Genetic Programming with Delayed Routing for Multi-Objective Dynamic Flexible Job Shop Scheduling

Binzi Xu 7151905016@vip.jiangnan.edu.cn
School of Electrical Engineering, Anhui Polytechnic University, Wuhu, 241000, PR China
School of IoT and Engineering, Jiangnan University, Wuxi, 214122, PR China
School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand

Yi Mei yi.mei@ecs.vuw.ac.nz
School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand

Yan Wang wangyan88@jiangnan.edu.cn
School of IoT and Engineering, Jiangnan University, Wuxi, 214122, PR China

Zhicheng Ji zcji@jiangnan.edu.cn
School of IoT and Engineering, Jiangnan University, Wuxi, 214122, PR China

Mengjie Zhang mengjie.zhang@ecs.vuw.ac.nz
School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand

Abstract

Dynamic Flexible Job Shop Scheduling (DFJSS) is an important and challenging problem, and can have multiple conflicting objectives. Genetic Programming Hyper-Heuristic (GPHH) is a promising approach to fast respond to the dynamic and unpredictable events in DFJSS. A GPHH algorithm evolves dispatching rules (DRs) that are used to make decisions during the scheduling process (i.e. the so-called heuristic template). In DFJSS, there are two kinds of scheduling decisions: the routing decision that allocates each operation to a machine to process it, and the sequencing decision that selects the next job to be processed by each idle machine. The traditional heuristic template makes both routing and sequencing decisions in a non-delay manner, which may have limitations in handling the dynamic environment. In this paper, we propose a novel heuristic template that delays the routing decisions rather than making them immediately. This way, all the decisions can be made under the latest and more accurate information. We propose three different delayed routing strategies, and automatically evolve the rules in the heuristic template by GPHH. We evaluate the newly proposed GPHH with Delayed Routing (GPHH-DR) on a multi-objective DFJSS that optimises the energy efficiency and mean tardiness. The experimental results show that GPHH-DR significantly outperformed the state-of-the-art GPHH methods. We further demonstrated the efficacy of the proposed heuristic template with delayed routing, which suggests the importance of delaying the routing decisions.

Keywords

dynamic flexible job shop scheduling, genetic programming, dispatching rule discovery, delayed routing, energy efficiency.

1 Introduction

In the past decades, lots of attention have been paid on flexible job shop scheduling (FJSS) (Jain and Meeran, 1999) with the rapid developments of economy and manufacturing technologies. In the real manufacturing industries, FJSS aims to process a set of jobs by a number of machines to optimise some objectives, such as the mean tardiness and energy efficiency. A good schedule can not only finish all the manufacturing tasks as soon as possible but also improve the quality of products, reduce the manufacturing cost and energy consumption. Hence, FJSS is an important problem for manufacturing industries.

As an NP-hard problem (Garey et al., 1976), FJSS has been extensively studied so far and many optimisation methods have been proposed to solve it properly. In the early days, researchers tried to find the optimal solution using exact methods, such as branch and bound (Brah and Hunsucker, 1991) and constraint programming (Pour et al., 2018). However, due to the NP-hardness of FJSS, these methods are unable to handle large scale practical problems. In this case, approximated optimisation approaches, such as genetic algorithm (Jensen, 2003), tabu search (Peng et al., 2015), and particle swarm optimisation (Karimi-Nasab et al., 2015), have shown promise to solve real-world FJSS problems.

FJSS can have multiple conflicting objectives. So far the tardiness-related objectives have been commonly considered (Romero-Silva et al., 2018). However, as sustainability is becoming increasingly important (May et al., 2017), energy efficiency is attracting more and more attention. In this paper, we focus on the multi-objective FJSS that optimises the mean tardiness (Tay and Ho, 2008; Zhang et al., 2018) and energy efficiency (Jose et al., 2019; Dai et al., 2019; Gahm et al., 2016; Lei et al., 2018), since both objectives are very important considerations in practice. Mean tardiness is a widely used objective in the FJSS that directly reflects production efficiency of a scheduling scheme, while energy efficiency demonstrates the efficiency of power utilisation (i.e. the so-called green degree of production).

In the real world, the scheduling environment is often dynamic with unpredictable events (e.g. job arrivals). In Dynamic FJSS (DFJSS), it is important to effectively and efficiently react to these unknown events (Nguyen et al., 2014; Shen and Yao, 2015). However, the traditional optimisation approaches, such as mathematical programming and genetic algorithm, cannot efficiently handle the dynamic environment due to their high computational cost.

Dispatching rules (DRs) have been shown as a promising heuristic for DFJSS since they can make effective scheduling decisions in a very short time. DRs essentially are priority functions that calculate the priority of each scheduling object (e.g. a job in the candidate operation set or a machine to be allocated) based on the temporal job shop state. Then, scheduling decisions could be made based on these priorities. Calculating the priorities based on DRs is very time-efficient. Hence, DRs can satisfy the special requirement of DFJSS.

There have been a variety of DRs designed manually, such as first-in-first-out (FIFO), shortest-processing-time (SPT) and other more advanced DRs (Nguyen et al., 2016). However, the manually designed DRs are hard to capture the complex interactions between the job and machine features in the job shop, and thus their performance is limited. Designing an effective DR manually requires much domain knowledge and many rounds of trial-and-error.

Recently, Genetic Programming Hyper-Heuristic (GPHH) has been widely used to automatically evolve DRs for solving dynamic problems (Hildebrandt and Branke,

2015), including DFJSS (Zhang et al., 2018; Nguyen et al., 2015). GPHH has also achieved much better performance than the manually designed DRs for the multi-objective scheduling problems (Tay and Ho, 2008; Nguyen et al., 2014).

To solve DFJSS with GPHH, there are two kinds of scheduling decisions to be made (Yska et al., 2018): routing decision and sequencing decision. Making a routing decision means to allocate a job operation to a candidate machine, while making a sequencing decision means to select a job operation from the candidate set of an idle machine to process next.

For developing GPHH for DFJSS, it is critical to design the heuristic template, which is the framework that the evolved DRs operate in (Burke et al., 2009). The existing GPHH (Nguyen et al., 2014; Zhang et al., 2018) employed a non-delay-based heuristic template. In this heuristic template, a routing decision is made as soon as a job operation becomes ready, and a sequencing decision is made as soon as a machine becomes idle. However, only the sequencing decision can take effect immediately. After the routing decision is made, the job still needs to wait in the candidate set of the allocated machine until the machine becomes idle and chooses this job to process next. The environment can change while the job is waiting, making the original allocation not desirable anymore (another machine becoming much better to process the job in the updated environment). The issue can be more serious when considering multiple conflicting objectives simultaneously.

To address the above issue, we propose a delayed routing strategy to postpone the routing decision to the last minute. In the delayed routing strategy, when a job becomes ready, it is placed in a pool without being allocated to any candidate machine. Once a machine becomes idle, it will select a subset of jobs from the pool for the sequencing decision. This way, we only consider the allocation of a job to the candidate machines when it can be processed immediately, and both the routing and sequencing decisions can consider the latest information during the dynamic scheduling process. We propose a novel heuristic template based on the delayed routing strategy to improve the performance of GPHH.

Although the delayed routing strategy can handle the dynamic environment better, it raises an extra challenge that the pool can consist of a large number of jobs, as it is essentially the union of all the machines' candidate sets. For each idle machine, it is challenging to decide the subset of the jobs for the sequencing decision, and it is extremely difficult to select the proper job to process next from such a large number of candidate jobs. To tackle this challenge, we propose three different delayed routing strategies to smartly select the subset of jobs for the idle machine.

The overall goal of the paper is to develop a novel GPHH algorithm with Delayed Routing (GPHH-DR) for solving the multi-objective DFJSS more effectively. More specifically, we have the following research objectives.

- Develop a novel heuristic template, which is a scheduling process with the delayed routing strategy.
- Propose three different delayed routing strategies to select the subset of ready jobs for the idle machine to make the sequencing decision.
- Develop the GPHH-DR to evolve the rules used in the novel heuristic template.
- Evaluate the proposed GPHH-DR on multi-objective DFJSS and analyse the evolved rules.

The rest of this paper is organised as follows. Section 2 gives the background of this paper, including a brief description of DFJSS, GPHH, and related work. The proposed GPHH-DR, including the novel heuristic template and three delayed routing strategies, is described in Section 3. Section 4 gives the experiment design and Section 5 gives the results and discussions. The conclusions and future work are shown in Section 6.

2 Background

This section first briefly describes DFJSS, then introduces GPHH, and finally gives the related work.

2.1 Problem Description

In DFJSS, there are a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ in the job shop. Each machine M_k has a standby power consumption ϵ_k, indicating the energy consumption if machine is idle for one time unit. The jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ arrive at the shop floor in real time. Each job J_i has an arrival time a_i and a due date d_i. A job J_i consists of a sequence of operations $(O_{i1}, \ldots, O_{in_i})$, where n_i is the number of operations of J_i. Each operation O_{ij} has a set of candidate machines $\mathcal{M}_{ij} \subseteq \mathcal{M}$, which contains the machines that can process O_{ij}. If O_{ij} is processed by machine $M_k \in \mathcal{M}_{ij}$, the processing time is p_{ijk}, and the energy consumption is e_{ijk}. Let $t(O_{ij})$ be the start time of O_{ij} and $t(M_k, q)$ be the start time of the qth operation processed by M_k. The problem is to process the jobs with the machines subject to the following constraints.

- A job is unseen (i.e. considered in the scheduling process) until it arrives.
- An operation cannot start being processed until its precedent operations have been completed. The first operation of a job cannot be processed until the job arrives at the shop floor.

$$t(O_{i1}) \geq a_i, \forall\, i = 1, \ldots, n \tag{1}$$

$$t(O_{ij}) + \sum_{k=1}^{m} p_{ijk} x_{ijk} \leq t(O_{i,j+1}), \forall\, i = 1, \ldots, N,\ j = 1, \ldots, n_i - 1 \tag{2}$$

  where the binary decision variable x_{ijk} equals 1 if O_{ij} is processed by M_k, and 0 otherwise.

- Each machine can process at most one job operation at a time.

$$t(M_k, q) + \sum_{i=1}^{n} \sum_{j=1}^{n_i} p_{ijk} y_{ijkq} \leq t(M_k, q+1), \forall\, k = 1, \ldots, m \tag{3}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n_i} y_{ijkq} = 1, \forall\, k = 1, \ldots, m,\ q = 1 \ldots, Q_k \tag{4}$$

  where the binary decision variable y_{ijkq} takes 1 if O_{ij} is the qth operation processed by M_k, and 0 otherwise. Q_k is the number of operations processed by M_k. The constraint (4) means that for each position of each machine, there is exactly one operation.

- Each operation must be processed by one of its candidate machines.

$$\sum_{k=1}^{m} x_{ijk} = 1, \forall\, i = 1, \ldots, n,\ j = 1, \ldots, n_i,\ M_k \in \mathcal{M}_{ij} \tag{5}$$

$$\sum_{k=1}^{m} x_{ijk} = 0, \forall\, i = 1, \ldots, n,\ j = 1, \ldots, n_i,\ M_k \notin \mathcal{M}_{ij} \tag{6}$$

$$\sum_{q=1}^{Q_k} y_{ijkq} = x_{ijk}, \forall\, i = 1, \ldots, n,\ j = 1, \ldots, n_i,\ k = 1, \ldots, m \tag{7}$$

$$t(O_{ij}) = \sum_{k=1}^{m} \sum_{q=1}^{Q_k} y_{ijkq} \cdot t(M_k, q) \tag{8}$$

  The constraint (7) indicates that if O_{ij} is processed by M_k (i.e. $x_{ijk} = 1$), it must be processed at some position. The constraint (8) is the matching constraint between $t(O_{ij})$ and $t(M_k, q)$.

- Once a machine starts processing an operation, it cannot be interrupted until the operation is completed.

The DFJSS considers two objective functions: (1) minimising the mean tardiness (MT), and (2) minimising the ratio of the idle energy consumption (REC_{idle}). The objective functions are calculated as follows.

$$\min \quad MT = \frac{1}{n} \sum_{i=1}^{n} \max\{c_i - d_i, 0\}, \tag{9}$$

$$\min \quad REC_{idle} = \frac{EC_{idle}}{EC_{idle} + \sum_{i=1}^{n} \sum_{j=1}^{n_i} \sum_{k=1}^{m} e_{ijk} x_{ijk}}, \tag{10}$$

where c_i is the completion time of the job J_i. The idle energy consumption EC_{idle} is calculated as

$$EC_{idle} = \sum_{k=1}^{m} \left(c_{\max} - \sum_{i=1}^{n} \sum_{j=1}^{n_i} p_{ijk} x_{ijk} \right) \epsilon_k, \tag{11}$$

where $c_{\max} = \max_{i=\{1,\ldots,n\}} c_i$ is the completion time of the last job.

Equation (10) looks similar to machine utilisation, however these two are different. REC_{idle} is calculated as the ratio of effective energy consumption to total energy consumption (Wang et al., 2013; He et al., 2015). Therefore, this objective is related to machine utilisation and the choice of idle machines. In addition to keeping machines busy, the standby power consumption ϵ_k also has an impact on energy efficiency. To improve energy efficiency, it is better to always choose the machine with small ϵ_k to be idle, especially when the production task is not heavy.

It can be seen that the mean tardiness MT depends on the completion time of the jobs. To minimise MT, the schedule tends to allocate the operations to the machines with smaller p_{ijk}. On the other hand, to minimise EC_{idle}, the routing decision tends to allocate the operations to the machines with larger ϵ_k. Therefore, the two objectives are potentially conflicting due to the potential conflict between p_{ijk} and ϵ_k of the machines.

The scheduling process consists of two kinds of decisions which are both made by the heuristic template and dispatching rules when using GPHH to generate scheduling schemes. The *routing* decision allocates an operation to a candidate operation set of the candidate machine (i.e. deciding the x_{ijk} values) when it becomes ready, and the *sequencing* decision selects the next job operation from the candidate operation set of an idle machine to process next (i.e. deciding the y_{ijkq} values). When an operation is finished by one of its candidate machines, the next operation will become ready if this finished operation is not the last operation of this job. A typical scheduling process where the routing and sequencing decisions collaborate with each other is illustrated in Figure 1.

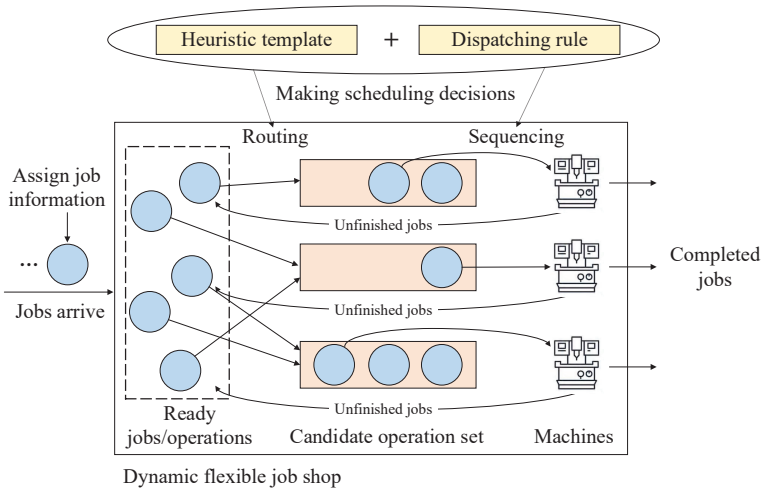

Figure 1: A typical DFJSS process.

2.2 Genetic Programming Hyper-Heuristic

GP (Banzhaf et al., 1998) is a type of evolutionary algorithm that evolves a population of computer programs. As a hyper-heuristic, GPHH has been successfully applied to evolve DRs for a variety of complex and dynamic problems (Burke et al., 2009; Nguyen et al., 2019). The general framework of GPHH is given in Algorithm 1. Unlike traditional optimisation algorithms, an individual in GPHH is a heuristic rather than a solution. To evaluate a heuristic during GPHH (line 3), a *heuristic template* is required.

Algorithm 1: The general framework of GPHH

```
1 Randomly initialise a population of DRs;
2 while stopping criteria not met do
3 |   Evaluate the DRs in the population;
4 |   Generate a new population using elitism, crossover, mutation and
  |     reproduction operators;
5 end
6 return the best heuristic;
```

A heuristic template is essentially an execution engine/simulator that makes

scheduling decisions and generates schedules (Burke et al., 2009). Given a problem instance and a heuristic, the heuristic template generates the corresponding solution to the problem instance. In GPHH, the fitness of a heuristic is normally defined as the average quality of the solutions generated by the heuristic on a set of training problem instances.

To represent a heuristic, the tree-based representation is commonly used. Specifically, a heuristic is represented as a tree in the simulation system, which is essentially a priority function. At each decision situation, the heuristic calculates the priority value of each candidate action (e.g. selecting the next job operation from the candidate operation set of an idle machine to process next), and takes the most prior action. Figure 2 shows two widely used manually designed routing and sequencing rules (Tay and Ho, 2008; Holthaus and Rajendran, 2000) as examples, where the terminals are the job shop features (e.g. WIQ stands for the total processing time of the operations waiting in the current machine's candidate operation set). The routing rule always chooses the machine with the least waiting time for routing while the sequencing rule chooses the job with minimal processing time for sequencing. GPHH typically uses the standard GP crossover and mutation operators such as the tree-swap crossover and tree-replacement mutation operators to generate new individuals.

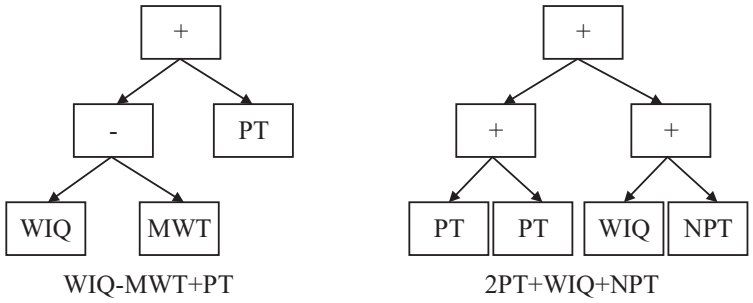

Figure 2: An example of the routing and sequencing rules for DFJSS.

For solving DFJSS with GPHH, the fitness of each individual (i.e. DR) is defined based on a set of training DFJSS instances. For each instance, the heuristic template, which is essentially the scheduling simulator, uses the DRs to make the routing and sequencing decisions at each decision situation during the simulation, and generates the schedule for the instance as shown in Figure 1. Then, during GPHH, the fitness of a DR is normally defined as the average objective value of the solutions generated on the training instances.

Based on the description by Burke et al. (2009), the development of GPHH consists of the following two key steps.

1. Develop an effective heuristic template that uses DRs to generate solutions.
2. Design the GP algorithm (e.g. representation, terminal and function sets, search mechanism) to evolve the heuristic.

Both steps are critical to the performance of GPHH. In this paper, we focus on improving the heuristic template by proposing three different delayed routing strategies.

2.3 Related Work

This section gives a literature review on recent related work. We firstly describe the different objectives considered in FJSS (e.g. mean tardiness, energy efficiency, etc.). Then, we introduce approaches for the dynamic FJSS, including non-DR and DR-based approaches. Finally, researches on GPHH for DFJSS are described, and their limitations are stated to motivate the study in this paper.

2.3.1 Different Objectives in FJSS

FJSS is one of the most famous problems in the field of optimisation, which is a static combinatorial optimisation problem that all jobs and related information are known at the very beginning (Nguyen et al., 2019). During the development of FJSS, many objectives have been considered.

As a practical issue, time related objectives are commonly used in the FJSS, including makespan (Wu and Sun, 2018; Gao and Pan, 2016; Zandieh et al., 2017), mean tardiness (Tay and Ho, 2008), and total flow time (Gao et al., 2018). These time related objectives are closely related to production efficiency, which is the top priority of manufacturing industries. These objectives highly depend on the choice of processing machines, and can effectively reflect the difference and performance of scheduling solutions. Considering the real situation, uncertainty (e.g. fuzzy processing time and fuzzy due date time) has also been introduced and made the FJSS more difficult in which the objectives are fuzzy makespan (Sun et al., 2019; Gao et al., 2016), fuzzy tardiness (Niroomand et al., 2016), etc. There are two main kinds of models to describe the uncertain number (Sun et al., 2019): fuzzy interval and fuzzy number. This uncertainty has extended the traditional FJSS to fuzzy FJSS in which the operators of optimisation algorithms should be improved (Lin, 2019).

Other manufacture related objectives are also considered in the previous work, such as production cost (Wang et al., 2018a), total energy cost (TEC) (Mokhtari and Hasani, 2017; Meng et al., 2019; Wang et al., 2018b), energy efficiency (Tang et al., 2016), and workload (Zhu and Zhou, 2020). Energy related objectives have attracted increasing attention recently because of the trend of green manufacturing and sustainable manufacturing (Gahm et al., 2016).

There are two main directions for energy efficiency oriented scheduling: (1) algorithm improvement, and (2) energy-efficiency specified methods. The former treats energy related objectives as normal objectives, and focuses on improving the performance of optimisation algorithms (Lei et al., 2018; Dai et al., 2019; Zhang and Chiong, 2016; Wang et al., 2018a; Rubaiee and Yildirim, 2019). The latter considers the complex characteristics of the energy consumption in the scheduling problem, and introduces some specific situations and requirements into the problem, e.g., turn off/turn on strategy (Liu et al., 2016; Wu and Sun, 2018) and speed level selection of machines (Meng et al., 2019; Salido et al., 2016). As pointed out by Zhang et al. (2016), processing planning also impacts the energy efficiency of scheduling.

It is obvious that most studies on energy efficiency oriented scheduling are solved by heuristic algorithms. The main drawback of heuristic algorithms is that the solutions they generated are not reusable, which means it is hard for them to deal with the dynamic environment (i.e. the unpredictable events). This is the reason why we consider to evolve DRs by GPHH in this paper.

2.3.2 Optimisation Approaches for Dynamic FJSS

Dynamic flexible JSS (DFJSS) is a more practical version of FJSS that considers dynamic and unpredictable events, which include random job arrivals, machine breakdowns, etc. In the DFJSS, information about a job is only available when this job arrives. Currently, there are two main directions on solving DFJSS: non-DR (e.g. heuristic algorithms) and DR-based approaches.

Heuristic algorithms are widely used non-DR based approaches that have already shown their advantages on solving FJSS. When it comes to DFJSS, using heuristic algorithm based rescheduling mechanisms (Ouelhadj and Petrovic, 2009) is a common way for heuristic algorithms to handle the dynamic environment of DFJSS, including completely reactive (Nie et al., 2013), predictive-reactive (Shen and Yao, 2015; Shahgholi Zadeh et al., 2019; Shoraneh and Kazuhiro, 2019) and pro-active scheduling (Mokhtari and Dadgar, 2015). Among them, predictive–reactive scheduling is the most popular method that can get a balance between computing time and scheduling performance.

Heuristic algorithms with rescheduling mechanism have two main problems on reacting dynamic and stochastic manufacturing circumstances. First, the stability is a key measure for rescheduling, which not only influences the implementation of rescheduling, but also has impacts on the practical optimisation objectives. Second, the high computational complexity of the heuristic algorithm makes it hard to obtain a suitable rescheduling solution within a limited time. Although many efforts have been done to reduce the difficulty of rescheduling (e.g. partial rescheduling method), they may make the overall scheduling solution less effective.

On the other hand, the capability of fast responding to dynamic and unknown changes in the DJSSP makes DR-based approaches more and more popular these days. As a priority function, DRs can make scheduling decisions in a very short time and generate a feasible scheduling solution consequently as they do not have the process of iterative improvement during scheduling. Another key advantage of DRs is that they can be applied to different job shops with little extra effort.

There are three categories of DRs in the literature. The research of DRs started from simple DRs (Panwalkar and Iskander, 1977), such as FIFO, SPT, EDD (Earliest Due Date), etc. These simple DRs only consider information of jobs (e.g. processing time and due date). Because of their simplicity and poor performance, combinations of simple DRs have been developed, which take different processing cases and situations into consideration, and select specific DR to make scheduling decisions (Vepsalainen and Morton, 1987), such as FIFO/SI, FCFS*S, 2CLASS. The third stage/type is weighted priority indexes, in which more different information and processing states are considered.

However, it is hard to manually design a good DR because of the requirement of vast domain knowledge and deep understanding of the changing processing environment. Therefore, automated design of DRs with hyper-heuristics (HH) has been shown as an effective approach compared with manually designed DRs (Li et al., 2013; Tay and Ho, 2008; Nguyen et al., 2013a, 2019). Instead of searching in the solution search space, hyper-heuristic algorithms aim to find a heuristic, i.e. DR in this context, in the heuristic search space.

2.3.3 GPHH Methods for DFJSS

In the field of HH algorithms, there are two main methods: HH for heuristic selection and HH for the heuristic generation. Heuristic selection (Huang and Süer, 2015) aims

to design a decision making strategy to select existing DRs according to the current system state. On the other hand, the heuristic generation tries to design a new DR automatically based on given terminal set (Branke et al., 2016).

GPHH is a popular HH approach and has been widely used in many applications because of its flexible representation and excellent search ability. Existing works have used GPHH to automatically generate mutation operators (Hong et al., 2018), optimise the running time of software (Langdon and Harman, 2015), generate the motion feature descriptor in a feature extraction method and design diverse classifiers with selected features (Nag and Pal, 2016). According to the previous work, GPHH can not only solve optimisation problems, but also be embedded in the other search approaches to improve their performance.

GPHH is also a promising approach for scheduling problems, especially DFJSS. Nguyen et al. (Nguyen et al., 2018) made much efforts on using GPHH to solve JSSP. They tested three kinds of representations of DRs and found that the mixed representation performed the best (Nguyen et al., 2013a). Besides, they used GPHH to solve dynamic multi-objective JSS based on cooperative co-evolution (Nguyen et al., 2014). Iterative dispatching rules (IDRs) (Nguyen et al., 2013b), surrogate assisted method (Nguyen et al., 2017) and iterated local search-based GPHH (Nguyen et al., 2015) are also proposed to improve the performance of the evolved DRs from different aspects. Besides, combination schemes (Park et al., 2018), multi-tree representation (Zhang et al., 2018), terminal selection (Mei et al., 2017), hybrid algorithms (Pickardt et al., 2013) and heuristic template (Durasević et al., 2016; Durasević and Jakobović, 2018) are also key factors that influence the performance of GPHH. Among them, Durasević et al. (2016) firstly proposed the basic idea of delayed routing strategy but only focused on the problem with unrelated machines.

Most GPHH methods adopted the same heuristic template which deals with routing decision and sequencing decision separately. The routing decision is made and the job is assigned to the candidate operation set of this selected machine immediately when a job is ready. However, the assigned machine may still be busy, and the job has to wait to be processed until the machine becomes idle again. In other words, there may be a time gap between the routing decision and the actual start time of the job processing. The environment can change during this time gap, making the previously made routing decision less effective.

To address the above issue, this paper proposes a GPHH-DR with a novel heuristic template considering delayed routing strategies for solving DFJSS. The delayed routing strategies change the way to make scheduling decisions. More specifically, the routing decision is not made immediately after a job operation becomes ready. Instead, it is made at the moment when a sequencing decision needs to be made. If this operation is not chosen this time, its routing decision can be remade again when the next sequencing decision is about to be made. This novel heuristic template ensures that the routing decision always be made with the latest job shop state and the previous routing decisions can be changed as long as the operation has not been processed yet. These two advantages make the proposed GPHH-DR more suitable for the dynamic environment in DFJSS.

3 GPHH with Delayed Routing

The proposed GPHH-DR is based on a novel heuristic template, which employs the delayed routing. The tree-based representation and evolutionary operators of GPHH-DR are the same as the standard GPHH for DFJSS.

3.1 The Novel Heuristic Template with Delayed Routing Strategy

Figure 3 gives a brief illustration of the framework of the proposed heuristic template with delayed routing strategies, in which the main contribution and improvement of this proposed heuristic template are highlighted by red boxes. In this novel heuristic template, a ready job is sent to a pool rather than immediately sent to one of the candidate machines' candidate operation sets by a routing rule. When a machine becomes idle, the delayed routing policy is used to form a candidate operation set for this idle machine. Then, a sequencing rule is used to select a job from this candidate operation set to process next.

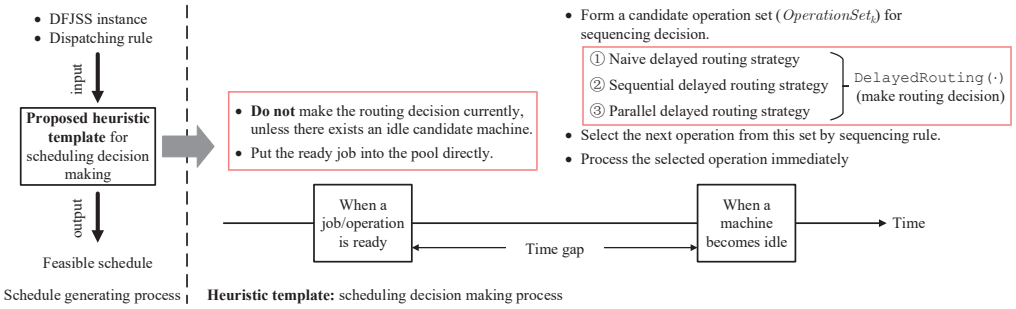

Figure 3: Framework of the proposed heuristic template with delayed routing strategy.

Algorithm 2 shows more details about the proposed heuristic template with delayed routing strategy. Initially, all the machines are idle and all the operations are not completed. Then, at each step, the algorithm identifies the earliest time when there exists operation that are ready to be processed (e.g. a new job arrives, or an operation is completed and its successive operation becomes ready), and there are idle machines to process these ready operations. Then, for each idle machine at that moment, it selects a subset of ready operations to form its candidate operation set (*batch delayed routing*), and further selects the next operation from the candidate operation set to be processed next (*sequencing*).

The main difference between the newly proposed heuristic template and the traditional heuristic template is that in the newly proposed heuristic template, when an operation becomes ready, it is not allocated to any machine's candidate operation set immediately. Instead, the candidate operation set of an idle machine is formed by selecting from all the ready operations before making the sequencing decision (line 6). The formation of $OperationSet_k$ involves a number of delayed routing decisions, i.e. whether each ready operation should be allocated to the idle machine or not.

3.2 Delayed Routing Strategy

A key issue in GPHH-DR is how to effectively make a number of delayed routing decisions simultaneously (line 6 of Algorithm 2), which leads to the candidate operation set $OperationSet_k$ of the idle machine M_k. Note that the delayed routing strategy has the following differences from the routing rule.

- A routing rule is to allocate a ready operation to a candidate machine (add an operation to an existing candidate operation set), while the delayed routing strategy is to form the entire candidate operation set of an idle machine for the sequencing decision.

Algorithm 2: The proposed heuristic template with delayed routing strategy

Input: A DFJSS instance and a dispatching rule (single-tree or multi-tree).
Output: A feasible schedule.

All machines are idle, all operations are not completed, schedule is empty;
while *there are operations not completed* **do**
  Go to the earliest time t when there exist ready operations and there are idle machines to process them;
  Get all the ready operations $\mathcal{O}^{(t)}$ and the idle machines $\mathcal{M}^{(t)}$ at time t;
  foreach $M_k \in \mathcal{M}^{(t)}$ **do**
    `// Make the delayed routing decisions simultaneously`
    $OperationSet_k = \texttt{DelayedRouting}(\mathcal{O}^{(t)}, M_k)$;
    `// Make the sequencing decision`
    Select the next operation from $OperationSet_k$ by the **sequencing rule**;
    Process the next operation by M_k, and add the processing to the schedule;
  end
end
return *the schedule;*

- After applying a routing rule, the ready operation is physically allocated to a candidate machine, and can no longer be changed. On the contrary, the "candidate operation set" formed by the delayed routing strategy is not physical. If an operation was placed into the candidate operation set of an idle machine but not selected for processing next by the sequencing rule, it can also be selected to form the candidate operation set of another idle machine later on.

In this paper, we propose three different delayed routing strategies to form the candidate operation set of an idle machine, namely (1) *naive*, (2) *sequential* and (3) *parallel*.

3.2.1 Naive Delayed Routing Strategy

In the naive delayed routing strategy, the $\texttt{DelayedRouting}(\cdot)$ function forms $OperationSet_k$ by simply selecting all the ready operations that can be processed by M_k. It is called "naive" since it does not consider any state feature except the candidate machine set of each ready operation. The pseudo code of the naive delayed routing policy is given by Algorithm 3.

Algorithm 3: The naive delayed routing strategy

Input: The ready operations $\mathcal{O}^{(t)}$, the machine M_k.
Output: The candidate operation set of M_k for sequencing.

$OperationSet_k = \emptyset$;
foreach $O_{ij} \in \mathcal{O}^{(t)}$ **do**
  if $M_k \in \mathcal{M}_{ij}$ **then** $OperationSet_k = OperationSet_k \cup \{O_{ij}\}$;
end
return $OperationSet_k$;

3.2.2 Sequential Delayed Routing Strategy

The sequential delayed routing strategy uses a routing rule to form the candidate operation set of the idle machines based on the current job shop state. It is expected to form better candidate operation sets than the naive strategy by considering more state features. The pseudo code of the sequential delayed routing strategy is given in Algorithm 4. Initially, the candidate operation set of all the machines are set to empty, and the ready operations are sorted by their ready time, so that if an operation becomes ready earlier, its routing decision is made first. The job shop state is cloned to make sure it is not changed by forming the candidate operation set. Then, each ready operation is allocated to a machine by the routing rule, and the cloned job shop state S is updated accordingly. Finally, the candidate operation set of M_k is returned.

Algorithm 4: The sequential delayed routing strategy

Input: The ready operations $\mathcal{O}^{(t)}$, the machine M_k, current job shop state $S^{(t)}$.
Output: The candidate operation set of M_k for sequencing.

```
Set the candidate operation set of each machine as empty;
Sort the operations in O^(t) by ready time;
Clone the current job shop state S = S^(t);
foreach O_ij ∈ O^(t) do
    Select the machine M for O_ij by the routing rule based on the job shop
      state S;
    Add O_ij into the candidate operation set of M;
    Update the cloned job shop state S;
end
return the candidate operation set of M_k;
```

The sequential delayed routing strategy is expected to consider more global information including the state of all the ready operations and all the machines. It is similar to the existing routing strategy (Yska et al., 2018; Zhang et al., 2018) but different in the following aspects.

- The sequential delayed routing strategy makes routing decisions using the latest information, which is more accurate.
- The sequential delayed routing strategy clones the job shop state and updates the cloned state. This way, an operation is not physically allocated to a machine's candidate operation set, and the routing decisions can still be changed later on.

Note that at a decision point t, there can be multiple idle machines. The sequential delayed routing strategy makes the sequencing decisions for the idle machines in a sequential manner (hence it is named "sequential"). The candidate operation set of an idle machine depends on the sequencing decisions made by the previous idle machines.

3.2.3 Parallel Delayed Routing Strategy

Note that in Algorithm 4, one can obtain the candidate operation set of all the machines, including all the idle machines. Therefore, one can make the routing decisions for the idle machines in a parallel way by calculating the candidate operation sets only once. The pseudo code of the parallel delayed routing strategy is described in Algorithm 5.

Algorithm 5: The parallel delayed routing strategy

Input: The ready operations $\mathcal{O}^{(t)}$, the machine M_k, current job shop state $S^{(t)}$.
Output: The candidate operation set of M_k for sequencing.
if *M_k is the first idle machine of $\mathcal{M}^{(t)}$* **then**
  Calculate the candidate operation set of all the machines by Algorithm 4 and store them globally;
  return *the candidate operation set of M_k;*
else
  return *the candidate operation set of M_k obtained from the global storage;*
end

The parallel delayed routing strategy is potentially more efficient than the sequential one, as it calculates the candidate operation sets only once. In addition, the sequencing decisions of the idle machines are independent of each other, and can be easily parallelised.

3.3 Overall Framework of GPHH-DR

The overall framework of GPHH-DR is given in Algorithm 6. It is a standard GPHH with the heuristic template with delayed routing (Algorithm 2) for fitness evaluation. To solve the multi-objective DFJSS, it uses the NSGA-II selection scheme to select the population.

Algorithm 6: The overall framework of GPHH-DR for Multi-Objective DFJSS

Input: A set of training DFJSS instances.
Output: A set of non-dominated individuals.
Randomly initialise a population of **individuals**;
while *stopping criteria not met* **do**
  `// Evaluation`
  foreach *ind* $\in$ *population* **do**
    Generate a schedule by applying this individual in Algorithm 2 on each training instance;
    Calculate the MT and REC_{idle} of the schedules by Eqs. (9) and (10);
    Calculate $F_1(indi)$ and $F_2(indi)$ as the mean of the MT and REC_{idle} values, respectively;
  end
  Calculate the non-dominated sorting and crowding distance (Deb et al., 2002) based on individuals' fitness.;
  Generate a new population using crossover, mutation and reproduction operators and the parents are selected based on the non-dominated sorting and crowding distance;
end
return *the final non-dominated individuals;*

There are three versions of GPHH-DR, depending on the delayed routing strategy used in the heuristic template. They have different individual representations as follows.

- **GPHH-DR-N** uses the *naive* delayed routing strategy. An individual in GPHH-DR-N is a single tree representing the sequencing rule, as no routing rule is required.

- **GPHH-DR-S** adopts the *sequential* delayed routing strategy. An individual in GPHH-DR-S is represented as two trees. The first tree represents the routing rule, and the second being the sequencing rule.

- **GPHH-DR-P** uses the *parallel* delayed routing strategy. An individual in GPHH-DR-P is the same as that in GPHH-DR-S, with one tree representing the routing rule, and the other the sequencing rule.

In GPHH-DR-N, we use the standard crossover and mutation operators for the single tree. In GPHH-DR-S and GPHH-DR-P, we use the multi-tree crossover and mutation operators proposed by Zhang et al. (2018).

3.4 Relationship Between Active Schedules and GPHH-DR

Active schedules with DR is also a well-known scheduling method, in which a non-delay factor α is required. This non-delay factor reflects the look-ahead ability of the heuristics, thus the determination of α is important for active schedules.

Different from active schedules, GPHH-DR is a parameter-free algorithm that there is no need for GPHH-DR to decide the non-delay factor α. Furthermore, in GPHH-DR, only the routing decision is delayed, but no job operation is allowed to be delayed if there is an idle machine. That is to say, the schedules generated by GPHH are non-delay schedules.

4 Experiment Design

To evaluate the proposed GPHH-DR, we empirically compare the three versions of GPHH-DR with the current state-of-the-art algorithms on a range of multi-objective DFJSS scenarios. Specifically, the following existing algorithms are included in the comparisons.

- **GPHH-LWT** (Tay and Ho, 2008) employs the heuristic template with immediate routing. It uses the Least Waiting Time as the routing rule and evolves the sequencing rule. It is the baseline GPHH for DFJSS.

- **GPHH-MT** (Zhang et al., 2018) employs the heuristic template with immediate routing. It uses the Multi-Tree representation to evolve the routing and sequencing rules simultaneously. It is the current state-of-the-art GPHH with immediate routing.

- **GPHH-UM** (Durasević et al., 2016) employs the heuristic template with delayed routing. It was proposed for solving JSS with Unrelated Machines. It evolves a single rule for both routing and sequencing. It is the only existing GPHH with delayed routing.

This section gives details about experiment design, including the simulation model, parameter setting of the compared algorithms and the performance measures.

4.1 DFJSS Simulation Model

The configuration of the DFJSS simulation model used in this paper is shown in Table 1. It has been commonly used in existing studies (Hildebrandt and Branke, 2015; Zhang et al., 2018; Yska et al., 2018). Note that in DFJSS, an operation can be processed by multiple machines. We assume that the processing time and energy consumption of the same operation by different machines are correlated with each other. Therefore, for each operation O_{ij}, we first randomly sample a mean processing time $\bar{p}_{ij}$ and a mean energy consumption $\bar{e}_{ij}$ from the uniform distribution $U(1, 99)$, and then sample the processing time p_{ijk} and energy consumption e_{ijk} for each machine from the normal distributions, i.e. $p_{ijk} \sim N(\bar{p}_{ij}, \bar{p}_{ij}/10)$ and $e_{ijk} \sim N(\bar{e}_{ij}, \bar{e}_{ij}/10)$. The standby power consumption of the 10 machines are sampled from a wide range, so that the REC_{idle} can be more sensitive to the schedule.

Table 1: Dynamic simulation configuration

Parameter	Value
Number of machines m	10
Number of arrived jobs n	5000
Number of warm-up jobs	1000
Number of operations per job	discrete $U(1, 10)$
Available machines per operation	discrete $U(1, 10)$
Job arrival process	Poisson process
Utilisation level σ	0.85, 0.95
Due date factor τ	2, 4, 6
Mean processing time $\bar{p}_{ij}$	discrete $U(1, 99)$
Mean energy consumption $\bar{e}_{ij}$	discrete $U(1, 99)$
Standby power consumption ϵ_k	{10, 12.5, 4.5, 3.6, 7.0, 1.5, 8.5, 2.2, 22.9, 6.4}

In the simulation model, the inter-arrival time of jobs follows the exponential distribution, and its rate parameter λ is calculated as (Rajendran and Holthaus, 1999)

$$\lambda = \frac{0.5(n_{low} + n_{up}) \times 0.5(p_{low} + p_{up})}{\sigma \times m} \tag{12}$$

where n_{low} and n_{up} are lower and upper bound of the number of operations respectively, p_{low} and p_{up} are lower and upper bound of the processing time, σ is the utilisation level and m is the number of machines.

The two utilisation levels and three due date factors lead to $2 \times 3 = 6$ scenarios denoted as "⟨utilisation level-due date factor⟩". For each scenario, 50 simulations are sampled independently to form the test set. During the training process, one simulation is used for fitness evaluation. To reduce overfitting, the training simulation is changed in each generation (Hildebrandt et al., 2010).

A simulation normally stops after the first 5000 arrived jobs have been completed. However, due to the application of random rules, such stopping criteria may never be reached, and some jobs can be in the candidate operation set forever. To address this issue, we stop the simulation if (1) the size of any machine's candidate operation set is larger than 100, or (2) more than 7500 jobs have arrived. If the simulation is stopped under the above criteria, we set a very poor fitness of the evaluated individual.

4.2 Parameter Setting

Table 2 shows the parameter setting of algorithms. The terminal set, which has been commonly used in the previous studies (Nguyen et al., 2019; Zhang et al., 2018; Li

et al., 2013), is given in Table 3. The function set is set to $\{+, -, *, /, \min, \max\}$, where the division returns 1 if divided by zero. The common terminals are used by all the compared algorithms. Note that GPHH-DR-N uses 7 additional terminals than GPHH-DR-S and GPHH-DR-P. This is because GPHH-DR-N uses a single rule to make both the routing and sequencing decisions. Unlike GPHH-DR-S and GPHH-DR-P, which use a separate routing rule to calculate the priority of each machine for each candidate operation, GPHH-DR-N only applies the single rule to the idle machine. Therefore, it requires additional information about other machines to calculate the priority of the candidate operations. The 7 additional terminals were designed to reflect the relative priority of the other machines to the idle machine from different aspects.

Table 2: Parameter setting of algorithms.

Parameter	Value
Initialisation	Ramped half-and-half
Population size	1024
Maximal depth	8
Crossover rates	80%
Mutation rates	15%
Reproduction rates	5%
Parent selection	Tournament selection with size 7
Elitism	10 best individuals
Number of generations	51

Table 3: The terminal set of the GPHH approaches.

Algorithm	Terminal	Description
Common	PT	Processing time of operation O_{ij} on the machine M_k
	EC	Energy consumption of operation O_{ij} on the machine M_k
	SL	Slack of the job J_i = d_i - t - WKR
	OWT	Waiting time of operation O_{ij} since ready
	NPT	Average processing time of the next operation
	NEC	Average energy consumption of the next operation
	WKR	Work remaining of the job J_i
	ECR	Remaining energy consumption of the job J_i
	NOR	Number of remaining operations of the job J_i
	TIS	Time of the job J_i in the job shop
	WIQ	Workload in the candidate operation set of machine M_k
	EIQ	Total energy consumption in the candidate operation set
	NIQ	Number of operations in the candidate operation set
	MP	The standby power consumption of the machine
	NOS	Number of candidate machines of the operation
	MWT	Machine waiting time
GPHH-DR-N	rPT	Relative processing time = $p_{ijk_{\text{idle}}}$ - min$\{p_{ijk} \mid M_k \in \mathcal{M}_{ij}\}$
	rEC	Relative energy consumption
	rWIQ	Relative workload in the candidate operation set
	rEIQ	Relative energy consumption in the candidate operation set
	rNIQ	Relative number of operations in the candidate operation set
	rMWT	Relative machine waiting time
	rMP	Relative standby power consumption

4.3 Performance Measures

The Hyper-Volume (HV) and Inverted Generational Distance (IGD), which are the two commonly used performance measures for multi-objective optimization, are adopted

in this paper. A higher HV value and a lower IGD value indicate a better performance of the algorithm. HV requires a reference point, and IGD relies on the true Pareto front. Since the true Pareto front is unknown in our problem, we combine the results of all the runs of all the compared algorithms and obtain the non-dominated solutions to approximate the Pareto front.

In our experiments, we first normalise the two objectives into the range $[0, 1]$. Then, we calculate IGD based on the approximated Pareto front, and HV with the reference point $1 + 1/(n_{\mathrm{APF}} - 1)$ (Ishibuchi et al., 2017), where n_{APF} is the number of points in the approximated Pareto front.

5 Results and Discussions

5.1 Overall Performance

Tables 4 and 5 show the mean and standard deviation of the HV and IGD values of the 30 independent runs of the compared algorithms on the tested six DFJSS scenarios.

Table 4: The mean and standard deviation of the HV values over the 30 independent runs of the compared algorithms.

Scenario	GP-LWT	GP-MT	GP-UM	GP-DR-N	GP-DR-P	GP-DR-S
$\langle 0.85\text{-}2 \rangle$	0.068(0.021)	0.920(0.016)	0.132(0.020)	0.222(0.020)	**0.934(0.016)**	**0.928(0.014)**
$\langle 0.85\text{-}4 \rangle$	0.070(0.043)	0.986(0.024)	0.136(0.039)	0.231(0.035)	0.994(0.024)	0.994(0.011)
$\langle 0.85\text{-}6 \rangle$	0.089(0.061)	1.032(0.012)	0.155(0.056)	0.253(0.051)	1.034(0.021)	1.025(0.022)
$\langle 0.95\text{-}2 \rangle$	0.179(0.038)	0.925(0.029)	0.414(0.084)	0.707(0.043)	**0.945(0.028)**	0.936(0.025)
$\langle 0.95\text{-}4 \rangle$	0.236(0.061)	0.999(0.012)	0.479(0.067)	0.853(0.027)	**1.013(0.011)**	1.004(0.010)
$\langle 0.95\text{-}6 \rangle$	0.213(0.112)	1.036(0.017)	0.485(0.112)	0.911(0.039)	**1.047(0.015)**	1.041(0.015)

Table 5: The mean and standard deviation of the IGD values over the 30 independent runs of the compared algorithms.

Scenario	GP-LWT	GP-MT	GP-UM	GP-DR-N	GP-DR-P	GP-DR-S
$\langle 0.85\text{-}2 \rangle$	0.651(0.036)	0.014(0.009)	0.592(0.036)	0.500(0.033)	**0.009(0.006)**	**0.010(0.007)**
$\langle 0.85\text{-}4 \rangle$	0.778(0.042)	0.024(0.022)	0.716(0.036)	0.624(0.034)	0.020(0.016)	0.017(0.009)
$\langle 0.85\text{-}6 \rangle$	0.822(0.077)	0.025(0.013)	0.760(0.071)	0.669(0.062)	0.028(0.024)	0.038(0.034)
$\langle 0.95\text{-}2 \rangle$	0.537(0.030)	0.019(0.009)	0.350(0.056)	0.143(0.025)	**0.012(0.009)**	**0.012(0.005)**
$\langle 0.95\text{-}4 \rangle$	0.575(0.059)	0.023(0.009)	0.373(0.057)	0.098(0.020)	**0.011(0.005)**	**0.016(0.010)**
$\langle 0.95\text{-}6 \rangle$	0.697(0.099)	0.026(0.016)	0.460(0.086)	0.086(0.017)	0.021(0.012)	0.026(0.016)

From the tables, one can see among the three GPHH-DR versions, GPHH-DR-P and GPHH-DR-S performed much better than GPHH-DR-N. They have much higher HV values and much lower IGD values. This is as expected since GPHH-DR-N uses a naive delayed routing strategy that does not consider how to allocate the ready operations into different machines. On the other hand, GPHH-DR-P and GPHH-DR-S make the delayed routing decisions by a routing rule, which can provide much better candidate operation sets for the sequencing rule.

Among the existing algorithms, GPHH-MT performed the best, which was much better than GPHH-LWT and GPHH-UM. This is also consistent with our expectations. First, it has been demonstrated in existing works (Zhang et al., 2018; Yska et al., 2018) that co-evolving the routing and sequencing rules can lead to much better results than fixing the routing rule (i.e. GPHH-LWT). GPHH-MT outperformed GPHH-LWT since it co-evolves the routing and sequencing rule. Second, although GPHH-UM adopts the

delayed routing rather than immediate routing, it uses a single rule for both routing and sequencing. Due to the different underlying mechanisms of the two kinds of decisions, it is hardly possible to have a single rule that can capture useful information for both routing and sequencing. From the above observations, we can see that the best-performing algorithms use separate rules for routing and sequencing respectively, either with immediate (GPHH-MT) or delayed routing (GPHH-DR-P and GPHH-DR-S).

Then, to verify the effectiveness of the proposed GPHH-DR, we compare GPHH-DR-P and GPHH-DR-S with GPHH-MT. Under the Wilcoxon's rank sum test with significance level of 0.05, if GPHH-DR or GPHH-MT is significantly better than the other compared algorithm, then the corresponding entry is marked in bold. From the tables, one can see that GPHH-DR-P and GPHH-DR-S outperformed GPHH-MT in most cases, and no worse than GPHH-MT in all the scenarios. GPHH-DR-P performed slightly better than GPHH-DR-S. In terms of HV, GPHH-DR-P significantly outperformed GPHH-MT in 4 scenarios. In terms of IGD, GPHH-DR-P was significantly better than GPHH-MT in 3 scenarios.

Figure 4 shows the final non-dominated solutions in the combined set of the 30 runs of the compared algorithms, which is an indicator of the best performance. From Figure 4, we can see that as the due date factor increases (e.g. from $\langle$0.85-2$\rangle$ to $\langle$0.85-6$\rangle$), the mean tardiness obtained by the algorithms tend to decrease, and the energy efficiency stays in the same range. This is because energy efficiency is independent of the due date factor. As the utilisation level increases (e.g. from $\langle$0.85-2$\rangle$ to $\langle$0.95-2$\rangle$), the mean tardiness of the algorithms tend to increase, and the ratio of idle energy consumption tends to decrease (the energy efficiency is better). This is because a higher utilisation level leads to a busier status of the machines, and thus a higher energy efficiency.

For all the scenarios, GPHH-LWT, GPHH-UM, and GPHH-DR-N showed obvious disadvantages in optimising the energy efficiency (the y-axis), although they were able to achieve almost the same mean tardiness as the other algorithms. GPHH-LWT performed the worst in terms of energy efficiency, since it uses the manually designed LWT routing rule, which does not consider the energy efficiency at all. GPHH-UM and GPHH-DR-N use a single rule, and thus are weaker in making decisions than GPHH-MT, GPHH-DR-P, and GPHH-DR-S, which use two rules for routing and sequencing respectively. The reason why all the algorithms reached almost the same best mean tardiness (which is nearly zero) is that it is easier to minimise the mean tardiness, e.g. by allocating the operations to the machine with minimal processing time. The best performance of GPHH-MT, GPHH-DR-P, and GPHH-DR-S are similar to each other. When we zoom in to some specific overlapping regions, we can see some slight difference, where GPHH-DR-P and GPHH-DR-S obtained slightly better solutions than GPHH-MT. The main advantage of GPHH-DR-P and GPHH-DR-S over GPHH-MT was that they can reach more extreme points with a smaller ratio of idle energy consumption (and larger mean tardiness).

Table 6 shows the average tree size for GPHH-MT, GPHH-DR-P, and GPHH-DR-S in each scenario. We can see that these three algorithms obtained a very similar size of the routing rule and sequencing rule, which is much smaller than the full tree size under the maximal depth of 8 ($2^8 - 1 = 255$). This implies that the change from immediate routing to delayed routing in the heuristic template does not change the complexity of the decision making.

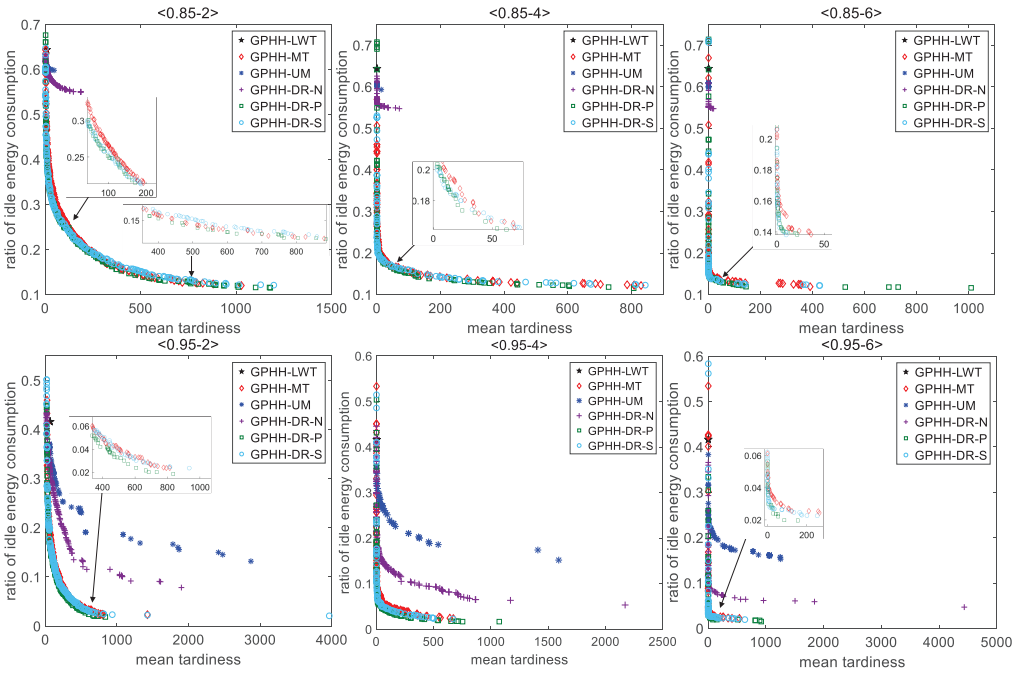

Figure 4: Combined non-dominate set of each algorithm in the six scenarios.

Table 6: The average tree size for three algorithms in each scenario

Rule type	Algorithms	$\langle 0.85\text{-}2 \rangle$	$\langle 0.85\text{-}4 \rangle$	$\langle 0.85\text{-}6 \rangle$	$\langle 0.95\text{-}2 \rangle$	$\langle 0.95\text{-}4 \rangle$	$\langle 0.95\text{-}6 \rangle$
routing	GP-MT	58.394	62.533	58.239	58.247	59.443	60.302
	GP-DR-P	56.956	62.024	58.819	60.385	57.396	57.101
	GP-DR-S	55.759	63.250	57.230	61.882	59.697	60.996
sequencing	GP-MT	51.049	55.044	48.683	53.295	55.682	51.292
	GP-DR-P	46.645	47.319	42.972	49.355	48.275	43.687
	GP-DR-S	48.944	47.338	47.296	46.633	47.387	47.584

5.2 Effectiveness on Single Objective Optimisation

Theoretically, the newly proposed GPHH-DR should work for any number of objectives. We have demonstrated the effectiveness of GPHH-DR on the multi-objective DFJSS. It is interesting to know whether the advantage of GPHH-DR still holds for single-objective DFJSS, which is considered easier than the multi-objective DFJSS. To this end, we reran the algorithms on the DFJSS that minimises the mean tardiness alone and compared their performance. We used the same experiment setting for the single-objective and multi-objective optimisation, including the simulation model and parameter setting. The only difference here is that the selection is based purely on the mean tardiness instead of dominance relation. For each run of each algorithm, the best rule is obtained using a training set. Then, the best rules are applied to a separate test set, and the test performance is calculated as the average mean tardiness of the generated schedules on the test set.

Table 7 shows the test performance of the compared algorithms. In addition, each GPHH-DR version is compared with GPHH-MT, which is the best existing GPHH algorithm, using Wilcoxon's rank sum test under the significance level of 0.05. The sig-

nificantly better results are marked in bold.

Table 7: The mean and standard deviation of the MT over the 30 independent runs of the compared algorithms.

Scenario	GP-LWT	GP-MT	GP-UM	GP-DR-N	GP-DR-P	GP-DR-S
$\langle 0.85\text{-}2 \rangle$	8.872(0.191)	3.563(0.140)	9999(9999)	**2.658(0.139)**	**2.684(0.174)**	**2.465(0.131)**
$\langle 0.85\text{-}4 \rangle$	0.116(0.058)	0.040(0.008)	0.056(0.034)	0.045(0.003)	**0.039(0.020)**	**0.036(0.008)**
$\langle 0.85\text{-}6 \rangle$	0.043(0.003)	0.020(0.010)	9999(9999)	0.023(0.002)	0.030(0.076)	0.044(0.154)
$\langle 0.95\text{-}2 \rangle$	68.44(0.946)	22.05(0.772)	9999(9999)	**17.44(0.388)**	**18.96(1.107)**	**17.61(0.806)**
$\langle 0.95\text{-}4 \rangle$	4.271(0.238)	0.128(0.049)	9999(9999)	**0.062(0.006)**	**0.109(0.048)**	**0.099(0.056)**
$\langle 0.95\text{-}6 \rangle$	0.423(0.603)	0.028(0.036)	5.800(6.250)	0.027(0.012)	0.028(0.023)	0.022(0.009)

From the table, it is obvious that all three GPHH-DR versions significantly outperformed all the existing GPHH methods. Note that the results of GP-UM are usually very huge (9999). This is because the evolved rules are not generalisable, and some simulations in the test set never stop. The test performances of these rules were then set to 9999.

An interesting observation from the table is that GPHH-DR-N performed competitively with GPHH-DR-P and GPHH-DR-S. This is consistent with Figure 4, which shows that GPHH-DR-N was able to reach very small mean tardiness. This indicates that to minimise the mean tardiness, the sequencing decision can be made considerably easily and much less sensitive to the pool of the candidate jobs. However, the energy efficiency is a much more complex objective, and highly depends on the allocation of the jobs to the machines. This makes the sequencing decision much more sensitive to the given pool.

Figure 5 shows the converge curve of the test performance of the six algorithms. The figure shows consistent patterns with Table 7. The three GPHH-DR versions converged the fastest, while GPHH-UM performed the worst.

Overall, we can see that GPHH-DR can significantly outperform the GPHH counterpart with immediate routing. If the objective is easy for the sequencing rule (e.g. mean tardiness), then GPHH-DR-N can already perform competitively. When taking more complex objectives such as energy efficiency into account, the more advanced GPHH-DR-P and GPHH-DR-S become superior.

5.3 Effectiveness of Delayed Routing

Note that the effectiveness of GPHH-DR-P and GPHH-DR-S rely on the following two components: (1) the novel heuristic template with delayed routing, and (2) the routing and sequencing rules evolved by GP. To further verify the effectiveness of the novel heuristic template with delayed routing, which is the major contribution in this work, we compared the performance of different heuristic templates by fixing the rules as the manually designed rules. This way, the impact of the GP-evolved rules can be excluded. Note that when using the manually designed rules, the algorithms become deterministic.

We select the Least Waiting Time (LWT) as the routing rule, and the ATC rule as the sequencing rule, both are well-known manually designed rules for DFJSS (Tay and Ho, 2008; Nguyen et al., 2013a).

Table 8 shows the mean tardiness obtained by the LWT+ATC rules on the heuristic templates with immediate routing, parallel delayed routing, and sequential delayed routing in different DFJSS scenarios. It is obvious that the two heuristic templates with delayed routing obtained much smaller mean tardiness than the one with immediate

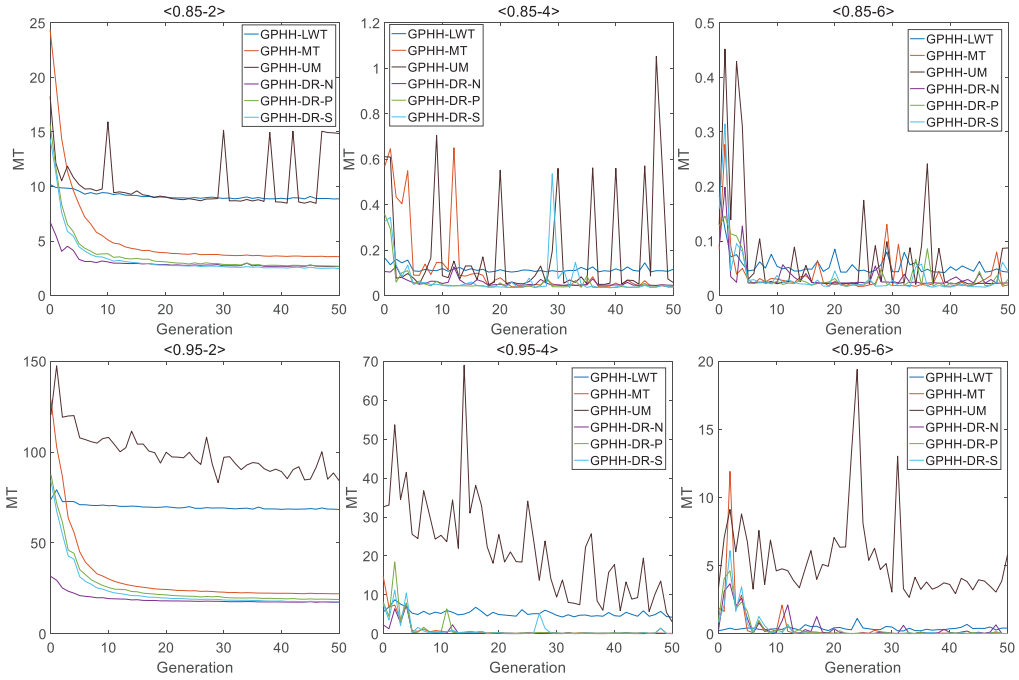

Figure 5: The converge curve of average MT over 30 runs.

Table 8: The mean tardiness of the LWT+ATC rule in the heuristic templates with immediate routing, parallel delayed routing and sequential delayed routing.

Heuristic template	$\langle 0.85\text{-}2 \rangle$	$\langle 0.85\text{-}4 \rangle$	$\langle 0.85\text{-}6 \rangle$	$\langle 0.95\text{-}2 \rangle$	$\langle 0.95\text{-}4 \rangle$	$\langle 0.95\text{-}6 \rangle$
immediate routing	37.387	10.147	3.922	332.848	217.855	161.701
parallel delayed routing	28.241	5.409	1.711	266.919	155.008	109.281
sequential delayed routing	27.084	5.340	1.746	257.458	147.687	103.197

routing for all the scenarios. This verifies the effectiveness of the proposed delayed routing strategies. Another observation from Table 8 is that the results obtained by LWT+ATC are much worse than the results obtained by GPHH in Table 7. This is consistent with previous studies (Li et al., 2013; Tay and Ho, 2008; Nguyen et al., 2013a, 2019) that GPHH can evolve much better DRs than manually designed DRs.

Recalling that the main advantage of using DRs to solve DFJSS is its ability to make decisions in real-time. Therefore, it is important to ensure that with the delayed routing, the decisions can still be made in real-time. For this purpose, we also calculated the time spent on each decision during the simulations. In the heuristic template with immediate routing, the average time spent by each decision is 0.0014ms. The heuristic templates with parallel and sequential delayed routing make each decision in 0.0022ms and 0.0060ms, respectively. This shows that although the delayed routing strategy is slower than the immediate routing strategy, it can still make sure that the decisions are made in real-time.

5.4 Semantic Analysis of Evolved Rules

To gain deeper understandings of the evolved rules, we consider one run of GPHH-DR-S in the $\langle$0.95-2$\rangle$ scenario. Again, we sort the individuals in the increasing order of MT, and select the first individual, the last individual, and the median individual. The three individuals are shown in Table 9, along with their two objective values. To facilitate analysis, we simplified the rules using algebraic transformation (e.g. $a-a=0$ and $a/a=1$).

Table 9: Examples of the GP-evolved individuals

Individual #1	(MT=18.732, REC_{idle}=0.508)
routing rule:	(PT^2+2NIQ×PT-MWT×PT-MP)×(max(PT, MP)+PT×NIQ)-2MWT-MP
sequencing rule:	2PT+2SL-WKR+max(3PT+2SL-2WKR, TIS/max(PT+WKR, MWT))
Individual #2	(MT=160.500, REC_{idle}=0.113)
routing rule:	min(WIQ-MP, 2PT)+min(WIQ-2MP, MWT^2)-PT-MWT-MP
sequencing rule:	3WKR×(WKR-NIQ)×(PT×SL+NOR)
Individual #3	(MT=783.547, REC_{idle}=0.031)
routing rule:	2PT×(NIQ-MP)-MP+NIQ×(2NIQ+PT-MWT)
sequencing rule:	max(WKR, 2PT+2SL+NPT+OWT)

Individual #1 has the smallest MT and the highest REC_{idle} among the three individuals. In other words, it focuses more on minimising MT. In its routing rule, PT is frequently used and plays an important role in the priority. This rule tends to allocate the operations to the machines with the smallest processing time. Another three main components in the routing rule are MWT, NIQ, and MP. Specifically, the operation is more likely to be allocated to a machine with smaller NIQ (fewer operations in the candidate operation set), larger MWT (has waited for longer), and larger standby power consumption. On one hand, choosing the machine with smaller PT, smaller NIQ and larger MWT tends to speed up the completion of the operations, and thus minimise the mean tardiness. On the other hand, making the machines with larger standby power consumption busier can effectively reduce REC_{idle}. It can also be seen that PT, NIQ, and MWT play more important roles than MP, as the priority function is of the second order of PT, NIQ, and MWT, while of the first order of MP.

The sequencing rule obviously tends to select the operations with smaller processing time, smaller slack and larger work remaining. This is consistent with intuition. Note that several dispatching rules have also been manually designed based on similar intuition to minimise mean tardiness and flowtime for JSS (e.g. PT+WINQ+SL (Holthaus and Rajendran, 2000) and the rule selecting the operation with most work remaining).

Individual #2 tries to find a trade-off between MT and REC_{idle}. That is, it has a reasonably good performance for both objectives in the test instances. The routing rule contains five terms. The first two terms are min(WIQ-MP, 2PT) and (WIQ-2MP, MWT^2), indicating that the machines with smaller work in the candidate operation set, smaller processing time, smaller waiting time, and larger standby power consumption are more likely to be allocated.

In most cases, MP is much smaller than WIQ which means WIQ directly affects the priority when PT and MWT are both large. When WIQ is large, PT and MWT show more impact on the priority. There are three -MP terms in this rule which shows its emphasis on REC_{idle}. The sequencing rule tends to select the operations with smaller work remaining, processing time and slack. Although it is reasonable to prefer the

operations with smaller processing time and slack to minimise the mean tardiness, it is a bit counter-intuitive to choose the operations with smaller work remaining. This may be because of the difficulty of balancing the two conflicting objectives in the sequencing rule.

Individual #3 mainly focuses on optimising REC_{idle}. To improve REC_{idle}, it is important to keep the machines, especially those with high standby power consumption, as busy as possible, so that the idle energy consumption is minimised. The routing rule of individual #3 is obviously for this purpose. It tends to allocate operations to the machines with smaller size of candidate operation set (which are more likely to become idle soon) and larger standby power consumption. In addition, it prefers the machines with smaller processing time and larger waiting time as well, which can also reduce machine idle time and speed up the completion of the operations, and thus reduce the mean tardiness. For the sequencing rule, we observe similar patterns as in individual #2, where the operations with smaller processing time, slack and work remaining are more likely to be selected. Again, it is counter-intuitive to select the operations with smaller work remaining. A possible explanation is that the objective REC_{idle} highly relies on the routing decisions, which should make all the machines, especially the ones with high standby power consumption, as busy as possible. As long as these machines are busy, the sequencing decisions become less important.

Overall, we can see that individuals with different preferences between MT and REC_{idle} have quite different routing and sequencing rules. The routing rule is very important in all cases, to prefer the machines with smaller processing time, smaller work in the candidate operation set, and larger standby power consumption. The sequencing rule prefers the operations with smaller processing time and slack in all cases. However, as the preference shifts more towards REC_{idle}, the sequencing rule becomes less sensitive to the performance.

6 GPHH-DR in the Practical Situation

6.1 Relationship between GPHH-DR and Production Management Methods

Production management methods like pull systems, Kanban and theory-of-constraints are very practical technologies implemented by manufacturing companies and used to deal with multi-variety and small batch production, in which the key problem of them is to reduce waste, stocks and etc. It is apparent that production management includes all aspects of manufacturing.

Although production management methods and GPHH-DR both focus on the scheduling, there still exist some differences because of the different angles of these two methods. The main difference is that GPHH-DR treats the scheduling problem as an optimisation problem and tries to find the optimal solution by iterative search, while it is hard for production management methods to guarantee the optimality of the solutions. Besides, GPHH-DR can focus on time-irrelevant objectives (e.g. energy efficiency) simultaneously, while the production management methods pay more attention to time-relevant objectives (e.g. job flow time) to ensure the regular operation of the manufacturing system and avoid excess capacity.

6.2 Managerial Implication of GPHH-DR

Job shop is essentially a discrete manufacturing system and is different from the flow shop. In the discrete manufacturing system, the semi-finished jobs are sent to a place called buffer and wait to be allocated to the next processing machine in real production. Buffer is a key component of the discrete manufacturing that balances the different

processing rates of different processing tasks. Actually, the "candidate operation set" mentioned in the traditional GPHH are essentially buffers that store the semi-finished jobs for the assigned machines in the job shop. The implication of proposed GPHH-DR includes two types: single buffer and multiple buffers, as shown in Figure 6.

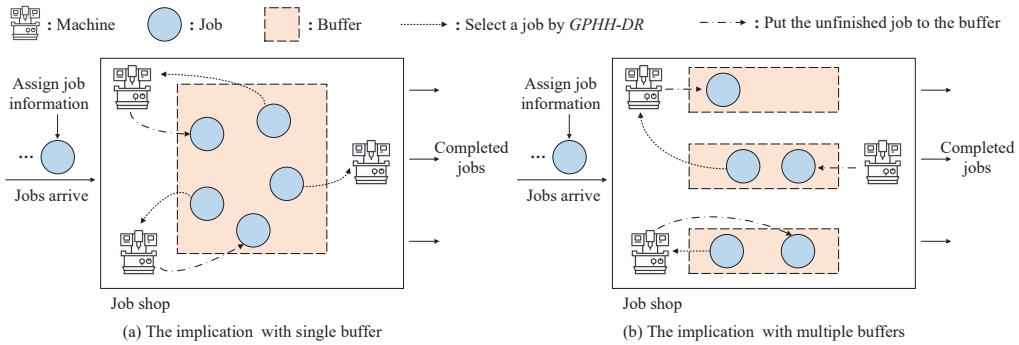

Figure 6: The managerial implication of GPHH-DR with different cases.

When there is only one big buffer in the job shop, it is the practical "pool" mentioned in the proposed algorithm. The semi-finished job is sent to this buffer physically after the previous operation is finished. All the ready jobs/operations are waiting in the big buffer until one of the machines becomes idle and chooses a job from this buffer. Then, this chosen job will be sent to the idle machine to process by transportation facility. If all the operations of a job are finished, it will be sent to the finished product warehouse directly.

Another feasible managerial implication of GPHH-DR requires multiple buffers that belong to machines. In this case, a job that has been processed by a machine but has not been allocated to the next machine will be placed in the machine's buffer. This way, the "pool" is essentially the union of all the buffers of the machines. Then, when an idle machine chooses this job to process next, the transportation facility will transport this job from the buffer of the current machine to the idle machine to process directly.

In summary, the implication of GPHH-DR does not require any extra place or changes. In this simplified version, the transportation cost was ignored. We will consider the transportation cost during the routing decision in our future work.

7 Conclusions and Future Work

In this paper, we develop a novel GPHH, named GPHH-DR, for solving multi-objective DFJSS that optimises the mean tardiness and energy efficiency simultaneously. We propose a novel heuristic template, which is a novel framework for the dispatching rules to operate in to generate schedules in real-time. In the new heuristic template, each routing decision is delayed rather than being made immediately. This way, the allocation of the ready operations to the machines can always take the latest information into account, and thus the algorithm has a potential to make better routing decisions.

Based on the novel heuristic template, we proposed a naive delayed routing strategy and two advanced delayed routing strategies. We found that simply delaying the routing decisions and using a single sequencing rule (i.e. GPHH-DR-N) cannot lead to promising results. This is because the sequencing rule cannot consider the routing decisions (e.g. balancing the operations on different machines), and thus may easily lead to

bottleneck machines. The more advanced strategies (i.e. GPHH-DR-P and GPHH-DR-S) with both routing and sequencing rules, on the other hand, showed superior performance over the current state-of-the-art GPHH with immediate routing for DFJSS. This demonstrates the effectiveness of delaying the routing decisions to the last minute, and making the sequencing decisions with smart allocation of the ready operations to the machines.

In the future, we will extend the delayed routing strategy to more complex DFJSS, such as many-objective problem and fuzzy DFJSS. Besides, it is also an urgent requirement for reducing the training cost of GPHH-DR. More efforts are needed to be done to improve the performance of GPHH-DR to find optimal DRs. In this paper, only job arrival is considered. Machine breaking down is also a dynamic event that worth to be considered.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grant No. 61572238), the Marsden Fund of New Zealand (Nos. VUW1614 and VUW1209), the Provincial Outstanding Youth Foundation of Jiangsu Province (Grant No. BK20160001) and the 111 Project (Grant No.B12018).

References

Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998). *Genetic Programming: An Introduction*, volume 1. Morgan Kaufmann San Francisco.

Brah, S. A. and Hunsucker, J. L. (1991). Branch and bound algorithm for the flow shop with multiple processors. *European Journal of Operational Research*, 51(1):88–99.

Branke, J., Nguyen, S., Pickardt, C. W., and Zhang, M. (2016). Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., and Woodward, J. R. (2009). Exploring hyper-heuristic methodologies with genetic programming. In *Computational Intelligence*, pages 177–201. Springer.

Dai, M., Tang, D., Giret, A., and Salido, M. A. (2019). Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints. *Robotics and Computer-Integrated Manufacturing*, 59:143–157.

Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Durasević, M. and Jakobović, D. (2018). Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. *Genetic Programming and Evolvable Machines*, 19(1-2):9–51.

Durasević, M., Jakobović, D., and Knežević, K. (2016). Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, 48:419–430.

Gahm, C., Denz, F., Dirr, M., and Tuma, A. (2016). Energy-efficient scheduling in manufacturing companies: a review and research framework. *European Journal of Operational Research*, 248(3):744–757.

Gao, K., Yang, F., Zhou, M., Pan, Q., and Suganthan, P. N. (2018). Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm. *IEEE Transactions on Cybernetics*, 49(5):1944–1955.

Gao, K. Z., Suganthan, P. N., Pan, Q. K., Tasgetiren, M. F., and Sadollah, A. (2016). Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion. *Knowledge-Based Systems*, 109:1–16.

Gao, L. and Pan, Q.-K. (2016). A shuffled multi-swarm micro-migrating birds optimizer for a multi-resource-constrained flexible job shop scheduling problem. *Information Sciences*, 372:655–676.

Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

He, Y., Li, Y., Wu, T., and Sutherland, J. W. (2015). An energy-responsive optimization method for machine tool selection and operation sequence in flexible machining job shops. *Journal of Cleaner Production*, 87:245–254.

Hildebrandt, T. and Branke, J. (2015). On using surrogates with genetic programming. *Evolutionary Computation*, 23(3):343–367.

Hildebrandt, T., Heger, J., and Scholz-Reiter, B. (2010). Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 257–264. ACM.

Holthaus, O. and Rajendran, C. (2000). Efficient jobshop dispatching rules: further developments. *Production Planning & Control*, 11(2):171–178.

Hong, L., Drake, J. H., Woodward, J. R., and Özcan, E. (2018). A hyper-heuristic approach to automated generation of mutation operators for evolutionary programming. *Applied Soft Computing*, 62:162–175.

Huang, J. and Süer, G. A. (2015). A dispatching rule-based genetic algorithm for multi-objective job shop scheduling using fuzzy satisfaction levels. *Computers & Industrial Engineering*, 86:29–42.

Ishibuchi, H., Imada, R., Setoguchi, Y., and Nojima, Y. (2017). Reference point specification in hypervolume calculation for fair comparison and efficient search. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 585–592. ACM.

Jain, A. S. and Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390–434.

Jensen, M. T. (2003). Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):275–288.

Jose, B. A., Katie, M., and Ruben, P. (2019). Energy cost minimization for unrelated parallel machine scheduling under real time and demand charge pricing. *Journal of Cleaner Production*, 208:232–242.

Karimi-Nasab, M., Modarres, M., and Seyedhoseini, S. (2015). A self-adaptive pso for joint lot sizing and job shop scheduling with compressible process times. *Applied Soft Computing*, 27:137–147.

Langdon, W. B. and Harman, M. (2015). Optimizing existing software with genetic programming. *IEEE Transactions on Evolutionary Computation*, 19(1):118–135.

Lei, D., Gao, L., and Zheng, Y. (2018). A novel teaching-learning-based optimization algorithm for energy-efficient scheduling in hybrid flow shop. *IEEE Transactions on Engineering Management*, 65(2):330–340.

Li, N., Liang, G., Li, P., and Li, X. (2013). A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing*, 24(4):763–774.

Lin, J. (2019). Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time. *Engineering Applications of Artificial Intelligence*, 77:186–196.

Liu, Y., Dong, H., Lohse, N., and Petrovic, S. (2016). A multi-objective genetic algorithm for optimisation of energy consumption and shop floor production performance. *International Journal of Production Economics*, 179:259–272.

May, G., Stahl, B., Taisch, M., and Kiritsis, D. (2017). Energy management in manufacturing: From literature review to a conceptual framework. *Journal of Cleaner Production*, 167:1464–1489.

Mei, Y., Nguyen, S., Xue, B., and Zhang, M. (2017). An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(5):339–353.

Meng, L., Zhang, C., Shao, X., and Ren, Y. (2019). Milp models for energy-aware flexible job shop scheduling problem. *Journal of Cleaner Production*, 210:710–723.

Mokhtari, H. and Dadgar, M. (2015). Scheduling optimization of a stochastic flexible job-shop system with time-varying machine failure rate. *Computers & Operations Research*, 61:31–45.

Mokhtari, H. and Hasani, A. (2017). An energy-efficient multi-objective optimization for flexible job-shop scheduling problem. *Computers & Chemical Engineering*, 104:339–352.

Nag, K. and Pal, N. R. (2016). A multiobjective genetic programming-based ensemble for simultaneous feature selection and classification. *IEEE Transactions on Cybernetics*, 46(2):499–510.

Nguyen, S., Mei, Y., Ma, H., Chen, A., and Zhang, M. (2016). Evolutionary scheduling and combinatorial optimisation: Applications, challenges, and future directions. In *2016 IEEE Congress on Evolutionary Computation (CEC)*, pages 3053–3060. IEEE.

Nguyen, S., Mei, Y., Xue, B., and Zhang, M. (2019). A hybrid genetic programming algorithm for automated design of dispatching rules. *Evolutionary Computation*, 27(3):467–496.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2013a). 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):621–639.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2013b). Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology*, 67(1-4):85–100.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2014). Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation*, 18(2):193–208.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2015). Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE Transactions on Cybernetics*, 45(1):1–14.

Nguyen, S., Zhang, M., and Tan, K. C. (2017). Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics*, 47(9):2951–2965.

Nguyen, S., Zhang, M., and Tan, K. C. (2018). Adaptive charting genetic programming for dynamic flexible job shop scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1159–1166. ACM.

Nie, L., Gao, L., Li, P., and Shao, X. (2013). Reactive scheduling in a job shop where jobs arrive over time. *Computers & Industrial Engineering*, 66(2):389–405.

Niroomand, S., Hadi-Vencheh, A., Mirzaei, N., and Molla-Alizadeh-Zavardehi, S. (2016). Hybrid greedy algorithms for fuzzy tardiness/earliness minimisation in a special single machine scheduling problem: case study and generalisation. *International Journal of Computer Integrated Manufacturing*, 29(8):870–888.

Ouelhadj, D. and Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417.

Panwalkar, S. S. and Iskander, W. (1977). A survey of scheduling rules. *Operations Research*, 25(1):45–61.

Park, J., Mei, Y., Nguyen, S., Chen, G., and Zhang, M. (2018). An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. *Applied Soft Computing*, 63:72–86.

Peng, B., Lü, Z., and Cheng, T. (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53:154–164.

Pickardt, C. W., Hildebrandt, T., Branke, J., Heger, J., and Scholz-Reiter, B. (2013). Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics*, 145(1):67–77.

Pour, S. M., Drake, J. H., Ejlertsen, L. S., Rasmussen, K. M., and Burke, E. K. (2018). A hybrid constraint programming/mixed integer programming framework for the preventive signaling maintenance crew scheduling problem. *European Journal of Operational Research*, 269(1):341–352.

Rajendran, C. and Holthaus, O. (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1):156–170.

Romero-Silva, R., Shaaban, S., Marsillac, E., and Hurtado, M. (2018). Exploiting the characteristics of serial queues to reduce the mean and variance of flow time using combined priority rules. *International Journal of Production Economics*, 196:211–225.

Rubaiee, S. and Yildirim, M. B. (2019). An energy-aware multiobjective ant colony algorithm to minimize total completion time and energy cost on a single-machine preemptive scheduling. *Computers & Industrial Engineering*, 127:240–252.

Salido, M. A., Escamilla, J., Giret, A., and Barber, F. (2016). A genetic algorithm for energy-efficiency in job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 85(5-8):1303–1314.

Shahgholi Zadeh, M., Katebi, Y., and Doniavi, A. (2019). A heuristic model for dynamic flexible job shop scheduling problem considering variable processing times. *International Journal of Production Research*, 57(10):3020–3035.

Shen, X. N. and Yao, X. (2015). Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems. *Information Sciences*, 298:198–224.

Shoraneh, K. M. and Kazuhiro, S. (2019). On optimal dynamic pegging in rescheduling for new order arrival. *Computers & Industrial Engineering*, 136:46–56.

Sun, L., Lin, L., Gen, M., and Li, H. (2019). A hybrid cooperative coevolution algorithm for fuzzy flexible job shop scheduling. *IEEE Transactions on Fuzzy Systems*, 27(5):1008–1022.

Tang, D., Dai, M., Salido, M. A., and Giret, A. (2016). Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. *Computers in Industry*, 81:82–95.

Tay, J. C. and Ho, N. B. (2008). Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473.

Vepsalainen, A. P. and Morton, T. E. (1987). Priority rules for job shops with weighted tardiness costs. *Management Science*, 33(8):1035–1047.

Wang, H., Jiang, Z., Wang, Y., Zhang, H., and Wang, Y. (2018a). A two-stage optimization method for energy-saving flexible job-shop scheduling based on energy dynamic characterization. *Journal of Cleaner Production*, 188:575–588.

Wang, Q., Liu, F., and Li, C. (2013). An integrated method for assessing the energy efficiency of machining workshop. *Journal of Cleaner Production*, 52:122–133.

Wang, S., Wang, X., Yu, J., Ma, S., and Liu, M. (2018b). Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of Cleaner Production*, 193:424–440.

Wu, X. and Sun, Y. (2018). A green scheduling algorithm for flexible job shop with energy-saving measures. *Journal of Cleaner Production*, 172:3249–3264.

Yska, D., Mei, Y., and Zhang, M. (2018). Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In *European Conference on Genetic Programming*, pages 306–321. Springer.

Zandieh, M., Khatami, A., and Rahmati, S. H. A. (2017). Flexible job shop scheduling under condition-based maintenance: improved version of imperialist competitive algorithm. *Applied Soft Computing*, 58:449–464.

Zhang, F., Mei, Y., and Zhang, M. (2018). Genetic programming with multi-tree representation for dynamic flexible job shop scheduling. In *Australasian Joint Conference on Artificial Intelligence*, pages 472–484. Springer.

Zhang, R. and Chiong, R. (2016). Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. *Journal of Cleaner Production*, 112:3361–3375.

Zhang, Z., Tang, R., Peng, T., Tao, L., and Jia, S. (2016). A method for minimizing the energy consumption of machining system: integration of process planning and scheduling. *Journal of Cleaner Production*, 137:1647–1662.

Zhu, Z. and Zhou, X. (2020). An efficient evolutionary grey wolf optimizer for multi-objective flexible job shop scheduling problem with hierarchical job precedence constraints. *Computers & Industrial Engineering*, 140:106280.