.85>	51(9)	48(9)(≈)	46(8)(≈)	48(8)(≈)
<Fmax, 0.95>	53(9)	48(7)**(–)**	47(7)**(–)**	49(7)**(–)**
<Fmean, 0.85>	48(8)	43(4)**(–)**	47(8)(≈)	47(7)(≈)
<Fmean, 0.95>	49(6)	47(7)(≈)	48(6)(≈)	46(7)(≈)
<WFmean, 0.85>	49(7)	48(7)(≈)	48(6)(≈)	47(8)(≈)
<WFmean, 0.95>	49(6)	47(9)(≈)	47(8)(≈)	47(7)(≈)

Table 3.10: The mean (standard deviation) of the objective values on test instances of M^3GP_2, M^3GP_3, M^3GP_4, and M^3GP_5 with the same number of generations according to 30 independent runs in six DFJSS scenarios.

Scenario	M^3GP_2	M^3GP_3	M^3GP_4	M^3GP_5
<Fmax, 0.85>	1232.39(31.70)	1228.03(37.98)(≈)	**1222.44(29.73)**(≈)	1230.54(33.47)(≈)
<Fmax, 0.95>	**1932.22(42.22)**	1948.53(45.67)(≈)	1960.67(120.22)(≈)	1954.50(68.28)(≈)
<Fmean, 0.85>	385.98(2.98)	385.33(2.06)(≈)	**384.93(1.73)**(≈)	385.59(3.11)(≈)
<Fmean, 0.95>	**552.50(5.07)**	553.43(5.30)(≈)	554.02(4.84)(≈)	555.82(6.76)**(+)**
<WFmean, 0.85>	831.24(5.22)	830.83(6.50)(≈)	831.36(6.56)(≈)	**830.79(4.27)**(≈)
<WFmean, 0.95>	**1114.36(8.88)**	1119.21(15.79)**(+)**	1122.82(17.39)**(+)**	1117.59(12.98)(≈)

narios. In one of the scenarios of M^3GP_3, M^3GP_4, and M^3GP_5, the performance is significantly worse than that of M^3GP_2. In terms of the mean and standard deviation, the performance of M^3GP_2 is better than other compared algorithms in half of the scenarios (i.e., <Fmax, 0.95>, <Fmean, 0.95>, and <WFmean, 0.95>) as shown in bold. In addition, the examined problems are not very sensitive to the number of surrogates as the performance of the proposed algorithm with different number of surrogates achieve similar performance.

Overall, the results show that the proposed algorithm with two surro-

gates achieves the best performance with the settings in this chapter. A possible reason is that the additional surrogate models in the experiments were not accurate enough, and thus introduced more noise than the first two surrogate models. The accuracy of surrogates can be different for different problem complexities, such as utilisation levels. In our case, we observe that using two surrogate models is a proper choice. There are several interesting but challenging questions are worth studying in the future. First, how to decide the optimal number of surrogates. Second, how to design efficient surrogate models according to the domain knowledge or information from the evolutionary process. Last but not least, how to design effective knowledge transfer mechanisms for a large number of surrogates since the interaction between more surrogates is even complex. However, this is out of the scope of this study, and we would like to investigate it in the future.

3.5.2 Sensitivity Analysis of Knowledge Transfer Ratio

The transfer ratio decides the frequency to transfer knowledge between different problems at each generation, and is further analysed in this subsection. Figure 3.7 shows the curve of average objective values on unseen instances of $\mathrm{M^3GP_2}$ with different transfer ratios obtained by the 30 independent runs in six different scenarios. The performance of $\mathrm{M^3GP_2}$ with different transfer ratios is almost the same in the scenarios <Fmax, 0.95>, <WFmean, 0.85>, and <WFmean, 0.95>. In the scenarios <Fmax, 0.85>, <Fmean, 0.85>, and <Fmean, 0.95>, there are some slight differences between the algorithms with different transfer ratios. From an overall perspective, $\mathrm{M^3GP_2}$ with a transfer ratio of 0.6 has a slightly better performance than its counterparts. Therefore, this chapter sets the transfer ratio to 0.6 to $\mathrm{M^3GP}$ to compare with other algorithms, as we mentioned earlier. However, in general, the performance of $\mathrm{M^3GP_2}$ is not sensitive to the transfer ratio.

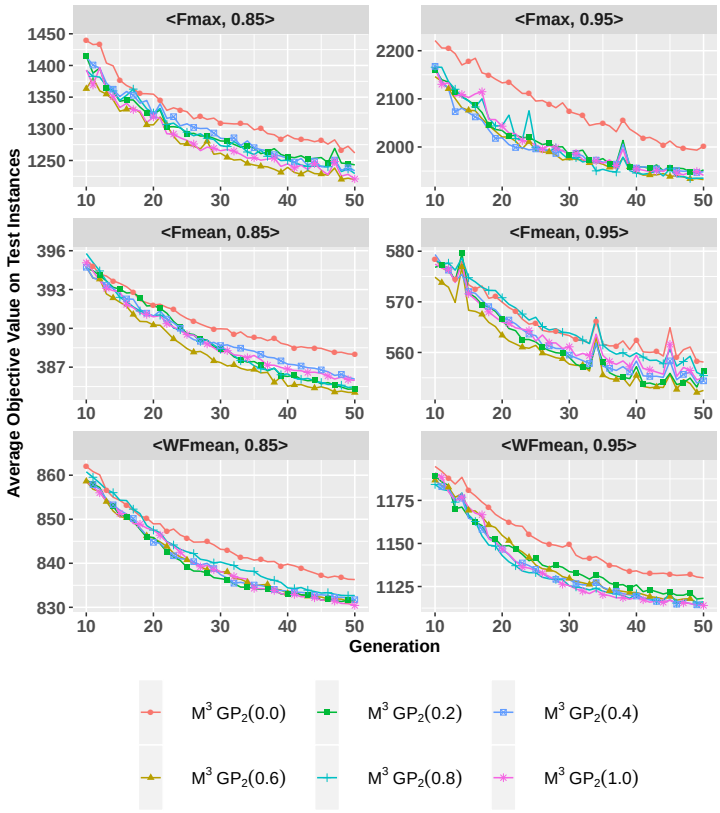

Figure 3.7: The curve of average objective values on test instances of $\mathrm{M^3GP_2}$ with different transfer ratios over 30 independent runs in six DFJSS scenarios.

3.6 Chapter Summary

The goal of this chapter is to develop an effective strategy that collaborates multi-fidelity surrogate models to improve the efficiency of GPHH to evolve scheduling heuristics automatically for DFJSS. The purpose is successfully achieved by proposing an effective collaboration framework in GP that allows the subpopulations with different surrogate models to learn from each other, and developing an effective knowledge transfer mechanism.

The results show that the proposed algorithm $\mathrm{M^3GP_2}$ can dramatically reduce the computational time of GPHH without losing its performance. Within the same training time, $\mathrm{M^3GP_2}$ can achieve significantly better performance in most of the scenarios, while no worse than its counterpart in all the scenarios. The efficiency of the proposed algorithm is verified by comparing the training time. The effectiveness of the proposed algorithm is verified by comparing the quality of evolved scheduling heuristics and the analysis of knowledge transfer mechanism. In summary, $\mathrm{M^3GP_2}$ can successfully improve the efficiency of GPHH, and achieve effective scheduling heuristics automatically for DFJSS. The proposed algorithm shows its superiority compared with the state-of-the-art surrogate-assisted GPHH algorithms for the JSS problems.

This chapter mainly uses surrogate techniques to improve the training efficiency and effectiveness of GPHH for DFJSS. In the next chapter, surrogate techniques will be used with other techniques to improve the efficiency for feature selection in GPHH for DFJSS.

Chapter 4

Search Space Reduction with Feature Selection

In Chapter 3, we investigate how to use surrogate techniques in GPHH for DFJSS. This chapter will focus on using surrogates along with individual adaptation and phenotypic characterisation techniques to reduce the search space of GPHH for evolving effective scheduling heuristics only with the selected features efficiently.

4.1 Introduction

A GP individual is a priority function, typically represented as a tree. GP evolves a population of such trees using a terminal set (i.e., the leaf nodes, reflecting the features of the job shop state) and a function set (i.e., non-leaf nodes, indicating the operations to combine the features in the priority function). The terminal set is a critical factor in the success of GPHH [220]. A compact terminal set can improve the effectiveness of GPHH. In DFJSS, a wide range of features about the job shop state (e.g., the processing time of each operation and the idle time of each machine) can be considered as terminals. However, the importance of a feature depends on job shop scenarios and objectives to be optimised. In practice, it is usually unknown

which features are useful, and which features are not important. Therefore, existing studies typically place all the possible job shop features in the terminal set. As a result, the evolved rules tend to have a large number of different features, making it hard to interpret the rules [81]. Besides, a large terminal set with redundant or unrelated features leads to exponentially large and noisy search space, and negatively affects the capability of GP in searching the solution space. This chapter aims to develop feature selection algorithms to reduce the search space of GPHH for DFJSS.

The feature selection in GPHH for DFJSS is different from and more challenging than in the traditional machine learning tasks, as the data is not available in advance, and should be generated before applying feature selection. The current state-of-the-art feature selection approach [152] used a short GP process with surrogate and niching techniques to evolve a diverse set of good GP individuals as the data for feature selection. Then the features were selected based on their importance to these individuals. There are two types of information obtained from the feature selection process [152]. The first is the selected features, and the second is the promising individuals found during the feature selection process. Most existing algorithms [89, 211] only use the former one and re-initialise the population using the selected features. This may not be effective, since the obtained information (e.g., evolved individual structures) during the feature selection process, except for the selected features, is not fully utilised.

4.1.1 Chapter Goals

The goal of this chapter is to *reduce the search space of GPHH with feature selection to evolve scheduling heuristics for DFJSS*. First, this chapter develops a new GPHH algorithm with feature selection for DFJSS, which aims at making feature selection for the two terminal sets (i.e., one for the routing rule, and the other for the sequencing rule) simultaneously. Second, individual adaptation strategies are proposed to utilise the information of the

selected features and examined individuals used in the feature selection stage. The proposed algorithms are expected to help GPHH find the routing and sequencing rules with only the selected features without sacrificing the performance. Specifically, this chapter has the following research objectives:

1. Propose a novel two-stage framework for GPHH with feature selection to evolve routing and sequencing rules simultaneously.

2. Develop a new two-stage GPHH algorithm to utilise the information of both the selected features and the examined individuals obtained from the feature selection process.

3. Propose novel individual adaptation strategies that inherit the information of the examined individuals obtained from the feature selection.

4. Analyse how the proposed algorithms influence the effectiveness and sizes of the evolved rules.

5. Analyse how the proposed individual adaptation strategies influence the efficiency of the proposed algorithms.

4.1.2 Chapter Organisation

The rest of this chapter is organised as follows. Detailed descriptions of the proposed algorithm are given in Section 4.2. The experiment design is shown in Section 4.3, followed by results and discussions in Section 4.4. Further analyses are conducted in Section 4.5. Finally, Section 4.6 concludes this chapter.

4.2 Proposed Algorithm

This section describes the proposed two-stage GPHH algorithm with feature selection for DFJSS. The framework of the proposed algorithm is first illustrated, followed by the details of its key components.

4.2.1 Proposed Two-stage GPHH with Feature Selection

The flowchart of the proposed approach is shown in Figure 4.1. The main steps, which are evaluation, selection and evolution, are the same as the classical GPHH algorithm. The difference is that there is a checkpoint (i.e., generation 50) for separating the whole GPHH process into two stages. *In the first stage* (i.e., before generation 50), GPHH proceeds with a niching evaluator and a surrogate model. *The output of stage 1 is a population for feature selection.* At generation 50, feature selection mechanism will be invoked to select two subsets of informative terminals for evolving routing and sequencing rules, respectively. Then, the terminal sets for evolving routing and sequencing rules are reset to the selected subsets. *In the second stage* (i.e., after generation 50), the population will be evaluated without niching and surrogate techniques. The mutation operator samples only from the selected terminals rather than the entire terminal set while generating the new sub-tree. The main process of the second stage is the same as the classical GPHH.

In conclusion, the two-stage GPHH approach contains two consecutive phases. The first stage is mainly for feature selection. Using the niching and surrogate techniques help GP quickly reach a diverse set of reasonably good individuals. In the second stage, the obtained information in the first stage is well utilised by inheriting the final population of stage 1. In addition, the selected features are used to guide the search space during the subsequent evolutionary search.

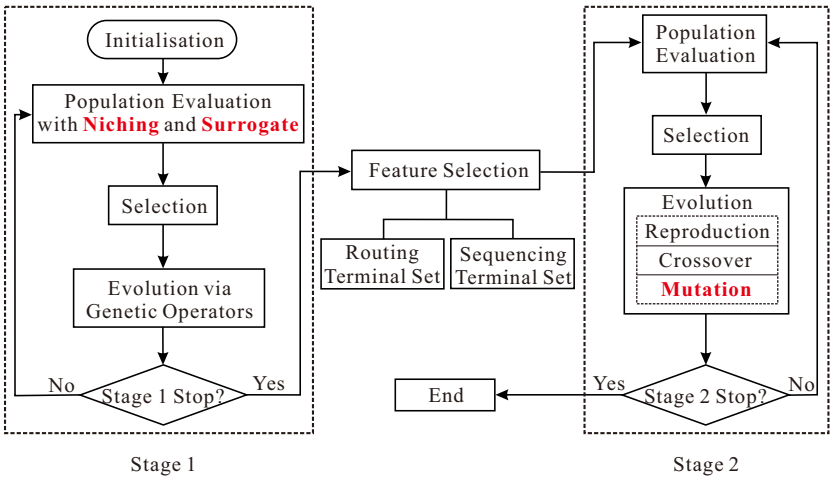

Figure 4.1: The flowchart of two-stage GPHH with feature selection for DFJSS.

4.2.2 Niching and Surrogate

The GPHH with feature selection is more computationally expensive compared with the classical GPHH algorithm because of the extra individual evaluations in the first stage for feature selection of the algorithm. To reduce extra computing costs, a niching based and surrogate assisted algorithm was proposed for feature selection in [152]. The algorithm aims to get a diverse set of good individuals quickly for feature selection. The niching technique [195] maintains the diversity of the population, and the surrogate is to speed up the evaluations. Briefly speaking, the niching technique maintains the diversity of individuals by building different niches and controlling the number of individuals in each niche. The surrogate technique was designed based on the assumption that the knowledge of solving simpler or auxiliary problems can be transferred to the original problems [176]. This chapter applies the idea but explores it to the DFJSS. In the original simulation, there are 5000 jobs and 10 machines. In the surrogate model, we shorten the simulation to 500 jobs and 5 machines.

Algorithm 5: Feature selection

Input : A diverse set of good individuals ($baseInds$) from stage 1
Output: The selected features F

```
1:  set F ← {}
2:  for i = 1 to |features| do
3:      vote_{f_i} ← 0 // the number of votes for feature f_i
4:      for j = 1 to |baseInds| do
5:          Calculate the contribution C_{f_i} of feature f_i
6:          ind ← baseInds_j
7:          if C_{f_i}^{ind} > 0 then
8:              vote_{f_i} ← vote_{f_i} +1
9:          end
10:     end
11:     if vote_{f_i} > 0.5 * |baseInds| then
12:         T ← T ∪ f_i
13:     end
14: end
15: return F
```

4.2.3 Feature Selection

There are three main steps of the feature selection algorithm. First, the top 10 individuals in the population based on fitness values are selected as a diverse set of good individuals $baseInds$. Second, the importance of each feature is measured according to its contributions to the fitness of the individuals in $baseInds$, and an individual in $baseInds$ will vote for a feature if the feature has contributions to it. Finally, if a feature can get more than half of the votes, the feature will be selected. The pseudo-code of feature selection is shown in Algorithm 5.

The Importance of Features. The importance of a feature f is measured by its contributions to a set of individuals $baseInds$ (from line 4 to line 10). To calculate the contribution of a feature f to an individual r (i.e., denoted by C_f^r), the feature f is first replaced with the constant of one, then the contribution is calculated as the difference between the fitness before

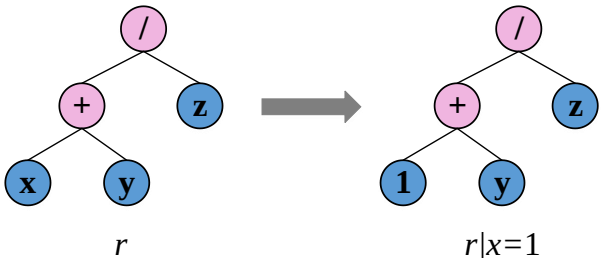

Figure 4.2: An example of how to examine the contribution (denoted as C_x) of a feature x for an individual r.

and after the replacement, as shown in Eq. 4.1.

$$C_f^r = fitness(r|f = 1) - fitness(r) \tag{4.1}$$

The DFJSS problem investigated in this thesis is a minimisation problem. Therefore, if $C_f > 0$, it means the fitness becomes worse without the measured feature, and the measured feature is important. Thus, the measured feature can get one vote from the individual r.

Figure 4.2 shows an example of a GP individual r with three features (x, y, and z). To examine the importance of feature x, x is firstly be replaced with 1, and the contribution of feature x is defined as $C_x^r = fitness(r|x = 1) - fitness(r)$.

Feature Selection Decision. This chapter makes two extensions of the feature selection algorithm in [152] to fit the DFJSS problems. First, two sets of individuals obtained from the two subpopulations for evolving routing rules and sequencing rules are selected, respectively. Second, the feature selection algorithm is applied for selecting the routing feature set and the sequencing feature set based on the two sets of individuals, respectively. Feature f is selected if it makes positive contributions to at least 50% of the selected individuals $baseInds$ (from line 11 to line 13).

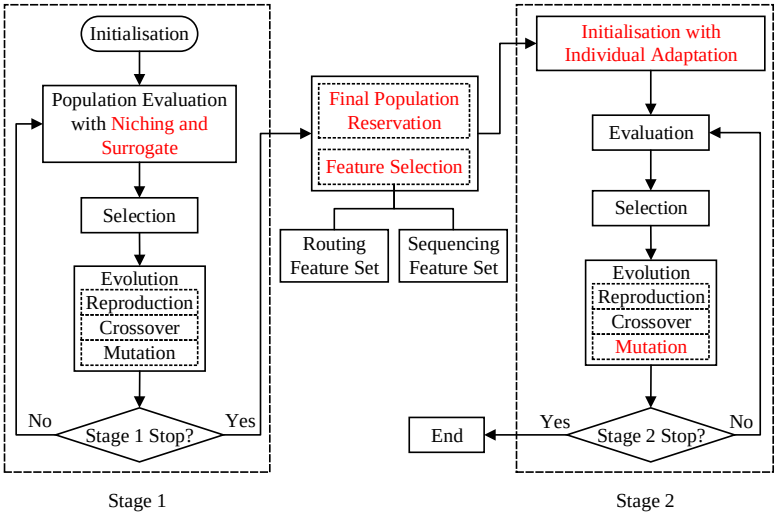

Figure 4.3: The flowchart of two-stage GPHH feature selection algorithm with individual adaptation strategies (i.e., the reddish font parts are the main steps of the proposed algorithm).

4.2.4 GPHH Feature Selection with Proposed Individual Adaptation Strategies

Based on the proposed two-stage GPHH feature selection algorithm, to eliminate unselected features in scheduling heuristics, we propose individual adaptation strategies to utilise the information of both the selected features and investigated individuals during the feature selection process. The flowchart of the two-stage GPHH feature selection algorithm with individual adaptation is shown in Figure 4.3. Stage 1 is designed to obtain an informative population with individuals for feature selection and individual adaptation. Stage 1 is the same as introduced in Section 4.2.1. The main difference between the proposed algorithm with the algorithm in Section 4.2.1 is the initialisation in stage 2.

Stage 2 of the proposed GPHH feature selection with individual adaptation strategies is developed for using the information (i.e., final population and selected features) obtained in stage 1 to evolve effective schedul-

ing heuristics with only selected features. This chapter develops a number of novel individual adaptation strategies to initialise the population in stage 2, consisting only of the selected features without deteriorating the performance. It is noted that the selected features will be used in two situations in stage 2. One is to generate new individuals for building a new population during the initialisation process by the ramped half-and-half method. The other is to generate a new sub-tree by the grow method with only the selected features as the terminals to replace a selected subtree of a parent by mutation. Readers interested in the genetic operators of GP can refer to [202].

Individual Adaptation Strategies

One important task is to inherit the information of the promising individuals in the final population of stage 1 to keep the quality of evolved scheduling heuristics with only selected features in stage 2. To this end, this chapter proposes two strategies to adapt the individuals in the final population of stage 1 to stage 2. The goal is to generate new individuals with only the selected features but still have the same or similar behaviour with the promising individuals obtained in stage 1.

The first individual adaptation strategy is to simply replace each unselected feature with a constant of one. This can completely remove the unselected features from the individuals, while still keeping the structures of individuals as much as possible. If a feature is not selected, it is expected to have little contribution to a majority of individuals, and thus replacing them by one would not change the fitness much. A potential drawback is that the average quality of individuals in the first generation of stage 2 might not be as well as the last generation of stage 1. One possible reason is that replacing a number of unselected features in an individual by one is more likely to change the behaviour of the individual in certain ways.

The second individual adaptation strategy is based on the idea of "mimicking". Specifically, it randomly generates a large number of individuals with only

the selected features. For each promising individual in the final population of stage 1, it is replaced by the randomly generated individual that has the closest behaviour with it. The behaviour is defined based on the phenotypic characterisation [99], which is a numeric vector. The details of the phenotypic characterisation calculation are shown as follows.

Phenotypic Characterisation

The phenotypic characterisation of an individual is a decision vector based on a set of decision situations [99]. In this chapter, decision situations are sampled from preliminary simulation runs with 5000 jobs on 10 machines using the reference rules (e.g., WIQ, work in the queue for routing, and SPT, shortest processing time for sequencing). The preliminary simulation generated about 50,000 routing and 50,000 sequencing decision situations. Following the steps in [99], we randomly sampled decision situations from the generated decision situations that contains 2 and 20 jobs. To balance the accuracy and complexity of the phenotypic characterisation, the number of candidates, i.e., machines for routing and operations for sequencing, in each decision situation, is set to 7 in this chapter. In other words, from all the generated decision situations, a subset of 20 routing situations and 20 sequencing situations with the length of 7 is sampled for measuring the phenotypic characteristic of an individual. A smaller distance between the phenotypic characterisations of two individuals suggests that the two individuals are more similar.

The phenotypic characterisation of a rule is a vector of rank numbers, where the number of dimensions equals the number of decision situations. The element in the i^{th} dimension indicates the rank of the most prior candidate, i.e., operation or machine, by the examined rule in the rank list of the reference rule (i.e., WIQ for routing, and SPT for sequencing). Table 4.1 shows an example of calculating the phenotypic characterisation of a routing rule with four decision situations, and each decision situation consists of three candidate machines. In the first decision situation, M_3

Table 4.1: An example of calculating the phenotypic characterisation of a routing rule with four decision situations and each with three candidate machines.

Decision Situation	Reference Rule	Examined Rule	PC_i
1 (M_1)	3	2	
1 (M_2)	2	3	1
1 (M_3)	<u>1</u>	<u>1</u>	
2 (M_1)	1	3	
2 (M_2)	<u>3</u>	<u>1</u>	3
2 (M_3)	2	2	
3 (M_1)	2	3	
3 (M_2)	3	2	1
3 (M_3)	<u>1</u>	<u>1</u>	
4 (M_1)	1	3	
4 (M_2)	<u>2</u>	<u>1</u>	2
4 (M_3)	3	2	

PC_i indicates the i^{th} dimension of phenotypic characterisation.

is the most prior machine by the characterised routing rule. When looking at the rank value of M_3 by the reference rule, we find that the rank value is 1. Therefore, the value of the phenotypic characterisation in the first situation PC_1 is set to 1. Similarly, PC_i can be obtained in other decision situations. The corresponding observation indicators for finalising the phenotypic characterisation are underlined in each decision situation. Finally, the phenotypic characterisation of this routing rule is [1, 3, 1, 2]. It is noted that the way to calculate the phenotypic characterisation of sequencing rule is the same as the routing rule, except that sequencing rule is examined with sequencing decision situations rather than routing decision situations.

This chapter extends the idea in [167, 168] to calculate the phenotypic characterisation for an individual in DFJSS by concatenating the pheno-

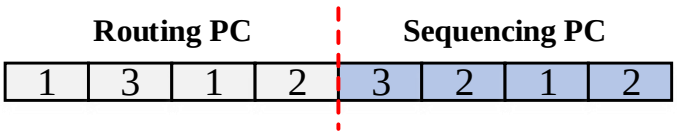

Figure 4.4: An example of the phenotypic characterisation of an individual in DFJSS (PC indicates phenotypic characterisation).

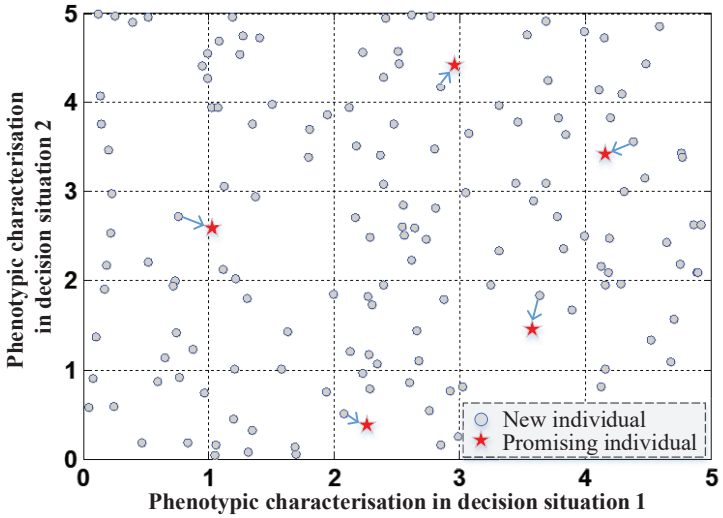

Figure 4.5: The process of mimicking individuals by generating new individuals only with selected features.

typic characterisations of routing and sequencing rules. An example of the phenotypic characterisation of an individual in DFJSS is shown in Figure 4.4. The phenotypic characterisation of an individual consists of the decision vectors of routing and sequencing heuristics. The individuals with both similar routing (left part) and sequencing (right part) phenotypic characterisations are considered to have similar behaviour.

Figure 4.5 shows an example of the process of mimicking individuals, where the phenotypic characterisation is 2-dimensional. Two decision situations are used to generate phenotypic characterisation. A decision situation is when a rule is to make a decision (i.e., a machine becomes idle or an

operation becomes ready). The stars indicate the promising individuals in stage 1 that need to be mimicked. A large number of new individuals (i.e., denoted as circles) are generated with only the selected features. The new individuals whose phenotypic vectors are closest to that of the mimicked individuals will be chosen to replace the promising individuals in stage 1. It is noted that it is not always possible to find individuals with the same behaviour (i.e., the distance between two individuals equals zero). The new individuals with the most similar behaviours with promising individuals will be saved in the initial population for stage 2.

It is noted that it is not meaningful to adapt all the individuals obtained from stage 1. If the number of adapted individuals is too small, there will be too many randomly generated individuals, and the performance will be close to purely re-initialisation. If it is too large, it will bring some noise, lose diversity, and increase the training time. Only the "promising" individuals are adapted in this chapter. This chapter identifies promising individuals using the knee point, which is a parameter-free approach. The knee point can be used as a demarcation point. The individuals whose fitnesses are smaller (i.e., minimising problem) than that of knee point are selected as promising individuals. The details of detecting promising individuals can be found in Section 3.2.3. It is noted that if there is more than one knee point, the knee point that has the largest distance to the generated line will be selected. The rest of the individuals in the initial population of stage 2 will be randomly initialised with selected features. Finally, a new population with only the selected features is obtained.

4.2.5 Algorithm Summary

This chapter proposes two GPHH feature selection algorithms for DFJSS. The first algorithm is a two-stage GPHH feature selection algorithm for DFJSS where the feature selection is conducted on both the routing terminal set and sequencing terminal set simultaneously. The output of the fea-

ture selection is two feature subsets, one for routing rule and the other for sequencing rule. The selected features are used during the evolutionary process of GPHH via mutation in stage 2. The second algorithm further extends the two-stage GPHH feature selection algorithm, where individual adaptation strategies are proposed to initialise the individuals to eliminate the unselected features in stage 2. The evolved scheduling heuristics are expected to be effective and contain only the selected features.

4.3 Experiment Design

4.3.1 Comparison Design

We introduce the framework in [233] with two subpopulations to evolve routing and sequencing rules simultaneously with the cooperation coevolutionary strategy for DFJSS. This is to separate the routing rules and sequencing rules into different subpopulations to make it easy to measure the importance of features for the routing rule and sequencing rule.

Five algorithms are taken into the comparison in this chapter. The cooperative coevolution genetic programming (CCGP) [233] (i.e., without feature selection and individual adaptation strategies) is selected as the baseline algorithm. The first algorithm proposed in this chapter is named as (CCGP^2) [249]. CCGP^2 uses the selected features in stage 2 only in mutation, and it is used to verify whether the new mutation operator will affect the performance. The proposed algorithms with the new individual adaptation strategies are named as $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ [242] (i.e., mimicking the behaviour of promising individuals) and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ (i.e., replacing the unselected features by one directly). To verify the effectiveness of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$, the algorithm (i.e., named as $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$) that randomly initialises all the individuals in the new population in stage 2 is also compared. This is because using the selected features to re-initialise the new population in stage 2 randomly is the most

Table 4.2: The specialised parameter settings of GPHH.

Parameter	Value
Number of subpopulations	2
Subpopulation size	512
Number of generations in stage 1 and stage 2	50 / 50

straightforward way to eliminate unselected features intuitively.

In order to verify the effectiveness and efficiency, the proposed algorithms are tested on six different scenarios. The scenarios consist of three objectives (i.e., max flowtime, mean flowtime, and mean weighted flowtime) and two utilisation levels (i.e., 0.85 and 0.95).

4.3.2 Specialised Parameter Settings of GPHH

The specialised parameter settings of GPHH are shown in Table 4.2. The representation with cooperative coevolution is used in this chapter to separate routing rule and sequencing rule to make it easy of the feature selection for the routing terminal set and the sequencing terminal set. There are two subpopulations, i.e., one is for evolving the routing rules, and the other is for evolving the sequencing rules. The GPHH with feature selection contains 100 generations. The first 50 generations are designed for feature selection, and the second 50 generations aims to evolve scheduling heuristics for DFJSS.

4.4 Results and Discussions

4.4.1 Quality of the Evolved Scheduling Heuristics

Table 4.3 shows the mean and standard deviation of involved five algorithms according to 30 independent runs in the six DFJSS scenarios. CCGP^2 which only uses the selected features in the mutation operator achieves

Table 4.3: The mean (standard deviation) of the objective values of the five algorithms over 30 independent runs for six DFJSS scenarios.

Sce.	CCGP	CCGP^2	$\mathrm{CCGP}^{2a}(\mathrm{rand})$	$\mathrm{CCGP}^{2a}(\mathrm{rep})$	$\mathrm{CCGP}^{2a}(\mathrm{mimic})$
1	1223.83(41.78)	1225.41(43.51)(≈)	1314.87(121.35)(+)	1237.53(81.28)(≈)	1238.34(99.27)(≈)
2	1959.24(46.63)	1998.09(115.26)(≈)	2054.56(204.36)(+)	2032.85(145.16)(≈)	2034.08(153.61)(≈)
3	385.42(2.65)	384.77(1.32)(≈)	387.34(2.23)(≈)	385.07(1.24)(≈)	385.14(1.87)(≈)
4	553.65(7.89)	552.88(6.78)(≈)	559.21(8.21)(+)	553.07(6.31)(≈)	551.20(6.11)(≈)
5	830.74(6.89)	829.58(5.56)(≈)	833.02(6.15)(+)	830.11(5.42)(≈)	831.51(6.52)(≈)
6	1109.89(13.07)	1110.86(12.01)(≈)	1112.35(12.91)(≈)	1109.58(7.96)(≈)	1112.94(14.62)(≈)

* 1: <Fmax, 0.85>, 2: <Fmax, 0.95>, 3: <Fmean, 0.85>

* 4: <Fmean, 0.95>, 5: <WFmean, 0.85>, 6: <WFmean, 0.95>

similar performance with CCGP. One possible reason is that the GP itself can detect important features automatically. The other is that there is not much difference when only applying selected features by mutation with a small rate (i.e., 0.15). However, the drawback is that the evolved rules by CCGP^2 still contain unselected features. It does not make it easier to interpret the rules, since all/most of the features are still included in the rules which makes the rules complex.

The performance of $\mathrm{CCGP}^{2a}(\mathrm{rand})$ is significantly worse than that of CCGP in most scenarios. One reason might be that a completely new random population which has worse performance (i.e., a new start), is generated for stage 2. It is hard to achieve good performance as CCGP (i.e., actually evolved for 100 generations). $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ and $\mathrm{CCGP}^{2a}(\mathrm{rep})$ (i.e., only with selected features) can achieve similar performance with CCGP^2 in most scenarios. It indicates that the proposed individual adaptation strategies are effective to take advantage of the population information from stage 1.

It is noted that all the algorithms have larger variances in scenario <Fmax, 0.85> and <Fmax, 0.95>, especially the algorithms with feature selection (i.e., CCGP^2, $\mathrm{CCGP}^{2a}(\mathrm{rand})$, $\mathrm{CCGP}^{2a}(\mathrm{rep})$, and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$).

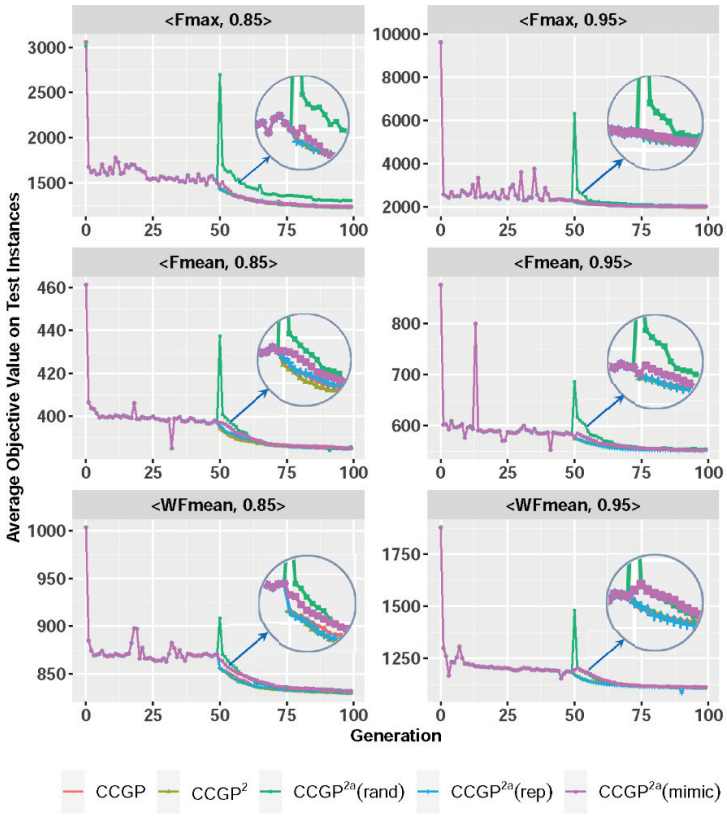

Figure 4.6: The curves of average objective values on test instances of the five algorithms according to 30 independent runs in six DFJSS scenarios.

One possible reason is that the max flowtime is more sensitive to the worst case of processing jobs than the mean flowtime. Another possible reason is that feature selection is not always accurate (depending on the individuals obtained in stage 1), and inaccurate feature selection results can lead to some outliers.

Figure 4.6 shows the convergence curves of the average objective value on unseen instances of CCGP, CCGP^2, $\mathrm{CCGP}^{2a}(\mathrm{rand})$, $\mathrm{CCGP}^{2a}(\mathrm{rep})$, and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ according to 30 independent runs in different scenarios. In all the scenarios, $\mathrm{CCGP}^{2a}(\mathrm{mimic})$, and $\mathrm{CCGP}^{2a}(\mathrm{rep})$ can mimic the behaviours well (i.e., shown at generation 50), and the performance does

not decrease too much. $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ can achieve similar performance in the scenario <Fmax, 0.95> after generation 50 compared with CCGP^{2}. However, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ can get similar performance with CCGP^{2} in all scenarios. This means that $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ has a more promising inheritance ability. In general, the effectivenesses of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ are better than that of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$. Both $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ can inherit the individuals' information well from stage 1 to stage 2.

4.4.2 Sizes of Evolved Scheduling Heuristics

Figure 4.7 and Figure 4.8 show the curves of the sizes (i.e., the mean value of 30 independent runs at each generation) of the routing rules and sequencing rules. From the figures, we can see that for both the routing rules and sequencing rules, the rule sizes of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$ decrease dramatically at the beginning of stage 2 due to the individual adaptation strategies, especially for the routing rules. It is noted that the size of the routing rules of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ is not that small compared with $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$. This is because the structures of the large rules in stage 1 are kept to stage 2.

The sizes of the best routing rules obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ are smaller than that of other algorithms in most scenarios (i.e., <Fmax, 0.85>, <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85>, and <WFmean, 0.95>). However, the sizes of the best routing rules of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ are similar with that of other compared algorithms in <Fmax, 0.95>. This may be because Fmax with a higher utilisation level (i.e., 0.95) is more difficult to be optimised due to its sensitiveness to the worst case (i.e., the longest finished time among all jobs). Compared with the curves of the sizes of sequencing rules, the proposed individual adaptation strategies have a great impact on the size of routing rules. The sizes of the best sequencing rules obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ are similar with that of other algorithms.

It is noted that the routing rule and sequencing rule work together

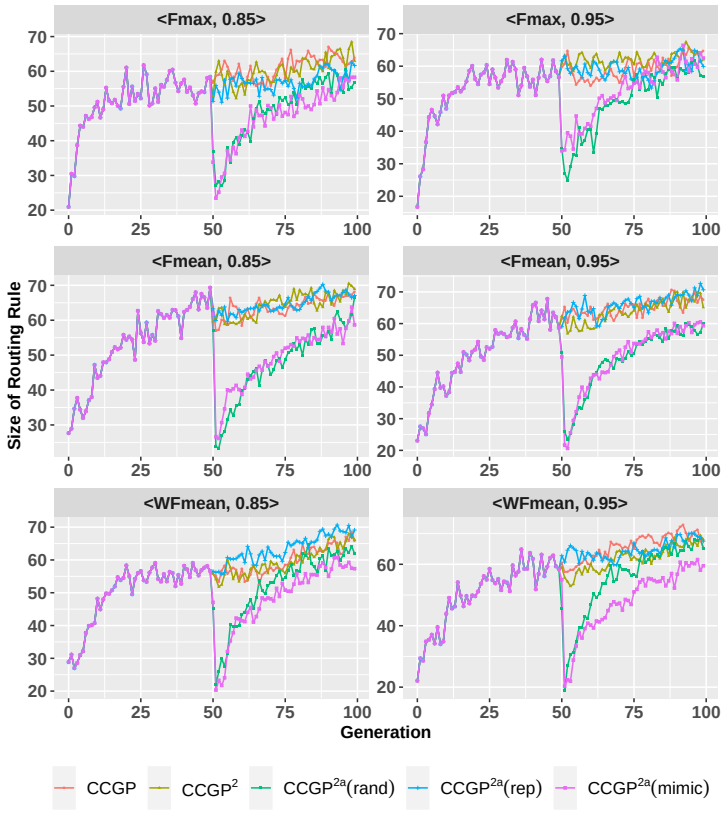

Figure 4.7: The curves of **the best routing rule sizes** of the population of the five algorithms according to 30 independent runs in six DFJSS scenarios.

in DFJSS. It makes sense to take a routing rule and a sequencing rule as a pair to measure the rule size. This is because a smaller routing rule and a larger sequencing can have the same ability for DFJSS compared with a larger routing rule and a smaller sequencing rule based on our observation. Based on the analyses as mentioned earlier, it turns out that $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ can achieve similar performance with $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ but with small scheduling heuristics. The rule sizes of three algorithms (i.e., CCGP, CCGP^2, and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$) are further compared.

Table 4.4 shows the mean and standard deviation of the rule sizes (i.e.,

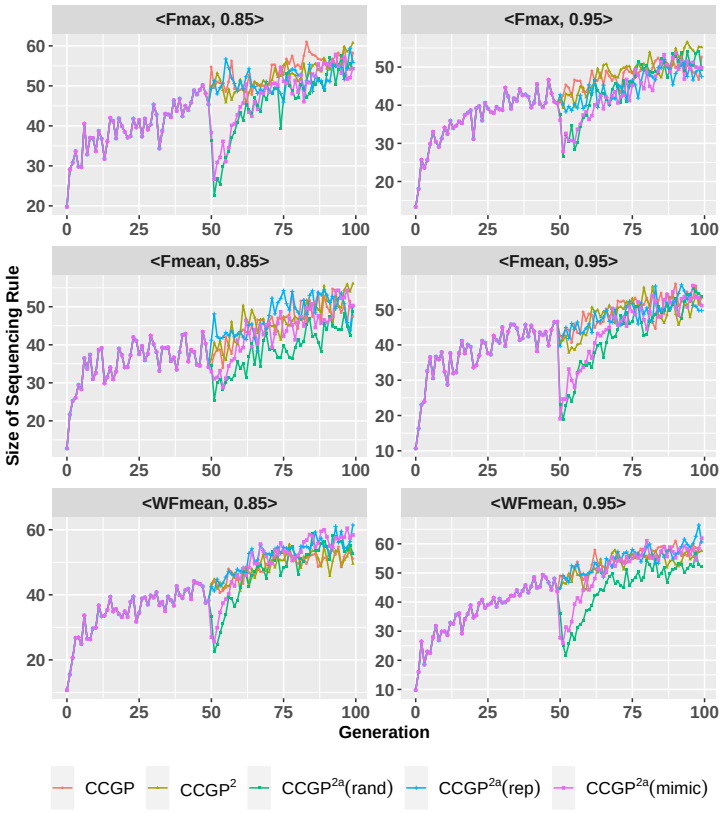

Figure 4.8: The curves of **the best sequencing rule sizes** of the population of the five algorithms according to 30 independent runs in six DFJSS scenarios.

routing rule size plus sequencing rule size) evolved by CCGP, CCGP^2, and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ according to 30 independent runs in six DFJSS scenarios. It shows that the rule sizes obtained by these three algorithms are similar. The main difference among the evolved rules is that the rules evolved by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ only contain selected features (i.e., fewer features) while the rules evolved by CCGP and CCGP^2 consist of all features possibly. The unique feature is further studied in the next subsection.

Table 4.4: The mean (standard deviation) of **the rule sizes** obtained by CCGP, CCGP^2 and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ over 30 independent runs in six DFJSS scenarios.

Scenario	CCGP	CCGP^2	$\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$
<Fmax, 0.85>	122.07(30.60)	123.80(26.72)(≈)	112.60(25.75)(≈)
<Fmax, 0.95>	117.27(25.22)	117.27(28.72)(≈)	112.27(31.38)(≈)
<Fmean, 0.85>	115.73(24.91)	125.07(18.18)(≈)	108.93(26.32)(≈)
<Fmean, 0.95>	121.27(17.60)	118.53(28.08)(≈)	110.47(22.31)(≈)
<WFmean, 0.85>	116.87(23.88)	115.67(26.53)(≈)	115.73(26.31)(≈)
<WFmean, 0.95>	127.40(24.40)	125.67(23.84)(≈)	121.47(22.80)(≈)

Table 4.5: The mean (standard deviation) of **the average number of unique features of routing rules** obtained by the five algorithms over 30 independent runs in six DFJSS scenarios.

Scenario	CCGP	CCGP^2	$\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$	$\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$	$\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$
<Fmax, 0.85>	8.40(1.33)	7.80(1.47)(≈)	6.27(1.68)**(–)**	6.57(1.74)**(–)**	6.67(1.81)**(–)**
<Fmax, 0.95>	8.67(0.92)	8.37(1.33)(≈)	6.73(1.72)**(–)**	6.83(1.64)**(–)**	6.70(1.66)**(–)**
<Fmean, 0.85>	8.03(1.03)	7.67(1.03)(≈)	5.70(1.62)**(–)**	5.83(1.46)**(–)**	5.63(1.52)**(–)**
<Fmean, 0.95>	8.40(1.10)	7.87(1.14)(≈)	5.57(1.33)**(–)**	5.73(1.53)**(–)**	5.83(1.56)**(–)**
<WFmean, 0.85>	8.27(1.23)	7.93(1.23)(≈)	5.60(1.63)**(–)**	6.17(2.09)**(–)**	5.70(1.99)**(–)**
<WFmean, 0.95>	8.20(1.13)	7.50(1.57)(≈)	5.70(1.47)**(–)**	5.90(1.73)**(–)**	5.70(1.99)**(–)**

4.4.3 Unique Feature Analysis

Table 4.5 shows the mean (standard deviation) of the number of unique features in routing rules in six DFJSS scenarios. There is no statistical difference between CCGP and CCGP^2 in terms of the number of unique features in routing rules in all scenarios. This means that applying selected features only to mutation is not an effective way for reducing the unique number of features of the evolved routing rules. In addition, no matter what kind of individual adaptation strategies are used (i.e., $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ with mimicking behaviour strategy, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$ with replacing by one

Table 4.6: The mean (standard deviation) of **the average number of unique features of sequencing rules** obtained by the five algorithms over 30 independent runs in six DFJSS scenarios.

Scenario	CCGP	CCGP^2	$\mathrm{CCGP}^{2a}(\mathrm{rand})$	$\mathrm{CCGP}^{2a}(\mathrm{rep})$	$\mathrm{CCGP}^{2a}(\mathrm{mimic})$
<Fmax, 0.85>	7.13(1.59)	6.53(1.14)**(–)**	5.23(1.41)**(–)**	5.00(1.20)**(–)**	5.20(1.27)**(–)**
<Fmax, 0.95>	7.40(1.57)	6.53(1.41)**(–)**	4.97(1.25)**(–)**	5.03(1.27)**(–)**	5.17(1.39)**(–)**
<Fmean, 0.85>	6.57(2.10)	5.47(1.36)**(–)**	3.53(1.07)**(–)**	3.40(1.00)**(–)**	3.70(1.06)**(–)**
<Fmean, 0.95>	6.90(1.60)	6.03(1.43)**(–)**	4.00(1.23)**(–)**	3.97(1.19)**(–)**	3.70(0.99)**(–)**
<WFmean, 0.85>	6.53(1.59)	5.17(1.05)**(–)**	4.00(0.79)**(–)**	3.93(0.69)**(–)**	4.00(0.79)**(–)**
<WFmean, 0.95>	6.80(1.52)	5.70(1.47)**(–)**	4.33(0.92)**(–)**	4.17(0.95)**(–)**	4.27(0.87)**(–)**

strategy and $\mathrm{CCGP}^{2a}(\mathrm{rand})$ with 100% randomly initialisation strategy), the number of unique features in routing rules are significantly smaller than CCGP.

Table 4.6 shows the mean (standard deviation) of the number of unique features in the sequencing rules over 30 independent runs in different DFJSS scenarios. For all the algorithms (i.e., $\mathrm{CCGP}^{2a}(\mathrm{mimic})$, $\mathrm{CCGP}^{2a}(\mathrm{rep})$, $\mathrm{CCGP}^{2a}(\mathrm{rand})$, and CCGP^2) that involve feature selection, the number of unique features in the sequencing rules is significantly smaller in all scenarios, especially the individual adaptation strategies related algorithms (i.e., $\mathrm{CCGP}^{2a}(\mathrm{mimic})$, $\mathrm{CCGP}^{2a}(\mathrm{rep})$, and $\mathrm{CCGP}^{2a}(\mathrm{rand})$).

Assuming that the rules evolved by $\mathrm{CCGP}^{2a}(\mathrm{mimic})$) with fewer features are easier to be simplified by the algebraic transformation. Simplification aims to simplify the complicated expression by some algebraic operations to make the evolved rules easier to be interpreted. For example, given a rule (A - B) / (A - B), it will be simplified to be one. Moreover, the rule A + B + A + A will become 3 * A + B, while the rule A * B / B will be simplified as A.

Figure 4.9 shows the violin plot of rule size after the simplification taking routing and sequencing rules as a pair obtained by CCGP, CCGP^2, and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ over 30 independent runs in six DFJSS scenarios. It shows

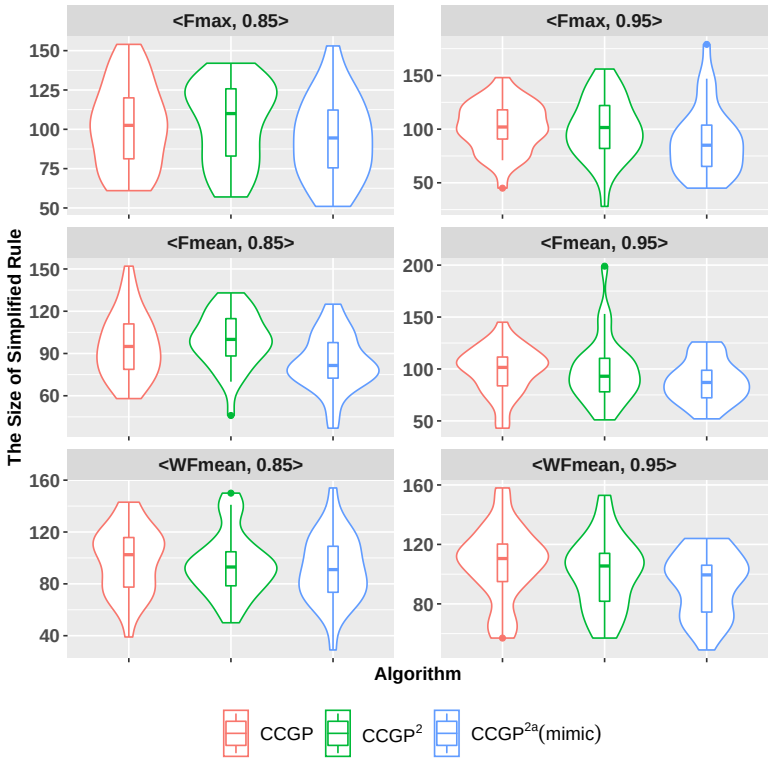

Figure 4.9: The violin plot of rule sizes (i.e., routing rule plus sequencing rule) obtained by CCGP, CCGP^2, and $\text{CCGP}^{2a}(\text{mimic})$ after simplification over 30 independent runs in six DFJSS scenarios.

that the sizes of simplified rules evolved by $\text{CCGP}^{2a}(\text{mimic})$ are much smaller than that of CCGP and CCGP^2. It indicates that $\text{CCGP}^{2a}(\text{mimic})$ has more potential to get smaller rules which are important for interpreting rules.

Figure 4.10 shows the scatter plot of the sizes of routing and sequencing rules before and after simplification. It shows that the routing rule sizes become much smaller (i.e., located downwards) in all scenarios. In addition, the sequencing rule sizes tend to be smaller (i.e., located towards the left) compared with the sizes without simplification in all scenarios. In summary, after simplification, $\text{CCGP}^{2a}(\text{mimic})$ can achieve smaller routing

Figure 4.10: The scatter plot of the sizes of routing rules and sequencing rules before and after simplification obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ over 30 independent runs in six DFJSS scenarios.

rules and sequencing rules than that of CCGP and CCGP^2.

4.4.4 Training Time

Table 4.7 shows the training time (in minutes) of the algorithms CCGP, CCGP^2, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$, and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$. The training time of CCGP^2 has no significant difference compared with that of CCGP. It is obvious that the training time of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rep})$, and $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ decrease dramatically compared with that of CCGP and CCGP^2. The main difference of the training time between CCGP, CCGP^2 and the algorithms with individual adaptation strategies (i.e., $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$,

Table 4.7: The mean(standard deviation) of **training time** (in minutes) by the five algorithms in six DFJSS scenarios.

Scenario	CCGP	CCGP^2	$\mathrm{CCGP}^{2a}(\mathrm{rand})$	$\mathrm{CCGP}^{2a}(\mathrm{rep})$	$\mathrm{CCGP}^{2a}(\mathrm{mimic})$
<Fmax, 0.85>	100(17)	96(14)(≈)	80(15)**(–)**	83(13)**(–)**	75(10)**(–)**
<Fmax, 0.95>	107(15)	111(20)(≈)	88(13)**(–)**	92(15)**(–)**	88(12)**(–)**
<Fmean, 0.85>	98(12)	99(13)(≈)	78(11)**(–)**	87(12)**(–)**	77(12)**(–)**
<Fmean, 0.95>	109(15)	108(16)(≈)	86(11)**(–)**	94(16)**(–)**	85(12)**(–)**
<WFmean, 0.85>	94(11)	98(11)(≈)	82(9)**(–)**	88(15)(≈)	83(15)**(–)**
<WFmean, 0.95>	113(16)	109(16)(≈)	90(13)**(–)**	98(16)**(–)**	88(13)**(–)**

$\mathrm{CCGP}^{2a}(\mathrm{rep})$, and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$), is caused by the individual adaptation strategy. Intuitively, for $\mathrm{CCGP}^{2a}(\mathrm{rand})$, $\mathrm{CCGP}^{2a}(\mathrm{rep})$ and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$, a longer training time might be needed due to the extra algorithm operators, i.e., individual adaptation. However, it turns out that the time of the proposed algorithms with the individual adaptation strategies are smaller than CCGP and CCGP^2, especially $\mathrm{CCGP}^{2a}(\mathrm{rand})$ and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$. This is because the sizes of the evolved scheduling heuristics obtained by $\mathrm{CCGP}^{2a}(\mathrm{rand})$ and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ are smaller than CCGP and CCGP^2. In summary, $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ is more efficient in terms of the mean values of the training time despite more algorithm operators (i.e., feature selection and mimicking individuals' behaviours operations in the algorithm) are involved.

When looking at the curve of average rule sizes in the population in Figure 4.11, the average rule sizes over the population of $\mathrm{CCGP}^{2a}(\mathrm{rand})$ and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ are smaller than its counterparts. The reason is that all the individuals in the population at generation 50 are re-initialised, and they have smaller sizes. The smaller rules tend to save computational time by reducing the individual evaluation time. Thus, $\mathrm{CCGP}^{2a}(\mathrm{rand})$ and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ can reduce the computational time significantly.

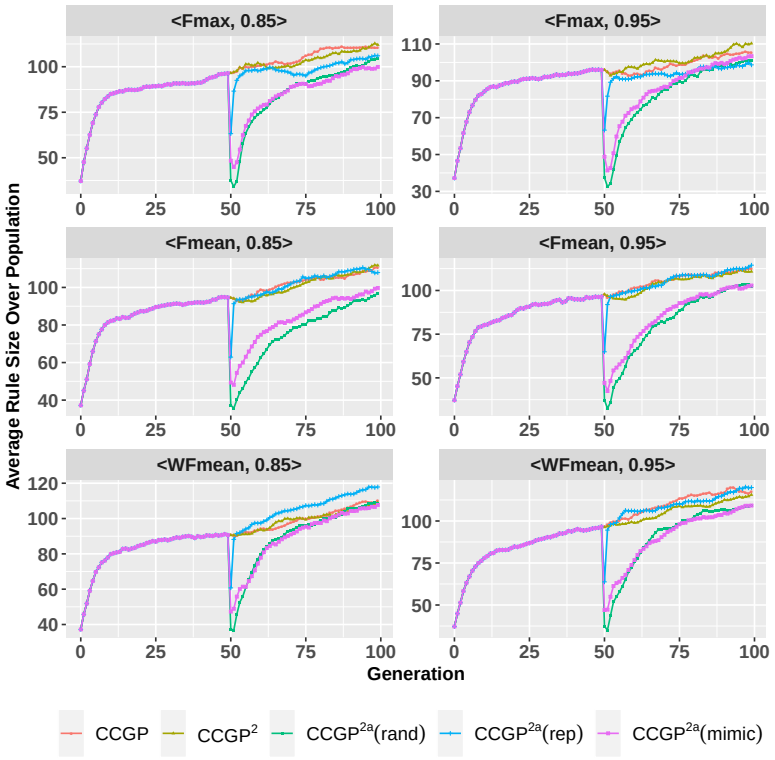

Figure 4.11: The **average rule (routing rules plus sequencing rules) sizes over population** of the five algorithms in six DFJSS scenarios.

4.5 Further Analyses

To deeply understand the effect of the proposed algorithm, the selected features and evolved rules are further analysed in this section.

4.5.1 Feature Analysis

Figure 4.12 and Figure 4.13 show the selected and unselected features of 30 runs by the sequencing rules and routing rules in the six DFJSS scenarios, respectively. For a feature, a bigger blue area (i.e., the corresponding feature is selected more frequently) means the corresponding feature is more important. It is noted that the selected features of CCGP^{2}, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{rand})$,

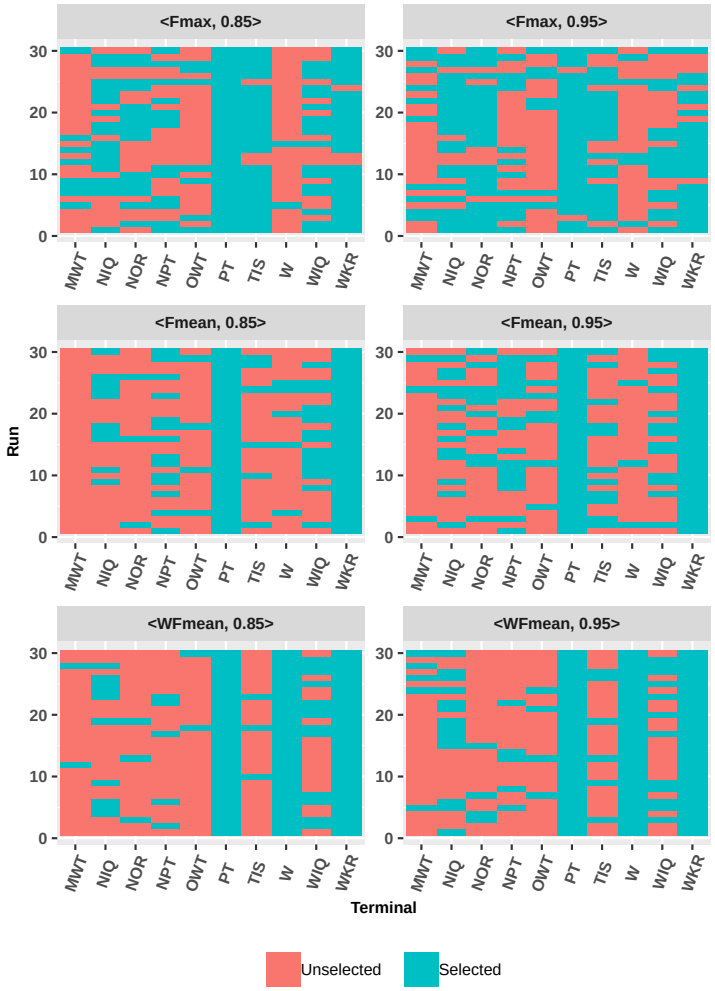

Figure 4.12: Selected and unselected features of **sequencing rules** of 30 independent runs in six DFJSS scenarios.

$\mathrm{CCGP}^{2a}(\mathrm{rep})$, and $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ are the same in the same run (i.e., with the same random seed) since the evolutionary processes are the same in stage 1. In each run, for the sequencing rules and routing rules, the selected features vary from each other based on the proposed feature selection algorithm. This means that the selected features can be adjusted adaptively with the proposed two-stage framework and that the selected

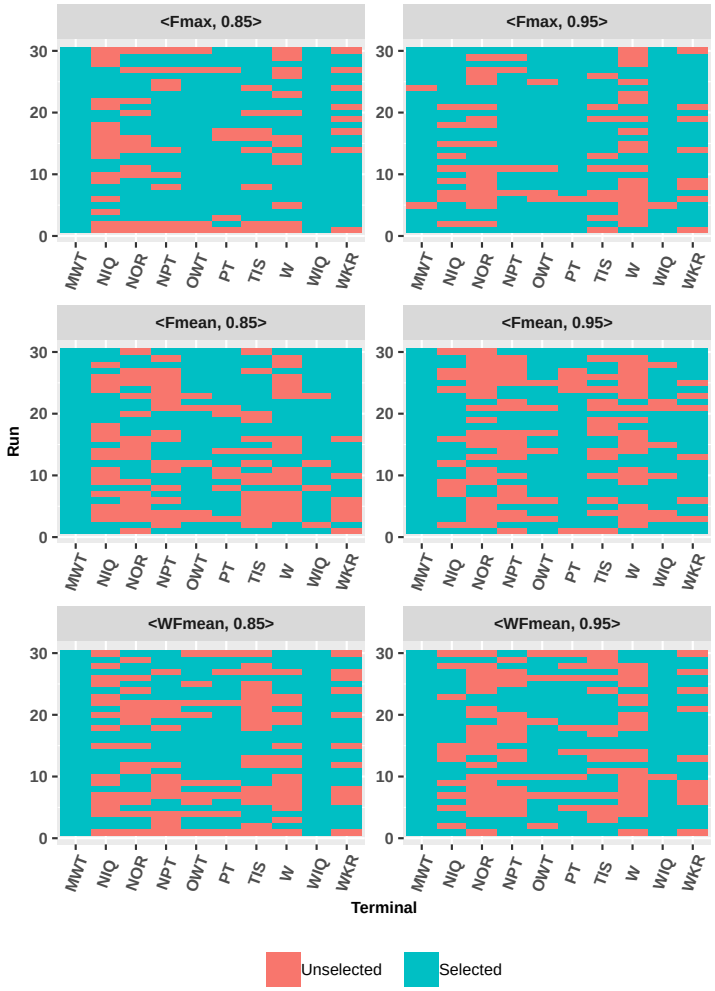

Figure 4.13: Selected and unselected features of **routing rules** of 30 independent runs in six DFJSS scenarios.

features are based on the specific problems (i.e., different runs).

For sequencing rules, the top three important features for the scenario <Fmax, 0.85> are PT, TIS, and WKR, as shown in Figure 4.12. Except for these three features, NIQ and NOR are also important in the scenario <Fmax, 0.95>. Compared with <Fmax, 0.85>, Figure 4.12 shows that more features are selected in the scenario <Fmax, 0.95>. It might be be-

cause a higher utilisation level makes the problem more difficult to be optimised. For minimising the mean flowtime (i.e., <Fmean, 0.85> and <Fmean, 0.95>), PT and WKR play an important role (i.e., they are selected in all the 30 runs). When taking the mean weighted flowtime into consideration (i.e., <WFmean, 0.85> and <WFmean, 0.95>), except for PT and WKR, W is also a significant feature. It is consistent with our intuition that PT (i.e., processing time) and WKR (i.e., median amount of work remaining for a job) are important factors for the flowtime related objectives. In addition, W is often chosen for minimising the mean weighted flowtime rather than max flowtime and mean flowtime. It is consistent with our expectation, since the calculations of mean flowtime and max flowtime do not involve W at all.

For routing rules, MWT, OWT, and WIQ, are significant for evolving routing rules in all the scenarios, as shown in Figure 4.13. It is consistent with our intuition that the machine with less workload (WIQ) and earlier ready time (MWT) is preferred, since the new operation has a higher chance to be processed early. In addition, this chapter will allocate a new operation once it becomes a ready operation (i.e., OWT, the operation waiting time equals zero). Compared with the selected features in the sequencing rules, more features are used in the routing rules (i.e., larger blue areas). It indicates that the evolved routing rules might be more complex than the sequencing rules.

Figure 4.14 shows the distributions of test objective values of the 30 independent runs of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ in the scenario <Fmax, 0.85>, categorised by whether each feature is selected or not in the sequencing rules. TIS is not selected in three runs, the test performance is much worse when it is selected. However, even NIQ is a relatively important feature (i.e., it is selected in half runs roughly) and it is not selected in half runs, the test performance is still very good. WKR is not selected in three runs, however, WKR has a greater impact on the quality of the evolved scheduling heuristics on two runs while it has no effect on one run. This indicates that

Figure 4.14: The distributions of the test objective values of the 30 independent runs of $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ in scenario <Fmax, 0.85>, categorised by whether each feature is selected or not in sequencing rules.

although the features used in the scheduling heuristics are important, the quality of the scheduling heuristics also depend on other factors, e.g., the structures of the rules.

4.5.2 Rule Analysis

This section considers the scenario <WFmean, 0.95> as an example for rule analysis. This is because the objective in the scenario <WFmean, 0.95> is more difficult to be optimised than other scenarios. Specially, we take the best routing rule and the corresponding sequencing rule (i.e., its objective value is 1096.02) for analysing. It is noted that the rules select the candidate (i.e., operation for sequencing or machine for routing) that has the lowest priority value.

For CCGP^2, the routing terminal set consists of NIQ, WIQ, MWT, PT, NPT, OWT, WKR, NOR, and W (i.e., **nine** features). There are **six** features which are NIQ, WIQ, MWT, PT, WKR, and W selected as sequencing terminal set. For the routing rules obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$, all the fea-

tures in the routing terminal set (i.e., **seven** features, NIQ, WIQ, MWT, PT, OWT, WKR, and W) are selected. When further looking into the sequencing rule obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$, **five** of them (i.e., NIQ, WIQ, PT, WKR, and W) are involved. Fewer features are used by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ compared with CCGP^{2}.

However, the evolved scheduling heuristics obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ have better test performance than that of CCGP^{2}. The routing rule obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ can be simplified, as shown in Eq. 4.2.

$$\begin{aligned} R_1 =& min\{2 * NIQ, max(\frac{NIQ * PT}{MWT}, \\ & PT * WIQ * min(WIQ, WKR))\}+ \\ & NIQ * \frac{PT}{W} - MWT \end{aligned} \tag{4.2}$$

It is obvious that this routing rule is quite small after simplification. In terms of the features related to machines, this routing rule prefers to choose the machine which has a large waiting time (i.e., MWT) and a smaller NIQ (i.e., the number of operations in the queue) and WIQ (i.e., the workload of machines). In terms of the features of operations, this routing rule tends to choose the machine which can process an operation more efficient with a smaller PT (i.e., processing time). In addition, from the perspective of operation, W (i.e., the weight that is used to indicate the importance of operation) is a constant when choosing a machine. From a machine's point of view, more important operation with a larger W has more priority to select a machine.

The corresponding sequencing rule obtained by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ is simplified, as shown in Eq. 4.3.

$$\begin{aligned} S_1 =& \frac{PT + WKR}{W} \\ & - \frac{W}{PT}(W * WIQ - W + WIQ) - \frac{W}{PT} * \\ & (W * WIQ + WKR + \frac{PT + WKR}{W * WIQ - W + WKR}) \end{aligned} \tag{4.3}$$

It is noted that for all the operations in the queue of a machine, the machine-related feature such as WIQ (i.e., the workload of a machine) is the same for all operations. This means that it is not a vitally important feature. This sequencing rule prefers to choose the operation with smaller PT (i.e., processing time) and larger W (i.e., weight, the importance of an operation). It is interesting that this sequencing rule prefers to smaller WKR (i.e., the median amount of work remaining for a job) based on the first line of S_1 while it tends to select the operations with larger WKR based on the third line of S_1 partially. It indicates that although we can interpret the rules to some extent, it is still hard to completely understand the behaviour of rules. We will continue to work on this topic in the future.

The corresponding routing and sequencing rules obtained by CCGP^2, can be simplified as shown in Eq. 4.4 and Eq. 4.5, respectively.

$$\begin{aligned} R_2 =& Max\{NIQ^2, (NIQ+NPT)*Min(NIQ,NOR)\} \\ & - Min(MWT, \frac{WKR}{MWT*WKR-1}) \\ & * Max\{WKR, -MWT*WKR+ \\ & \frac{NIQ*PT*Max\{WIQ, \frac{Min(MWT,PT)}{(W+WKR)}\}}{Max\{NIQ^2, \frac{NOR-W+WKR}{W}, \frac{NIQ+NPT}{(Min(NIQ,NOR))^{-1}}\}}\} \end{aligned} \tag{4.4}$$

$$\begin{aligned} S_2 =& NIQ(PT-W)(PT+\frac{WKR}{W}*Max\{PT, \frac{WIQ}{W}, \\ & \frac{Max\{WIQ, \frac{WKR}{W}\}}{W}\}) + Max\{\frac{WIQ}{W^2}, \frac{NIQ}{W^2}+WIQ\} \\ & * (MWT+W+Max\{\frac{WKR}{W^2}, NIQ-1\}) \\ & * \frac{Max\{WIQ, \frac{WKR}{W}\}}{W} \end{aligned} \tag{4.5}$$

The structures of both this routing rule and sequencing rule are more complex than that of $\mathrm{CCGP}^{2a}(\mathrm{mimic})$ even after simplification. From the perspective of the components, the function Max and Min are used a lot and nested inside each other. It is hard to know which component has

played a real role since it relies on multiple factors. It is not easy to understand it from a human perspective.

In summary, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ can evolve effective scheduling heuristics with smaller number of unique features and sizes. This can benefit the real-world applications, since the evolved scheduling heuristics tend to be understood by human easier.

4.6 Chapter Summary

The goal of this chapter is to develop an effective GPHH feature selection algorithm to evolve scheduling heuristics for DFJSS without compromising any performance. To achieve this goal, this chapter firstly proposes the two-stage feature selection algorithm CCGP^2 for DFJSS. The proposed algorithm is based on the idea that promising features can benefit more for the evolutionary process and can be used to guide the process effectively. In addition, this chapter proposes GPHH feature selection algorithm with novel individual adaptation strategies to eliminate unselected features in the evolved scheduling heuristics. The quality of the scheduling heuristics is kept by utilising both the information of the selected features and the investigated individuals in the feature selection process.

The results show the effectiveness of the proposed GPHH feature selection algorithm CCGP^2. Moreover, the results show that the evolved rules by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ are smaller and contain fewer unique features due to feature selection. Thesis kinds of rules tend to be interpreted easier. This is also verified by the semantic analyses of the routing and sequencing rules evolved by $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$. In terms of training time, $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ is more efficient than that of the baseline algorithm, since it can reduce the average rule size of the population. In summary, the proposed algorithm $\mathrm{CCGP}^{2\mathrm{a}}(\mathrm{mimic})$ can evolve effective scheduling heuristics with a smaller number of unique features and smaller size efficiently.

It is noted that the surrogate technique is used to improve the train-

ing efficiency of GPHH in both the last chapter and this chapter. The last chapter focuses on the design of surrogate models for DFJSS. However, this chapter focuses on using the individual information and selected features to maintain the quality of individuals only with the selected features. The surrogate technique used in this chapter is to improve the efficiency of obtaining individuals for feature selection, and the surrogate itself is not the focus in this chapter.

This chapter uses the phenotypic characterisation technique to measure the behaviour/performance of GP individuals for finding their similar individuals with only the selected features. In the next chapter, phenotypic characterisation will be used to measure the subtree importance for improving the search mechanism of GPHH for DFJSS.

Chapter 5

New Search Mechanism with Specialised Genetic Operators

In Chapter 4, we develop the phenotypic characterisation technique to measure the behaviour/performance of individuals for feature selection. This chapter will focus on using the phenotypic characterisation technique to measure the subtree importance to improve the search mechanism of GPHH for DFJSS.

5.1 Introduction

For an evolutionary computation algorithm, genetic operators play important roles for generating offspring. The crossover operator is an essential genetic operator for GP to produce offspring during the evolutionary process. An effective crossover is expected to generate offspring with good quality. In essence, the crossover is a recombination of different materials from the parents. In traditional GP, subtrees are randomly chosen from two parents to swap to produce two offspring. However, the importance of subtrees in each individual can be different. Some subtrees are redundant or less important, and removing them might not affect the fitness of an individual too much. On the other hand, some subtrees play essen-

tial roles for an individual and losing them will cause considerable loss to the fitness. The random way of recombination may disrupt beneficial building-blocks.

To the best of our knowledge, little is yet known to improve the recombinative effectiveness of GPHH via the crossover for DFJSS. Riccardo et al. provided a comprehensive general schema theory for GP with subtree-swapping crossover in [203, 204]. This theory suggests that the biases of GP operators can be beneficial for different purposes, such as improving the quality of the offspring and controlling the size or shape of the offspring. However, it is challenging when we apply bias in GP to practice. *One critical challenge* is how to measure the subtrees based on the desired purpose. *The other challenge* is how to apply the expected "biases" to GP for a specific problem.

5.1.1 Chapter Goals

The goal of this chapter is to *develop an effective GPHH with specialised genetic operators (i.e., the crossover operator) with recombination guidance to evolve effective scheduling heuristics for DFJSS*. The key is to design effective strategies to measure the importance of subtrees. This chapter develops two importance measures to reflect the degree of relationship between the behaviour of the subtree and the entire tree. First, this chapter proposes a subtree importance measure based on the feature importance information. Second, this chapter proposes to use the correlation between the behaviour (i.e., phenotypic) of a subtree and the whole tree to measure the importance of a subtree. The probability of a subtree to be chosen for the crossover operator is then set based on its importance. An offspring is generated by replacing an unimportant subtree from one parent with an important subtree from the other. The proposed algorithm is expected to help GPHH find better scheduling heuristics more efficiently by improving the quality of the produced offspring. Specifically, this chapter has the

following research objectives:

1. Develop effective strategies to measure the importance of subtrees of an individual according to the characteristics of the investigated DFJSS problem.
2. Propose a novel recombinative guidance mechanism for the crossover operator in GPHH.
3. Analyse the effectiveness of the proposed algorithm in terms of the quality of evolved scheduling heuristics.
4. Analyse the efficiency of the proposed algorithm based on the convergence speed and training time.
5. Analyse how the proposed recombinative guidance influences the behaviour of GPHH to select crossover points.
6. Analyse the evolved scheduling heuristics in terms of size and rule structure.

5.1.2 Chapter Organisation

The rest of this chapter is organised as follows. Detailed descriptions of the proposed algorithm are given in Section 5.2. The experiment design is shown in Section 5.3, followed by results and discussions in Section 5.4. Further analyses are conducted in Section 5.5. Finally, Section 5.6 concludes this chapter.

5.2 Proposed Algorithm

5.2.1 Framework of the Proposed Algorithm

Figure 5.1 shows the flowchart of the proposed algorithm. The main process is the same as the traditional GP. It starts with initialising the pop-

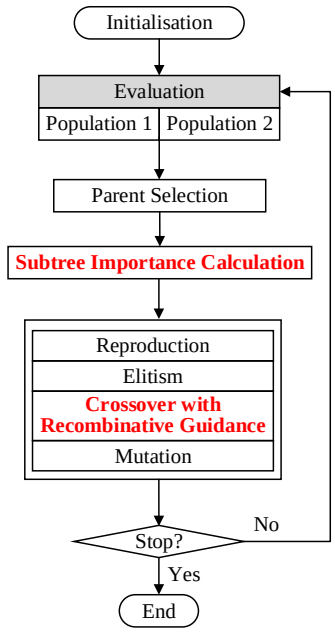

Figure 5.1: The flowchart of the proposed algorithm.

ulation randomly, and then evaluates the individuals in the population. It is noted that there are two subpopulations. One subpopulation is designed for evolving the routing rules, and the other for the sequencing rules. However, there are two new components that are different from the traditional GP, which are highlighted in red in Figure 5.1. First, the importance of each subtree of the parents is calculated before the mating process. Second, during the mating process, the crossover is conducted based on the proposed recombinative guidance mechanism.

According to the framework of the proposed algorithms, the two research questions in this chapter are (1) how to measure the importance of subtrees, and (2) how to apply the subtree importance information to guide the recombination between the parents via the crossover operator. This chapter proposes two strategies for subtree importance calculation for GP.

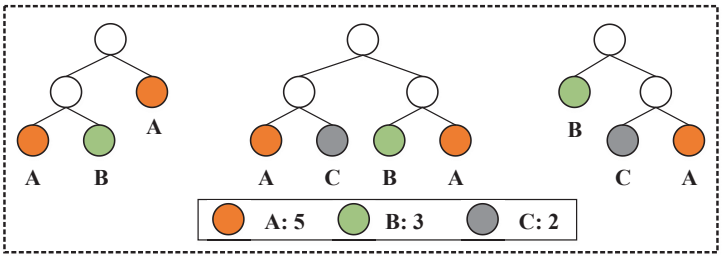

Figure 5.2: The occurrences of features in the top three individuals.

5.2.2 Subtree Importance Measure Based on Feature Importance

An advantage of GP is that it can automatically select important features to build individuals. The features of individuals with good fitness are more likely to be important. The individuals that contain important features are more likely to be promising individuals. This indicates that the features involved in promising individuals can be used to measure the importance of subtrees.

The Occurrences of Features

The occurrences of features in the top ten individuals are used to assess the importance of subtrees of an individual, since our preliminary studies show that the top ten individuals tend to have promising fitness which is good for detecting feature characteristics. Another advantage of using the occurrence information of features is that we do not need to put too much extra effort to obtain useful information, since the information is already generated during the evolutionary process. Figure 5.2 shows an example of how to extract feature occurrence information based on three individuals. These three individuals contain different numbers of features and have different structures. Assuming that they are the top three individuals in the population based on the fitness. According to the three individuals, the occurrences of features in all three individuals are counted. The occur-

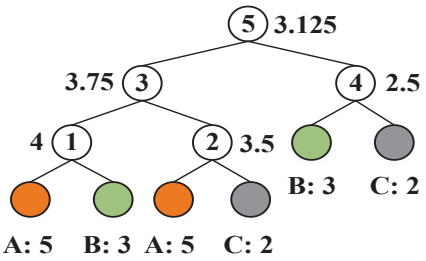

Figure 5.3: The importance (i.e., scores) of subtrees of an individual.

rences of feature A, B, and C are 5, 3, and 2, respectively. This information will be used to measure the importance of subtrees of an individual.

The Importance of Subtrees

An individual (i.e., a tree) can be considered to be composed of multiple subtrees. After a function node is selected, the subtree is determined. The importance of subtrees is measured from bottom to top, and we use the concept *score* to indicate the importance of a subtree. Each feature has its occurrence information at the bottom level of an individual, and the score of their parent node (i.e., the importance of subtree) is set as the average occurrence number of its child nodes. Assuming that the importance (i.e., occurrence) of feature A, B and C are ranked as $A > B > C$ (i.e., $5 > 3 > 2$). If only considering the simplest subtrees (i.e., depth is two) and only take two features, there will be three possible combinations for the subtree which are A and B, A and C, and B and C. The importance of the subtrees should be ranked as $subtree(A, B) > subtree(A, C) > subtree(B, C)$.

Figure 5.3 shows an example of how to measure the importance of each subtree for an individual. For example, the $subtree_1$ (i.e., in the bottom-left corner) contains two features (i.e., A and B), the score of their parent node is set as 4 (i.e., (5 + 3) / 2). The importance of $subtree_3$ (i.e., 3.75, (4 + 3.5) / 3) is assigned as the average scores of its two subtrees (i.e., $subtree_1$ and $subtree_2$). By analogy, the scores of all the subtrees will be assigned, as shown in Figure 5.3. Taking the subtrees with the depth of

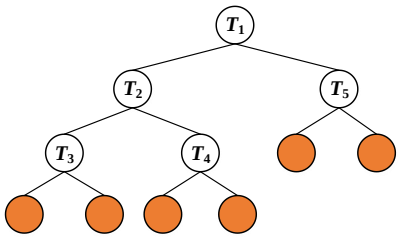

Figure 5.4: An example of a labelled tree-based GP individual.

two into consideration, there are three subtrees (i.e., indicated by subtree 1, 2 and 4) whose importance are marked as 4, 3.5 and 2.5, respectively. The importance of subtree 1, 2 and 4 are ranked as $subtree_1 > subtree_2 > subtree_4$, which is consistent with the importance measurement design. When looking at all the subtrees, the importance rank of all subtrees in this individual is $subtree_1 > subtree_3 > subtree_2 > subtree_5 > subtree_4$.

5.2.3 Subtree Importance Measure Based on the Correlation Between the Behaviour of Subtrees and the Whole Tree

Intuitively, a subtree is considered to be more important to the tree if it can make more consistent decisions with the entire tree. Figure 5.4 shows an example of a GP individual with five subtrees. Each subtree can be considered as an independent "individual", which has its own decision-making ability. To characterise the behaviour of a subtree T_i under a decision situation, this chapter uses a decision vector $\vec{d_i}$ (i.e., phenotypic characterisation) which is the list of the ranks of the candidates (i.e., machines for routing decision situations, or operations for sequencing decision situations) decided by T_i.

Table 5.1 shows an example of how to calculate the decision vectors of subtrees. The individual is a routing rule, and it has four subtrees. For simplicity, the decision situation is to allocate a ready operation to one of the three candidate machines. The numbers in the machine columns (i.e.,

Table 5.1: An example of the calculation for **decision vector** of the subtrees of an individual.

Subtree (T_i)	M_1	M_2	M_3	Decision Vector (d_i)
T_1	100 ①	150 ②	200 ③	(1, 2, 3)
T_2	300 ①	320 ②	350 ③	(1, 2, 3)
T_3	140 ③	120 ②	110 ①	(3, 2, 1)
T_4	100 ①	160 ③	130 ②	(1, 3, 2)

M_1, M_2 and M_3) are the priority values (i.e., real numbers) based on the corresponding subtrees (i.e., routing rules) and the ranks of the machines based on the priority values. A machine with a smaller priority value has a better priority than other machines. Finally, the decision vectors are composed of the ranks. It shows that different subtrees can have the same decisions (T_1 and T_2), opposite decisions (T_1 and T_3) or partially same decisions (T_1 and T_4). Since a decision is made solely based on the ranks rather than the exact priority values of the candidates, this chapter focuses on the relationship in terms of the ranks rather than the priority values.

Pearson and Spearman correlation coefficients [54] are two commonly used measures of the relationship between the two variables. Pearson's correlation coefficient assesses linear relationships [163], while Spearman's correlation coefficient assesses monotonic relationships (i.e., regardless of whether they are linear or not). Specifically, the Spearman correlation coefficient measures the statistical dependence between the rank values of two variables. The decision making processes of subtrees in DFJSS are based on the ranks of machines or operations. Therefore, the Spearman correlation coefficient is a natural candidate for measuring the correlation between the behaviour of subtrees. This chapter uses correlation c_i between the decisions (i.e., $\vec{d_i}$ and $\vec{d_1}$) made by T_i and T_1 (i.e., the whole tree) to measure the importance of a subtree T_i. The values range between -1 and 1. If $|c_i|$ is close to 1, the behaviour of T_i is highly consistent with T_1 (i.e., either positively or negatively), and T_i is an important subtree for an

Table 5.2: An example of the calculations for **correlation** of subtrees of an individual in a decision situation.

Subtree (T_i)	Decision Vector (d_i)	Correlation (c_i)
T_1	(1, 2, 3, 4, 5, 6)	1
T_2	(1, 2, 3, 4, 5, 6)	1
T_3	(1, 3, 2, 6, 4, 5)	0.77
T_4	(6, 5, 1, 2, 3, 4)	-0.43
T_5	(6, 5, 4, 3, 2, 1)	-1

individual. If $|c_i|$ is close to 0, the behaviour of T_i is almost irrelevant with the behaviour of T_1, and thus T_i is not important subtree for T_1.

Table 5.2 shows an example of the calculations for the correlation of subtrees of the individual shown in Figure 5.4. Different subtrees have different correlations (i.e., either positive or negative values). T_2 makes exactly the same decisions with T_1, and thus is a very important subtree of T_1. On the other hand, T_5 has a correlation of -1, which means its behaviour is completely reverse as the behaviour of T_1. In this case, T_5 is also a very important subtree of T_1, since its behaviour can be converted to be the same as that of T_1 by a slight modification, i.e., "$0 - T_1$". In contrast, T_3 and T_4 show relatively weaker relationship with T_1, and thus are considered to be less important than T_2 and T_5.

The pseudo-code of measuring the importance of a subtree is shown in Algorithm 6. An absolute value of the correlation closer to 1 leads to a more important subtree. It is noted that the correlation between the behaviours of two trees can vary across different decision situations since the characteristics of jobs (e.g., processing time) and machines (e.g., the workload) can be different. To have a reliable measure of the relationship, we sample a set of representative decision situations [99], and define the relationship between the behaviours of two trees to be the average correlation values over all the sampled decision situations. To sample a set of representative decision situations, this chapter uses the WIQ (i.e., work

Algorithm 6: Calculation of the importance of a subtree

Input : An individual T, a subtree T_i of T, and a set of decision situations
Output: The importance of the subtree T_i

1: $S(T_i) \leftarrow null$, $\vec{d_i} \leftarrow null$
2: $c_i \leftarrow 0$, $sum(c_i) \leftarrow 0$
3: **for** *j = 1 to* $|decisionSituations|$ **do**
4: Calculate the priority values of machines or operations based on the subtree T_i
5: Rank machines or operations based on the priority values
6: $\vec{d_i} \leftarrow$ get the decision vector of subtree T_i based on the ranks
7: $c_i \leftarrow$ calculate the correlation of $\vec{d_i}$ and $\vec{d_1}$
8: $sum(c_i) \leftarrow sum(c_i) + |c_i|$
9: **end**
10: $S(T_i) \leftarrow \frac{sum(c_i)}{|decisionSituations|}$
11: **return** $S(T_i)$

in the queue) rule for routing and the SPT (i.e., shortest processing time) rule for sequencing, and runs a preliminary simulation with 5000 jobs on 10 machines, which generate about 50,000 routing and 50,000 sequencing decision situations. In [99], decision situations were created randomly containing between 2 and 20 jobs, which have been proven to be good for measuring the behaviour of individuals by phenotypic characteristic. Taking the complexity of DFJSS into consideration, the number of candidates, either machines or operations, is 7 in this chapter for both the routing and sequencing decisions. Then, we randomly select 50 routing and 50 sequencing decision situations from the generated routing and sequencing decisions with a length of 7. This means that each subtree has a decision vector with a dimension of 7. The fixed dimension length aims to set the same length decision vectors for all the individuals to get feasible correlation value with the Spearman's correlation coefficient. In each decision situation, the priority values of machines or operations are calculated (line 4) to get their ranks (line 5). A vector $\vec{d_i}$ denotes the decision made by T_i (line 6). The correlation c_i between T_i and T_1 is used to measure the im-

portance of T_i (line 7). The final importance of subtree T_i is the average c_i over all the decision situations (line 10).

5.2.4 Crossover with Recombinative Guidance

A GP crossover operator is typically conducted on two parents ($parent_1$ and $parent_2$) which are both considered to be promising individuals in the population (e.g., selected by tournament selection). For each parent, it is reasonable to choose the unimportant subtree and replace it with an important subtree from the other. Based on the importance of subtrees $S(T)$, two probability calculations are designed for different purposes. One is designed for selecting important subtrees, while the other for selecting unimportant subtrees. Then, we design the crossover operator with recombinative guidance according to the probabilities.

The probability for each subtree. Based on the subtree importance information, this chapter uses the idea of roulette wheel selection to choose the desired subtrees. The probability of each subtree is proportional to its importance. We use the example shown in Table 5.2 with the correlation to show how to calculate the probability of each subtree. We assume that there is only one decision situation, and the calculated importance (i.e., using Algorithm 1) of subtrees from T_1 to T_5 are 1, 1, 0.77, 0.43, and 1.

Figure 5.5 shows an example of two different ways to calculate the probability of each subtree in an individual for crossover. Figure 5.5 (a) shows the way for choosing unimportant subtrees. A subtree with a larger score has a lower probability of being selected, which is shown as "↓" in the caption. Figure 5.5 (b) shows the way for choosing important subtrees. A subtree with a larger score has a higher the probability of being selected, which is indicated as "↑" in the caption. The ways to calculate the probabilities of subtrees follow the roulette-wheel selection according to the correlation values or the converted correlation values. As shown in Figure 5.5, at the beginning, the correlation values of subtrees are the same, as

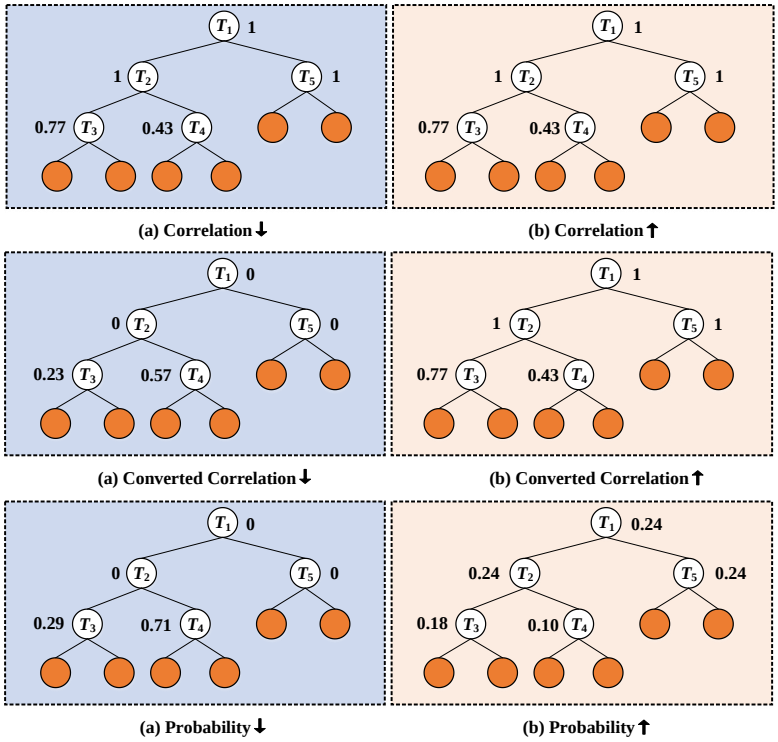

Figure 5.5: An example of calculating the probabilities for subtrees. Figure 5.5 (a) tends to choose unimportant subtrees while Figure 5.5 (b) tends to choose important subtrees.

shown in Figure 5.5 "(a) Correlation ↓" and Figure 5.5 "(b) Correlation ↑". However, different from Figure 5.5 (b), the correlation value of each subtree will be converted to $1 - S(T)$, as shown in Figure 5.5 "(a) Converted Correlation ↓", since we are choosing unimportant subtrees. The probabilities of subtrees are shown beside the function nodes in the last row of Figure 5.5. The rank of the probability of subtrees in Figure 5.5 (a) is $T_4 > T_3 > T_1 = T_2 = T_5$, and in Figure 5.5 (b) is $T_1 = T_2 = T_5 > T_3 > T_4$. In this way, this chapter can make sure that important and unimportant subtrees can be selected in accordance with the requirements.

The recombinative guidance mechanism. The pseudo-code of the proposed crossover operator is shown in Algorithm 7. The importance

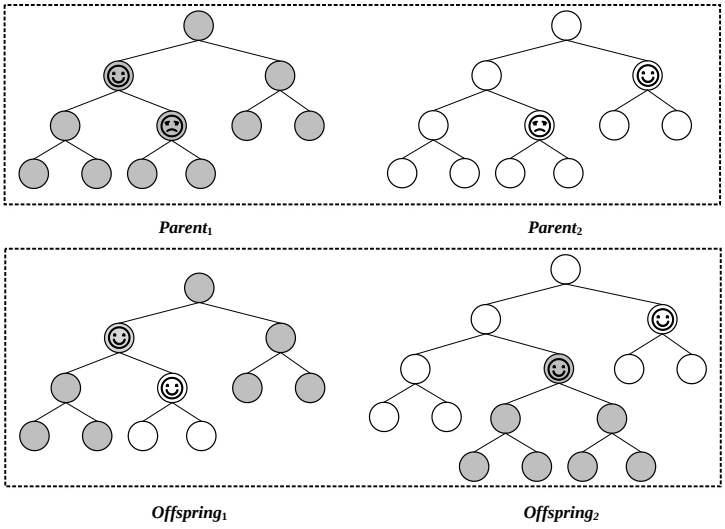

Figure 5.6: An example of produced offspring from two parents with the proposed recombinative guidance mechanism.

of subtrees of an individual is calculated before choosing important and unimportant subtrees based on roulette wheel selection (line 2 to line 8 for $parent_1$, line 9 to line 13 for $parent_2$). Finally, one offspring is produced by replacing the unimportant subtree $parent_1(T^*)^n$ from $parent_1$ with the important subtree $parent_2(T^*)^p$ from $parent_2$ (line 14). The other offspring is produced by replacing the unimportant subtree $parent_2(T^*)^n$ from $parent_2$ with the important subtree $parent_1(T^*)^p$ from $parent_1$ (line 15).

Continuing the example in Figure 5.5, Figure 5.6 shows an example of the produced offspring with the proposed recombinative guidance. For $parent_1$ ($parent_2$), the unimportant subtree with an unhappy face from $parent_1$ ($parent_2$) is expected to be replaced by the important subtree with a happy face from $parent_2$ ($parent_1$), aiming to produce an even better offspring. The produced offspring are expected to preserve the promising building-blocks of one parent and incorporate good building-blocks from the other parent (i.e., produce offspring with more happy faces).

Algorithm 7: Crossover with recombinative guidance

Input : Two parents for the crossover ($parent_1$ and $parent_2$)
Output: The generated offspring ($offspring$)

1: set $offspring \leftarrow$ null
2: **if** $parent_1$ **then**
3: $S(T) \leftarrow$ Calculate the importance of subtrees (**Algorithm 1**)
4: $S(T)^p \leftarrow |S(T)|$
5: $S(T)^n \leftarrow 1 - |S(T)|$
6: $parent_1(T^*)^p \leftarrow$ Selected important subtree based on roulette wheel selection with $S(T)^p$
7: $parent_1(T^*)^n \leftarrow$ Selected unimportant subtree based on roulette wheel selection with $S(T)^n$
8: **end**
9: **if** $parent_2$ **then**
10: repeat from line 3 to line 5
11: $parent_2(T^*)^p \leftarrow$ Selected important subtree based on roulette wheel selection with $S(T)^p$
12: $parent_2(T^*)^n \leftarrow$ Selected unimportant subtree based on roulette wheel selection with $S(T)^n$
13: **end**
14: $offspring_1 \leftarrow$ produce offspring by replacing the subtree chosen from $parent_1(T^*)^n$ with $parent_2(T^*)^p$
15: $offspring_2 \leftarrow$ produce offspring by replacing the subtree chosen from $parent_2(T^*)^n$ with $parent_1(T^*)^p$
16: $offspring \leftarrow offspring_1 \cup offspring_2$
17: **return** $offspring$

5.2.5 Algorithm Summary

The proposed algorithm aims to improve the effectiveness of crossover by introducing recombinative guidance mechanism rather than choosing subtrees randomly. We assume that removing an unimportant subtree from an individual does not make a big difference to its fitness. However, introducing an important subtree to the position of the removed unimportant subtree has a high probability of making the individual better. It is noted that the idea in this chapter is not limited to DFJSS but can benefit

GP in general. An important issue is to design a proper measure for the subtree importance based on the specific problem to be solved. Taking the symbolic regression problem as an example, the subtree importance can be measured with sampling semantics [222].

5.3 Experiment Design

5.3.1 Comparison Design

The goal of this chapter is to improve the effectiveness and efficiency of the crossover operator of GPHH with recombinative guidance mechanism to evolve effective scheduling heuristics for DFJSS. Three algorithms are taken into the comparison in this chapter. The cooperative coevolution genetic programming (CCGP) [233] is selected as the baseline algorithm. CCGP uses the uniform crossover operator. The proposed algorithm in this chapter that measures the subtree importance based on feature importance is named as $\mathrm{CCGP}^{\mathrm{f}}$ [238], while the proposed algorithm that measures the subtree importance based on the correlation between subtrees and the whole tree is named as $\mathrm{CCGP}^{\mathrm{c}}$. In addition, to further verify the proposed subtree importance measure and recombinative guidance mechanism, we compare with a reverse algorithm named $\mathrm{CCGP}^{\mathrm{!c}}$ that uses unimportant subtrees to replace important subtrees to GPHH.

In order to verify their effectiveness and efficiency, the proposed algorithm is tested on six scenarios. The scenarios consist of three objectives (i.e., max-flowtime, mean-flowtime, and mean-weighted-flowtime) and two utilisation levels (i.e., 0.85 and 0.95).

5.3.2 Specialised Parameter Settings of GPHH

The specialised parameter settings of GPHH are shown in Table 5.3. This chapter adopts the representation with cooperative coevolution along with

Table 5.3: The specialised parameter settings of GPHH.

Parameter	Value
Number of subpopulations	2
Subpopulation size	512
The number of elites for each subpopulation	5

two subpopulations, since the crossover operation of the routing and sequencing individuals is independent in this way. Therefore, there are two subpopulations of the algorithms in this chapter. This makes it convenient for validating the effectiveness of the proposed crossover operator.

5.4 Results and Discussions

5.4.1 Quality of the Evolved Scheduling Heuristics

Table 5.4 shows the mean (standard deviation) of the objective values of the four compared algorithms on unseen instances based on 50 independent runs in six DFJSS scenarios. It can be seen that $\mathrm{CCGP}^{\mathrm{f}}$ shows similar performance with CCGP in all the scenarios. One possible reason is that using the occurrences of features to measure the importance of subtrees may not be accurate due to the redundant branches in GP. $\mathrm{CCGP}^{\mathrm{c}}$ is significantly better than CCGP in half of the scenarios (i.e., <Fmean, 0.85>, <WFmean, 0.85> and <WFmean, 0.95>) and no worse in all other scenarios. Although $\mathrm{CCGP}^{\mathrm{c}}$ is not significantly better than that of CCGP in the scenarios <Fmax, 0.85> and <Fmean, 0.95>, it still shows its superiority in terms of the mean and standard deviation values obtained. In addition, $\mathrm{CCGP}^{\mathrm{c}}$ is significantly better than $\mathrm{CCGP}^{\mathrm{f}}$ in the most complex scenario (<WFmean, 0.95>), which is shown in bold. $\mathrm{CCGP}^{\mathrm{!c}}$ is significantly worse than all other algorithms, which is as expected. This verifies the effectiveness of proposed subtree importance measure with correlation coefficient

Table 5.4: The mean (standard deviation) of the objective values of CCGP, $\mathrm{CCGP}^{\mathrm{f}}$, $\mathrm{CCGP}^{\mathrm{c}}$ and $\mathrm{CCGP}^{!\mathrm{c}}$ on unseen instances over 50 independent runs in six DFJSS scenarios.

Scenario	CCGP	$\mathrm{CCGP}^{\mathrm{f}}$	$\mathrm{CCGP}^{\mathrm{c}}$	$\mathrm{CCGP}^{!\mathrm{c}}$
<Fmax, 0.85>	1212.05(34.68)	1215.55(32.62)(≈)	1211.83(27.45)(≈)	1291.96(48.23)**(+)**
<Fmax, 0.95>	1941.98(29.93)	1939.84(32.97)(≈)	1942.09(29.16)(≈)	2026.88(80.15)**(+)**
<Fmean, 0.85>	385.95(3.22)	384.66(1.19)(≈)	384.68(1.92)**(–)**	389.79(3.96)**(+)**
<Fmean, 0.95>	551.18(5.78)	551.11(3.81)(≈)	550.30(3.72)(≈)	563.76(10.14)**(+)**
<WFmean, 0.85>	831.41(6.08)	829.89(4.76)(≈)	828.98(3.57)**(–)**	841.22(9.78)**(+)**
<WFmean, 0.95>	1111.01(12.02)	1109.52(11.27)(≈)	**1105.84(7.21)(–)**	1141.54(23.04)**(+)**

technique and recombinative guidance from an opposite perspective.

Figure 5.7 shows the violin plot of the test objective values of CCGP, $\mathrm{CCGP}^{\mathrm{f}}$, $\mathrm{CCGP}^{\mathrm{c}}$ and $\mathrm{CCGP}^{!\mathrm{c}}$ over 50 independent runs in six DFJSS scenarios. It shows that although $\mathrm{CCGP}^{\mathrm{f}}$ is not significantly better than CCGP in any scenario, it achieves better performance than CCGP in most scenarios (i.e., <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>). For $\mathrm{CCGP}^{\mathrm{c}}$, most obtained test objective values distribute at a lower position (i.e., the smaller, the better) than that of $\mathrm{CCGP}^{\mathrm{f}}$ and CCGP in the scenarios <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>. In addition, the obtained objectives of $\mathrm{CCGP}^{!\mathrm{c}}$ are larger than the other compared algorithms, since the idea of $\mathrm{CCGP}^{!\mathrm{c}}$ is the opposite of that of the proposed algorithm $\mathrm{CCGP}^{\mathrm{c}}$. This verifies the effectiveness of the proposed subtree importance measure and the proposed recombinative guidance based on replacing unimportant subtrees with important ones.

Figure 5.8 shows the converges curves of the average objective values based on 50 independent runs on the unseen instances of CCGP, $\mathrm{CCGP}^{\mathrm{f}}$, $\mathrm{CCGP}^{\mathrm{c}}$ and $\mathrm{CCGP}^{!\mathrm{c}}$ in six DFJSS scenarios. In most scenarios (i.e., <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>), $\mathrm{CCGP}^{\mathrm{c}}$ achieves better performance than its counterparts. In half of the scenar-

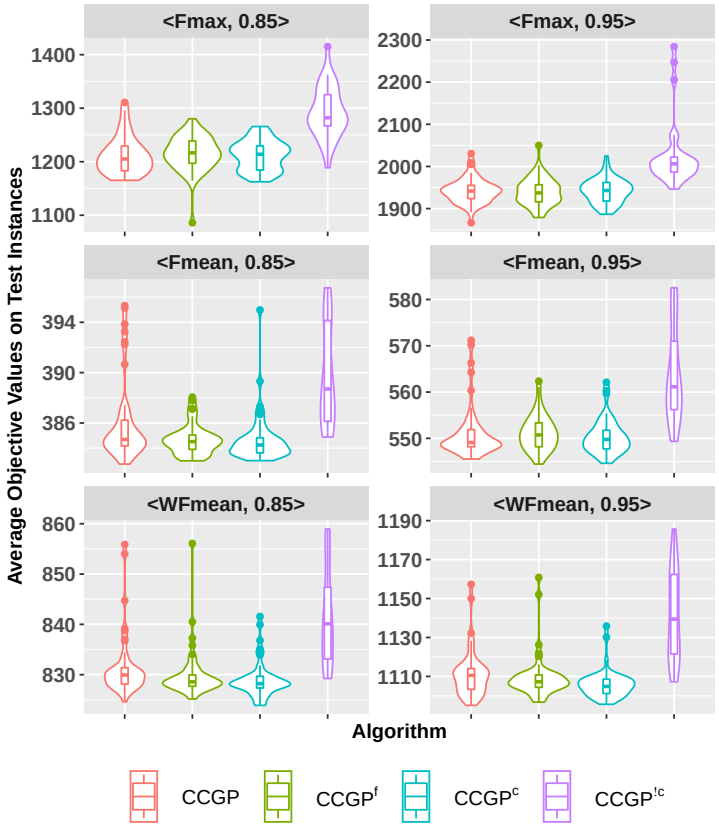

Figure 5.7: The violin plot of the average objective values of CCGP, CCGP^{f}, CCGP^{c} and $\text{CCGP}^{\text{!c}}$ on unseen instances over 50 independent runs in six DFJSS scenarios.

ios (i.e., <Fmean, 0.85>, <WFmean, 0.85> and <WFmean, 0.95>), CCGP^{c} convergence much faster than CCGP and CCGP^{f}. In addition, the individuals evolved by $\text{CCGP}^{\text{!c}}$ are much worse than the other algorithms over generations, which also demonstrates the effectiveness of CCGP^{c}. For the max-flowtime related scenarios (i.e., <Fmax, 0.85> and <Fmax, 0.95>), the performance of the involved three algorithms do not have obvious difference. This may be because max-flowtime is not easy to be optimised due to its sensitivity to the worst case.

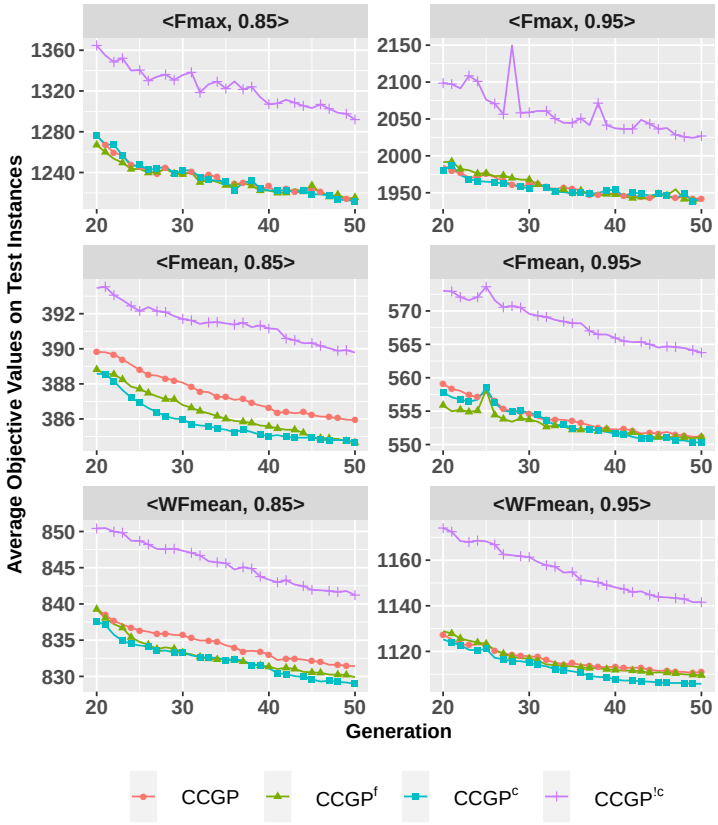

Figure 5.8: The curves of the average objective values of CCGP, CCGP$^{\mathrm{f}}$, CCGP$^{\mathrm{c}}$ and CCGP$^{\mathrm{lc}}$ on unseen instances over 50 independent runs in six DFJSS scenarios.

5.4.2 Depth Ratios of Selected Subtrees

Both CCGP$^{\mathrm{c}}$ and CCGP$^{\mathrm{f}}$ attempt to choose proper crossover points, however, only CCGP$^{\mathrm{c}}$ shows its superiority. It is interesting to analyse the different behaviours of CCGP$^{\mathrm{c}}$ and CCGP$^{\mathrm{f}}$. We define the *depth ratio* to measure the location of the selected subtrees of a tree. The depth ratio is the division of the depth where a selected subtree on and the depth of the tree. A smaller (larger) ratio lends to a closer location of the selected subtree to the root node (terminals) of a tree.

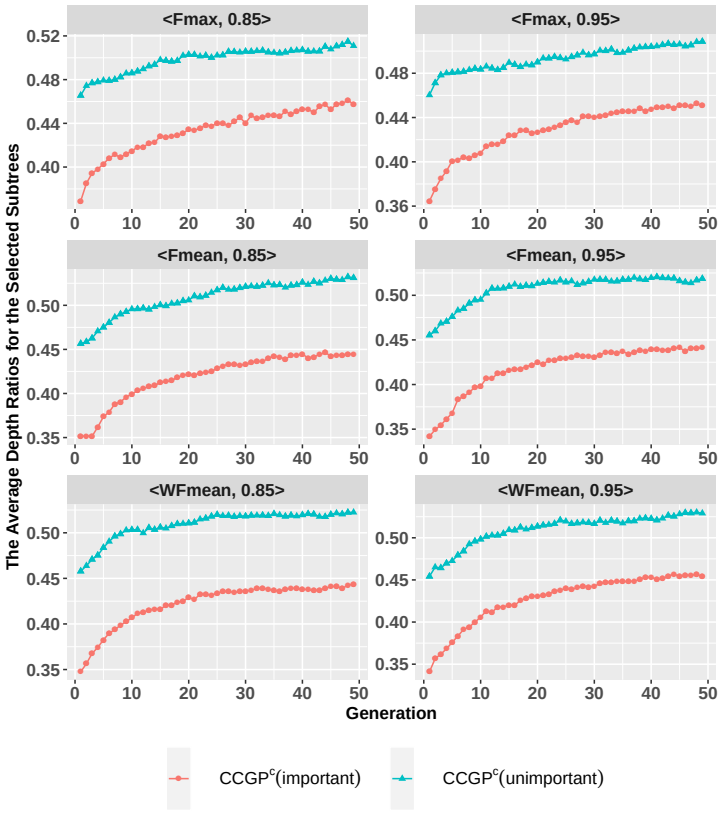

Figure 5.9: The curves of the average depth ratios for the selected important and unimportant subtrees of $\mathrm{CCGP^c}$ over 50 independent runs in six DFJSS scenarios.

Figure 5.9 and Figure 5.10 show the average depth ratios for the selected important and unimportant subtrees of $\mathrm{CCGP^c}$ and $\mathrm{CCGP^f}$ over 50 independent runs in six DFJSS scenarios, respectively. For both $\mathrm{CCGP^c}$ and $\mathrm{CCGP^f}$, the depth ratios of important subtrees are smaller than that of unimportant subtrees. This is consistent with our intuition that the subtrees closer to the root are more likely to be important subtrees because they contain more comprehensive components. The gaps in depth ratios between important subtrees and unimportant subtrees of $\mathrm{CCGP^c}$ is much bigger than that of $\mathrm{CCGP^f}$. In addition, it can be seen that $\mathrm{CCGP^c}$ can de-

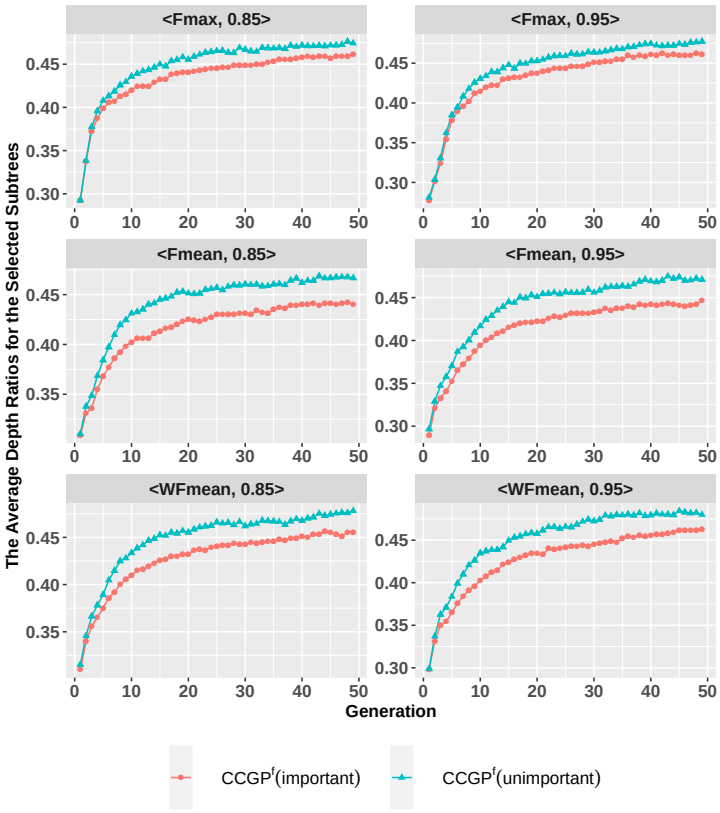

Figure 5.10: The curves of the average depth ratios for the selected important and unimportant subtrees of $\mathrm{CCGP^f}$ over 50 independent runs in six DFJSS scenarios.

tect important and unimportant subtrees better than $\mathrm{CCGP^f}$ in the early stage (i.e., before generation 10).

Figure 5.11 shows the curves of average depth ratios of important subtrees obtained by the 50 independent runs of CCGP, $\mathrm{CCGP^f}$ and $\mathrm{CCGP^c}$ in six DFJSS scenarios. It shows that the depth ratios of the selected important trees of CCGP, $\mathrm{CCGP^f}$ and $\mathrm{CCGP^c}$ are similar to each other. The average depth ratios of important subtrees of CCGP, $\mathrm{CCGP^f}$, and $\mathrm{CCGP^c}$ are consistently between 0.4 and 0.45 after generation 10, which means we do not usually select the important subtrees towards the root. In general,

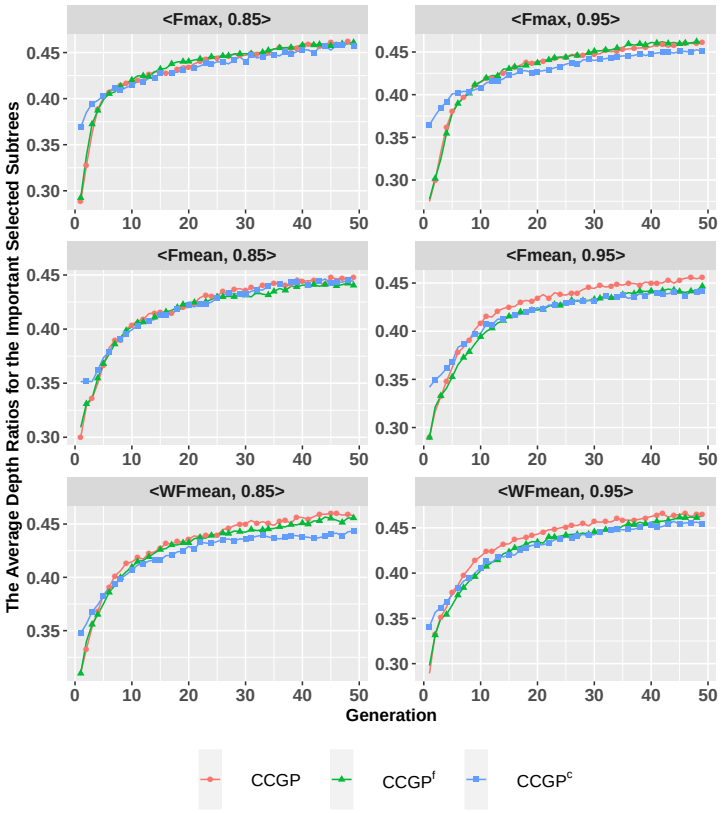

Figure 5.11: The curves of the average depth ratios of **important subtrees** obtained by CCGP, $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ over 50 independent runs in six DFJSS scenarios.

the depth ratios of the selected important subtrees of $\mathrm{CCGP}^{\mathrm{c}}$ are slightly smaller than its counterparts. However, the main difference is that $\mathrm{CCGP}^{\mathrm{c}}$ prefers to choose the subtrees which are further away from root node with a larger depth ratio while CCGP and $\mathrm{CCGP}^{\mathrm{f}}$ tend to select the subtrees that are closer to the root node with a smaller depth ratio in the early stage (i.e., from generation 1 to generation 5 roughly). It implies that the ability of $\mathrm{CCGP}^{\mathrm{f}}$ to detect promising subtrees is limited at the early stage. One possible reason is that the occurrences of features are not accurate to measure the importance of features. This shortcoming is very obvious at the

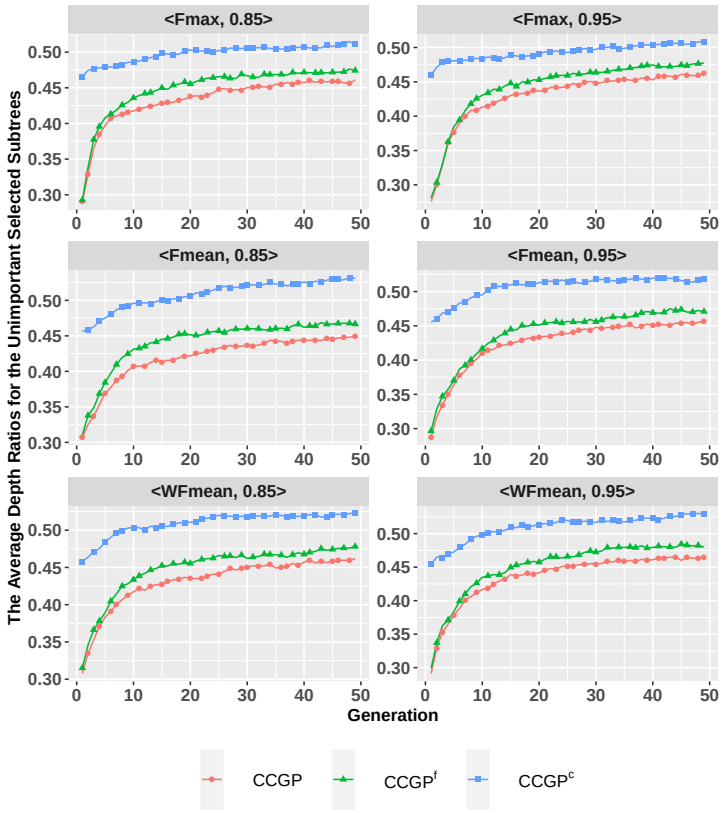

Figure 5.12: The curves of the average depth ratios of **unimportant subtrees** of CCGP, $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ over 50 independent runs in six DFJSS scenarios.

early stage because the individuals have not evolved well yet, and the occurrence information of features is not reliable.

Figure 5.12 shows the curves of the average depth ratios of unimportant subtrees obtained by CCGP, $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ based on 50 independent runs in six DFJSS scenarios. Figure 5.12 shows that CCGP, $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ make clearly different decisions when selecting unimportant subtrees. On one hand, both $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ tend to choose the subtrees with larger tree depths, i.e., on the lower parts of an individual. On the other hand, compared with $\mathrm{CCGP}^{\mathrm{f}}$, the depth ratios of the unimportant

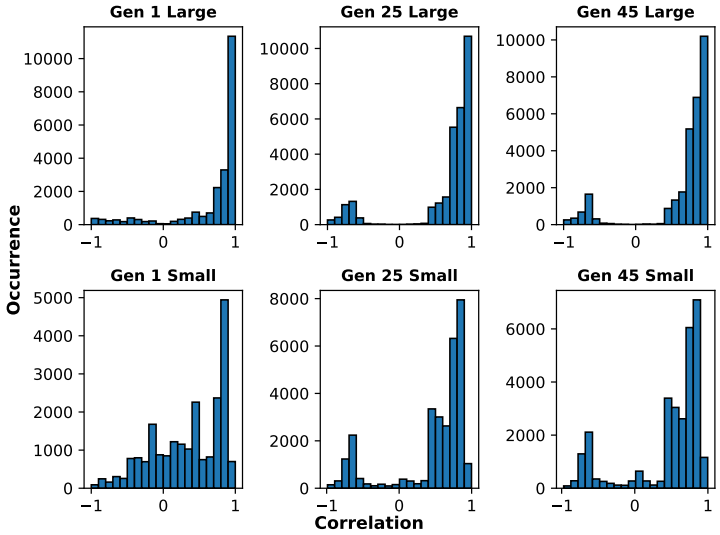

Figure 5.13: The histogram plot for the **correlations** of the selected subtrees of $\mathrm{CCGP}^{\mathrm{c}}$ at generation 1, 25, and 45 in the scenario <WFmean, 0.95> over 50 independent runs.

subtrees of $\mathrm{CCGP}^{\mathrm{c}}$ are much larger. At the later stage (i.e., from generation 10 to generation 50 roughly), the depth ratios of $\mathrm{CCGP}^{\mathrm{f}}$ fluctuate around 0.45 while the depth ratios of $\mathrm{CCGP}^{\mathrm{c}}$ show a trend of fluctuation around 0.5. This means that $\mathrm{CCGP}^{\mathrm{c}}$ treats the subtrees closer to the leaf nodes as unimportant subtrees.

5.4.3 Correlations of Selected Subtrees

The correlations of subtrees determine the subtree selection probabilities. Figure 5.13 shows the histogram plot for the correlations of the selected subtrees of $\mathrm{CCGP}^{\mathrm{c}}$ at early, middle and late stages over 50 independent runs in the scenario <WFmean, 0.95>. The "Gen X Large (Small)" in the subtitle indicates that the subtree with a larger (smaller) score has a higher (lower) chance of being chosen. All the correlations are between -1 and 1. From the sub-figures in the first row (i.e., for selecting important subtrees),

we find that the subtrees with correlations of zero are seldom selected, and the absolute values of the correlations of the selected subtrees are close to 1. This is in line with our expectation because we tend to choose important subtrees with larger absolute correlation values.

For selecting unimportant subtrees, as shown in the sub-figures of the second row, much more selected subtrees have their correlations close to zero, especially at the early stage (generation 1). However, it is inconsistent with our intuition that there are still lots of correlations of the selected unimportant subtrees between 0.5 and 1 at generation 25 and generation 45. When we further look at the correlations during the process of selecting unimportant subtrees, we find that it can occur that all of the subtrees in an individual are important with the correlations range between 0.5 and 1, especially in the middle and late stages. This is the reason why the correlations of the selected unimportant subtrees show in such a distribution. In other words, the proposed algorithm still chooses relatively unimportant subtrees.

5.4.4 Probability Difference

The basic idea in this chapter is to differentiate the probabilities of subtrees to be chosen instead of choosing subtrees randomly. We use *probability difference* to measure how the proposed algorithm influences the chance of subtrees to be selected. The probability difference is defined as the difference (i.e., subtraction) between the assigned probability by the proposed recombinative guidance mechanism and the uniform probability of the selected subtree. It is noted that the probability difference can be positive, negative, and zero. A positive probability difference indicates that the current subtree is selected with a higher chance compared with uniform probability. A negative probability difference means that the current subtree is selected with a lower chance than uniform probability. If the probability difference is zero, the assigned probability is the same as the uniform

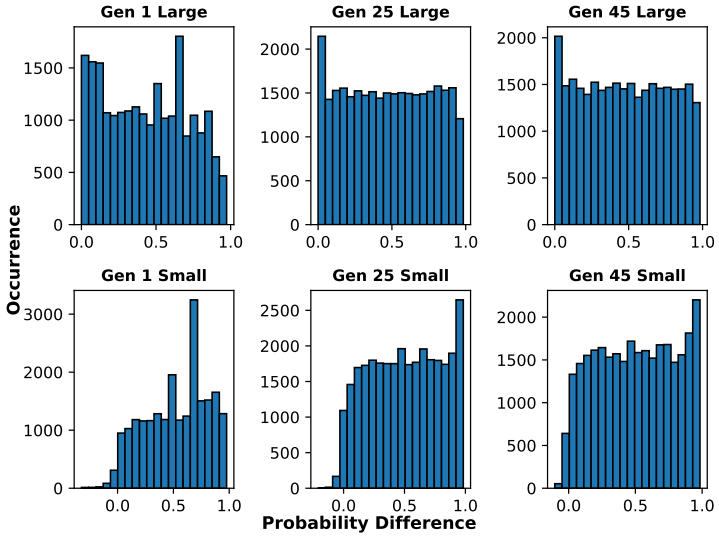

Figure 5.14: The histogram plot of **probability difference** of the selected subtrees of $\mathrm{CCGP}^{\mathrm{c}}$ at generation 1, 25, and 45 in the scenario <WFmean, 0.95> over 50 independent runs.

probability. This means that the proposed algorithm does not have effective guidance on choosing the crossover point for producing offspring.

We take $\mathrm{CCGP}^{\mathrm{c}}$ in the scenario <WFmean, 0.95> as an example to investigate how $\mathrm{CCGP}^{\mathrm{c}}$ affects subtree selection, since $\mathrm{CCGP}^{\mathrm{c}}$ performs significantly better than the other two algorithms in this scenario. Figure 5.14 shows the histogram plot of the probability difference in the early (generation 1), middle (generation 25) and late (generation 45) stages of the evolutionary process in the scenario <WFmean, 0.95> over 50 independent runs. Overall, most of the probability differences are positive numbers. This indicates that the proposed algorithm increases the probabilities of both selecting important and unimportant subtrees. This is in line with our expectation that $\mathrm{CCGP}^{\mathrm{c}}$ can successfully guide GPHH to choose important or unimportant subtrees for crossover as required.

Table 5.5: The mean (standard deviation) of training time (in minutes) of CCGP, CCGP^{f}, and CCGP^{c} over 50 independent runs in six DFJSS scenarios.

Scenario	CCGP	CCGP^{f}	CCGP^{c}
<Fmax, 0.85>	73(9)	74(13)(≈)	74(11)(≈)
<Fmax, 0.95>	87(15)	88(13)(≈)	89(12)(≈)
<Fmean, 0.85>	71(10)	72(10)(≈)	72(9)(≈)
<Fmean, 0.95>	80(13)	81(11)(≈)	81(12)(≈)
<WFmean, 0.85>	73(13)	75(16)(≈)	74(15)(≈)
<WFmean, 0.95>	82(13)	82(12)(≈)	83(13)(≈)

5.4.5 Training Time

Table 5.5 shows the mean and standard deviation of the training time (in minutes) of CCGP, CCGP^{f} and CCGP^{c} over 50 independent runs in six DFJSS scenarios. It is obvious that there is no significant difference among CCGP, CCGP^{f} and CCGP^{c} in terms of training time. In other words, the extra subtree importance calculations of CCGP^{f} and CCGP^{c} are efficient compared with the GP evolution and evaluation, since it does not incur significantly longer training time.

For CCGP^{f}, it verifies the advantages of taking the information such as the occurrences of terminals during the evolutionary process of GP to improve the algorithm further. For CCGP^{c}, it verifies the advantages of taking some techniques such as correlation coefficient that can be quickly utilised along with the information during the evolutionary process of GP to enhance the performance of the algorithm.

5.5 Further Analyses

To deeply understand the effects of the proposed algorithms, the occurrences of the potential unsuccessful crossover whose offspring exceed the

maximum depth limit are firstly studied. Then, the sizes of evolved rules, the insight of the evolved scheduling heuristics of $\mathrm{CCGP}^{\mathrm{c}}$, and the occurrences of features of $\mathrm{CCGP}^{\mathrm{f}}$ are further analysed.

5.5.1 Occurrences of Potential Invalid Crossover

The sizes of offspring highly depend on the depth ratios of selected subtrees from parents. Intuitively, a subtree closer to the root (with larger depth ratio) tends to be more important, while a subtree closer to the leaf nodes (with smaller depth ratio) tends to be less important. If a subtree closer to the root of a parent replaces a subtree closer to the leaf nodes of the other parent, the produced offspring tends to have a large size. We are interested in how the proposed algorithm affects the size of offspring, since the offspring whose depths are larger than eight (the maximal depth set in the experiment) will be ignored during the crossover.

We record the number of "invalid" crossover which generates an offspring whose depth is larger than eight. We name the "invalid" crossover as *potential invalid replacements* since the produced offspring are ignored, and the crossover actually does not happen. The number of potential invalid replacements can be used to investigate how the proposed algorithm influences the process of generating offspring. Figure 5.15 shows the average potential invalid replacements for the crossover of CCGP, $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ at each generation over 50 independent runs in the six DFJSS scenarios. In all the scenarios, $\mathrm{CCGP}^{\mathrm{c}}$ leads to more potential invalid replacements than $\mathrm{CCGP}^{\mathrm{f}}$ over the generations. It is consistent with the analyses in Section 5.4.4. In $\mathrm{CCGP}^{\mathrm{c}}$, the unimportant subtrees with larger depth ratios are more likely to be replaced by the important subtrees with smaller depth ratios, which leads to more potential invalid replacements. Fortunately, it does not have a significant impact on the effectiveness of the proposed algorithm.

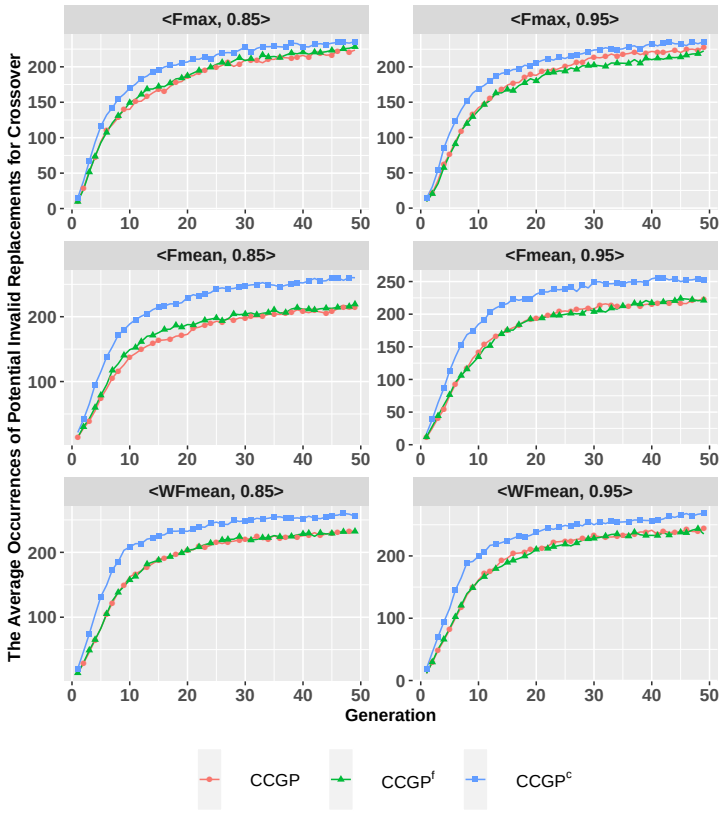

Figure 5.15: The curve of average occurrences of **potential invalid replacements** of $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$ over 50 independent runs in six DFJSS scenarios.

5.5.2 Sizes of Evolved Scheduling Heuristics

The size (i.e., the number of nodes) can be a measure for the "interpretability" [81] of the evolved rules. A smaller rule can be more easily interpreted than a larger rule. In this subsection, we investigate how the proposed algorithm influences the sizes of the evolved rules in terms of the sizes of the evolved best rules. Table 5.6 shows the mean and standard deviation of the sizes of the evolved best routing and sequencing rules in six DFJSS scenarios. Compared with CCGP, there is no statistical significant difference between the sizes of evolved routing and sequencing rules obtained

Table 5.6: The mean (standard deviation) of the sizes of evolved the best routing and sequencing rules of CCGP, $\mathrm{CCGP}^{\mathrm{f}}$, and $\mathrm{CCGP}^{\mathrm{c}}$ over 50 independent runs in six DFJSS scenarios.

	Routing Rule		
Scenario	CCGP	$\mathrm{CCGP}^{\mathrm{f}}$	$\mathrm{CCGP}^{\mathrm{c}}$
<Fmax, 0.85>	61.48(18.30)	68.68(18.68)(≈)	62.32(19.07)(≈)
<Fmax, 0.95>	59.28(19.20)	66.28(20.30)(≈)	60.68(18.87)(≈)
<Fmean, 0.85>	59.84(15.05)	61.52(16.21)(≈)	59.40(18.09)(≈)
<Fmean, 0.95>	64.16(19.42)	65.28(16.45)(≈)	59.60(16.52)(≈)
<WFmean, 0.85>	59.00(17.35)	63.12(17.99)(≈)	64.52(17.20)(≈)
<WFmean, 0.95>	63.88(15.95)	61.44(15.30)(≈)	65.20(18.11)(≈)
	Sequencing Rule		
Scenario	CCGP	$\mathrm{CCGP}^{\mathrm{f}}$	$\mathrm{CCGP}^{\mathrm{c}}$
<Fmax, 0.85>	54.40(18.12)	51.36(15.01)(≈)	53.72(18.23)(≈)
<Fmax, 0.95>	51.32(16.34)	53.92(16.77)(≈)	50.08(19.32)(≈)
<Fmean, 0.85>	46.64(19.98)	47.32(18.43)(≈)	45.32(15.22)(≈)
<Fmean, 0.95>	44.92(16.04)	44.80(15.47)(≈)	42.12(19.94)(≈)
<WFmean, 0.85>	46.44(18.77)	50.92(13.93)(≈)	46.32(18.32)(≈)
<WFmean, 0.95>	47.04(18.45)	52.92(20.33)(≈)	51.68(19.11)(≈)

by $\mathrm{CCGP}^{\mathrm{f}}$ and $\mathrm{CCGP}^{\mathrm{c}}$. We can conclude that the proposed $\mathrm{CCGP}^{\mathrm{c}}$ with recombinative guidance achieves better performance without having an impact on the sizes of the evolved rules.

5.5.3 Insight on the Evolved Scheduling Heuristics

To study the behaviours of the evolved rules obtained by $\mathrm{CCGP}^{\mathrm{c}}$, this section conducts structural analyses on the evolved sequencing rules. Specifically, the best sequencing rules obtained by $\mathrm{CCGP}^{\mathrm{c}}$ for minimising max-flowtime and mean-weighted-flowtime with utilisation level of 0.95 are further investigated, respectively.

Evolved Rule for Max-flowtime. Figure 5.16 shows one of the best

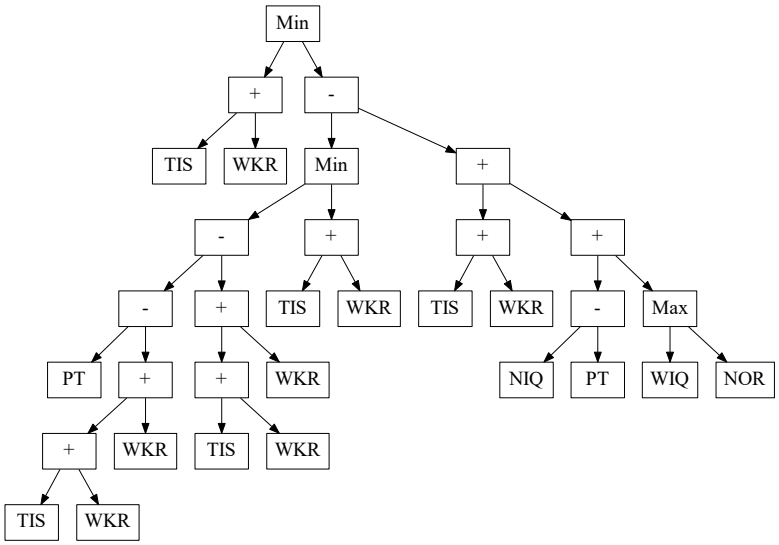

Figure 5.16: One of the best evolved sequencing rules evolved by $\mathrm{CCGP}^{\mathrm{c}}$ in scenario <Fmax, 0.95>.

evolved sequencing rules by $\mathrm{CCGP}^{\mathrm{c}}$ in the scenario <Fmax, 0.95>. It is observed that the rule is a combination of six simple terminals (TIS, WKR, PT, NIQ, WIQ, and NOR), and TIS and WKR are the most frequently used terminals for building this rule. In addition, "TIS + WKR" might be an effective constructed building block for this sequencing rule, since it appears five times in this rule.

To make analysis easy, the rule in Figure 5.16 is further simplified, as shown in Eq. 5.1.

$$\begin{aligned} S_1 =& Min\{TIS + WKR, \\ & Min\{PT - 2TIS - 4WKR, TIS + WKR\} - \\ & (TIS + WKR + NIQ - PT + Max\{WIQ, NOR\})\} \\ \approx& Min\{TIS + WKR, \\ & 2PT - 3TIS - 5WKR - NIQ - WIQ\} \\ \approx& 2PT - 3TIS - 5WKR - NIQ - WIQ \\ =& 2PT - 3TIS - 5WKR \end{aligned} \tag{5.1}$$

From step 1 to step 2, "Min{PT - 2TIS - 4WKR, TIS + WKR}" is simplified as "PT - 2TIS - 4WKR", since "PT - 2TIS - 4WKR" is almost always smaller than "TIS + WKR". In addition, "Max{WIQ, NOR}" is represented as WIQ, since WIQ (time) tends to be larger than NOR (between 1 and 10). Similarly, the rule in step 2 can be mostly replaced by the rule in step 3. Finally, NIQ and WIQ are the same for all operations in the same queue, and can be safely ignored, since they do not affect the final decision of choosing an operation. This rule suggests that when a machine is idle, the machine should process the operation with small processing time first. In addition, the jobs that arrive at the shop floor earlier or have more remaining work should be processed earlier. Otherwise, if they are completed too late, the max-flowtime will be increased. It is consistent with our intuition for minimising max-flowtime due to its sensitivity to the worst case. The weights of the terminals may require domain knowledge and many rounds of trial-and-error if manually designed.

Evolved Rule for Mean-weighted-flowtime. Figure 5.17 shows one of the best evolved sequencing rules obtained by $\mathrm{CCGP}^{\mathrm{c}}$ in the scenario <WFmean, 0.95>. This rule consists of four simple terminals (WKR, W, PT, MWT), and four functions (+, /, Max, Min). "WKR / W" is an important learned component in this rule, and it appears four times. W tends to play its role as a denominator. PT is also an important terminal which mainly plays its role as a component for addition.

The simplification of the sequencing rule in Figure 5.17 is shown in Eq. 5.2.

$$\begin{aligned} S_2 =& (Min\{WKR/W + PT, WKR\} + PT \\ & + Min\{WKR/W + PT, WKR/W + WKR\} + PT \\ & + PT + Max\{WKR/W, PT\} + W)/(W + MWT) \\ =& (Min\{WKR/W + PT, WKR\} + 3PT \\ & + Min\{PT, WKR\} \\ & + Max\{WKR/W, PT\} + W)/(W + MWT) \end{aligned} \tag{5.2}$$

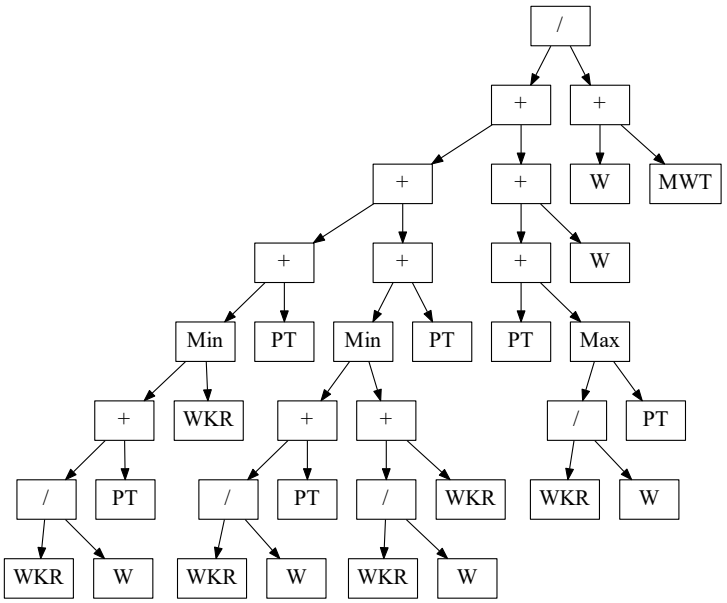

Figure 5.17: One of the best evolved sequencing rules evolved by $\mathrm{CCGP}^{\mathrm{c}}$ in the scenario <WFmean, 0.95>.

This rule suggests to process the important operation with a large W earlier. In addition, the operations with short processing time and the jobs with small remaining work are preferred to be processed as soon as possible. Otherwise, the weighted-flowtime will increase.

In summary, this section shows the advantage of evolving scheduling heuristics with the proposed algorithm. The evolved scheduling heuristics consist of simple heuristics but are combined in an effective way, which is not easy to be designed manually. In addition, the evolved scheduling heuristics have good interpretability, which is important for real-world applications.

5.5.4 Occurrences of Features

Figure 5.18 shows the curves of the occurrence of features in routing rules during the evolutionary process of $\mathrm{CCGP}^{\mathrm{c}}$. The MWT (i.e., machine wait-

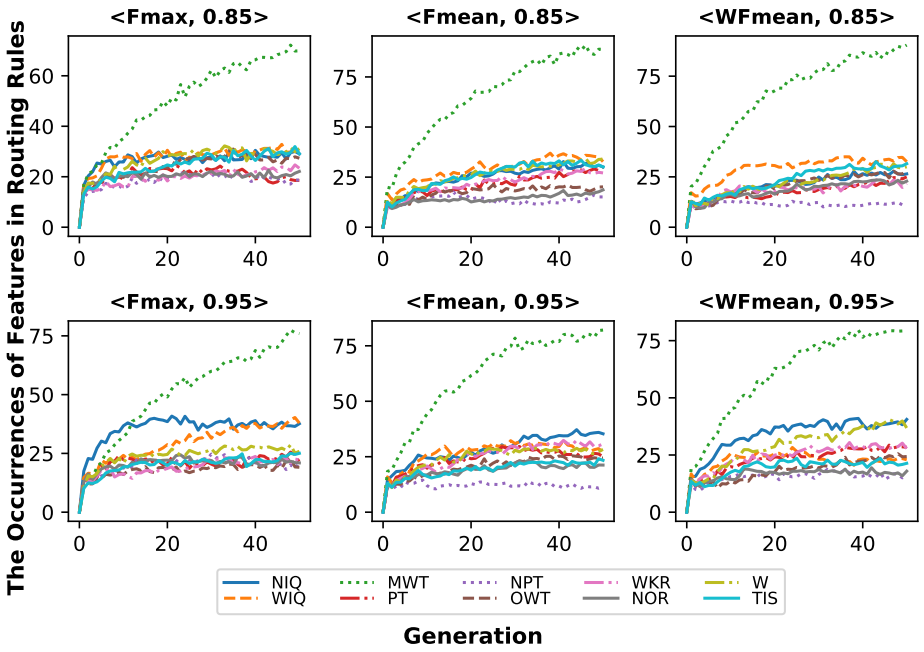

Figure 5.18: The curves of the occurrence of features in **routing rules** during the evolutionary process of CCGP^c.

ing time) is the most important feature for the routing rules in all the scenarios. The importance of MWT is much higher than the other features. In the scenarios whose utilisation levels are 0.85, WIQ (i.e., the workload in the queue) plays a second important role. In the scenarios with a high utilisation level (i.e., 0.95), NIQ (i.e., the number of operations in the queue) plays a significant role. Intuitively, both WIQ and NIQ are important indicators for measuring the workload for machines, they might have the same functionality, and one might take over the other. However, we do not know how they work in different scenarios. It is interesting to see that the role of NIQ is significantly higher than that of WIQ in the scenarios that have higher utilisation level. This indicates that NIQ is an important factor in busy scenarios.

Figure 5.19 shows the curves of the occurrence of terminals in sequencing rules during the evolutionary process of CCGP^c. Different from rout-

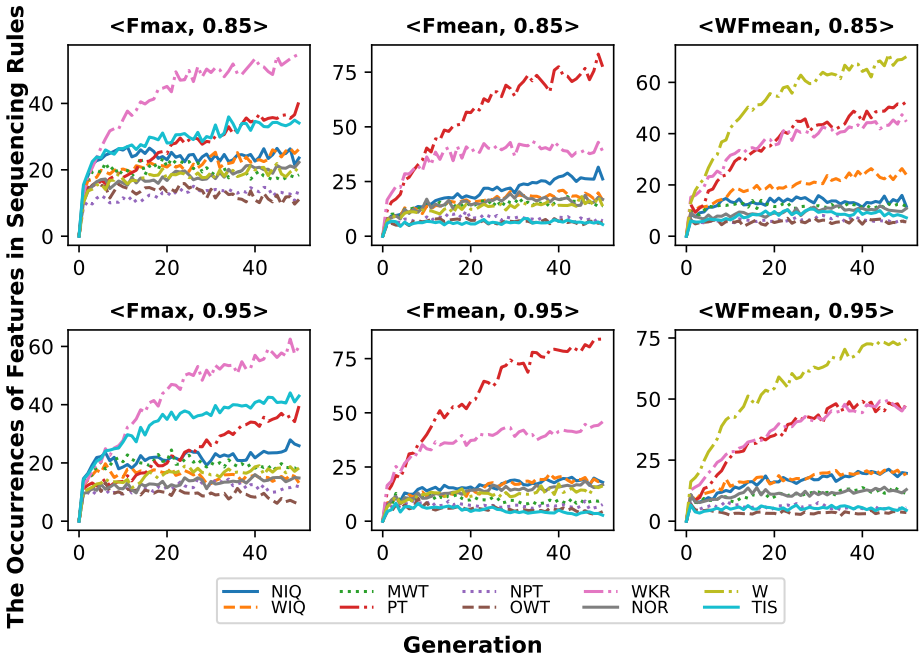

Figure 5.19: The curves of the occurrence of features in **sequencing rules** during the evolutionary process of CCGP^{c}.

ing rules, three terminals (i.e., WKR, TIS, and PT) play a vital role in minimising max-flowtime. PT and WKR are also the two important terminals in minimising mean-flowtime and weighted mean-flowtime. Except for them, W plays a dominant role in minimising the weighted mean-flowtime, which is consistent with our intuition. In addition, W plays its role mainly in sequencing rules instead of routing rules.

5.6 Chapter Summary

The goal of this chapter is to develop an effective recombinative guidance strategy for GPHH to automatically evolve effective scheduling heuristics by improving the quality of produced offspring for DFJSS. To achieve this goal, this chapter firstly proposes to measure the subtree importance with the feature importance information. The proposed algorithm is based on

the idea that the subtrees contain important features are more important for the whole tree. In addition, this chapter proposes an effective way to measure the importance of subtrees of an individual based on the characteristics of DFJSS with the correlation coefficient technique. A properly designed recombinative guidance mechanism is then developed for the crossover operator of GP to generate offspring by replacing the unimportant subtrees of one parent with the important subtrees from the other parent.

The results show that the evolved rules by the proposed algorithm with the correlation coefficient based recombinative guidance have better performance in most scenarios while no worse in all the other scenarios due to its effectiveness for producing offspring. This is also verified by the analyses in terms of the depth ratios, the correlations, and the probability difference of selected subtrees during the evolutionary process. In addition, the proposed algorithm does not need extra computational time compared with its counterparts. This verifies the advantages of utilising the information produced by GP during the evolutionary process and the efficient information calculation techniques such as correlation coefficient.

In Chapters 3-5, we have already studied different techniques (i.e., surrogate, knowledge sharing, phenotypic characteristic) for different chapter goals. However, Chapters 3-5 only solve a single DFJSS task at a time. In the next two chapters, multitask GPHH will be studied to solve multiple DFJSS tasks simultaneously with the investigated techniques in the previous chapters.

Chapter 6

Multitask Genetic Programming Hyper-heuristic

Chapters 3-5 aim to develop effective GPHH methods for solving a **single** DFJSS task. In Chapters 6 and 7, we will focus on how multitask learning can help improve the multiple task solving ability of GPHH with the investigated techniques in the previous chapters for solving **multiple** DFJSS tasks simultaneously. This chapter will focus on adapting the traditional evolutionary multitask algorithm to GPHH for DFJSS, focusing on knowledge sharing.

6.1 Introduction

Evolutionary multitask learning has achieved great success due to its ability to handle multiple tasks simultaneously [90, 91]. Figure 6.1 shows the flowchart of an evolutionary multitask learning algorithm named multifactorial evolutionary algorithm (MFEA) with k tasks [91]. In the beginning, a population of individuals are randomly initialised based on the predefined unified representation. Then, each individual is evaluated on all tasks, and the individual is allocated for its fittest task. As an evolutionary algorithm, the working of MFEA is based on the transmission of

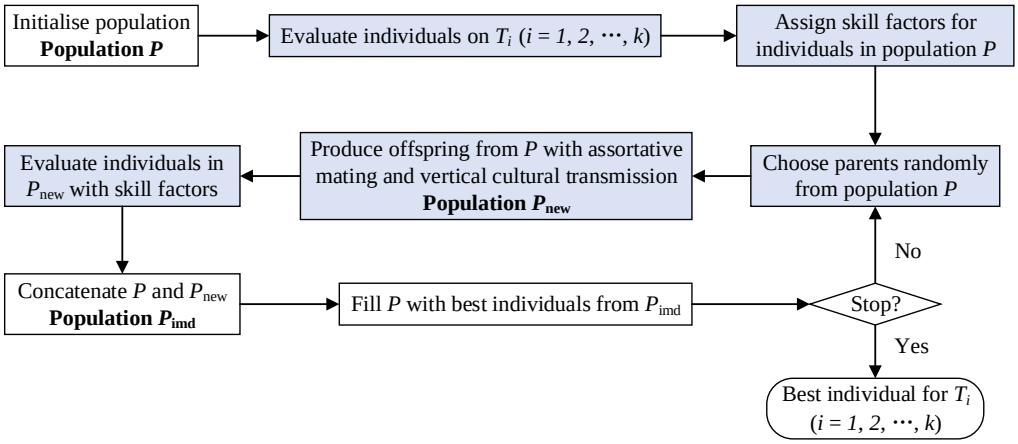

Figure 6.1: The flowchart of MFEA with k tasks, where P, P_{new}, and P_{imd} denote the evaluated, newly generated offspring, and the concatenated population.

cultural genetic materials from parents to their offspring. This is an important step of MFEA that plays a role of transferring knowledge between tasks. In particular, assortative mating and vertical cultural transmission are used with two randomly selected parents to generate offspring. Assortative mating states that individuals prefer to mate with those belonging to the same cultural background. In MFEA, the skill factor is regarded as an individual's cultural bias. The individuals with the same or different skill factor(s) have the same or different culture. Vertical cultural transmission is a mode of inheritance that the phenotype of an offspring is directly affected by the phenotype of its parents. In MFEA, it is realised by allowing offspring to inherit the skill factor of their parents. Finally, the parent population P and offspring population P_{new} are concatenated as an intermediate population P_{imd}, and the top $popsize$ fittest individuals from P_{imd} are kept into the next generation. The output of MFEA is k individuals, each for one task. The unique features of MFEA are highlighted in blue.

However, multitask learning is rarely used in the hyper-heuristic domain to generate heuristics rather than solutions. Multitask selective hyper-

heuristic has been investigated in [96] on exam timetabling and graph colouring problems, however, to the best of our knowledge, there is no study on multitask generative hyper-heuristic.

GP hyper-heuristic (i.e., generative hyper-heuristic) with tree-based representations has been successfully used for evolving scheduling heuristics for complex combinatorial optimisation problems such as dynamic JSS [71, 167, 173, 236, 249] and routing [227]. While, GP is rarely used in multitask learning. Multitask GP was successfully used to the symbolic regression problem in [255]. However, the GP approach in [255] worked on the solution space rather than heuristic space. There are a number of related dynamic tasks in real-world applications such as cloud computing [94, 218] and JSS [166] with a preference for hyper-heuristic approaches. Thus, this chapter presents an attempt to fill the gap of multitask in generative hyper-heuristic by adapting the idea of MFEA to multitask GP hyper-heuristic for evolving scheduling heuristics in DFJSS. The main benefit of multitask GP learning in this chapter is that by handling a number of tasks simultaneously, each task can be solved more effectively with the help from solving the other tasks than being solved independently.

GP and hyper-heuristics have their own features so that directly applying MFEA [91] to GP and hyper-heuristics may not achieve satisfactory effectiveness and efficiency. First, GP uses a different way from the genetic algorithm [50] to control the selection pressure. In GP, the selection pressure is typically implemented by the parent selection for breeding. However, MFEA follows a common genetic algorithm framework that combines the parent and offspring populations, and selects the best individuals from the combined population to the next generation. Using both the parent selection in GP and offspring selection in MFEA simultaneously will make MFGP too greedy and lose the population diversity. Second, in addition to the training performance, hyper-heuristic approaches ultimately aim to achieve good performance on the unseen test instances, which is known as generalisation. To improve generalisation, a commonly

used strategy used in GP hyper-heuristic is to rotate the training instances at each generation [21]. Selecting the best individuals from both the parent population and the offspring population requires to re-evaluate the individuals in the parent population on the same training instances of the offspring. This can make the training process less efficient. Last but not least, MFEA allocates individuals to different tasks by calculating their fitness on all the tasks at the first generation, and assigns them to their the best task. It is time-consuming due to the requirement of extra individual evaluations.

6.1.1 Chapter Goals

The goal of this chapter is to develop an effective multitask GPHH algorithm based on the characteristics of GP. Specifically, this chapter removes the concatenation operation, and proposes to use multiple subpopulations for the tasks. In addition, this chapter develops an effective knowledge sharing mechanism for the tasks. The proposed algorithm is expected to improve the quality of the evolved high-level heuristics for all the tasks considered in the multitask scenarios. Specifically, this chapter has the following research objectives:

1. Propose an effective framework for multitask GP based generative hyper-heuristic according to the characteristics of GP. Specifically, the tasks are solved independently via multiple subpopulations.

2. Develop a novel multitask GPHH algorithm with an origin-based offspring reservation strategy to share knowledge between the tasks via crossover.

3. Verify the effectiveness of the algorithm variations obtained from the adaptation of MFEA to GPHH.

4. Verify the effectiveness of the proposed multitask GPHH algorithm with origin-based offspring reservation strategy on a wide range of

multitask DFJSS scenarios.

5. Analyse the structure and behaviour of the evolved scheduling heuristics and show how the proposed algorithm can solve multiple tasks collaboratively.

6.1.2 Chapter Organisation

The rest of this chapter is organised as follows. Detailed descriptions of the proposed algorithm are given in Section 6.2. The experiment design is shown in Section 6.3, followed by results and discussions in Section 6.4. Finally, Section 6.5 concludes this chapter.

6.2 Proposed Algorithm

6.2.1 Framework of the Proposed Algorithm

This chapter proposes to use multiple subpopulations to solve multiple related tasks and keep knowledge sharing among them. To handle multiple tasks (T_1, T_2, ... , T_k) simultaneously, this chapter groups the GP individuals by equally dividing the entire population into k subpopulations ($Subpop_1$, $Subpop_2$, ... , $Subpop_k$). The individuals in the same (different) subpopulation are used to optimise the same (different) task. On one hand, each subpopulation is independent from each other, and the individuals in different subpopulations are evolved for different tasks. Each subpopulation can be seen as individuals with the same skill factor in MFEA. On the other hand, different subpopulations assist with each other by sharing their knowledge with others, which is realised by crossover.

The main framework of the proposed algorithm is presented in Algorithm 8. The input is the k tasks that are expected to be solved. The output of a GP run consists of k best evolved rules (h_1^*, h_2^*, ... , h_k^*), each for a task. $subpopsize_i$ indicates the number of individuals of subpopulation i.

The fitness of a heuristic h_i is denoted by $fitness_{h_i}$. There are three main differences between the proposed multitask GP hyper-heuristic and the traditional GP for a single task. Specifically, *at the initialisation stage*, the population consists of multiple subpopulations, and each subpopulation is designed to solve one task (line 1). *During the evaluation process*, the individuals in different subpopulations are evaluated independently (from line 6 to line 11). *During the evolution stage*, the crossover operator is applied to share knowledge between the tasks. If the knowledge sharing condition is met, the offspring of each subpopulation are generated according to the proposed knowledge sharing mechanism with the origin-based offspring reservation strategy (from line 25 to line 28). Otherwise, traditional GP crossover will be used to produce two offspring (from line 29 to line 32). It is noted that the proposed knowledge sharing mechanism via crossover is conducted on the individuals from different subpopulations (for different tasks), while the traditional GP crossover works on the individuals from the same subpopulation (for the same task).

6.2.2 Knowledge Sharing

Figure 6.2 shows the evolutionary process of the proposed multitask GP based generative hyper-heuristic with a focus on the knowledge sharing. At generation 0, a population with k subpopulations is initialised for k tasks. The individuals in each subpopulation are assigned to the corresponding task and fixed for that task during the evolutionary process. Specifically, the individuals with white, grey and blue colours in Figure 6.2 are initialised for T_1 (task 1), T_2 (task 2), and T_k (task k), respectively. When generating offspring to the next generation, we use the crossover operator for knowledge transfer but keep the individuals for each task fixed, and the offspring for each subpopulation are produced sequentially. Thus, the best evolved high-level heuristic for each task consists of the genetic materials of individuals that originally belong to that tasks.

Algorithm 8: The Framework of the Proposed Algorithm

Input : k tasks T_1, T_2, ... , T_k
Output: The best evolved heuristics for each task h_1^*, h_2^*, ... , h_k^*

Initialisation: Randomly initialise the population with k subpopulations
set h_1^*, h_2^*, ... , $h_k^* \leftarrow null$
set $fitness_{h_1^*}$, $fitness_{h_2^*}$, ... , $fitness_{h_k^*} \leftarrow +\infty$
$gen \leftarrow 0$
while $gen < maxGen$ **do**
    // **Evaluation**: Evaluate the individuals in the population
    for *i = 1 to* k **do**
        for *j = 1 to* $subpopsize_i$ **do**
            Calculate $fitness_{h_j}$ based on fitness function of T_i
        end
    end
    for *i = 1 to* k **do**
        for *j = 1 to* $subpopsize_i$ **do**
            if $fitness_{h_j} < fitness_{h_i^*}$ **then**
                $h_i^* \leftarrow h_j$
            end
        end
    end
    if $gen < maxGen - 1$ **then**
        // Instance rotation with a new random seed
        // **Evolution**: Generate offspring for each subpopulation
        for *i = 1 to* k **do**
            for *j = 1 to* $subpopsize_i$ **do**
                if *Crossover is applied* **then**
                    if *rand* $\leq$ *rmp* **then**
                        Choose the first parent from $Subpop_i$
                        Choose the second parent from $Subpop_{\neg i}$
                        Produce one offspring with the proposed ***origin-based offspring reservation*** strategy
                    else
                        Choose two parents from $Subpop_i$, and produce two offspring with traditional GP crossover
                    end
                else
                    Choose one parent from $Subpop_i$
                    Do mutation or reproduction accordingly
                end
            end
        end
    end
    $gen \leftarrow gen + 1$
end
return h_1^*, h_2^*, ... , h_k^*

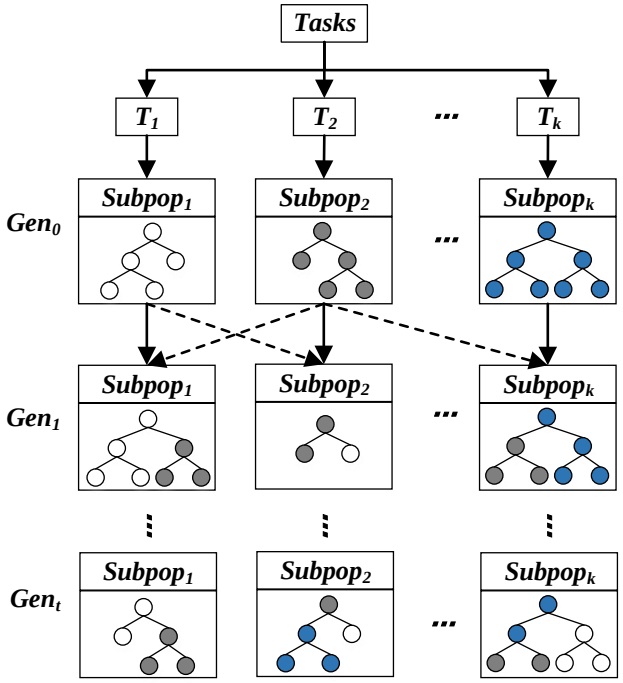

Figure 6.2: The framework of the proposed multitask GP based generative hyper-heuristic with a focus on the knowledge sharing.

This chapter uses a knowledge transfer ratio of rmp to control the frequency to gain knowledge from other subpopulations at each generation. If the knowledge share mechanism is triggered ($rand \leq rmp$), the first parent $parent_1$ will be selected from the current subpopulation, and the other parent $parent_2$ will be selected from other subpopulations. It is noted that if there are more than two subpopulations, when producing offspring for one subpopulation, one of the remaining subpopulations will be selected randomly. Taking $Subpop_1$ as an example, a random subpopulation is chosen to share knowledge with $Subpop_1$. Assume that $Subpop_2$ is selected, the generated offspring with knowledge transfer for $Subpop_1$ consists of white (from $Subpop_2$) and grey (from $Subpop_1$) elements as shown in Figure 6.3 (b), thus, the knowledge transfer among the tasks is realised. We can see that the newly generated offspring contain genetic materials from

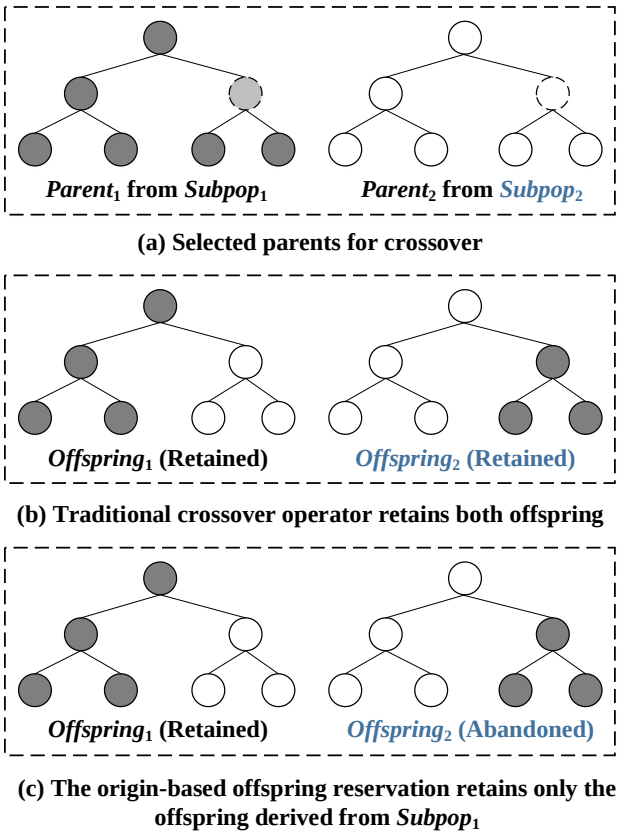

Figure 6.3: An example of generating offspring for $Subpop_1$ by sharing knowledge from $Subpop_2$.

individuals for different tasks. Otherwise ($rand > rmp$), two parents will be selected from the current subpopulation (i.e., the same subpopulation) to produce two offspring for the next generation.

For an individual, an effective knowledge sharing mechanism should not only make the individual obtain useful information but also maintain the original characteristics of the individual. We call this the *origin-based offspring reservation*. When producing offspring for the current task, only the offspring generated based on the parent from the corresponding subpopulation (e.g., $Subpop_1$ in this example) will be kept. Specifically, Figure 6.3 (b) shows the traditional crossover operator that retains both offspring.

It is noted that the knowledge sharing used in Chapter 3 is the same, as shown in Figure 6.3 (b). On the other hand, Figure 6.3 (c) shows the origin-based offspring reservation, which retains only the offspring generated based on the parent from $Subpop_1$.

6.2.3 Algorithm Summary

The proposed algorithm is an extension of the traditional multifactorial evolutionary multitask in [90] by optimising the multitask framework according to the characteristics of GP for developing hyper-heuristic approach. It is noted that the proposed algorithm is not problem-dependent, and can be applied to other domains but with proper construction of related tasks for the specific problems such as evolving routing rules for the arc routing problems. There are a number of improvements compared with the traditional MFEA. First, in terms of the number of evaluations, the proposed algorithm requires fewer evaluations than MFEA. The individuals in the initialised population do not need to be evaluated for all tasks for deciding the skill factor for each individual (i.e., can save $popsize * k$ evaluations). Second, in terms of the individual selection pressure, there is no concatenation operation of the parent and offspring populations, which brings another benefit. If we handle dynamic problems with changed training instance at each generation, we will avoid the extra re-evaluation of the individuals in the parent population (i.e., potentially $popsize * maxGen$ evaluations). Third, in terms of evaluation resource, the number of individuals for each task is fixed and equal, which is easy to manage. This also reduces the parameters in the algorithm, such as skill factor. From the perspective of computational cost, the proposed algorithm can save up to $popsize * (k + maxGen)$ evaluations compared with directly applying MFEA.

In summary, the proposed multitask GP based generative hyper-heuristic approach modifies the evolutionary multitask learning process in [90] to

match the characteristics of GP from the following aspects. First, the proposed algorithm saves the evaluation requirements for assigning individuals to different tasks at the first generation. Second, the proposed algorithm can handle dynamic problems with changed training instances successfully without increasing the number of evaluations, which can promote the application of multitask learning in dynamic problems. Finally, the proposed algorithm can successfully obtain promising offspring without extra evolutionary operations such as concatenating parent and offspring population.

6.3 Experiment Design

6.3.1 Multitask DFJSS Task Definition

Although there are lots of studies related to multitask learning, most of them work on benchmark problems, for which the relatedness between tasks has been widely studied [53]. However, what kinds of DFJSS problems are related and can be optimised in a multitask scenario is not clear. In this section, this chapter defines the related tasks based on the characteristics of DFJSS.

Tasks with the Same Objective but Different Utilisation Levels

In real-world applications, the demand for a specific product varies over time rather than fixed [76]. For example, the amount of orders of T-shirt in summer is likely to be larger than that in winter. A larger amount leads to more complex scheduling. Although the complexities of job shops can be different, they are commonly similar in production, and have the same goal such as minimising the total production time. Thus, we define the tasks with different utilisation levels (i.e., indicate different complexities) but with the same objective to be naturally related tasks for building multitask scenarios.

Tasks with Different Objectives but the Same Utilisation Level

For a scheduling task, different customers may have different requirements [200]. One may require to minimise the flowtime to reduce the total cost. Others may prefer to minimise the tardiness to hand out products to the customers in time. Although the objectives are different, they all involve reducing the idle time of the machines in the shop floor. The knowledge learned from one objective might be also helpful for the others. Therefore, we define the tasks with different objectives but with the same utilisation level as the related tasks considered in multitask scenarios.

For simplicity, we name the multitask problem with the same objective but with the different utilisation levels as *homogeneous multitask*, while the multitask problem with different objectives but with the same utilisation level as *heterogeneous multitask*.

6.3.2 Comparison Design

We consider three homogeneous multitask scenarios, each with a different objective, i.e., mean flowtime, mean tardiness, and mean weighted tardiness. The utilisation levels of 0.75, 0.85, and 0.95 are used in the homogeneous multitask scenarios, since they are three typical distinct configurations in DFJSS [242, 248] with different complexities. With the same objective in each homogeneous multitask scenario, each utilisation level represents a task in the homogeneous multitask [137], which is used to verify the effectiveness of the proposed multitask learning algorithm on the tasks with the same objective but different complexities. The details of the designed homogeneous multitask scenarios represented by optimised objective and utilisation level are shown in Table 6.1.

We also consider three heterogeneous multitask scenarios and choose the most complex scenario with a utilisation level of 0.95 for investigation [167]. For each heterogeneous multitask scenario, two different objectives are involved without varying the utilisation levels between the tasks. In

Table 6.1: The designed **homogeneous** multitask scenarios with tasks represented by optimised objective and utilisation level.

Scenario	task 1	task 2	task 3
Scenario 1	<Fmean, 0.75>	<Fmean, 0.85>	<Fmean, 0.95>
Scenario 2	<Tmean, 0.75>	<Tmean, 0.85>	<Tmean, 0.95>
Scenario 3	<WTmean, 0.75>	<WTmean, 0.85>	<WTmean, 0.95>

Table 6.2: The designed **heterogeneous** multitask scenarios with tasks represented by optimised objective and utilisation level.

Scenario	task 1	task 2
Scenario 1	<Fmax, 0.95>	<Tmax, 0.95>
Scenario 2	<Fmean, 0.95>	<Tmean, 0.95>
Scenario 3	<WFmean, 0.95>	<WTmean, 0.95>

this way, this chapter can focus on verifying the effectiveness of the proposed algorithm on the tasks with the same utilisation level but different objectives. The details of the designed heterogeneous multitask scenarios are shown in Table 6.2.

The GP algorithm with k subpopulations to solve k tasks independently is regarded as the baseline algorithm. The second compared algorithm adapts MFEA [91] to GP without rotating training instances (i.e., no need to do re-evaluation), which is named MFGP. In addition, the algorithm adapts MFEA to GP with *rotating* training instances but without re-evaluation is named $\mathrm{MFGP}^{\mathrm{r}-}$. While the algorithm adapts MFEA to GP with both rotating training instances and re-evaluation is named $\mathrm{MFGP}^{\mathrm{r}+}$. Table 6.3 summaries the characteristics of MFGP, $\mathrm{MFGP}^{\mathrm{r}-}$, and $\mathrm{MFGP}^{\mathrm{r}+}$ according to whether they involves instance rotation or re-evaluation or not. The proposed multitask GP based generative hyper-heuristic approach without the proposed offspring reservation strategy is named $\mathrm{M}^2\mathrm{GP}$, since it involves both multitask and multi-population. $\mathrm{M}^2\mathrm{GP}$ with the proposed

Table 6.3: The availability of instance rotation and re-evaluation of MFGP, $\mathrm{MFGP}^{\mathrm{r}-}$, and $\mathrm{MFGP}^{\mathrm{r}+}$.

	MFGP	$\mathrm{MFGP}^{\mathbf{r-}}$	$\mathrm{MFGP}^{\mathbf{r+}}$
Instance Rotation		$\surd$	$\surd$
Re-evaluation			$\surd$

Table 6.4: The specialised parameter settings of GPHH.

Parameter	Value
*Number of subpopulations	k
*Subpopulation size	400
**Number of tasks	k
**Population size with re-evaluation	200 * k
**Population size without re-evaluation	400 * k
The transfer ratio	0.6

* : for the algorithms with multiple subpopulations only

** : for the algorithms with one population only

offspring reservation strategy named $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$.

To verify the adaptability of MFEA to GP in dynamic scheduling, $\mathrm{MFGP}^{\mathrm{r}-}$, $\mathrm{MFGP}^{\mathrm{r}+}$ and MFGP are compared. To verify the effectiveness of the proposed $\mathrm{M}^2\mathrm{GP}$ and the origin-based offspring reservation strategy, GP, MFGP, $\mathrm{M}^2\mathrm{GP}$, and $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ are compared. In addition, the effectiveness of the proposed $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ on the common tasks between homogeneous and heterogeneous scenarios is further compared with MFGP. The effectiveness of the multitask learning mechanism is also examined by analysing the evolved scheduling heuristics for each task in a multitask scenario.

Table 6.5: The mean (standard deviation) of the objective values on test instances of $\mathrm{MFGP}^{\mathrm{r}-}$, $\mathrm{MFGP}^{\mathrm{r}+}$ and MFGP over 30 independent runs in three homogeneous multitask scenarios.

Scenario	Task	$\mathrm{MFGP}^{\mathrm{r}-}$	$\mathrm{MFGP}^{\mathrm{r}+}$	MFGP
1	<Fmean, 0.75>	339.97(1.11)	337.01(1.39)**(–)**	336.60(1.21)**(–)**(≈)
	<Fmean, 0.85>	396.21(2.90)	387.91(3.72)**(–)**	386.67(2.93)**(–)**(≈)
	<Fmean, 0.95>	586.77(6.50)	560.04(8.64)**(–)**	556.55(5.83)**(–)**(≈)
2	<Tmean, 0.75>	16.08(0.95)	13.90(0.66)**(–)**	13.60(0.25)**(–)**(≈)
	<Tmean, 0.85>	46.32(2.74)	41.51(1.92)**(–)**	40.54(0.66)**(–)**(≈)
	<Tmean, 0.95>	202.54(3.40)	182.88(5.60)**(–)**	180.39(4.46)**(–)**(≈)
3	<WTmean, 0.75>	33.56(2.55)	28.44(1.87)**(–)**	27.26(0.61)**(–)(–)**
	<WTmean, 0.85>	97.12(5.22)	79.90(4.79)**(–)**	76.95(2.19)**(–)(–)**
	<WTmean, 0.95>	381.03(29.32)	310.91(15.34)**(–)**	303.05(8.84)**(–)(–)**

6.3.3 Specialised Parameter Settings of GPHH

The specialised parameter settings are shown in Table 6.4. k subpopulations are used to solve k tasks, and each subpopulation contains 400 individuals. To keep the number of individual evaluations is the same between different algorithms for fair comparison, population size of the algorithm with re-evaluation mechanism is set to 200 * k, and the population size of the algorithm without re-evaluation mechanism is set to 400 * k. The transfer ratio for knowledge transfer is set to 0.6.

6.4 Results and Discussions

6.4.1 Adaptation of MFEA to GPHH

Table 6.5 shows the mean and standard deviation of the objective values of $\mathrm{MFGP}^{\mathrm{r}-}$, $\mathrm{MFGP}^{\mathrm{r}+}$ and MFGP on the unseen instances according to 30 independent runs in the three homogeneous multitask scenarios. Compared

Table 6.6: The mean (standard deviation) of the objective values on test instances of GP, MFGP, $\mathrm{M^2GP}$, and $\mathrm{M^2GP^f}$ over 30 independent runs in three **homogeneous multitasking** scenarios.

Sce.	Task	GP	MFGP	$\mathrm{M^2GP}$	$\mathrm{M^2GP^f}$
1	<Fmean, 0.75>	337.57(1.80)	336.60(1.21)(≈)	335.86(0.91)**(–)(–)**	336.17(1.04)**(–)**(≈)(≈)
	<Fmean, 0.85>	388.79(4.30)	386.67(2.93)(≈)	385.14(1.94)**(–)(–)**	385.73(2.33)**(–)(–)**(≈)
	<Fmean, 0.95>	561.35(9.16)	556.55(5.83)(≈)	553.11(4.26)**(–)(–)**	552.74(4.75)**(–)(–)**(≈)
2	<Tmean, 0.75>	14.08(1.10)	13.60(0.25)(≈)	13.34(0.27)**(–)(–)**	13.33(0.25)**(–)(–)**(≈)
	<Tmean, 0.85>	41.61(2.73)	40.54(0.66)(≈)	39.75(0.87)**(–)(–)**	39.78(0.84)**(–)(–)**(≈)
	<Tmean, 0.95>	182.34(7.72)	180.39(4.46)(≈)	176.84(3.10)**(–)(–)**	176.65(4.17)**(–)(–)**(≈)
3	<WTmean, 0.75>	28.81(2.66)	27.26(0.61)(≈)	27.27(0.99)**(–)(–)**	26.92(0.72)**(–)(–)**(≈)
	<WTmean, 0.85>	81.23(7.63)	76.95(2.19)(≈)	76.43(3.19)**(–)(–)**	75.34(2.06)**(–)(–)(–)**
	<WTmean, 0.95>	312.26(15.86)	303.05(8.84)**(–)**	297.72(10.38)**(–)(–)**	295.67(8.44)**(–)(–)**(≈)

with $\mathrm{MFGP^{r-}}$, the performance of $\mathrm{MFGP^{r+}}$ is significantly better. This indicates that rotating training instances requires the re-evaluation of the individuals in the next generation to achieve accurate fitness. The results also show that MFGP without rotating training instances can achieve similar (better) performance with $\mathrm{MFGP^{r+}}$ in six (three) scenarios, respectively. MFGP will be used for the comparisons in the subsequent experiments, since it performs the best among the adapted algorithms from MFEA.

6.4.2 Quality of the Evolved Scheduling Heuristics

Homogeneous Multitask Scenarios

Table 6.6 shows the mean and standard deviation of the objective values on the test instances of GP, MFGP, $\mathrm{M^2GP}$, and $\mathrm{M^2GP^f}$ over 30 independent runs in three homogeneous multitask scenarios. MFGP does not show any significant difference from GP in most of the scenarios, which indicates that directly applying the idea of MFEA is not effective in the context of

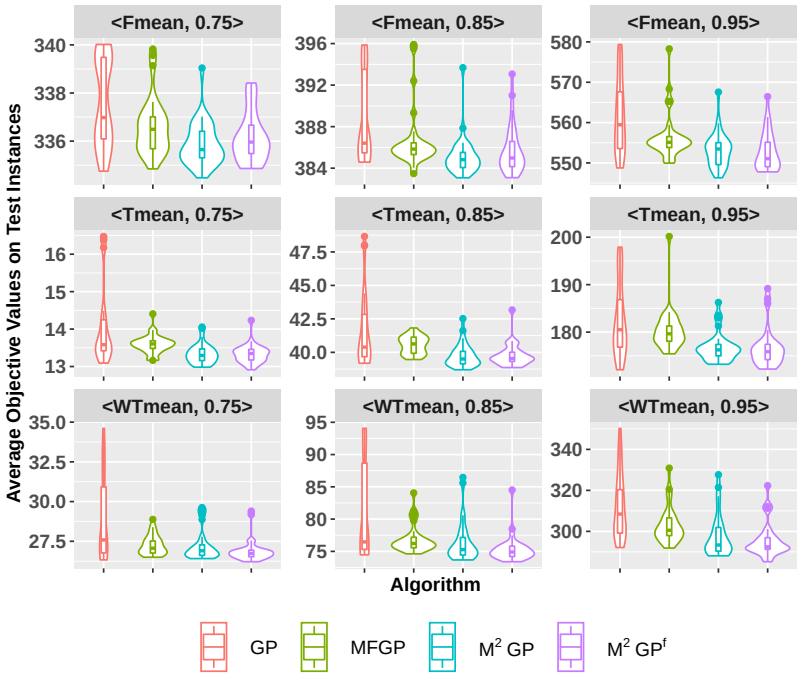

Figure 6.4: The violin plot of the average objective values on test instances of GP, MFGP, and $\mathrm{M}^2\mathrm{GP}$ based on 30 independent runs in three *homogeneous multitask* scenarios (each row is a multitask scenario).

GP hyper-heuristic. The characteristics of GP are quite different from the genetic algorithm that is used in MFEA. $\mathrm{M}^2\mathrm{GP}$ performs significantly better than both GP and MFGP in all the examined scenarios. This verifies the effectiveness of the proposed $\mathrm{M}^2\mathrm{GP}$ in the homogeneous multitask scenarios. $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ also shows it superiority compared with GP and MFGP. However, $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ only significantly better than $\mathrm{M}^2\mathrm{GP}$ in the one scenario (i.e., <WTmean, 0.85>).

Figure 6.4 shows the violin plot of the average objective values on the test instances based on 30 independent runs of GP, MFGP, $\mathrm{M}^2\mathrm{GP}$, and $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ in the three homogeneous multitask scenarios. According to the distribution of the 30 objective values, MFGP can achieve smaller values than that of GP but not significantly better. We can see that $\mathrm{M}^2\mathrm{GP}$ shows its

Table 6.7: The mean (standard deviation) of the objective values on test instances of GP, MFGP, $\mathrm{M^2GP}$, and $\mathrm{M^2GP^f}$ over 30 independent runs in three **heterogeneous multitask** scenarios.

Sce.	Task	GP	MFGP	$\mathrm{M^2GP}$	$\mathrm{M^2GP^f}$
1	<Fmax, 0.95>	2032.96(98.29)	2081.77(76.40)**(+)**	1991.15(88.56)**(–)(–)**	1981.28(37.19)**(–)(–)**(≈)
	<Tmax, 0.95>	1580.81(54.13)	1647.75(55.45)**(+)**	1576.03(54.94)(≈)**(–)**	1575.13(37.84)(≈)**(–)**(≈)
2	<Fmean, 0.95>	560.72(10.18)	556.10(6.10)(≈)	556.52(9.11)(≈)(≈)	553.79(7.43)**(–)(–)**(≈)
	<Tmean, 0.95>	180.81(6.83)	178.76(3.47)(≈)	180.20(6.51)(≈)(≈)	177.55(5.72)**(–)(–)(–)**
3	<WFmean, 0.95>	1136.33(25.65)	1121.67(12.74)**(–)**	1123.87(20.45)**(–)**(≈)	1121.05(22.20)**(–)(–)**(≈)
	<WTmean, 0.95>	311.20(16.82)	301.12(8.07)**(–)**	303.06(15.23)**(–)**(≈)	300.00(15.38)**(–)(–)**(≈)

superiority with smaller objective values than GP and MFGP. This shows that the tasks with the same objective but different utilisation levels in DFJSS can be solved simultaneously in a mutually reinforcing way. In addition, the objective values of $\mathrm{M^2GP^f}$ tend to be smaller than $\mathrm{M^2GP}$ in most of the scenarios (i.e., <Fmean, 0.95>, <Tmean, 0.75>, <Tmean, 0.95>, <WTmean, 0.75>, <WTmean, 0.85>, and <WTmean, 0.95>). This indicates that the proposed origin-based offspring strategy can benefit the algorithm $\mathrm{M^2GP}$.

Heterogeneous Multitask Scenarios

Table 6.7 shows the mean and standard deviation of the objective values on the test instances of GP, MFGP, $\mathrm{M^2GP}$ and $\mathrm{M^2GP^f}$ based on 30 independent runs in the three heterogeneous multitask scenarios. Different from the observations in the homogeneous multitask scenarios, MFGP and $\mathrm{M^2GP}$ do not outperform GP, since they are not significantly better than GP in most scenarios. While, $\mathrm{M^2GP^f}$ achieves significantly better results than GP and MFGP in most of the scenarios, and it outperforms $\mathrm{M^2GP}$ in one scenario (i.e., <Tmean, 0.95>). One possible reason is that the tasks with different objectives in the heterogeneous multitask scenarios are less

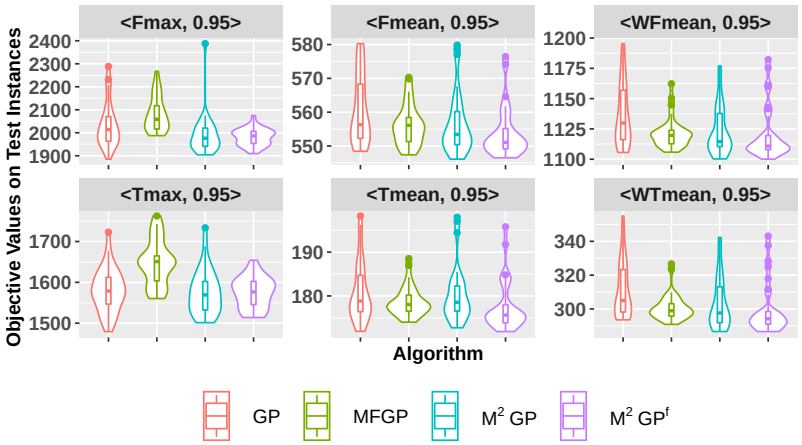

Figure 6.5: The violin plot of the average objective values on test instances of GP, MFGP, $\mathrm{M}^2\mathrm{GP}$, and $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ based on 30 independent runs in three **heterogeneous multitask** scenarios (each column is a multitask scenario).

related, and GP, MFGP, and $\mathrm{M}^2\mathrm{GP}$ cannot maintain the quality of individuals for each task. The proposed origin-based offspring reservation strategy can help GPHH maintain the quality of individuals for each task, since it aims to keep the main characteristics of the individuals for the corresponding tasks.

Figure 6.5 shows the violin plot of the average objective values of GP, MFGP, $\mathrm{M}^2\mathrm{GP}$, and $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ on the unseen instances based on 30 independent runs in the three heterogeneous multitask scenarios. It is obvious that the objective values achieved by $\mathrm{M}^2\mathrm{GP}$ tend to be much smaller than that of GP and MFGP. This indicates that the tasks with different objectives but the same utilisation level in DFJSS can also be solved simultaneously in a mutually reinforcing way. In addition, we can see that the objective values obtained by $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ tend to be smaller than $\mathrm{M}^2\mathrm{GP}$, which confirms the positive effect of the proposed origin-based offspring reservation strategy.

Homogeneous Versus Heterogeneous Multitask Scenarios

There are three common tasks (i.e., <Fmean, 0.95>, <Tmean, 0.95>, and <WTmean, 0.95>) solved in both the homogeneous and heterogeneous multitask scenarios. It is interesting to know the quality of the evolved scheduling heuristics obtained for the same task in different types of multitask scenarios.

Figure 6.6 shows the violin plot of the average objective values of MFGP and $\mathrm{M^2GP^f}$ on the unseen data for the common tasks (i.e., <Fmean, 0.95>, <Tmean, 0.95>, and <WTmean, 0.95>) between the homogeneous and heterogeneous multitask scenarios. Overall, for all the scenarios, $\mathrm{M^2GP^f}$ performs better than MFGP in both homogeneous and heterogeneous multitask. Based on the distributions of the achieved objective values, both MFGP and $\mathrm{M^2GP^f}$ show its superiority in heterogeneous multitask scenarios rather than the homogeneous multitask scenarios for most of the common tasks. The objective values obtained from heterogeneous multitask are distributed in a relatively lower position.

In summary, $\mathrm{M^2GP^f}$ can achieve effective scheduling heuristics for both the homogeneous and heterogeneous multitask scenarios. In addition, we find that learning in a heterogeneous multitask scenario has more potential to improve the quality of scheduling heuristics.

6.4.3 Evolved High-level Scheduling Heuristics

This section chooses the evolved scheduling heuristics, including the routing and sequencing rules for the tasks in a heterogeneous multitask scenario to investigate how the tasks help with each other from the perspective of the genotypes of individuals. The second heterogeneous multitask scenario is selected, since it contains two out of three common tasks (i.e., <Fmean, 0.95>, <Tmean, 0.95>, and <WTmean, 0.95>) with homogeneous multitask scenarios. The routing rules are the best evolved rules for task 1 and task 2 from the same run in heterogeneous multitask scenario

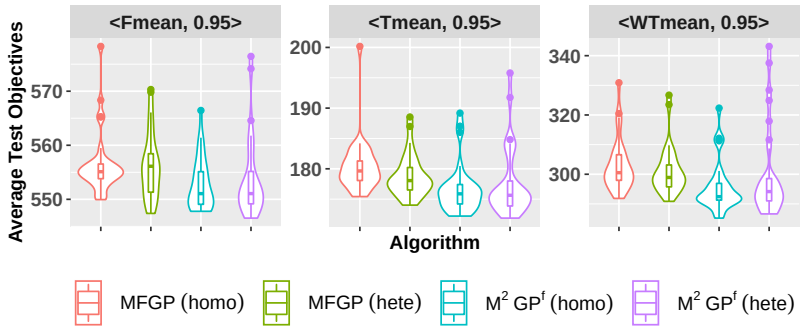

Figure 6.6: The violin plot of the average objective values on test instances of MFGP and $\mathrm{M}^2\mathrm{GP}^{\mathrm{f}}$ in both homogeneous (denoted as homo) and heterogeneous (denoted as hete) multitask scenarios for their common tasks based on 30 independent runs.

2, and the sequencing rules are the corresponding sequencing rules of the routing rules mentioned above. It is noted that a machine or an operation with a smaller priority value is more prior.

Routing Rules

Figure 6.7 and Figure 6.8 show one of the evolved routing rules for the tasks <Fmean, 0.95> and <Tmean, 0.95> in the second heterogeneous multitask scenario, respectively. It is obvious that these two scheduling heuristics share knowledge with each other, since the major part of the rules is the same which is highlighted in grey.

We further investigate the behaviour of the routing rule for minimising mean-tardiness, as shown in Figure 6.8. It can be further simplified, as

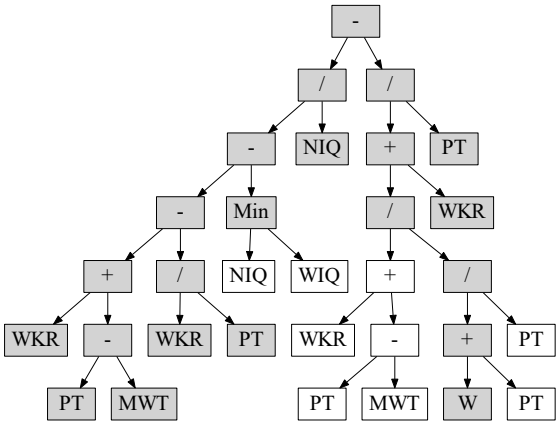

Figure 6.7: One of the best evolved routing rules for task 1 <Fmean, 0.95> in heterogeneous multitask scenario 2.

shown in Eq. 6.1.

$$
\begin{aligned}
R =& \{WKR + PT - MWT - \frac{WKR}{PT} - Min\{ \\
& \frac{Max\{WKR, MWT\}}{PT}, \frac{WKR}{PT} - NOR\}\}/NIQ \\
& - \frac{Min\{WKR, TIS\}}{PT(NPT + W)} - \frac{WKR}{PT} \\
\approx& \{PT - MWT - \frac{WKR}{PT} - Min\{ \\
& \frac{Max\{WKR, MWT\}}{PT}, \frac{WKR}{PT} - NOR\}\}/NIQ \\
& - \frac{Min\{WKR, TIS\}}{PT(NPT + W)} - \frac{WKR}{PT}
\end{aligned}
\tag{6.1}
$$

WKR and PT are the two most commonly used low-level heuristics for this routing rule based on the occurrences of heuristics. However, the value of WKR (i.e., the remaining work of the corresponding job of an operation for all the machines) is the same. This indicates that WKR is not an important factor for this rule to distinguish the machines, and it can be considered as a constant. Similarly, W (i.e., the importance of a job), NOR (i.e., the

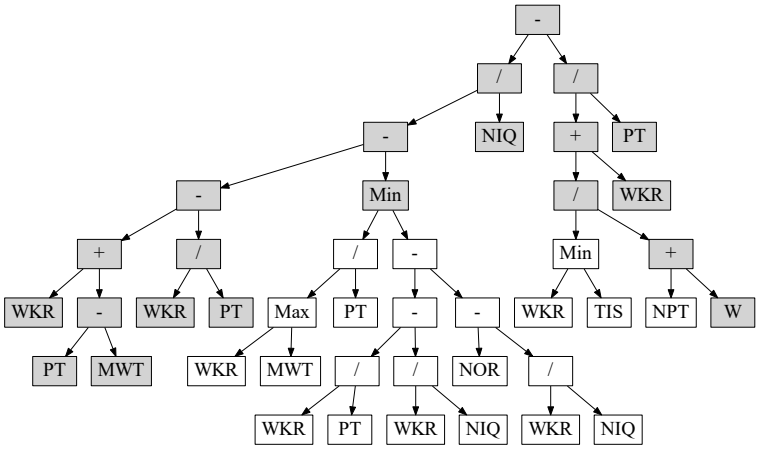

Figure 6.8: One of the best evolved routing rules for task 2 <Tmean, 0.95> in heterogeneous multitask scenario 2.

number of remaining operations of a job), NPT (i.e., the median processing time of the next operation), and TIS (i.e., the time that an operation in the job shop floor) are also considered as constants here. Without considering the constant features (i.e., WKR, W and NOR) where it is possible, this rule can be further simplified as shown in step 2 of Eq. 6.1.

This rule tends to choose the machine with smaller processing time (i.e., efficient machine for an operation) and a longer waiting time (i.e., the time when the machine is idle). It is consistent with our intuition, since we would like to use efficient machines to reduce the producing time for products. In addition, this rule prefers to allocate an operation to a machine with a larger number of operations (NIQ). It is noted that a larger NIQ does not mean an overhead workload for a machine, since the processing time of the operations can be small.

Sequencing Rules

Figure 6.9 and Figure 6.10 show the corresponding sequencing rules of the routing rules, as shown in Figure 6.7 and Figure 6.8, respectively. The main structure of these two sequencing rules for different tasks in a mul-

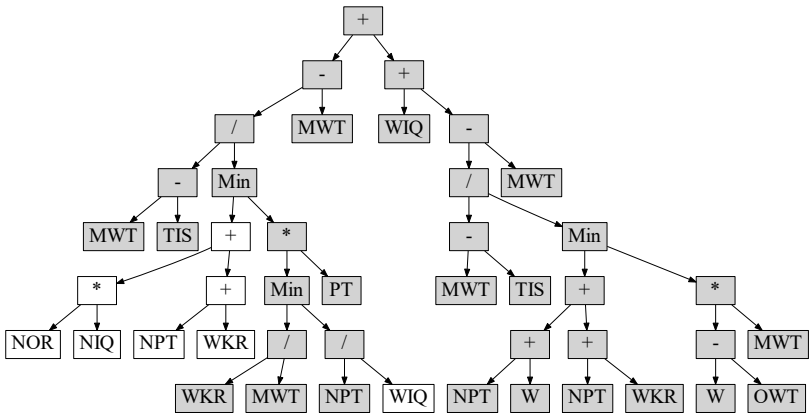

Figure 6.9: One of the best evolved sequencing rules for task 1 <Fmean, 0.95> in heterogeneous multitask scenario 2.

titask scenario is the same as shown in grey, and only two smaller parts of them are different. This means that these two sequencing rules strongly share their knowledge in the evolutionary process. In the grey area, MWT and NPT are the two most important terminals based on the frequency of terminals. This indicates that MWT and NPT are both important operation sequencing in minimising mean-flowtime and mean-tardiness.

The corresponding sequencing rule (Figure 6.10) of the routing rule in Figure 6.8 is shown in Eq. 6.2. From step 1 to step 2, "Min{W, PT * Min{$\frac{WKR}{MWT}$, $\frac{NPT}{WKR}$}}" can be further simplified to W, since W (i.e., 1, 2 and 4) is more likely to be smaller than "PT * Min{$\frac{WKR}{MWT}$, $\frac{NPT}{WKR}$}". "(W - OWT) * MWT" is more likely to be a negative number, because W is usually smaller than OWT. Therefore, "Min{2NPT + W + WKR, (W - OWT) * MWT}" can be further simplified as "(W - OWT) * MWT". Finally, since the value of W is much smaller than OWT, W is removed, as shown in the

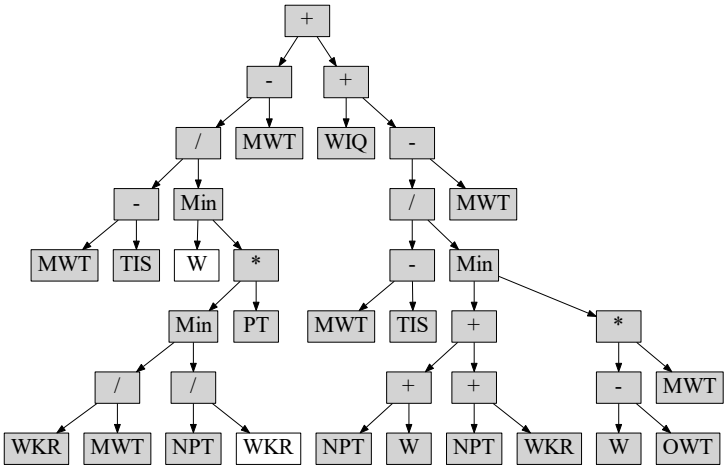

Figure 6.10: One of the best evolved sequencing rules for task 2 <Tmean, 0.95> in heterogeneous multitask scenario 2.

last step of Eq. 6.2.

$$
\begin{aligned}
S =& \frac{MWT - TIS}{Min\{W, PT * Min\{\frac{WKR}{MWT}, \frac{NPT}{WKR}\}\}} - \\
& \frac{MWT - TIS}{Min\{2NPT + W + WKR, (W - OWT) * MWT\}} \\
& - 2MWT + WIQ \\
\approx& \frac{MWT - TIS}{W} + \frac{MWT - TIS}{(OWT - W) * MWT} \\
\approx& \frac{MWT - TIS}{W} + \frac{MWT - TIS}{OWT * MWT}
\end{aligned}
\tag{6.2}
$$

For the operations in the queue of a machine, MWT (i.e., machine waiting time) is the same for all the operations, and can be considered as a constant. This sequencing rule suggests to choose the operation that has stayed in the job shop floor for a long time (i.e., large TIS), and has waited in the queue of a machine for a long time (i.e., large OWT). It is consistent with our intuition that a long waiting time will delay the production, and it is conducive to minimise the tardiness. In addition, the machine prefers to choose an important operation with a large W. This is also con-

sistent with our intuition that important jobs should be processed earlier to reduce the delay and improve customer satisfaction.

In summary, both the routing and sequencing rules for the tasks in heterogeneous multitask scenarios share lots of knowledge with each other. We observe the same pattern in the homogeneous multitask scenarios. We can conclude that the proposed algorithm can solve the tasks in a mutually reinforcing way.

6.5 Chapter Summary

The goal of this chapter is to develop an effective multitask GP hyper-heuristic algorithm to solve multiple DFJSS problems simultaneously. To achieve this purpose, this chapter proposes a multi-population based GP framework for solving multiple related DFJSS tasks simultaneously. This chapter also develops an effective knowledge sharing mechanism with origin-based offspring reservation strategy for sharing knowledge between the tasks. The effectiveness of the proposed algorithm is examined on both homogeneous and heterogeneous multitask DFJSS scenarios.

The results show that the proposed $\mathrm{M^2GP^f}$ can achieve effective scheduling heuristics in all homogeneous and heterogeneous multitask DFJSS scenarios. In addition, $\mathrm{M^2GP^f}$ is robust in terms of the performance in both homogeneous and heterogeneous multitask scenarios. We also found that the task with a heterogeneous multitask scenario has more potential to be optimised well. The effectiveness of the proposed multitask GP hyper-heuristic was examined by not only comparing the quality of evolved scheduling heuristics, but also the structures and behaviours of the evolved scheduling heuristics for all tasks in a multitask scenario. It has also been observed that the proposed algorithm manages to solve the tasks in a mutually reinforcing way.

This chapter proposes a novel multitask GPHH algorithm by adapting the traditional evolutionary multitask learning algorithm with an effective

knowledge sharing strategy based on the characteristics of GP. It is the first time to use multitask GPHH for solving multiple DFJSS tasks simultaneously. It expands the paradigm of evolutionary multitask to GPHH for DFJSS (i.e., complex dynamic combinatorial optimisation problems). Chapter 6 is a start point for using multitask GPHH for DFJSS, and provides a basic framework for studying multitask GPHH for DFJSS. In the next chapter, we will further enhance the effectiveness of multitask GPHH for DFJSS.

Chapter 7

Surrogate-Assisted Multitask Genetic Programming

In Chapter 6, we propose a novel multitask GPHH algorithm with an effective knowledge sharing strategy for DFJSS by adapting the traditional evolutionary multitask algorithm to GP. This chapter will focus on using the investigated techniques in Chapters 3-6 (i.e., surrogate, knowledge sharing, and phenotypic characterisation) to further enhance the effective of multitask GPHH in Chapter 6 for DFJSS. Specifically, the surrogate technique will be used to improve the efficiency of individual evaluation and share knowledge between tasks.

7.1 Introduction

Although surrogate and multitask techniques can improve the efficiency and effectiveness of evolutionary optimisation in different aspects, they are often used independently. Based on the studies of multitask GPHH for DFJSS in Chapter 6, this chapter aims at further introducing the surrogate to assist multitask GPHH. To the best of our knowledge, there are only a few studies on surrogate-assisted multitask learning [62, 104, 146, 157]. In [146], Gaussian Process was introduced to build a surrogate model for the

designed global search component of the algorithm. In [104], surrogates are applied to reduce the fitness evaluations for each task on benchmark problems. In [62, 157], computationally cheap models are applied to handle expensive optimisation problems by sharing its acquired knowledge. Although these studies have shown good performance for multitask learning, the surrogate is only used to improve the effectiveness for each single task independently rather than multiple tasks simultaneously. In other words, these studies are not about using surrogate for the core mechanism of multitask such as knowledge transfer between the tasks.

Moreover, it is not easy to apply existing approaches [62, 104, 146, 157] in DFJSS with multitask GPHH directly. First, allocating individuals for different tasks by evaluating individuals on all the tasks is computationally expensive for DFJSS. Second, existing surrogate and multitask learning approaches are mostly applied to benchmarks with continuous, numeric optimisation problems [255] rather than discrete, combinatorial optimisation problems, thus, they are not directly applicable for DFJSS.

7.1.1 Chapter Goals

The goal of this chapter is to *develop an effective surrogate-assisted multitask GPHH algorithm to evolve scheduling heuristics for multiple DFJSS tasks simultaneously.* The built surrogates are used for improving not only the effectiveness of solving a single task but also the effectiveness of knowledge sharing between the tasks. The proposed algorithm is expected to improve the quality of scheduling heuristics of multitask GPHH for DFJSS. Specifically, this chapter has the following research objectives:

1. Build surrogates for different tasks accordingly by taking the behaviour of individuals and their real fitness into account.

2. Propose a new knowledge mechanism by allocating the newly generated individuals to appropriate tasks incorporated with the surrogate models for multitask learning.

3. Analyse the effectiveness of the proposed algorithm in terms of the quality of the evolved scheduling heuristics and the constructed surrogate.

4. Analyses the diversity of individuals for tasks, and individual allocation for tasks of the proposed algorithm.

5. Analyse the size, structure, and schematics of the evolved scheduling heuristics obtained by the proposed algorithm.

7.1.2 Chapter Organisation

The rest of this chapter is organised as follows. Detailed descriptions of the proposed algorithm are given in Section 7.2. The experiment design is shown in Section 7.3, followed by results and discussions in Section 7.4. Further analyses are conducted in Section 7.5. Finally, Section 7.6 concludes this chapter.

7.2 Proposed Algorithm

7.2.1 Framework of the Proposed Algorithm

The framework of the proposed algorithm is presented in Algorithm 9. The inputs are the k tasks to be solved. The output is a set of best evolved scheduling heuristics obtained from the subpopulations, one for each task. *At the initialisation stage*, the population is formed with k subpopulations to solve the k tasks (line 1). *During the evaluation process*, the individuals in different subpopulations are evaluated with different training instances according to the tasks (from line 8 to line 18). In addition, the surrogates are built with the phenotypic characterisations of the individuals in each subpopulation and their real fitness (from line 19 to line 22). It is noted that there are k surrogates generated at each generation, one for each task. The

surrogates are updated at the next generation. *During the evolution stage*, $n * subpopsize$ number of offspring are generated for each subpopulation to build an offspring pool (from line 24 to line 26). The duplicated individuals are then removed from the offspring pool according to their phenotypic characterisations (line 27). To obtain the final offspring $newInds$ for $Subpop_i$ for task T_i, the fitness of all the individuals in the offspring pool are estimated by the surrogate S_i (line 30). Then, the individuals in the offspring pool are ranked according to the fitness estimated by S_i, and the top $subpopsize$ individuals are selected as the final offspring population for $Subpop_i$ (from line 28 to line 33). With surrogates, the knowledge carried by individuals can be evaluated more efficiently. The surrogates can also help with knowledge transfer between tasks by allocating the newly generated individuals from different subpopulations to proper tasks.

7.2.2 Surrogate Model

The main idea in this chapter is to use surrogate to assist the multitask learning, and the surrogate design itself is not the focus in this chapter. KNN with phenotypic characterisation has been successfully proposed in [99] for dynamic JSS with GP. Since KNN is efficient and straightforward, it is chosen as the surrogate to estimate the fitness of individuals by finding the most similar individual based on the phenotypic characterisations of individuals with Euclidean distance as suggested in [55]. The Euclidean distance is a reasonable technique to measure the similarity between the behaviour of individuals, since the phenotypic characterisation (i.e., the details can be found in Section 4.2.4) is a vector of ranking numbers that indicate the decisions made by a rule. The individuals that have been evaluated using the real fitness evaluation in the previous generation are used to build the KNN surrogate, and the surrogate is updated at each generation.

Algorithm 9: Framework of the Proposed Algorithm

Input : Tasks T_1, T_2, ... , T_k
Output: The best evolved heuristics for each task h_1^*, h_2^*, ... , h_k^*

Initialisation: Randomly initialise the population with k subpopulations
set h_1^*, h_2^*, ... , $h_k^* \leftarrow null$
set $fitness(h_1^*)$, $fitness(h_2^*)$, ... , $fitness(h_k^*) \leftarrow +\infty$
$gen \leftarrow 0$
while $gen < maxGen$ **do**
    set $S \leftarrow null$
    set $newPop \leftarrow null$
    // **Evaluation**: Evaluate the individuals in the population
    for *i = 1 to* k **do**
        for *j = 1 to* $subpopsize$ **do**
            Run a DFJSS simulation with h_j according to task T_i to get the schedule $Schedule_j$
            $fitness_{h_j} \leftarrow Obj(Schedule_j)$
        end
        for *j = 1 to* $subpopsize$ **do**
            if $fitness_{h_j} < fitness_{h_i^*}$ **then**
                $h_i^* \leftarrow h_j$
            end
        end
        Calculate the phynotypic characterisation vector for each individual in $Subpop_i$
        Build surrogate model S_i with the phenotypic characterisations and the corresponding fitness of indivudlas in $Subpop_i$
        $S \leftarrow S \cup S_i$
    end
    if $gen < maxGen - 1$ **then**
        // **Evolution**: generate new population
        Generate $n * subpopsize$ offspring for each subpopulation by genetic operators, respectively.
        Offspring Pool: Put the offspring of all subpopulations together
        Clearing the individuals in the offspring pool
        // **Assign individuals to tasks**
        for *i = 1 to* k **do**
            Estimate the fitness of individuals in the offspring pool using the surrogate model S_i built for $Subpop_i$
            $newInds$: Choose the top $subpopsize$ individuals from offspring pool based on the estimated fitness
            $newPop \leftarrow newPop \cup newInds$
        end
    end
    $gen \leftarrow gen + 1$
end
return h_1^*, h_2^*, ... , h_k^*

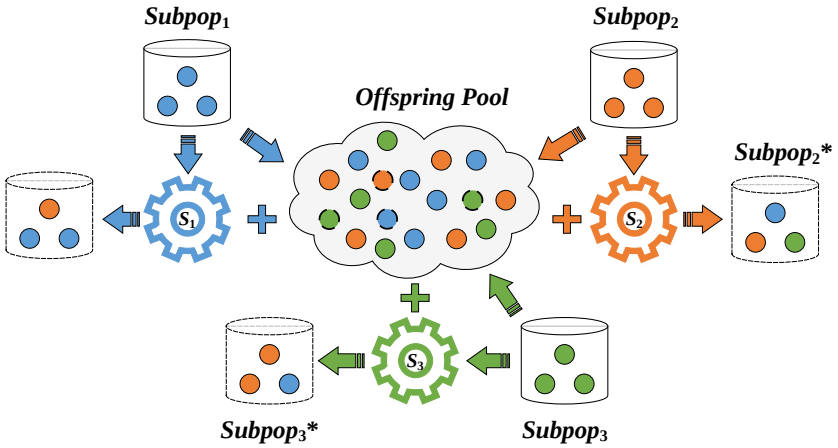

Figure 7.1: An example of the proposed surrogate-assisted multitask GPHH with three tasks in terms of the surrogate and knowledge sharing mechanism.

Regarding the way of using the surrogate model, previous studies [99, 174, 176] show that pre-selection is an effective approach. Specifically, an intermediate population with a large number of offspring is generated at each generation, then the surrogate is used to estimate the fitness of all the offspring efficiently. Only the individuals with good surrogate fitness are pre-selected to the next generation, and are re-evaluated with the real fitness evaluation. This chapter uses the surrogate in a pre-selection manner at each generation but in a different way. Specifically, the surrogate is used to not only improve the effectiveness of solving a single task but also sharing knowledge between different tasks.

7.2.3 Knowledge Sharing with Surrogate

To find suitable individuals for knowledge transfer, we employ the KNN-based surrogate mechanism with phenotypic characterisation to help transfer knowledge between different tasks due to its effectiveness in distinguishing the behaviour of individuals. Figure 7.1 illustrates the knowledge transfer process in a multitask scenario with three tasks via surro-

gate. In this example, there are three subpopulations, and each subpopulation consists of three individuals. Each subpopulation is designed to solve a single task. The individuals in $Subpop_1$, $Subpop_2$, and $Subpop_3$ are marked in solid blue, orange, and green circles, respectively. If there is no knowledge transfer between them, the three subpopulations can be considered as three independent evolutionary processes. For multitask learning, the main consideration is to design the mechanism for transferring knowledge between these three subpopulations. First, the phenotypic characterisations and the fitness of individuals in the subpopulations are used to build the corresponding KNN surrogate model for each subpopulation (i.e., S_1, S_2, S_3), respectively. Second, to get more potentially useful knowledge, $n * subpopsize$ (where n offspring is two in this example) are generated based on each subpopulation independently and put into the offspring pool. The offspring pool is composed of the offspring from $Subpop_1$, $Subpop_2$ and $Subpop_3$. The individuals in the offspring pool are then cleared by removing the duplicated individuals according to their phenotypic characterisations. The clearing procedure for offspring pool is shown in Algorithm 10. In Figure 7.1, the removed individuals in the offspring pool are marked in dotted circles. Finally, the individuals in the offspring pool are evaluated based on S_1 (S_2 or S_3 respectively), and the best $subpopsize$ individuals according to fitness are selected as the final offspring for $Subpop_1$ ($Subpop_2$ or $Subpop_3$ respectively). The newly generated subpopulations are shown as ${Subpop_1}^*$, ${Subpop_2}^*$, and ${Subpop_3}^*$. It is noted that an offspring in the offspring pool can be allocated to multiple subpopulations, since an offspring is possible to work well on multiple tasks.

From the perspective of transferred materials, the knowledge is transferred with a *FullTree* strategy via the surrogate-assisted multitask mechanism. However, in fact, the knowledge is also shared in a *SubTree* manner via the crossover operator in each subpopulation, since the chosen parents in each subpopulation can come from different subpopulations. As a re-

Algorithm 10: Pseudo-code of clearing procedure of offspring pool

Input : All the individuals in the offspring pool Ind
Output: Cleared individuals with different behaviour $NonRepInd$

```
set PC ← null
distance ← ∞
for i = 1 to |Ind| do
    PC_i: Calculate the phenotypic characteristic of Ind_i
    PC ← PC ∪ PC_i
end
for i = 1 to |PC| do
    for j = i +1 to |PC| do
        distance: Calculate the distance between PC_i and PC_j
        if distance == 0 then
            remove Ind_j from Ind
            distance ← ∞
        end
    end
end
NonRepInd ← Ind
return NonRepInd
```

sult, the proposed approach can take advantage of transferring knowledge in terms of both the *FullTree* and *SubTree*.

7.2.4 Algorithm Summary

The proposed algorithm combines the advantages of the surrogate and multitask learning to GPHH for DFJSS. It improves the effectiveness of not only solving a single task but also sharing knowledge between tasks in a multitask scenario. From the perspective of solving each single task, the surrogate is used in a pre-selection way by estimating the fitness of a large number of individuals in the offspring pool. From the perspective of solving multiple related tasks simultaneously, the surrogate plays a role to share knowledge between different tasks by reallocating the generated

individuals to different tasks. The surrogate technique makes it possible to efficiently examine more useful materials carried by the individuals in the offspring pool, while multitask with the surrogate can make a better decision of individual allocations for the tasks. This chapter provides a good case study to illustrate the effectiveness of the surrogate-assisted evolutionary multitask algorithm for solving dynamic discrete and combinatorial optimisation problems.

7.3 Experiment Design

7.3.1 Comparison Design

The homogeneous multitask scenarios designed in Section 6.3.2 are used to examine the proposed algorithm in this chapter. The details of the designed multitask scenarios are shown in Table 6.1. Four algorithms are taken into comparison. The GP with k subpopulations to solve k tasks independently named MTGP (GP with multi-tree representation) [244], is selected as the baseline algorithm, since there is no surrogate or multitask mechanism involved. The multitask algorithm proposed in chapter 6 [239] named $\mathrm{M^2GP^f}$, is also taken into consideration. It is the state-of-the-art algorithm that introduces multitask learning in DFJSS. The algorithm in [99] that uses KNN to build surrogate only, named SMTGP in this chapter, is compared by applying it to each single task separately. The proposed surrogate-assisted multitask algorithm in this chapter is named $\mathrm{SMT^2GP}$, since it involves surrogate, multitask, and multi-tree GP.

To verify the effectiveness of the proposed $\mathrm{SMT^2GP}$, the approaches of MTGP, $\mathrm{M^2GP^f}$ and SMTGP are compared with $\mathrm{SMT^2GP}$ in terms of the test objective values on the unseen instances. The effectiveness of the constructed surrogates in DFJSS is illustrated by the comparison between MTGP and SMTGP. The effect of the proposed multitask mechanism of $\mathrm{SMT^2GP}$ is examined by the comparison between SMTGP and $\mathrm{SMT^2GP}$.

Table 7.1: The specialised parameter settings of GPHH.

Parameter	Value
Number of subpopulations	3
Subpopulation size	400
The number of offspring for each task	400*12
Intermediate population size of SMTGP / $\mathrm{SMT^2GP}$	400*12*3 / 400*4*3

7.3.2 Specialised Parameter Settings of GPHH

Table 7.1 shows the specialised parameter settings of GPHH in this chapter. Three subpopulations with a size of 400 individuals are used to solve the three tasks simultaneously. It is noted that increasing the number of individuals in the intermediate population can improve the performance of SMTGP. However, the improvement for SMTGP is marginal, especially when the number of individuals increases to a certain number. This is consistent with the finding drawn in [99]. Our preliminary experiments show that 400*12 is a proper intermediate subpopulation size for SMTGP, and no further improvement can be found if the size is further increased. Accordingly, the number of generated offspring from each subpopulation is set to 400*4 for $\mathrm{SMT^2GP}$ in order to keep the same number of evaluations with a surrogate (400*4*3 = 400*12) as SMTGP for each task.

7.4 Results and Discussions

7.4.1 Quality of the Evolved Scheduling Heuristics

Figure 7.2 shows the violin plot of the average objective values on the test instances of MTGP, $\mathrm{M^2GP^f}$, SMTGP, and $\mathrm{SMT^2GP}$ over 30 independent runs in the three multitask scenarios. The obtained objective values of the problems with utilisation level 0.95 or 0.85 are much larger than the problems with utilisation level 0.85 or 0.75. This indicates that a higher

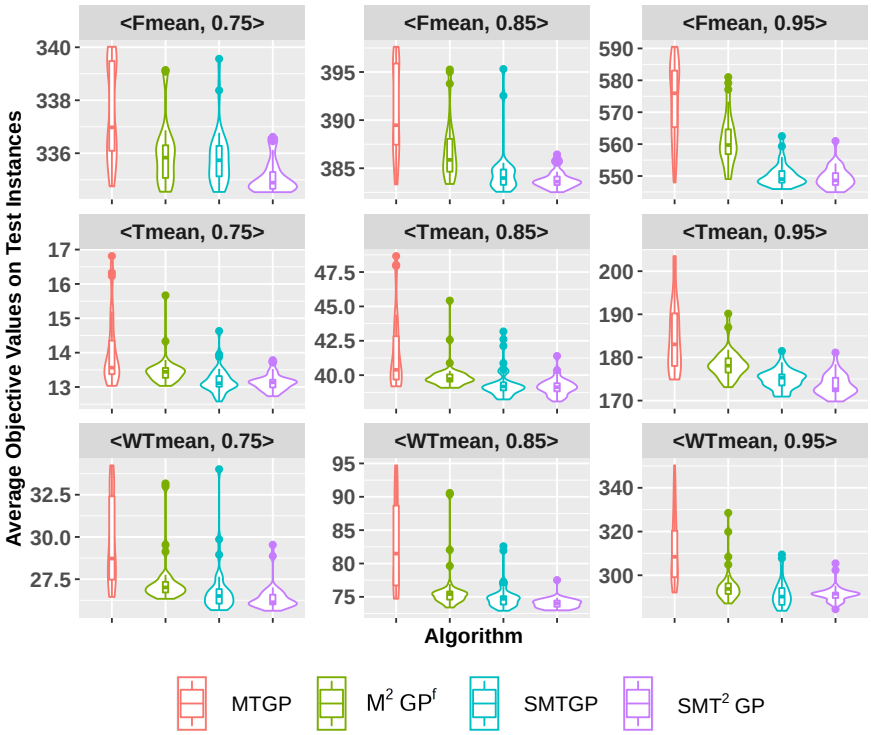

Figure 7.2: The violin plot of average objective values on test instances of MTGP, $\mathrm{M^2GP^f}$, SMTGP, and $\mathrm{SMT^2GP}$ over 30 independent runs in three multitask DFJSS scenarios (each row is a multitask scenario).

utilisation level leads to a more complex problem. Compared with MTGP, both $\mathrm{M^2GP^f}$ and SMTGP achieve better performance in all the scenarios. This shows the effectiveness of using surrogate and multitask techniques in DFJSS in an independent way, which is consistent with the conclusion drawn in [99, 239]. Compared with $\mathrm{M^2GP^f}$ and SMTGP, $\mathrm{SMT^2GP}$ achieves better performance with smaller average objective values and standard deviations, illustrating the effectiveness of $\mathrm{SMT^2GP}$.

Figure 7.3 shows the curves of the average objective values on the test instances based on 30 independent runs of MTGP, $\mathrm{M^2GP^f}$, SMTGP, and $\mathrm{SMT^2GP}$ in the three multitask scenarios with totally nine tasks. The results show that $\mathrm{SMT^2GP}$ converges faster than MTGP, $\mathrm{M^2GP^f}$, and SMTGP

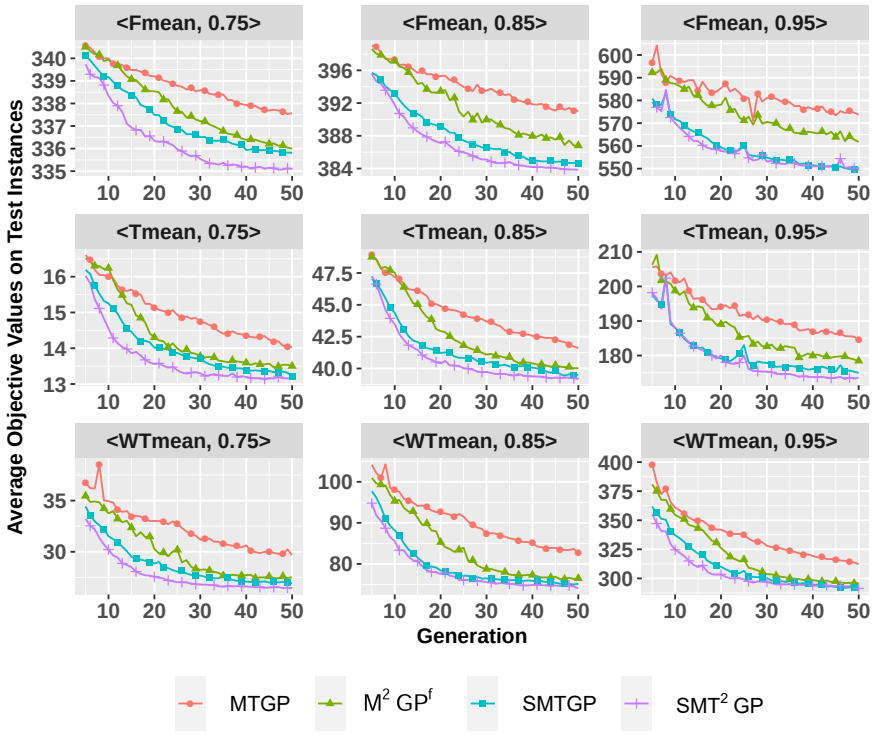

Figure 7.3: The curves of the average objective values on test instances based on 30 independent runs of MTGP, $\mathrm{M}^2\mathrm{GP}^\mathrm{f}$, SMTGP, and $\mathrm{SMT}^2\mathrm{GP}$ in three multitask DFJSS scenarios (each row is a multitask scenario).

in all the nine tasks of the three multitask scenarios. As stated earlier, the only difference between SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ is the allocation of individuals for different tasks with the surrogate. The advantage of $\mathrm{SMT}^2\mathrm{GP}$ over SMTGP indicates that the proposed surrogates are capable of sharing knowledge between different tasks by assigning appropriate individuals from the offspring pool to different tasks. This demonstrates the effectiveness of the proposed surrogate-assisted multitask GPHH algorithm. The effectiveness of $\mathrm{SMT}^2\mathrm{GP}$ mainly depends on the proposed knowledge sharing mechanism via surrogate.

Table 7.2: The mean (standard deviation) of the objective values on test instances of MTGP and SMTGP over 30 independent runs in three multitask scenarios.

Scenario	Task	MTGP	SMTGP
1	<Fmean, 0.75>	337.57(1.80)	335.60(1.20)**(–)**
	<Fmean, 0.85>	391.04(4.65)	384.35(1.86)**(–)**
	<Fmean, 0.95>	573.70(12.17)	548.77(4.60)**(–)**
2	<Tmean, 0.75>	14.03(1.01)	13.15(0.48)**(–)**
	<Tmean, 0.85>	41.61(2.73)	39.16(0.81)**(–)**
	<Tmean, 0.95>	184.60(8.04)	173.32(1.25)**(–)**
3	<WTmean, 0.75>	29.67(2.55)	26.53(0.59)**(–)**
	<WTmean, 0.85>	82.75(6.71)	74.84(2.63)**(–)**
	<WTmean, 0.95>	312.26(15.86)	289.48(6.73)**(–)**

7.4.2 Effectiveness of the Constructed Surrogate

A proper surrogate for DFJSS is important for the success of surrogate-assisted multitask learning. Table 7.2 shows the mean and standard deviation of the objective values on unseen instances of MTGP and SMTGP according to 30 independent runs in three multitask scenarios with nine DFJSS tasks. The results show that SMTGP significantly outperforms MTGP on all the tasks of the examined multitask scenarios. This shows the effectiveness of the surrogates for DFJSS in multitask scenarios. In addition, for both MTGP and SMTGP, the objective values in each task increase along with the utilisation level. This shows that a higher utilisation level leads to a more complex task, which is harder to optimise.

Figure 7.4 shows the curves of the average objective values on the test instances based on 30 independent runs of MTGP and SMTGP in the nine tasks of the three multitask scenarios. SMTGP performs much better for all the scenarios than MTGP for all tasks with a higher convergence speed after roughly five generations in the examined multitask scenarios. In most

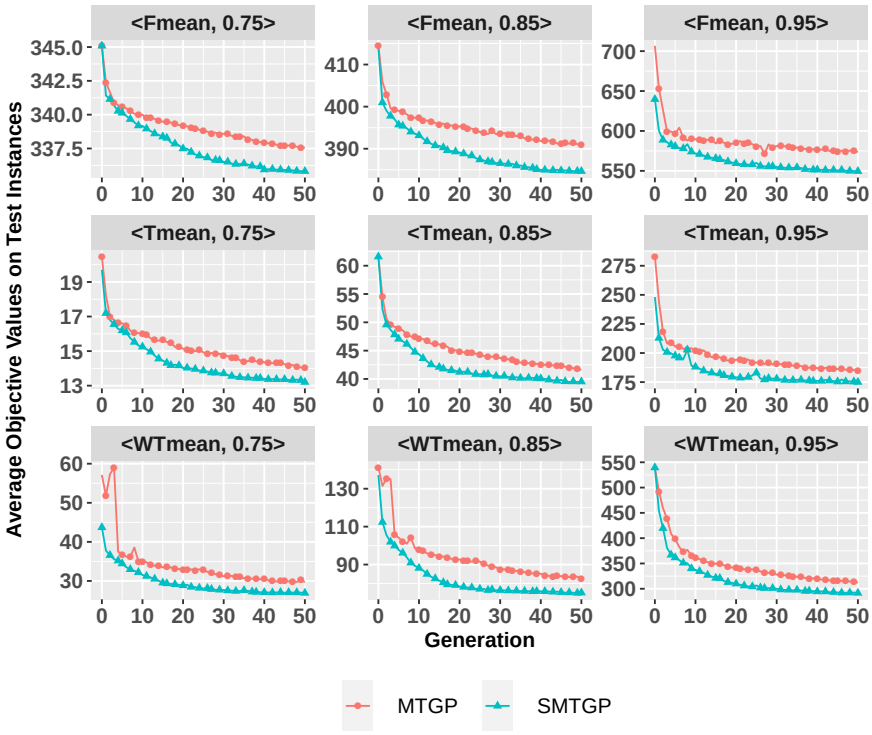

Figure 7.4: The curves of the average objective values on test instances based on 30 independent runs of MTGP and SMTGP in three multitask scenarios (each row is a multitask scenario).

cases, before generation five roughly, the difference between MTGP and SMTGP is not clear. One possible reason is that the sample data in the surrogate with KNN are not accurate before generation five, since the individuals in the population are not well evolved at the beginning of the evolutionary process. In summary, the results show the effectiveness of constructed surrogates for the multitask DFJSS scenarios.

7.4.3 Effectiveness of Diversity Preservation

The individuals in the offspring pool are cleared (as shown in Algorithm 10) to reduce the duplicated individuals based on their behaviour before

Figure 7.5: The curves of the average number of cleared individuals for tasks of SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in three multitask DFJSS scenarios (each row is a multitask scenario).

assigning to different subpopulations by the surrogates. Among the four algorithms, SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ are designed with the clearing process. It is interesting to see the effect of the proposed $\mathrm{SMT}^2\mathrm{GP}$ on the number of cleared individuals in the offspring pool, since the number of cleared individuals is a good indicator for measuring the diversity of the offspring pool. A smaller number of cleared individuals generally indicates a higher diversity of the offspring pool.

Figure 7.5 shows the curves of the number of cleared individuals by SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in the nine tasks of the three multitask scenarios. The results show that the number of cleared individuals in the tasks with the utilisation level 0.75 and 0.85 of $\mathrm{SMT}^2\mathrm{GP}$ is

much smaller than that of SMTGP during the evolutionary process. This indicates that $\mathrm{SMT^2GP}$ can improve the diversity of generated offspring of $Subpop_1$ for task 1 and $Subpop_2$ for task 2. One possible reason is that the final offspring for task 1 in $Subpop_1$ and task 2 in $Subpop_2$ contain different individuals from all subpopulations. In this case, the newly generated offspring for task 1 in $Subpop_1$ and task 2 in $Subpop_2$ vary due to the diversity of selected parents (i.e., also can be produced by the individuals for different tasks). However, it is not the case for task 3 in $Subpop_3$ that optimises the task with a utilisation level of 0.95, especially in <WTmean, 0.95>.

7.4.4 Individual Allocation for Tasks

One of the main ideas in this work is to allocate individuals to appropriate tasks. It is interesting to see the allocation of individuals for each task, i.e., the individuals originally generated for which task. Figure 7.6 shows the curves of the average number of assigned individuals by $\mathrm{SMT^2GP}$ over 30 independently runs to each of the nine tasks of the three multitask scenarios. The results show that the utilisation level is a more important factor for individual allocation than the objective. The tasks with the same utilisation level but with different objectives (i.e., in the same column) have a similar trend of allocated individuals.

For the subpopulation of task 1 with utilisation level of 0.75, the number of individuals originally generated for task 1 fluctuates around 40% over generations. The number of individuals originally generated for task 2 and task 3 shares a similar trend during the evolutionary process, which is 30% roughly. For task 2, with utilisation level of 0.85, the number of individuals from $Subpop_1$ for task 1 is the largest, and is approximately 45%. The number of individuals generated in $Subpop_2$ for task 2 ranks second, fluctuating between 30% and 35%. The least number of individuals is generated in $Subpop_3$ for task 3, which is lower than 25% roughly. For task 3 with the utilisation level of 0.95, the number of individuals gener-

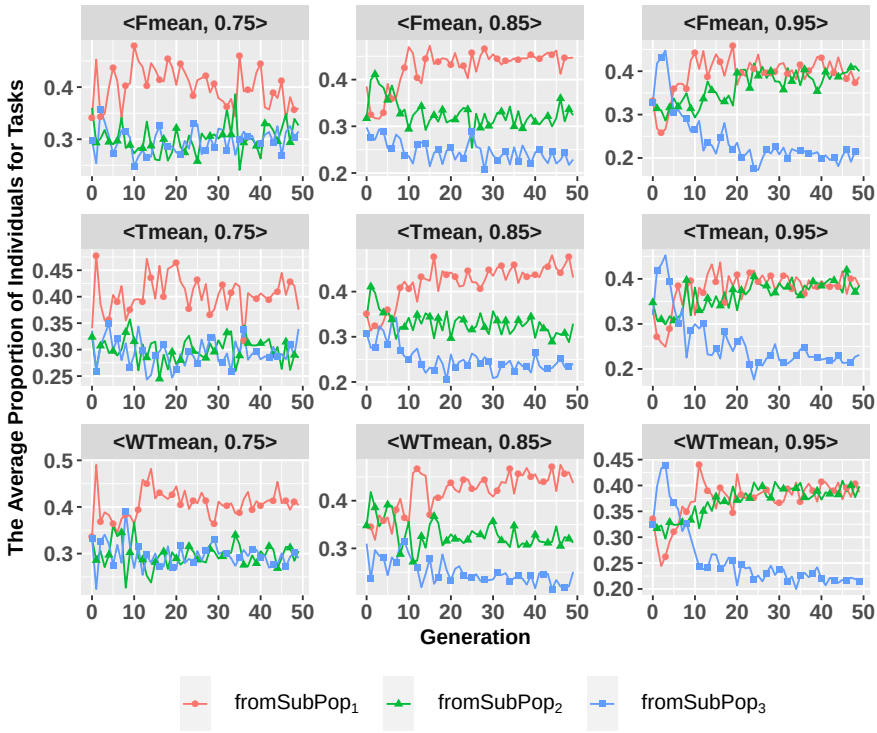

Figure 7.6: The curves of the average number of assigned individuals for a specific task from different tasks of $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in three multitask DFJSS scenarios (each row is a multitask scenario).

ated for task 1 and task 2 is quite similar, fluctuating between 35% and 40%. However, compared with the number of individuals originally generated for task 1 and task 2, the number of individuals generated for task 3 is quite small. In other words, the individuals for task 3 are not well allocated. This might be why the diversity of individuals in $Subpop_3$ for task 3 of $\mathrm{SMT}^2\mathrm{GP}$ is not as high as expected, which is shown in the last column of Figure 7.5. In addition, the offspring generated by the current subpopulation are not guaranteed to perform well in the corresponding problem.

The samples in the pools of the surrogates contain individuals from different subpopulations because of the knowledge transfer. According to the

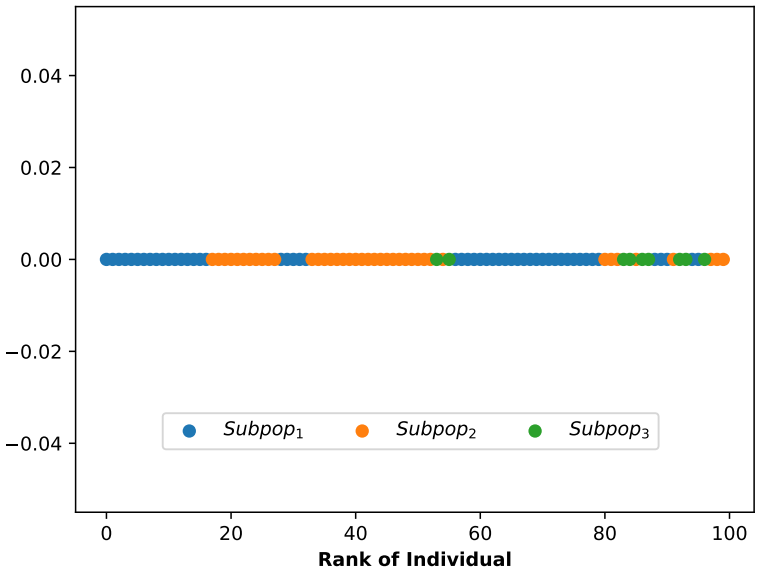

Figure 7.7: The ranks of individuals samples of the surrogate for task 3.

mechanism of KNN, the newly generated individuals in the intermediate population tend to be selected into the next generation if they have similar phenotypic behaviour with the top-ranked individual samples in the surrogate pool. To investigate the effect of surrogate on choosing individuals, we set the $subpopsize$ to 100 and take task 3 in the multitask scenario with mean-flowtime as an example. We collect the individual samples in the surrogate for task 3 at generation 20 (i.e., a steady state), and evaluate the individuals on task 3 (i.e., mean-flowtime with utilisation level 0.95).

Figure 7.7 shows the ranks of individuals samples of the surrogate for task 3. For task 3, the results show that individuals from task 1 and task 2 have better ranks than those generated by task 3. The individuals which are close to the individuals originally from task 1 and task 2 based on phenotypic characteristics tend to be selected. This is the reason why the proportions of individuals for task 3 from $Subpop_1$, $Subpop_2$, and $Subpop_3$ are 0.4 (large), 0.4 (large), and 0.2 (small), respectively.

Individuals in the surrogate (named as *mapped individuals*) are close to

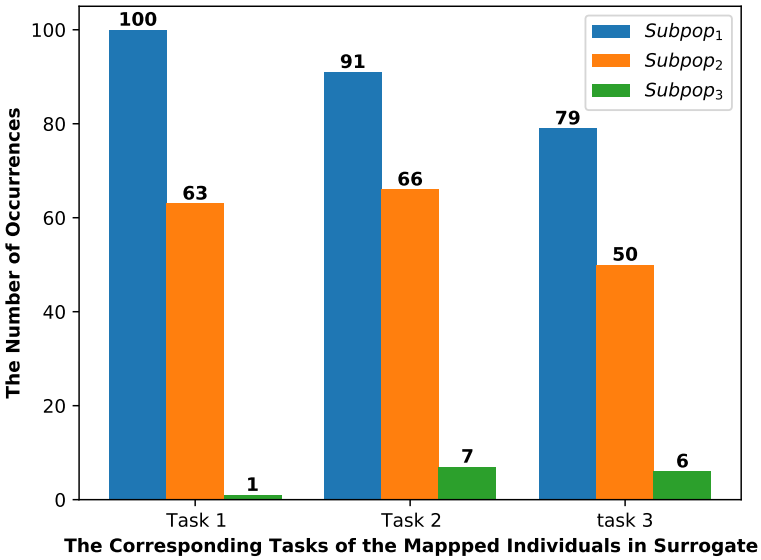

Figure 7.8: The corresponding tasks of the mapped individuals in surrogate of the newly generated individuals for tasks.

the generated individuals for tasks that directly affect the choices of individuals in the intermediate population. Figure 7.8 shows the corresponding tasks of the mapped individuals in the surrogates for the individuals in the intermediate population for task 1, task 2 and task 3. The results show that the newly generated individuals for the current task T_i tend to near the individual samples that are originally generated from the corresponding subpopulation $Subpop_i$ (i.e., $100 > 91 > 79$ corresponding to task 1, $66 > 63 > 50$ corresponding to task 2, and $6 \approx 7 > 1$ corresponding to task 3). It is consistent with our intuition that the newly generated individuals based on the corresponding subpopulation tend to benefit the current task. Taking Figure 7.7 into consideration, this is one reason for the bias to choose individuals from task 1 and task 2 for task 3, as shown in Figure 7.6.

When we further look at the mapped individuals in task 3, most newly generated individuals for task 3 are close to the surrogate samples that are originally from task 1 and task 2. It is noted that although the surrogate

samples for task 3 consist of large proportions of individuals that are originally for task 1 and task 2, the samples are good for task 3 (as shown in Figure 7.7) and become the individuals for optimising task 3. The newly generated individuals for task 3, which have good quality, can still be kept into the next generation. Although the proportion of individuals samples in the surrogate S_3 is not high, this does not mean the individuals generated for task 3 are useless for task 3.

In summary, we find that among the individuals in the surrogate pool, the ones from task 1 and task 2 have better ranks than that generated from task 3. On the other hand, the newly generated individuals tend to have similar phenotypic characterisation with the samples in the surrogate pool that are originally for the same task. Therefore, the offspring from task 1 and 2 tend to have better predicted fitness than that from task 3, and thus more likely to be selected into the next generation. In addition, Figure 7.6 shows that the number of removed individuals increases from task 1 to task 3. This indicates that task 3 has the smallest number of individuals to be allocated. These explain why the proportion of individuals from $Subpop_3$ is consistently smaller than that from other subpopulations.

7.5 Further Analyses

To understand the effect of the proposed algorithm for solving the DFJSS problems, the sizes, structures and behaviour of the evolved scheduling heuristics are further analysed in this section.

7.5.1 Sizes of Evolved Scheduling Heuristics

Figure 7.9 shows the curves of the average sizes of the routing and sequencing rules obtained by MTGP, SMTGP, and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in each task of the three multitask scenarios. The results show that a more complex task requires a larger rule size for all the algorithms.

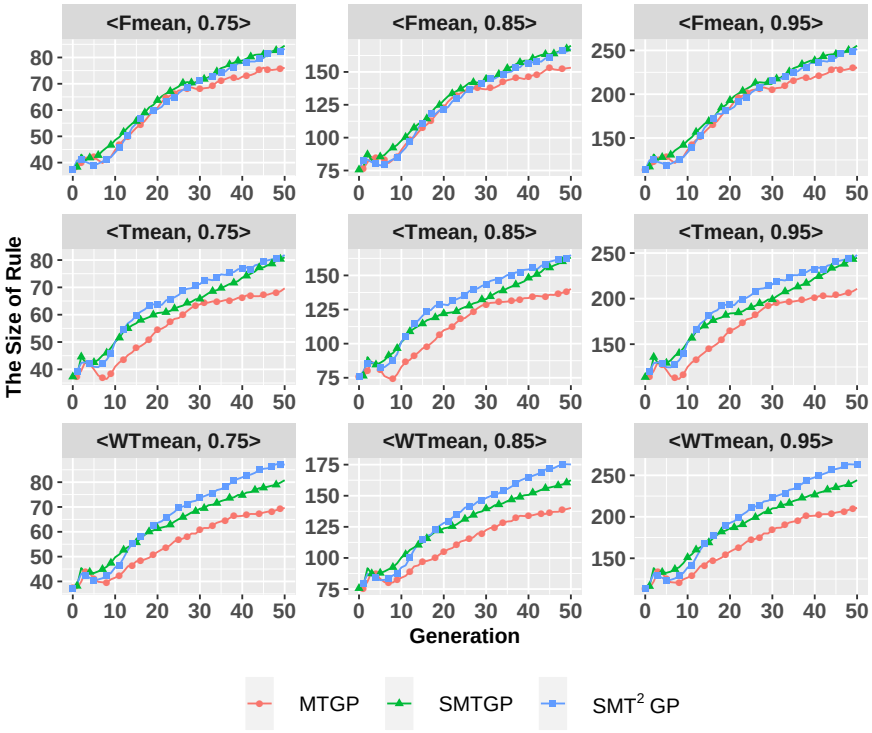

Figure 7.9: The curves of the average rule sizes (routing plus sequencing rule) for tasks of MTGP, SMTGP, and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in three multitask DFJSS scenarios (each row is a multitask scenario).

For example, the rule sizes for the tasks with utilisation level 0.95 are larger than those for the tasks with utilisation level 0.75 and 0.85. The rule sizes of SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ are similar over generations, and the rule sizes of SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ are larger than MTGP in all the scenarios from the early stage. This indicates that surrogate and multitask techniques tend to increase sizes of the evolved scheduling heuristics compared to MTGP. For SMTGP, one possible reason is that it tends to choose scheduling heuristics with large sizes due to their good quality in the intermediate population. For $\mathrm{SMT}^2\mathrm{GP}$, another possible reason is that the parents may have large sizes, since they may come from other subpopulations for complex tasks with large rules. This might also be a reason that the scheduling heuris-

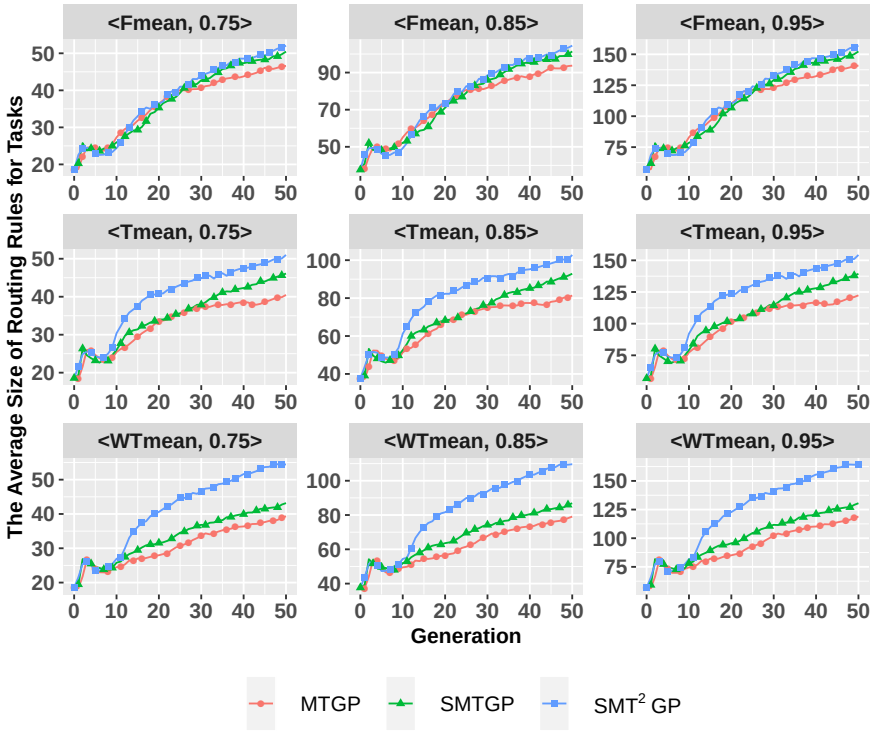

Figure 7.10: The curves of the average **routing** rule sizes for tasks of MTGP, SMTGP, and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in three multitask DFJSS scenarios (each row is a multitask scenario).

tics with good qualities can be obtained by SMTGP and $\mathrm{SMT}^2\mathrm{GP}$ from the early stage.

To further study the effect of the proposed $\mathrm{SMT}^2\mathrm{GP}$ on the rule size, the routing and sequencing rule are examined respectively due to the similarity of rule sizes of SMTGP and $\mathrm{SMT}^2\mathrm{GP}$. Figure 7.10 and Figure 7.11 show the curves of the average routing and sequencing rule sizes obtained by MTGP, SMTGP, and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs for each of the nine tasks in the three multitask scenarios, respectively. It can be seen that the sizes in each multitask (i.e., the tasks with the same objective but with different utilisation level) show a similar trend but with different scales. It may be due to the tasks in each multitask scenario are handled simulta-

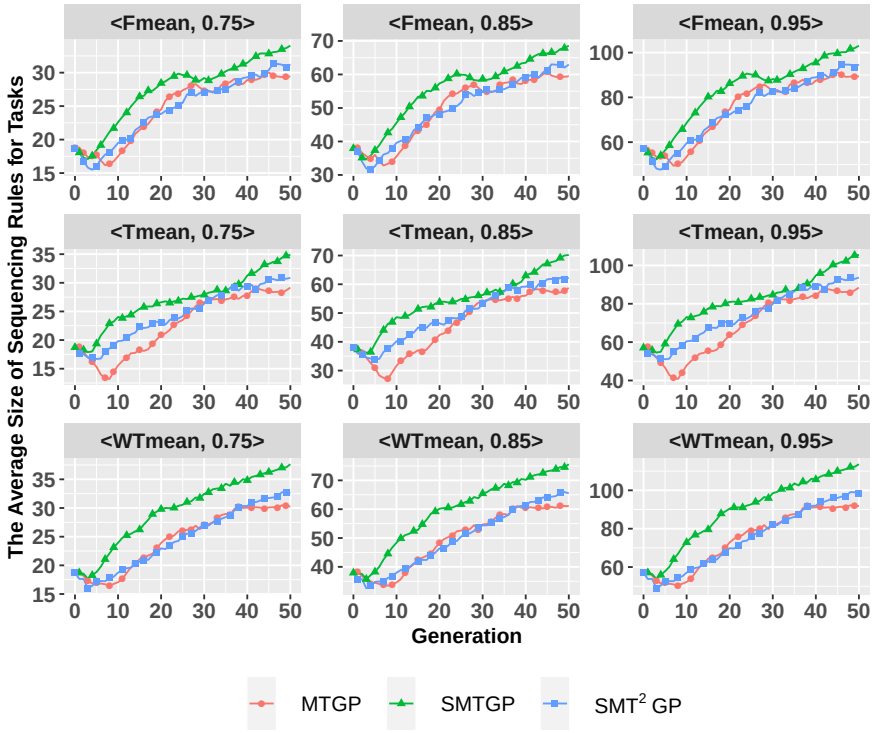

Figure 7.11: The curves of the average **sequencing** rule sizes for tasks of MTGP, SMTGP, and $\mathrm{SMT}^2\mathrm{GP}$ over 30 independent runs in three multitask DFJSS scenarios (each row is a multitask scenario).

neously by one population with three subpopulations, and the rule sizes of the subpopulations highly interact with each other. We can see that the trend of individual allocation is highly related to the utilisation level, while the trend of the rule size, either routing or sequencing, is highly related to the objective examined.

For the sizes of routing rules, as shown in Figure 7.10, $\mathrm{SMT}^2\mathrm{GP}$ tends to evolve larger rules than that of SMTGP for most tasks (i.e., <Tmean, 0.75>, <Tmean, 0.85>, <Tmean, 0.95>, <WTmean, 0.75>, <WTmean, 0.85>, and <WTmean, 0.95>). In addition, the sizes of the evolved routing rules by SMTGP is similar to that of MTGP. For the sizes of sequencing rules, as shown in Figure 7.11, SMTGP obtains larger rule sizes than that

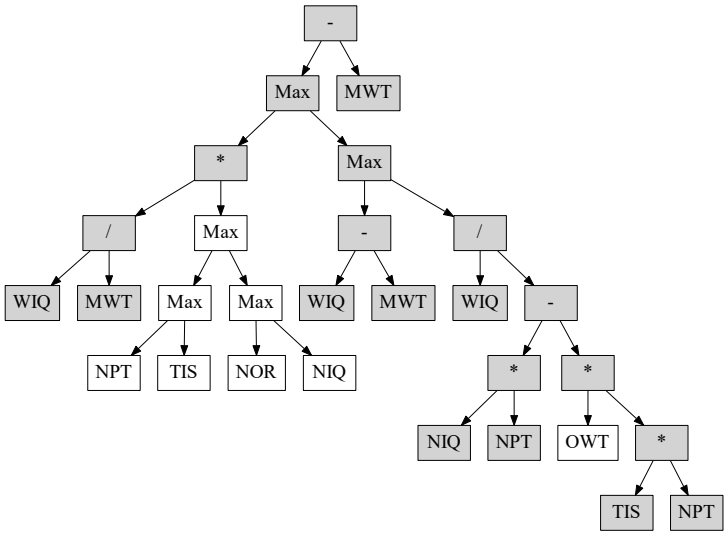

Figure 7.12: One of the best evolved routing rules for task 1 <WTmean, 0.75> in multitask scenario 3.

of $\mathrm{SMT^2GP}$ for all the tasks. The sizes of the evolved sequencing rules of $\mathrm{SMT^2GP}$ and MTGP are similar. This indicates that the sizes of rules for tasks can be similar but with various routing and sequencing combinations. In addition, $\mathrm{SMT^2GP}$ improves the quality of the evolved scheduling heuristics via routing rule, while SMTGP improves its performance by sequencing rule. It shows that enhancing the quality of routing rule may be an effective way to improve effectiveness of the final schedules in DFJSS.

7.5.2 Insight of Evolved Scheduling Heuristics

As stated in the previous subsection, the proposed algorithm $\mathrm{SMT^2GP}$ has a significant impact on the routing rule. In this section, we choose three routing rules evolved by $\mathrm{SMT^2GP}$ for each task in the multitask scenario 3 related to WTmean for further analysis. Figs. 7.12-7.14 show the best routing rules for task <WTmean, 0.75>, <WTmean, 0.85> and <WTmean, 0.95> in a multitask scenario, respectively. These three routing rules are

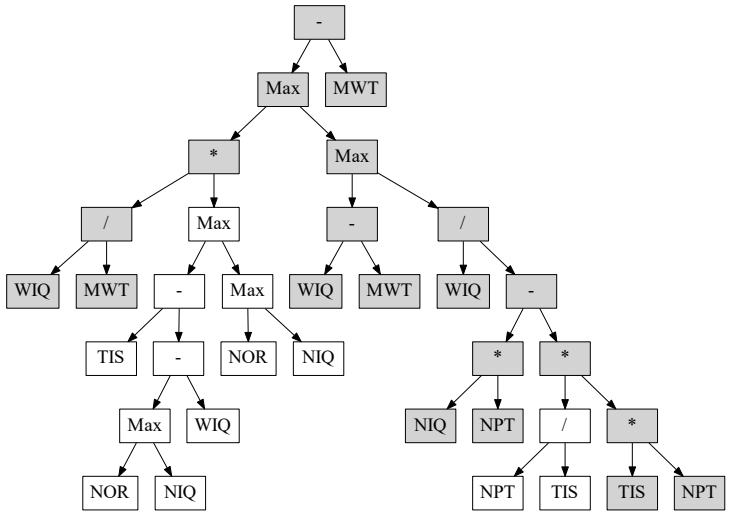

Figure 7.13: One of the best evolved routing rules for task 2 <WTmean, 0.85> in multitask scenario 3.

evolved together in the same multitask scenario.

In terms of the structures of the routing rules, we can see that the major part of the structure of the three routing rules is the same (as shown in grey). One possible reason is that the optimised objective in a multitask scenario is the same, and the rules for different tasks have inherent similarities. For the evolved building-blocks, except for the building-blocks shown in grey, the routing rule for task 1 shares the same component (i.e., Max{*, Max{NOR, NIQ}}) with the routing rule for task 2. Similarly, "Max{NOR, NIQ}" and "NPT / TIS" are evolved building-blocks of both the routing rule for task 2 and task 3. In addition, "Max{NOR, NIQ}" is a common constructed feature shared by the routing rules of task 1, task 2 and task 3. This is a sign that the tasks in a multitask scenario learn from each other, which is as expected. In terms of the rule size, the sizes of the routing rules for task 1, task 2, and task 3 are 29, 35 and 37, respectively. We can see that complex tasks often require larger rules.

To make it easy for analysis, we simplify the routing rules by calculating its different components. Note that a machine with a smaller priority

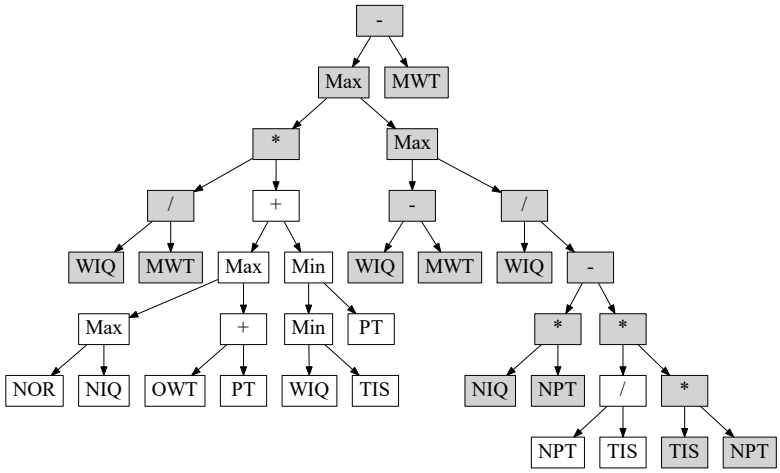

Figure 7.14: One of the best evolved routing rules for task 3 <WTmean, 0.95> in multitask scenario 3.

value is considered to be more prior. The routing rule for task 1 in Figure 7.12 can be further simplified, as shown in Eq. 7.1.

$$\begin{aligned} R_1 =& Max\{\frac{WIQ}{MWT} * Max\{NPT, TIS, NOR, NIQ\}, \\ & WIQ - MWT, \\ & \frac{WIQ}{NIQ * NPT - OWT * TIS * NPT}\} - MWT \\ \approx& Max\{\frac{WIQ}{MWT} * TIS, WIQ - MWT, \\ & \frac{WIQ}{NPT(NIQ - OWT * TIS)}\} - MWT \\ =& \frac{WIQ}{MWT} * TIS - MWT \end{aligned} \tag{7.1}$$

The Max function connects the main parts of this rule, and the key of this priority function is the component with the largest value. "Max{NPT, TIS, NOR, NIQ}" is simplified as TIS from step 1 to step 2, since TIS is almost always larger than NPT, NOR, and NIQ. Similarly, this rule can be further simplified as $\frac{WIQ}{MWT} * TIS - MWT$. We can see that this routing rule suggests the choice of the machine with a small workload (small WIQ) and a long

idle time (large MWT). It is noted that TIS is a constant here that will not significantly impact the final decision, since TIS is a property of operation and it is the same for all the candidate machines of an operation.

The routing rule for task 2 in Figure 7.13 can be simplified, as shown in Eq. 7.2.

$$\begin{aligned} R_2 =& Max\{\frac{WIQ}{MWT} * Max\{ \\ & TIS - Max\{NIQ, NOR\} + WIQ, NOR, NIQ\}, \\ & WIQ - MWT, \frac{WIQ}{NPT(NIQ - NPT)}\} - MWT \\ \approx& Max\{\frac{WIQ}{MWT} \\ & * Max\{TIS - Max\{NIQ, NOR\} + WIQ\}, \\ & WIQ - MWT, \frac{WIQ}{NPT(NIQ - NPT)}\} - MWT \\ =& \frac{WIQ}{MWT} * (TIS + WIQ - Max\{NIQ, NOR\}) \\ & - MWT \end{aligned} \tag{7.2}$$

From step 1 to step 2, "Max{TIS-Max{NIQ, NOR} + WIQ, NOR, NIQ}" can be simplified as "TIS - Max{NOR, NIQ} + WIQ", since "TIS - Max{NOR, NIQ} + WIQ" is more likely to be larger than NOR and NIQ. Compared with Eq. (2), except for WIQ and MWT, this routing rule takes NIQ or NOR into consideration. If NIQ is larger than NOR, this rule will be simplified as $\frac{WIQ}{MWT} * (TIS + WIQ - NIQ) - MWT$. Compared with the routing rule for task 1, except for WIQ and MWT, this rule takes the number of operations in the queue (NIQ) into consideration. This rule suggests selecting the machine with a larger number of operations but a smaller work in queue. This means that this rule prefers to select the machine with operations that need a short processing time. In this way, the newly allocated operations have a high probability of being processed earlier, since the number of sequencing decision point is increased. If NOR is larger than NIQ, this rule will be simplified as $\frac{WIQ}{MWT} * (TIS + WIQ - NOR) - MWT$. Like TIS, NOR

(the number of operations remaining of a job) is a characteristic of the operation, which is a constant for candidate machine and will not affect the routing decision too much. In addition, compared with the routing rule for task 1, WIQ plays an important role in this rule since it appears twice.

The routing rule for task 3 in Figure 7.14 can be simplified, as shown in Eq. 7.3. "Max{NOR, NIQ, OWT + PT}" is denoted as "OWT + PT", since "OWT + PT" is almost always larger than NOR and NIQ. PT is used to indicate "Min{WIQ, TIS, PT}", since PT is usually the smallest one among them. Different from R_1 and R_2, R_3 suggests choosing the machine that has high processing efficiency for a specific operation. It is consistent with our intuition that operations tend to be assigned to the most efficient machine for processing it. In addition, operation waiting time (OWT) can be considered as a constant as it is the same for all its candidate machines of an operation that are expected to be allocated.

$$\begin{aligned} R_3 =& Max\{\frac{WIQ}{MWT} * \{Max\{NOR, NIQ, OWT + PT\} \\ & + Min\{WIQ, TIS, PT\}\}, WIQ - MWT, \\ & \frac{WIQ}{NPT(NIQ - NPT)}\} - MWT \\ \approx& Max\{\frac{WIQ}{MWT} * (OWT + PT + PT), \\ & WIQ - MWT, \frac{WIQ}{NPT(NIQ - NPT)}\} - MWT \\ =& \frac{WIQ}{MWT} * (OWT + 2PT) - MWT \end{aligned} \tag{7.3}$$

In summary, the routing rules evolved in a multitask scenario that optimises the same objective have similarities. All the rules are highly related to WIQ and MWT, and "WIQ / MWT" is a shared pattern of the routing rules. However, they differ from each other according to the complexities of tasks. For the slightly complex task (task 2), the routing rule pays more attention to the workload of the machine. For the most complex task (task 3), the routing rule focuses on the processing efficiency of the machine.

This shows the effectiveness of the proposed surrogate-assisted multitask algorithm in DFJSS, since it not only shares the knowledge between different tasks but also keeps the unique characteristics for each task.

7.6 Chapter Summary

The goal of this chapter is to develop an effective surrogate-assisted evolutionary multitask approach to improve the effectiveness of GPHH for DFJSS. To achieve this purpose, this chapter builds surrogate models for all the tasks. The built surrogate models are used to improve the effectiveness of GPHH to solve a single task and transfer knowledge between the tasks.

The proposed surrogate-assisted multitask algorithm has three main features as compared to traditional multitask framework. First, a large number of new offspring are generated for providing useful materials for tasks. Second, the newly generated individuals are evaluated with surrogate rather than actual simulation evaluations. Third, individuals are assigned to optimise tasks based on the estimated fitness by surrogates directly rather than computationally expensive simulation revaluations. The results show that the proposed $\mathrm{SMT^2GP}$ can evolve highly-competitive scheduling heuristics for DFJSS with high convergence speed for all the examined multitask scenarios. The effectiveness of $\mathrm{SMT^2GP}$ is examined by comparing the quality of the evolved scheduling heuristics, the analyses of the diversity of individuals for tasks, structures and behaviour of the evolved scheduling heuristics. It has also been observed that the individual allocations for the tasks are highly related to the utilisation level. This implies that the complexities significantly impact the knowledge transfer between the tasks in a multitask scenario. In addition, we found that the sizes of the evolved rules over generations are highly related to the objective to be optimised in the task rather than the utilisation level.

Chapters 6-7 propose effective multitask GPHH algorithms to solve

multiple DFJSS tasks simultaneously. Chapter 6 focuses on an developing effective multitask GPHH algorithm based on the characteristic of GP, which is the first time to use multitask GPHH for DFJSS. Chapter 7 further enhances the performance of multitask GPHH with the surrogate, knowledge sharing, and phenotypic characterisation techniques. The proposed multitask GPHH algorithms have successfully used to evolve scheduling heuristics for multiple DFJSS tasks simultaneously.

In this thesis, we have developed a series of new GPHH methods for DFJSS, i.e., GPHH with surrogates, GPHH with the feature selection technique, GPHH with specialised genetic operators, and multitask GPHH. The next chapter will provide an overall summary on what objectives we have achieved and conclude this thesis.

Chapter 8

Conclusions

This thesis focuses on developing new GPHHs for the DFJSS problems. The overall goal is to improve the performance of GPHHs to evolve effective scheduling heuristics for DFJSS efficiently. This goal has been successfully achieved by investigating different aspects of GPHHs, including efficiency improvement with multi-fidelity surrogates, search space reduction with feature selection, search mechanism improvement with specialised genetic operators, and multitask GPHH. The effectiveness of each proposed algorithm is measured using the DFJSS simulation with a range of measurements and analyses.

The rest of this chapter highlights the achieved objectives in this thesis, followed by the main conclusions. Then, insightful discussions are provided to help understand the key issues in this research area. Finally, this chapter presents some potential research directions which are motivated by the studies in this thesis.

8.1 Achieved Objectives

The following research objectives have been fulfilled by this thesis.

1. This thesis has proposed an effective GPHH algorithm with multi-

fidelity surrogate models to improve training efficiency for evolving effective scheduling heuristics for DFJSS efficiently (Chapter 3). This thesis builds multiple surrogate models with different fidelities by shorting the simulation of the original problem. In addition, we propose a collaborative strategy to promote the effective of multi-fidelity surrogate models by developing an effective knowledge transfer mechanism. With the collaborations between the multiple surrogate models with different fidelities, the proposed algorithm has successfully achieved better efficiency and effectiveness to evolve scheduling heuristics for DFJSS. This work not only motivates the study of multi-fidelity surrogate models on GPHH but also broadens the applications of surrogate techniques to evolve rules for dynamic scheduling problems.

2. This thesis has developed an effective two-stage feature selection GPHH approach with individual adaptation strategies to automatically evolve scheduling heuristics only with the selected features for DFJSS efficiently (Chapter 4). First, a two-stage feature selection algorithm is developed to select only the important features and reduce the search space of GPHH. The first stage is a GPHH with a niching and surrogate model to quickly collect the data for the feature selection. The second stage uses the selected features to evolve promising scheduling heuristics by GPHH further. Second, based on the proposed two-stage feature selection algorithm, this thesis develops a new GPHH algorithm with feature selection to evolve effective scheduling heuristics with only the selected features. The proposed novel individual adaptation strategies are used to inherit the information of the examined individuals during the feature selection to maintain the quality of individuals with only the selected features. The results show that the proposed algorithms can help GPHH find rules with only selected features without sacrificing the performance. This thesis also shows how feature selection can im-

prove the effectiveness and sizes of the evolved rules, and how the proposed individual adaptation strategies can improve the efficiency of the proposed algorithms.

3. This thesis has proposed a new effective search mechanism with specialised genetic operators for GPHH to generate offspring based on the importance of the subtrees (Chapter 5). Specifically, this thesis develops recombinative guidance for the crossover to generate effective offspring by swapping subtrees between the parents according to their importance. The new crossover tends to retain the more important subtrees, and replace the less important subtrees with the important subtrees from the other parent. This thesis designs two measures for subtree importance. The first one is based on the importance of features. The second is based on the correlation between the behaviour of subtrees and the whole tree. This thesis then proposes a novel recombinative guidance mechanism for the crossover operator that defines the probabilities of each subtree to be selected to be (negatively) proportional to its importance. The results show that the GPHH algorithm with the correlation-based subtree importance performs better than the compared algorithms. The way of measuring the importance of subtrees in our problem can provide guidance for developing subtree importance measures for other problems such as regression. In addition, the analyses between the algorithms with feature frequency and correlation-based subtree importance measures provide a better understanding of the mechanism of GP based algorithms from the perspective of building-block recombination.

4. This thesis has developed an effective multitask GPHH approach, and a surrogate-assisted multitask GPHH approach to improving the effectiveness of GPHH to evolve scheduling heuristics for multiple DFJSS tasks, respectively (Chapters 6 and 7). First, this thesis

develops a novel and effective evolutionary multitask framework for GPHH by adapting the traditional evolutionary multitask algorithms based on the characteristics of GP and dynamic scheduling to optimise multiple DFJSS tasks simultaneously (Chapter 6). The results show that the proposed multitask GPHH is effective for dynamic scheduling. Second, this thesis proposes to transfer knowledge by allocating the newly generated individuals to appropriate tasks incorporated with the surrogate technique. Specifically, this thesis builds surrogates for different tasks accordingly by taking the behaviour of individuals and their real fitness into account. The proposed surrogates can not only improve the efficiency for solving a single task but also help share knowledge between the tasks. The results show that the proposed knowledge sharing mechanism can effectively transfer knowledge between tasks. The proposed surrogate-assisted multitask GPHH algorithm outperformed the state-of-the-art algorithm, and can evolve highly-competitive scheduling heuristics for DFJSS for different tasks in all examined multitask scenarios.

8.2 Main Conclusions

This section describes the main conclusions for this thesis drawn from the five major contribution chapters, i.e., Chapter 3 to Chapter 7.

8.2.1 Efficiency Improvement with Multi-fidelity Surrogates

To improve the efficiency of GPHH to evolve scheduling heuristics with the surrogate technique, two main issues need to be considered. First, how to generate surrogate models? Second, how to use the surrogate models in an effective and efficient way?

Chapter 3 develops an effective GPHH with multi-fidelity surrogate models to improve the efficiency for evolving scheduling heuristics in the

DFJSS problems. This thesis uses the degree of simplification of the problem to indicate the fidelity of the surrogate model. The surrogate models are designed by shortening the simulation of the original problem. The key idea of chapter 3 is to solve the desired problem by utilising the advantages of surrogate models with different fidelities. The problem with lower fidelity surrogates is computationally cheaper but less accurate. On the contrary, the problem with higher fidelity surrogates is computationally more expensive but more accurate. The most important thing is that an effective knowledge transfer mechanism is used to collaborative the multiple surrogate models to learn from each other.

The results verify the effectiveness and efficiency of the proposed algorithm, and the proposed algorithm is insensitive to the surrogate models. We can see that for improving training efficiency with the surrogate technique, how we manage to develop an effective collaboration strategy is important. This work broadens the studies of the surrogate technique to JSS by investigating a new collaborative way to improve training efficiency with surrogates rather than only focusing on improving the accuracy of the surrogates. Similar ideas of multi-fidelity based surrogate techniques can also be applicable to other problems.

In real-world applications, factories generally have a large production scale, making it impossible to learn scheduling heuristics using all the production data. Using surrogate techniques can help learn effective scheduling heuristics for factories in a reasonable time, which is more practical for handling real-world problems. However, the proposed algorithms have not investigated how to simplify a problem appropriately as the surrogate model. We will explore more in this direction in the future.

8.2.2 Search Space Reduction with Feature Selection

To reduce the search space of GPHH using feature selection, two main issues need to be considered. First, how to measure the importance of

features? Second, how to use the selected features and other information from the feature selection process properly?

Chapter 4 proposes a two-stage feature selection algorithm to select important features for the routing rule and the sequencing rule in DFJSS, respectively. The selected features are used in the mutation operator in the second stage of the proposed algorithm. Chapter 4 further proposes novel individual adaptation strategies to help GPHH evolve scheduling heuristics only with the selected features without losing the qualities of the rules. The success of the proposed individual adaptation based on mimicking the behaviour of individuals lies in the conversion of variable-length representation to vector-based fixed representation for GP individuals with phenotypic characterisation.

The results show that the proposed algorithm can detect important features effectively and efficiently. The results also show that the proposed algorithm can reduce the size of the evolved scheduling heuristics, thus to improve their interpretability potentially. The individual adaptation strategies that generate individuals by mimicking the behaviours of individuals with only the selected features are also applicable to other problems, if one can design proper mechanisms to mimic individual behaviour with only the selected features based on the problem-specific characteristics.

From the perspective of a practical problem, a number of features (i.e., factors) can affect the decision making of production schedules. However, the importance of the features usually is unclear. In addition, the scheduling heuristics with many features are not easy to be understood by the job shop operators. Feature selection algorithms proposed in this thesis can handle these two issues in real-world applications well. However, the proposed feature selection algorithms in this thesis treat the selected features equally to learn scheduling heuristics. A more effective way to use the selected features obtained by the feature selection algorithm is worth studying.

8.2.3 New Search Mechanism with Specialised Genetic Operators

Crossover is the main operator for GP to generate offspring by exchanging genetic materials (i.e., subtrees) of the parents. To generate high-quality offspring via crossover, we propose to improve the crossover operator by replacing unimportant subtrees from one parent with important subtrees from the other parent. Two main issues need to be considered. First, how to measure the importance of subtrees of an individual? Second, how to implement the subtree exchange according to the subtree importance information?

Chapter 5 proposes two strategies to measure the importance of subtrees. The first strategy is based on the frequency of features, assuming that important subtrees tend to have more important features. The second strategy is based on the coefficient between the behaviour of subtrees and the whole tree. If the behaviour of a subtree has a high (low) coefficient with the behaviour of the whole tree, the subtree is important (unimportant). A recombinative guidance mechanism is then designed for crossover to replace unimportant subtrees of one parent with important subtrees from the other parent.

The results show that the selected subtrees for crossover have a significant effect on the quality of the generated offspring. The evolved rules by the proposed algorithm with correlation-based recombinative guidance have better performance in most scenarios while no worse in all other scenarios due to its effectiveness for producing offspring. This is also verified by the analyses in terms of the depth ratios of selected subtrees, the correlations of selected subtrees, and the probability difference during the evolutionary process. The proposed algorithm does not need extra computational time compared with its counterparts. This verifies the advantages of utilising the information produced by GP during the evolutionary process and the efficient information calculation techniques such as corre-

lation coefficient. It is noted that the idea in this chapter is not limited to GP for DFJSS but can benefit GP in general. An important issue is to design a proper measure for the subtree importance based on the specific problem to be solved. Taking the symbolic regression problem as an example, the subtree importance can be measured with sampling semantics [222].

One key factor of the proposed GP algorithm with specialised genetic operators is measuring the subtree behaviour with decision situations. A better way of selection/design decision situations can positively affect the performance of the proposed algorithm. However, this thesis has not investigated how to extract representative decision situations for handling real-world applications. We will work on this in the near future.

8.2.4 Multitask Learning Ability in GPHH

To improve the multiple task solving ability of GPHH, two main issues need to be considered. First, is it applicable to use the traditional evolutionary multitask algorithms to GPHH for DFJSS directly? Second, how to improve the knowledge transfer effectiveness with GPHH for DFJSS?

Chapter 6 firstly adapts the traditional MFEA to GPHH with a number of modifications due to the specific characteristics of our problem. In addition, a novel origin-based offspring reservation strategy is proposed to enhance the knowledge sharing between tasks via crossover. The proposed algorithm in Chapter 6 is the first multitask GPHH approach for DFJSS. Based on the study in Chapter 6, Chapter 7 makes a deep study of the surrogate for multitask learning by considering two issues. First, how to build surrogates effectively? Second, how to use the surrogate to transfer knowledge between tasks properly? Chapter 7 uses the phenotypic characterisation and the fitness of fully-evaluated individuals to build the surrogates for all the tasks. The built surrogates are used to estimate the fitness of all the individuals for all the tasks, and allocates indi-

viduals to different tasks with the estimated fitness. The built surrogates are not only used to improve the effectiveness of solving a single task but also for knowledge transfer between tasks.

The results show that the proposed algorithm in Chapter 6 is effective for multitask GPHH to evolve scheduling heuristics for multiple DFJSS tasks simultaneously. This is a start point for the study in multitask GPHH for DFJSS. The results in Chapter 7 show that the proposed surrogate-assisted GPHH can evolve highly-competitive scheduling heuristics for DFJSS with high convergence speed for all the examined multitask scenarios. In addition, the scheduling heuristic evolving processes for different tasks highly interact with each other in a multitask scenario. It is a good case study to illustrate the effectiveness of surrogate-assisted evolutionary multitask for solving dynamic discrete and combinatorial optimisation problems.

It is not uncommon that a factory in practice involves different production tasks, and the tasks are typically related to each other. The proposed multitask algorithms can handle multiple production tasks simultaneously. However, the proposed multitask GP algorithms have not considered the effect of the relatedness between tasks on the performance of proposed algorithms. This is a promising direction to explore in the future.

8.3 Further Discussions

The previous section gives a summary of the key findings in this thesis. This section further provides discussions on general issues covered by this thesis and related to this research field.

8.3.1 Genotype vs Phenotype

Genotype considers the structures of GP individuals, while phenotype focuses on the behaviour of GP individuals [190]. Although phenotypic

characteristic has been successfully used in the literature (i.e., also in Chapter 4, Chapter 5 and Chapter 7) to measure the quality of GP individuals, genotype can also play an important role for investigation of GP individuals. For example, the genotype can provide us a better understanding of the genetic materials used in GP individuals. The variation of GP individuals occurs at the genotypic level but fitness is evaluated at the phenotypic level. Therefore, the mapping between the genotype and phenotype of GP individuals [11] in DFJSS is an interesting topic.

8.3.2 Implicit vs Explicit Knowledge Transfer

The knowledge sharing mechanisms in Chapter 3, Chapter 6, Chapter 7 are implemented implicitly via crossover. Explicit knowledge transfer represents another form to share knowledge, which can guarantee the effectiveness of knowledge sharing in different tasks [75]. As an explicit knowledge transfer, task adaptation aims to convert the search space of one task being good for an other task. A linearised domain adaptation was developed to transform the search space of a simple task to the search space similar to its constitutive complex task [10]. The transformed search space resembled a high correlation with its constitutive task and provided a platform for efficient knowledge transfer. All tasks were transformed into new space while maintaining the same geometric properties of solutions in [62]. The individuals were transformed to fit an other task by task space mapping strategy for generating higher-quality offspring in [142]. These kinds of studies [45] focus on adapting solutions of one task being good for other tasks, which is good to be also investigated in GPHH for DFJSS.

8.3.3 Surrogate

The surrogates in Chapter 3, Chapter 4 and Chapter 7 (i.e., either based on the KNN model or simplified problem) have shown their effectiveness

to estimate the fitness of GP individuals. There is a lot of information we can obtain from the simulation execution, such as the processing information in the early and middle stages. If one can utilise the information in an early stage to build surrogate models to estimate the performance of an individual in the later stage, the simulation can be dramatically shortened. Thus, training efficiency can be significantly improved. This is an interesting direction that uses the historical data to predict the performance of an individual in the future.

8.4 Future Work

8.4.1 Interpretability of Scheduling Heuristics

Evolving interpretable models is a hot topic in machine learning [81], especially in recent years. GP provides better interpretability due to its tree-based representation. However, the concepts or measures of the interpretability of the evolved scheduling heuristics by GPHH are still unclear. This thesis mainly uses the number of nodes, the number of unique features, and semantic analyses to investigate the interpretability of the evolved scheduling heuristics. More promising measures for evaluating the interpretability of the obtained scheduling heuristics by GPHH are worth to be studied. In addition, new algorithms to improve the interpretability of the evolved scheduling heuristics need to be further studied.

Some potential directions of the interpretability of the evolved scheduling heuristics obtained by GPHH are: (1) propose effective criteria to measure the interpretability of the evolved rules, for example, considering the types of features such as time-related feature, e.g., processing time, and number related feature, e.g., the number of operations left for a job. If the evolved subtrees contain the same type of features, it tends to be easier to be understood, (2) develop grammar-based GPHH to restrict the search space to evolve interpretable scheduling heuristics automatically during

the evolutionary process.

8.4.2 Local Search

Local search [1] is a widely used technique in evolutionary algorithms to exploit good solutions by looking around their neighbourhoods. However, it is rarely applied in GPHH, since it is challenging to define neighbourhood relationship for GP individuals with the variable-length tree-based representation. A small change in the GP tree may lead to a big influence on the behaviour of a GP individual. In addition, the fitness evaluation of GP individuals in DFJSS is time-consuming, since there are lots of priority calculations in the simulation. Thus, measuring the quality of the neighbours of an individual is time-consuming.

To handle these issues, some potential directions are: (1) consider the phenotypic space of GP individuals to define the neighbourhood relationship. If two GP individuals have similar phenotypic characteristics, we can say they are neighbours in the local search domain, (2) utilise the surrogate technique to estimate the fitness of the neighbours of a good solution.

8.4.3 Relatedness Between Tasks

The relatedness between tasks is an important prerequisite for the effectiveness of multitask algorithm, and measuring the relatedness between tasks is still an open question [182], especially in combinatorial optimisation. The existing studies mainly focus on the relatedness between continuous benchmark functions, where the characteristics of the problem, such as the fitness landscape are known. However, it is not the case in combinatorial optimisation. It is challenging to decide what kinds of information can be used for measuring the relatedness between tasks, and how to measure the relatedness between tasks.

Some potential future directions are: (1) use the individual information

during the evolutionary process to measure the relatedness between tasks, since the individuals become more and more specific to the optimised task along with the evolutionary process, (2) apply the machine learning technique such as Kullback–Leibler divergence [132] to measure the relatedness between tasks based on the obtained individual information.

8.4.4 Multi/many-objective Optimisation

Scheduling problems normally involve multiple objectives. A major challenge for evolving scheduling heuristics for DFJSS is to balance and handle conflicting objective simultaneously such as minimising max-flowtime and mean-flowtime. This is particularly true in DFJSS, since its parameters and variables can change over time [207]. To the best of our knowledge, handling multiple or many conflicting objectives for DFJSS has not tacked by GPHH due to the difficulty of the problem.

Some potential directions that worth to be investigated are: (1) develop a new multi/many-objective GPHH algorithm for DFJSS based on the characteristics of Pareto-front optimisation, (2) develop preference-based multi/many-objective GPHH algorithms for DFJSS. The preference-based algorithms can be used to narrow down the Pareto-front search space of GPHH, (3) develop new algorithms to analyses the behaviour of evolved scheduling heuristics to identify how the trade-offs between different objectives are made.

Bibliography

[1] AARTS, E., AARTS, E. H., AND LENSTRA, J. K. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] AITZAI, A., BENMEDJDOUB, B., AND BOUDHAR, M. Branch-and-bound and pso algorithms for no-wait job shop scheduling. *Journal of Intelligent Manufacturing 27*, 3 (2016), 679–688.

[3] AMJAD, M. K., BUTT, S. I., KOUSAR, R., AHMAD, R., AGHA, M. H., FAPING, Z., ANJUM, N., AND ASGHER, U. Recent research trends in genetic algorithm based flexible job shop scheduling problems. *Mathematical Problems in Engineering 2018* (2018), 1–32.

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

[5] ARDEH, M. A., MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic with knowledge transfer for uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 334–335.

[6] ASSIMI, H., JAMALI, A., AND NARIMAN-ZADEH, N. Multi-objective sizing and topology optimization of truss structures using genetic programming based on a new adaptive mutant operator. *Neural Computing and Applications 31*, 10 (2019), 5729–5749.

[7] BÄCK, T., FOGEL, D. B., AND MICHALEWICZ, Z. *Handbook of evolutionary computation*. CRC Press, 1997.

[8] 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.

[9] BAI, R., AND KENDALL, G. An investigation of automated planograms using a simulated annealing based hyper-heuristic. In *Metaheuristics: Progress as real problem solvers*. Springer, 2005, pp. 87–108.

[10] BALI, K. K., GUPTA, A., FENG, L., ONG, Y. S., AND SIEW, T. P. Linearized domain adaptation in evolutionary multitasking. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2017), IEEE, pp. 1295–1302.

[11] BANZHAF, W. Genotype-phenotype-mapping and neutral variation—a case study in genetic programming. In *Proceedings of the International Conference on Parallel Problem Solving from Nature* (1994), Springer, pp. 322–332.

[12] BANZHAF, W., NORDIN, P., KELLER, R. E., AND FRANCONE, F. D. *Genetic programming: an introduction*, vol. 1. Morgan Kaufmann San Francisco, 1998.

[13] BAYKASOĞLU, A., HAMZADAYI, A., AND KÖSE, S. Y. Testing the performance of teaching–learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. *Information Sciences 276* (2014), 204–218.

[14] BENNETT, S., NGUYEN, S., AND ZHANG, M. A hybrid discrete particle swarm optimisation method for grid computation scheduling. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2014), IEEE, pp. 483–490.

[15] BERTSEKAS, D. P. *Dynamic programming and optimal control, 3rd Edition*. Athena Scientific, 2005.

[16] BEYER, H.-G., AND SCHWEFEL, H.-P. Evolution strategies–A comprehensive introduction. *Natural computing 1*, 1 (2002), 3–52.

[17] BLAZEWICZ, J., ECKER, K., PESCH, E., SCHMIDT, G., AND WEGLARZ, J. *Handbook on scheduling*. Springer, 2019.

[18] BOKHORST, J. A., NOMDEN, G., AND SLOMP, J. Performance evaluation of family-based dispatching in small manufacturing cells. *International Journal of Production Research 46*, 22 (2008), 6305–6321.

[19] BOYLE, E., AL-AKASH, M., GALLAGHER, A. G., TRAYNOR, O., HILL, A. D., AND NEARY, P. C. Optimising surgical training: use of feedback to reduce errors during a simulated surgical procedure. *Postgraduate medical journal 87*, 1030 (2011), 524–528.

[20] BRANKE, J., HILDEBRANDT, T., AND SCHOLZ-REITER, B. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary Computation 23*, 2 (2015), 249–277.

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

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

[23] BRUCKER, P., AND SCHLIE, R. Job-shop scheduling with multipurpose machines. *Computing 45*, 4 (1990), 369–375.

[24] 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*. Springer, 2003, pp. 457–474.

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

[26] BURKE, E. K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E., AND WOODWARD, J. R. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[27] BURKE, E. K., HYDE, M. R., KENDALL, G., OCHOA, G., OZCAN, E., AND WOODWARD, J. R. Exploring hyper-heuristic methodologies with genetic programming. In *Proceedings of the Computational intelligence*. Springer, 2009, pp. 177–201.

[28] BURKE, E. K., HYDE, M. R., KENDALL, G., AND WOODWARD, J. R. A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics. *IEEE Transactions Evolutionary Computation 14*, 6 (2010), 942–958.

[29] BURKE, E. K., KENDALL, G., AND SOUBEIGA, E. A tabu-search hyperheuristic for timetabling and rostering. *Journal of heuristics 9*, 6 (2003), 451–470.

[30] CAMACHO-HERNANDEZ, G. A., AND TAYLOR, P. Lessons from nature: structural studies and drug design driven by a homologous surrogate from invertebrates, achbp. *Neuropharmacology 179* (2020), 108108.

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

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

[33] CARLIER, J., AND PINSON, E. A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research 26*, 1 (1990), 269–287.

[34] CARLIER, J., AND PINSON, E. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research 78*, 2 (1994), 146–161.

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

[36] CHARNIAK, E. *Introduction to artificial intelligence*. Pearson Education India, 1985.

[37] CHEN, H., CHU, C., AND PROTH, J.-M. An improvement of the lagrangean relaxation approach for job shop scheduling: a dynamic programming method. *IEEE Transactions on Robotics and Automation 14*, 5 (1998), 786–795.

[38] CHEN, K., XUE, B., ZHANG, M., AND ZHOU, F. Novel chaotic grouping particle swarm optimization with a dynamic regrouping strategy for solving numerical optimization tasks. *Knowledge-Based Systems* (2020), 105568.

[39] CHEN, K., XUE, B., ZHANG, M., AND ZHOU, F. An evolutionary multitasking-based feature selection method for high-dimensional classification. *IEEE Transactions on Cybernetics* (2020. Doi: 10.1109/TCYB.2020.3042243).

[40] CHEN, K., ZHOU, F., AND XUE, B. Particle swarm optimization for feature selection with adaptive mechanism and new updating strategy. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence* (2018), Springer, pp. 419–431.

[41] CHEN, K., ZHOU, F., AND YUAN, X. Hybrid particle swarm optimization with spiral-shaped mechanism for feature selection. *Expert Systems with Applications 128* (2019), 140–156.

[42] CHEN, Q., ZHANG, M., AND XUE, B. Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Transactions on Evolutionary Computation 21*, 5 (2017), 792–806.

[43] CHEN, Q., ZHANG, M., AND XUE, B. Geometric semantic genetic programming with perpendicular crossover and random segment mutation for symbolic regression. In *Proceedings of the Simulated Evolution and Learning* (2017), pp. 422–434.

[44] CHEN, Q., ZHANG, M., AND XUE, B. Structural risk minimization-driven genetic programming for enhancing generalization in symbolic regression. *IEEE Transactions on Evolutionary Computation 23*, 4 (2019), 703–717.

[45] CHEN, Z., ZHOU, Y., HE, X., AND ZHANG, J. Learning task relationships in evolutionary multitasking for multiobjective continuous optimization. *IEEE Transactions on Cybernetics* (2020, Doi: 10.1109/TCYB.2020.3029176).

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

[47] CHIANG, T.-C., AND FU, L.-C. Using dispatching rules for job shop scheduling with due date-based objectives. *International Journal of Production Research 45*, 14 (2007), 3245–3262.

[48] CHONG, C. S., SIVAKUMAR, A. I., LOW, M. Y. H., AND GAY, K. L. A bee colony optimization algorithm to job shop scheduling. In *Pro-*

ceedings of the Conference on Winter Simulation (2006), Winter Simulation Conference, pp. 1954–1961.

[49] CHUGH, T., JIN, Y., MIETTINEN, K., HAKANEN, J., AND SINDHYA, K. A surrogate-assisted reference vector guided evolutionary algorithm for computationally expensive many-objective optimization. *IEEE Transactions Evolutionary Computation 22*, 1 (2018), 129–142.

[50] CONROY, G. Handbook of genetic algorithms. *The Knowledge Engineering Review 6*, 4 (1991), 363–365.

[51] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A hyperheuristic approach to scheduling a sales summit. In *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling* (2000), Springer, pp. 176–190.

[52] COWLING, P., KENDALL, G., AND SOUBEIGA, E. A parameter-free hyperheuristic for scheduling a sales summit. In *Proceedings of the Metaheuristic International Conference* (2001), vol. 2001, Citeseer, pp. 127–131.

[53] DA, B., ONG, Y.-S., FENG, L., QIN, A. K., GUPTA, A., ZHU, Z., TING, C.-K., TANG, K., AND YAO, X. Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results. *arXiv:1706.03470* (2017).

[54] DANIEL, W. W., ET AL. *Applied nonparametric statistics*. Houghton Mifflin, 1978.

[55] DANIELSSON, P.-E. Euclidean distance mapping. *Computer Graphics and Image Processing 14*, 3 (1980), 227–248.

[56] DASH, M., AND LIU, H. Feature selection for classification. *Intelligent Data Analysis 1*, 3 (1997), 131–156.

[57] DAVIS, J. P., EISENHARDT, K. M., AND BINGHAM, C. B. Developing theory through simulation methods. *Academy of Management Review 32*, 2 (2007), 480–499.

[58] 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.

[59] DICK, G. Sensitivity-like analysis for feature selection in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2017), pp. 401–408.

[60] DICK, G., RIMONI, A. P., AND WHIGHAM, P. A. A re-examination of the use of genetic programming on the oral bioavailability problem. In *Proceedings of the Conference on Genetic and Evolutionary Computation* (2015), pp. 1015–1022.

[61] DIMOPOULOS, C., AND ZALZALA, A. M. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software 32*, 6 (2001), 489–498.

[62] DING, J., YANG, C., JIN, Y., AND CHAI, T. Generalized multitasking for evolutionary optimization of expensive problems. *IEEE Transactions on Evolutionary Computation 23*, 1 (2017), 44–58.

[63] DINH, T. T. H., CHU, T. H., AND NGUYEN, Q. U. Transfer learning in genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2015), IEEE, pp. 1145–1151.

[64] DOERR, B., KÖTZING, T., LAGODZINSKI, J. G., AND LENGLER, J. Bounding bloat in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2017), pp. 921–928.

[65] DOMINIC, P. D., KALIYAMOORTHY, S., AND KUMAR, M. S. Efficient dispatching rules for dynamic job shop scheduling. *The International Journal of Advanced Manufacturing Technology 24*, 1-2 (2004), 70–75.

[66] DORIGO, M., BIRATTARI, M., AND STUTZLE, T. Ant colony optimization. *IEEE Computational Intelligence Magazine 1*, 4 (2006), 28–39.

[67] DOUGUET, D. e-LEA3D: a computational-aided drug design web server. *Nucleic Acids Research 38*, suppl_2 (2010), W615–W621.

[68] 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.

[69] DURASEVIC, M., AND JAKOBOVIC, D. Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. *Genetic Programming and Evolvable Machines 19*, 1-2 (2018), 9–51.

[70] DURASEVIC, M., AND JAKOBOVIC, D. A survey of dispatching rules for the dynamic unrelated machines environment. *Expert Systems with Applications 113* (2018), 555–569.

[71] DURASEVIC, M., JAKOBOVIC, D., AND KNEZEVIC, K. Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing 48* (2016), 419–430.

[72] EIBEN, A. E., AND SMITH, J. E. What is an evolutionary algorithm? In *Introduction to Evolutionary Computing*. Springer, 2015, pp. 25–48.

[73] EMMERICH, M., GIOTIS, A., ÖZDEMIR, M., BÄCK, T., AND GIANNAKOGLOU, K. Metamodel—Assisted evolution strategies. In *International Conference on Parallel Problem Solving from Nature* (2002), Springer, pp. 361–370.

[74] FATTAHI, P., AND FALLAHI, A. Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability.

CIRP Journal of Manufacturing Science and Technology 2, 2 (2010), 114–123.

[75] FENG, L., ZHOU, L., ZHONG, J., GUPTA, A., ONG, Y.-S., TAN, K.-C., AND QIN, A. Evolutionary multitasking via explicit autoencoding. *IEEE Transactions on Cybernetics 49*, 9 (2018), 3457–3470.

[76] FISHER, M., AND RAMAN, A. Reducing the cost of demand uncertainty through accurate response to early sales. *Operations Research 44*, 1 (1996), 87–99.

[77] FOGEL, D. B. An overview of evolutionary programming. In *Proceedings of the Evolutionary Algorithms* (1999), Springer, pp. 89–109.

[78] GARZA-SANTISTEBAN, F., CRUZ-DUARTE, J. M., AMAYA, I., ORTIZ-BAYLISS, J. C., CONANT-PABLOS, S. E., AND TERASHIMA-MARÍN, H. Influence of instance size on selection hyper-heuristics for job shop scheduling problems. In *Proceedings of the IEEE Symposium Series on Computational Intelligence* (2019), IEEE, pp. 1708–1715.

[79] GARZA-SANTISTEBAN, F., SÁNCHEZ-PÁMANES, R., PUENTE-RODRÍGUEZ, L. A., AMAYA, I., ORTIZ-BAYLISS, J. C., CONANT-PABLOS, S., AND TERASHIMA-MARÍN, H. A simulated annealing hyper-heuristic for job shop scheduling problems. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2019), IEEE, pp. 57–64.

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

[81] GILPIN, L. H., BAU, D., YUAN, B. Z., BAJWA, A., SPECTER, M., AND KAGAL, L. Explaining explanations: An overview of interpretability of machine learning. In *Proceedings of the IEEE Interna-*

tional Conference on Data Science and Advanced Analytics (2018), IEEE, pp. 80–89.

[82] GOMES, M. C., BARBOSA-PÓVOA, A. P., AND NOVAIS, A. Q. Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach. *International Journal of Production Research 51*, 17 (2013), 5120–5141.

[83] GONG, G., CHIONG, R., DENG, Q., AND GONG, X. A hybrid artificial bee colony algorithm for flexible job shop scheduling with worker flexibility. *International Journal of Production Research 58*, 14 (2020), 4406–4420.

[84] GONG, M., TANG, Z., LI, H., AND ZHANG, J. Evolutionary multitasking with dynamic resource allocating strategy. *IEEE Transactions on Evolutionary Computation 23*, 5 (2019), 858–869.

[85] GOODFELLOW, I., BENGIO, Y., AND COURVILLE, A. Machine learning basics. *Deep learning 1* (2016), 98–164.

[86] GRABOWSKI, J., NOWICKI, E., AND ZDRZAŁKA, S. A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research 26*, 2 (1986), 278–285.

[87] GRAVES, S. C. A review of production scheduling. *Operations Research 29*, 4 (1981), 646–675.

[88] GROMICHO, J. A., VAN HOORN, J. J., SALDANHA-DA GAMA, F., AND TIMMER, G. T. Solving the job-shop scheduling problem optimally by dynamic programming. *Computers & Operations Research 39*, 12 (2012), 2968–2977.

[89] GU, S., CHENG, R., AND JIN, Y. Feature selection for high-dimensional classification using a competitive swarm optimizer. *Soft Computing 22*, 3 (2018), 811–822.

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

[91] GUPTA, A., ONG, Y.-S., AND FENG, L. Multifactorial evolution: toward evolutionary multitasking. *IEEE Transactions on Evolutionary Computation 20*, 3 (2015), 343–357.

[92] GUPTA, A., ONG, Y.-S., AND FENG, L. Insights on transfer optimization: Because experience is the best teacher. *IEEE Transactions on Emerging Topics in Computational Intelligence 2*, 1 (2017), 51–64.

[93] GUYON, I., AND ELISSEEFF, A. An introduction to variable and feature selection. *Journal of Machine Learning Research 3* (2003), 1157–1182.

[94] GUZEK, M., BOUVRY, P., AND TALBI, E.-G. A survey of evolutionary computation for resource management of processing in cloud computing. *IEEE Computational Intelligence Magazine 10*, 2 (2015), 53–67.

[95] HAJIRAMEZANALI, E., DADANEH, S. Z., KARBALAYGHAREH, A., ZHOU, M., AND QIAN, X. Bayesian multi-domain learning for cancer subtype discovery from next-generation sequencing count data. *arXiv preprint arXiv:1810.09433* (2018).

[96] HAO, X., QU, R., AND LIU, J. A unified framework of graph-based evolutionary multitasking hyper-heuristic. *IEEE Transactions on Evolutionary Computation* (2020. Doi: 10.1109/TEVC.2020.2991717).

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

[98] HELD, M., AND KARP, R. M. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics 10*, 1 (1962), 196–210.

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

[100] HILDEBRANDT, T., HEGER, J., AND SCHOLZ-REITER, B. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *Proceedings of the Conference on Genetic and Evolutionary Computation* (2010), ACM, pp. 257–264.

[101] HO, N. B., AND TAY, J. C. Evolving dispatching rules for solving the flexible job-shop problem. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2005), pp. 2848–2855.

[102] HOFFMANN, P. A dynamic limit order market with fast and slow traders. *Journal of Financial Economics 113*, 1 (2014), 156–169.

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

[104] HUANG, S., ZHONG, J., AND YU, W. Surrogate-assisted evolutionary framework with adaptive knowledge transfer for multi-task optimization. *IEEE Transactions on Emerging Topics in Computing* (2019).

[105] HUNT, R. *Genetic Programming Hyper-heuristics for Job Shop Scheduling*. Victoria University of Wellington, 2016.

[106] HUNT, R., JOHNSTON, M., AND ZHANG, M. Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the Conference on Genetic and Evolutionary Computation* (2014), ACM, pp. 927–934.

[107] HYDE, M. R. *A genetic programming hyper-heuristic approach to automated packing*. PhD thesis, University of Nottingham, UK, 2010.

[108] IBA, H., AND DE GARIS, H. Extending genetic programming with recombinative guidance. *Advances in Genetic Grogramming 2* (1996), 69–88.

[109] IQBAL, M., XUE, B., AL-SAHAF, H., AND ZHANG, M. Cross-domain reuse of extracted knowledge in genetic programming for image classification. *IEEE Transactions on Evolutionary Computation 21*, 4 (2017), 569–587.

[110] ITO, T., IBA, H., AND SATO, S. A self-tuning mechanism for depth-dependent crossover. *Advances in Genetic Programming 3* (1999), 377.

[111] IWASAKI, Y., SUZUKI, I., YAMAMOTO, M., AND FURUKAWA, M. Job-shop scheduling approach to order-picking problem. *Transactions of the Institute of Systems, Control and Information Engineers 26*, 3 (2013), 103–109.

[112] JAKOBOVIĆ, D., AND BUDIN, L. Dynamic scheduling with genetic programming. In *Proceedings of the European Conference on Genetic Programming* (2006), Springer, pp. 73–84.

[113] JAKOBOVIĆ, D., JELENKOVIĆ, L., AND BUDIN, L. Genetic programming heuristics for multiple machine scheduling. In *Proceedings of the European Conference on Genetic Programming* (2007), Springer, pp. 321–330.

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

[115] JAYAMOHAN, M., 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.

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

[117] JIN, Y., WANG, H., CHUGH, T., GUO, D., AND MIETTINEN, K. Data-driven evolutionary optimization: An overview and case studies. *IEEE Transactions on Evolutionary Computation 23*, 3 (2018), 442–458.

[118] JONES, A., RABELO, L. C., AND SHARAWI, A. T. Survey of job shop scheduling techniques. *NISTIR, National Institute of Standards and Technology, Gaithersburg, MD* (1998).

[119] JORDAN, M. I., AND MITCHELL, T. M. Machine learning: Trends, perspectives, and prospects. *Science 349*, 6245 (2015), 255–260.

[120] JURADO, S., NEBOT, À., MUGICA, F., AND AVELLANA, N. Hybrid methodologies for electricity load forecasting: Entropy-based feature selection with machine learning and soft computing techniques. *Energy 86* (2015), 276–291.

[121] KABAN, A., OTHMAN, Z., AND ROHMAH, D. Comparison of dispatching rules in job-shop scheduling problem using simulation: a case study. *International Journal of Simulation Modelling 11*, 3 (2012), 129–140.

[122] KANET, J. J., AND LI, X. A weighted modified due date rule for sequencing to minimize weighted tardiness. *Journal of Scheduling 7*, 4 (2004), 261–276.

[123] KARUNAKARAN, D., MEI, Y., AND ZHANG, M. Multitasking genetic programming for stochastic team orienteering problem with time windows. In *Proceedings of the IEEE Symposium Series on Computational Intelligence* (2019), IEEE, pp. 1598–1605.

[124] KENNEDY, J. Particle swarm optimization. *Encyclopedia of Machine Learning* (2010), 760–766.

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

[126] KIM, M. H., MCKAY, R. I. B., HOAI, N. X., AND KIM, K. Operator self-adaptation in genetic programming. In *Proceedings of the European Conference on Genetic Programming* (2011), Springer, pp. 215–226.

[127] KOULAMAS, C. A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research 105*, 1 (1998), 66–71.

[128] KOZA, J. R. *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*, vol. 34. Stanford University, Department of Computer Science Stanford, CA, 1990.

[129] KOZA, J. R. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing 4*, 2 (1994), 87–112.

[130] KOZA, J. R., KEANE, M. A., STREETER, M. J., MYDLOWEC, W., YU, J., AND LANZA, G. *Genetic programming IV: Routine human-competitive machine intelligence*, vol. 5. Springer Science & Business Media, 2006.

[131] KOZA, J. R., AND POLI, R. Genetic programming. In *Search Methodologies*. Springer, 2005, pp. 127–164.

[132] KULLBACK, S. *Information theory and statistics*. Courier Corporation, 1997.

[133] LAMÉ, G., AND DIXON-WOODS, M. Using clinical simulation to study how to improve quality and safety in healthcare. *BMJ Simulation and Technology Enhanced Learning 6*, 2 (2020), 87–94.

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

[135] LAWLER, E. L., AND WOOD, D. E. Branch-and-bound methods: A survey. *Operations Research 14*, 4 (1966), 699–719.

[136] LENSEN, A., XUE, B., AND ZHANG, M. Particle swarm optimisation representations for simultaneous clustering and feature selection. In *Proceedings of the IEEE Symposium Series on Computational Intelligence* (2016), pp. 1–8.

[137] LEUSIN, M. E., FRAZZON, E. M., URIONA MALDONADO, M., KÜCK, M., AND FREITAG, M. Solving the job-shop scheduling problem in the industry 4.0 era. *Technologies 6*, 4 (2018), 107.

[138] LI, G., LIN, Q., AND GAO, W. Multifactorial optimization via explicit multipopulation evolutionary framework. *Information Sciences 512* (2020), 1555–1570.

[139] LI, H., ONG, Y.-S., GONG, M., AND WANG, Z. Evolutionary multitasking sparse reconstruction: Framework and case study. *IEEE Transactions on Evolutionary Computation 23*, 5 (2018), 733–747.

[140] LI, L., ZHANG, F., LIU, C., AND NIU, B. A hybrid artificial bee colony algorithm with bacterial foraging optimization. In *Proceedings of the IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems* (2015), IEEE, pp. 127–132.

[141] LIAN, Z., JIAO, B., AND GU, X. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. *Applied Mathematics and Computation 183*, 2 (2006), 1008–1017.

[142] LIANG, Z., ZHANG, J., FENG, L., AND ZHU, Z. A hybrid of genetic transform and hyper-rectangle search strategies for evolutionary multi-tasking. *Expert Systems with Applications 138* (2019), 112798.

[143] LIN, J., LIU, H.-L., TAN, K. C., AND GU, F. An effective knowledge transfer approach for multiobjective multitasking optimization. *IEEE Transactions on Cybernetics* (2020. Doi: 10.1109/TCYB.2020.2969025).

[144] LIN, T.-L., HORNG, S.-J., KAO, T.-W., CHEN, Y.-H., RUN, R.-S., CHEN, R.-J., LAI, J.-L., AND KUO, I.-H. An efficient job-shop scheduling algorithm based on particle swarm optimization. *Expert Systems with Applications 37*, 3 (2010), 2629–2636.

[145] LIPOWSKI, A., AND LIPOWSKA, D. Roulette-wheel selection via stochastic acceptance. *Physica A: Statistical Mechanics and its Applications 391*, 6 (2012), 2193–2196.

[146] LIU, D., HUANG, S., AND ZHONG, J. Surrogate-assisted multitasking memetic algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2018), pp. 1–8.

[147] LIU, D., JIANG, T., WANG, Y., MIAO, R., SHAN, F., AND LI, Z. Clearness of operating field: a surrogate for surgical skills on in vivo clinical data. *International Journal of Computer Assisted Radiology and Surgery 15*, 11 (2020), 1817–1824.

[148] LIU, H., AND MOTODA, H. *Feature extraction, construction and selection: A data mining perspective*, vol. 453. Springer Science & Business Media, 1998.

[149] LIU, Y., WANG, L., WANG, X. V., XU, X., AND ZHANG, L. Scheduling in cloud manufacturing: state-of-the-art and research challenges. *International Journal of Production Research 57*, 15-16 (2019), 4854–4879.

[150] MANNE, A. S. On the job-shop scheduling problem. *Operations Research 8*, 2 (1960), 219–223.

[151] MCPHEE, N. F., DRAMDAHL, M. K., AND DONATUCCI, D. Impact of crossover bias in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2015), pp. 1079–1086.

[152] MEI, Y., NGUYEN, S., XUE, B., AND ZHANG, M. An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. *IEEE Transactions on Emerging Topics in Computational Intelligence 1*, 5 (2017), 339–353.

[153] MEI, Y., NGUYEN, S., AND ZHANG, M. Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling. In *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning* (2017), Springer, pp. 435–447.

[154] MEI, Y., NGUYEN, S., AND ZHANG, M. Evolving time-invariant dispatching rules in job shop scheduling with genetic programming. In *Proceedings of the European Conference on Genetic Programming* (2017), Springer, pp. 147–163.

[155] MEI, Y., ZHANG, M., AND NYUGEN, S. Feature selection in evolving job shop dispatching rules with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2016), ACM, pp. 365–372.

[156] MILLER, J. F., AND HARDING, S. L. Cartesian genetic programming. In *Proceedings of the Conference on Genetic and Evolutionary Computation* (2008), pp. 2701–2726.

[157] MIN, A. T. W., ONG, Y.-S., GUPTA, A., AND GOH, C.-K. Multiproblem surrogates: transfer evolutionary multiobjective optimiza-

tion of computationally expensive problems. *IEEE Transactions on Evolutionary Computation 23*, 1 (2017), 15–28.

[158] MIYASHITA, K. Job-shop scheduling with genetic programming. In *Proceedings of the Conference on Genetic and Evolutionary Computation* (2000), Morgan Kaufmann Publishers Inc., pp. 505–512.

[159] MOHAN, J., LANKA, K., AND RAO, A. N. A review of dynamic job shop scheduling techniques. *Procedia Manufacturing 30* (2019), 34–39.

[160] MOHRI, M., ROSTAMIZADEH, A., AND TALWALKAR, A. *Foundations of machine learning*. MIT press, 2018.

[161] NADAFZADEH, M., MEHDIZADEH, S. A., AND SOLTANIKAZEMI, M. Development of computer vision system to predict peroxidase and polyphenol oxidase enzymes to evaluate the process of banana peel browning using genetic programming modeling. *Scientia Horticulturae 231* (2018), 201–209.

[162] NAG, K., AND PAL, N. R. A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification. *IEEE Transactions on Cybernetics 46*, 2 (2016), 499–510.

[163] NESHATIAN, K., AND ZHANG, M. Unsupervised elimination of redundant features using genetic programming. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence* (2009), Springer, pp. 432–442.

[164] NGUYEN, Q. U., PHAM, T. A., NGUYEN, X. H., AND MCDERMOTT, J. Subtree semantic geometric crossover for genetic programming. *Genetic Programming and Evolvable Machines 17*, 1 (2016), 25–53.

[165] NGUYEN, S., MEI, Y., XUE, B., AND ZHANG, M. A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation 27*, 3 (2019), 467–496.

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

[167] NGUYEN, S., ZHANG, M., ALAHAKOON, D., AND TAN, K. C. Visualizing the evolution of computer programs for genetic programming. *IEEE Computational Intelligence Magazine 13*, 4 (2018), 77–94.

[168] NGUYEN, S., ZHANG, M., ALAHAKOON, D., AND TAN, K. C. People-centric evolutionary system for dynamic production scheduling. *IEEE Transactions on Cybernetics* (2019).

[169] 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 *IEEE Congress on Evolutionary Computation* (2012), pp. 1–8.

[170] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation 17*, 5 (2013), 621–639.

[171] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Hybrid evolutionary computation methods for quay crane scheduling problems. *Computers & Operations Research 40*, 8 (2013), 2083–2093.

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

[173] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Genetic programming for evolving due-date assignment models in job shop environments. *Evolutionary Computation 22*, 1 (2014), 105–138.

[174] NGUYEN, S., ZHANG, M., JOHNSTON, M., AND TAN, K. C. Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning* (2014), Springer, pp. 656–667.

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

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

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

[178] NIEHAUS, J., AND BANZHAF, W. Adaption of operator probabilities in genetic programming. In *Proceedings of the European Conference on Genetic Programming* (2001), pp. 325–336.

[179] NIU, B., ZHANG, F., LI, L., AND WU, L. Particle swarm optimization for yard truck scheduling in container terminal with a cooperative strategy. In *Intelligent and Evolutionary Systems*. Springer, 2017, pp. 333–346.

[180] NOWICKI, E., AND SMUTNICKI, C. A fast taboo search algorithm for the job shop problem. *Management Science 42*, 6 (1996), 797–813.

[181] OLTEAN, M., AND GROSAN, C. A comparison of several linear genetic programming techniques. *Complex Systems 14*, 4 (2003), 285–314.

[182] ONG, Y.-S., AND GUPTA, A. Evolutionary multitasking: a computer science view of cognitive multitasking. *Cognitive Computation 8*, 2 (2016), 125–142.

[183] ONG, Y.-S., ZHOU, Z., AND LIM, D. Curse and blessing of uncertainty in evolutionary algorithm using approximation. In *Proceedings of the IEEE International Conference on Evolutionary Computation* (2006), IEEE, pp. 2928–2935.

[184] O'REILLY, U.-M., AND OPPACHER, F. Program search with a hierarchical variable length representation: Genetic programming, simulated annealing and hill climbing. In *International Conference on Parallel Problem Solving from Nature* (1994), Springer, pp. 397–406.

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

[186] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., AND ZHANG, M. Evolutionary multitask optimisation for dynamic job shop scheduling using niched genetic programming. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence* (2018), Springer, pp. 739–751.

[187] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., AND ZHANG, M. Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling. In *Proceedings of the European Conference on Genetic Programming* (2018), Springer, pp. 253–270.

[188] PARK, J., MEI, Y., NGUYEN, S., CHEN, G., AND ZHANG, M. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing 63* (2018), 72–86.

[189] PARK, J., AND SANDBERG, I. W. Universal approximation using radial-basis-function networks. *Neural Computation 3*, 2 (1991), 246–257.

[190] PATERSON, N. R., AND LIVESEY, M. Distinguishing genotype and phenotype in genetic programming. *Late Breaking Papers at the Genetic Programming* (1996), 141–150.

[191] PAWLAK, T. P., AND KRAWIEC, K. Synthesis of constraints for mathematical programming with one-class genetic programming. *IEEE Transactions on Evolutionary Computation 23*, 1 (2018), 117–129.

[192] PEARL, J. *Heuristics - intelligent search strategies for computer problem solving*. Addison-Wesley, 1984.

[193] PENG, C., WU, G., LIAO, T. W., AND WANG, H. Research on multi-agent genetic algorithm based on tabu search for the job shop scheduling problem. *PloS one 14*, 9 (2019), e0223182.

[194] PETERSON, L. E. K-nearest neighbor. *Scholarpedia 4*, 2 (2009), 1883.

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

[196] PEZZELLA, F., MORGANTI, G., AND CIASCHETTI, G. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research 35*, 10 (2008), 3202–3212.

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

[198] PILLAY, N., AND BANZHAF, W. A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In *Proceedings of the Portuguese Conference on Aritficial Intelligence* (2007), pp. 223–234.

[199] PINEDO, M. *Planning and scheduling in manufacturing and services*. Springer, 2005.

[200] PINEDO, M. *Scheduling*, vol. 29. Springer, 2012.

[201] POLI, R. Evolution of graph-like programs with parallel distributed genetic programming. In *Proceedings of the International Conference on Genetic Algorithms* (1997), pp. 346–353.

[202] POLI, R., LANGDON, W. B., MCPHEE, N. F., AND KOZA, J. R. *A field guide to genetic programming*. Lulu. com, 2008.

[203] POLI, R., AND MCPHEE, N. F. General schema theory for genetic programming with subtree-swapping crossover: Part I. *Evolutionary Computation 11*, 1 (2003), 53–66.

[204] POLI, R., AND MCPHEE, N. F. General schema theory for genetic programming with subtree-swapping crossover: Part II. *Evolutionary Computation 11*, 2 (2003), 169–206.

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

[206] 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.

[207] RAQUEL, C., AND YAO, X. Dynamic multi-objective optimization: a survey of the state-of-the-art. In *Evolutionary computation for dynamic optimization problems*. Springer, 2013, pp. 85–106.

[208] RASMUSSEN, C. E. Gaussian processes in machine learning. In *Summer School on Machine Learning* (2003), Springer, pp. 63–71.

[209] RUSSELL, S. J., AND NORVIG, P. *Artificial intelligence - A modern approach*. Pearson Education, 2010.

[210] SAIDI, R., BOUAGUEL, W., AND ESSOUSSI, N. Hybrid feature selection method based on the genetic algorithm and pearson correlation coefficient. In *Machine Learning Paradigms: Theory and Application*. 2019, pp. 3–24.

[211] SAYED, G. I., HASSANIEN, A. E., AND AZAR, A. T. Feature selection via a novel chaotic crow search algorithm. *Neural Computing and Applications 31*, 1 (2019), 171–188.

[212] 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 50*, 15 (2012), 4255–4270.

[213] SIMON, F. Y.-P., ET AL. Integer linear programming neural networks for job-shop scheduling. In *Proceedings of the IEEE International Conference on Neural Networks* (1988), IEEE, pp. 341–348.

[214] SLOAN, T. W. 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.

[215] SONG, H.-B., AND LIN, J. A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times. *Swarm and Evolutionary Computation 60* (2021), 100807.

[216] SOTSKOV, Y. N., AND SHAKHLEVICH, N. V. NP-hardness of shop-scheduling problems with three jobs. *Discrete Applied Mathematics 59*, 3 (1995), 237–266.

[217] SUN, X., GONG, D., JIN, Y., AND CHEN, S. A new surrogate-assisted interactive genetic algorithm with weighted semisupervised learning. *IEEE Transactions on Cybernetics 43*, 2 (2013), 685–698.

[218] TAN, B., MA, H., AND MEI, Y. A hybrid genetic programming hyper-heuristic approach for online two-level resource allocation in container-based clouds. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2019), IEEE, pp. 2681–2688.

[219] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering 54*, 3 (2008), 453–473.

[220] TAY, J. C., AND HO, N. B. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering 54*, 3 (2008), 453–473.

[221] TERAMOTO, K., MORINAGA, E., WAKAMATSU, H., AND ARAI, E. A neighborhood limitation method for job-shop scheduling based on simulated annealing. *Transactions of the Institute of Systems, Control and Information Engineers 33*, 6 (2020), 171–181.

[222] UY, N. Q., HOAI, N. X., O'NEILL, M., MCKAY, R. I., AND LÓPEZ, E. G. Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genetic Programming and Evolvable Machines 12*, 2 (2011), 91–119.

[223] UYSAL, A. K. An improved global feature selection scheme for text classification. *Expert Systems with Applications 43* (2016), 82–92.

[224] VAN BREEDAM, A. Improvement heuristics for the vehicle routing problem based on simulated annealing. *European Journal of Operational Research 86*, 3 (1995), 480–490.

[225] VAN LAARHOVEN, P. J., AND AARTS, E. H. Simulated annealing. In *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

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

[227] WANG, S., MEI, Y., AND ZHANG, M. Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), pp. 1093–1101.

[228] XIE, H., ZHANG, M., AND ANDREAE, P. An analysis of depth of crossover points in tree-based genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2007), pp. 4561–4568.

[229] XIONG, J., XING, L.-N., AND CHEN, Y.-W. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. *International Journal of Production Economics 141*, 1 (2013), 112–126.

[230] XU, M., ZHANG, F., MEI, Y., AND ZHANG, M. Genetic programming with archive for dynamic flexible job shop scheduling. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2021), IEEE, pp. 2117–2124.

[231] XUE, B., ZHANG, M., BROWNE, W. N., AND YAO, X. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions Evolutionary Computation 20*, 4 (2016), 606–626.

[232] YIN, W.-J., LIU, M., AND WU, C. Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2003), vol. 2, IEEE, pp. 1050–1055.

[233] YSKA, D., MEI, Y., AND ZHANG, M. Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In *European Conference on Genetic Programming* (2018), Springer, pp. 306–321.

[234] YUAN, Y., ONG, Y.-S., GUPTA, A., TAN, P. S., AND XU, H. Evolutionary multitasking in permutation-based combinatorial optimization problems: Realization with tsp, qap, lop, and jsp. In *Proceedings of the IEEE Region 10 Conference* (2016), IEEE, pp. 3157–3164.

[235] ZARROUK, R., BENNOUR, I. E., AND JEMAI, A. A two-level particle swarm optimization algorithm for the flexible job shop scheduling problem. *Swarm Intelligence 13*, 2 (2019), 145–168.

[236] ZHANG, F., MEI, Y., NGUYEN, S., TAN, K. C., AND ZHANG, M. Multitask genetic programming-based generative hyper-heuristics: A case study in dynamic scheduling. *IEEE Transactions on Cybernetics* (2021, Doi: 10.1109/TCYB.2021.3065340).

[237] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization* (2020), Springer, pp. 214–230.

[238] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Guided subtree selection for genetic operators in genetic programming for dynamic flexible job shop scheduling. In *Proceedings of the European Conference on Genetic Programming* (2020), Springer, pp. 262–278.

[239] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. A preliminary approach to evolutionary multitasking for dynamic flexible job shop scheduling via genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2020), ACM, pp. 107–108.

[240] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Collaborative multi-fidelity based surrogate models for genetic programming in dynamic flexible job shop scheduling. *IEEE Transactions on Cybernetics* (2020. Doi: 10.1109/TCYB.2021.3050141).

[241] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Correlation coefficient-based recombinative guidance for genetic programming hyperheuristics in dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation 25*, 3 (2021), 552–566.

[242] ZHANG, F., MEI, Y., NGUYEN, S., AND ZHANG, M. Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job shop scheduling. *IEEE Transactions on Cybernetics 51*, 4 (2021), 1797–1811.

[243] ZHANG, F., MEI, Y., NGUYEN, S., ZHANG, M., AND TAN, K. C. Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. *IEEE Transactions on Evolutionary Computation 25*, 4 (2021), 651–665.

[244] ZHANG, F., MEI, Y., AND ZHANG, M. Genetic programming with multi-tree representation for dynamic flexible job shop scheduling. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence* (2018), Springer, pp. 472–484.

[245] ZHANG, F., MEI, Y., AND ZHANG, M. Surrogate-assisted genetic programming for dynamic flexible job shop scheduling. In *Proceedings of the Australasian Joint Conference on Artificial Intelligence* (2018), Springer, pp. 766–772.

[246] ZHANG, F., MEI, Y., AND ZHANG, M. Can stochastic dispatching rules evolved by genetic programming hyper-heuristics help in dynamic flexible job shop scheduling? In *Proceedings of the IEEE Congress on Evolutionary Computation* (2019), IEEE, pp. 41–48.

[247] ZHANG, F., MEI, Y., AND ZHANG, M. Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation* (2019), IEEE, pp. 1366–1373.

[248] ZHANG, F., MEI, Y., AND ZHANG, M. A new representation in genetic programming for evolving dispatching rules for dynamic flexible job shop scheduling. In *Proceedings of the European Conference on Evolutionary Computation in Combinatorial Optimization* (2019), Springer, pp. 33–49.

[249] ZHANG, F., MEI, Y., AND ZHANG, M. A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference* (2019), ACM, pp. 347–355.

[250] ZHANG, G., HU, Y., SUN, J., AND ZHANG, W. An improved genetic algorithm for the flexible job shop scheduling problem with multiple time constraints. *Swarm and Evolutionary Computation 54* (2020), 100664.

[251] ZHANG, J., DING, G., ZOU, Y., QIN, S., AND FU, J. Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing 30*, 4 (2019), 1809–1830.

[252] ZHANG, J., JIE, J., WANG, W., AND XU, X. A hybrid particle swarm optimisation for multi-objective flexible job-shop scheduling problem with dual-resources constrained. *International Journal of Computing Science and Mathematics 8*, 6 (2017), 526–532.

[253] ZHANG, M., GAO, X., AND LOU, W. A new crossover operator in genetic programming for object classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B 37*, 5 (2007), 1332–1343.

[254] ZHANG, X., TIAN, Y., AND JIN, Y. A knee point-driven evolutionary algorithm for many-objective optimization. *IEEE Transactions Evolutionary Computation 19*, 6 (2015), 761–776.

[255] ZHONG, J., FENG, L., CAI, W., AND ONG, Y.-S. Multifactorial genetic programming for symbolic regression problems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2018).

[256] ZHOU, L., FENG, L., TAN, K. C., ZHONG, J., ZHU, Z., LIU, K., AND CHEN, C. Toward adaptive knowledge transfer in multifactorial evolutionary computation. *IEEE Transactions on Cybernetics* (2020, Doi: 10.1109/TCYB.2020.2974100).

[257] ZHOU, L., FENG, L., ZHONG, J., ONG, Y.-S., ZHU, Z., AND SHA, E. Evolutionary multitasking in combinatorial search spaces: A case study in capacitated vehicle routing problem. In *Proceedings of the IEEE Symposium Series on Computational Intelligence* (2016), IEEE, pp. 1–8.

[258] ZHOU, Q., JIANG, P., SHAO, X., HU, J., CAO, L., AND WAN, L. A variable fidelity information fusion method based on radial basis function. *Advanced Engineering Informatics 32* (2017), 26–39.

[259] ZHOU, Y., YANG, J.-J., AND ZHENG, L.-Y. Hyper-heuristic coevolution of machine assignment and job sequencing rules for multi-objective dynamic flexible job shop scheduling. *IEEE Access 7* (2018), 68–88.

[260] 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.