Genetic Programming for Evolving Due-Date Assignment Models in Job Shop Environments

Su Nguyen su.nguyen@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, Wellington, New Zealand

Mengjie Zhang mengjie.zhang@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, Wellington, New Zealand

Mark Johnston mark.johnston@msor.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, Wellington, New Zealand

Kay Chen Tan eletankc@nus.edu.sg
Department of Electrical and Computer Engineering, National University of Singapore, Singapore

doi:10.1162/EVCO_a_00105

Abstract

Due-date assignment plays an important role in scheduling systems and strongly influences the delivery performance of job shops. Because of the stochastic and dynamic nature of job shops, the development of general due-date assignment models (DDAMs) is complicated. In this study, two genetic programming (GP) methods are proposed to evolve DDAMs for job shop environments. The experimental results show that the evolved DDAMs can make more accurate estimates than other existing dynamic DDAMs with promising reusability. In addition, the evolved operation-based DDAMs show better performance than the evolved DDAMs employing aggregate information of jobs and machines.

Keywords

Genetic programming, job shop, due-date assignment, hyper-heuristics.

1 Introduction

Job shop scheduling (JSS) has been one of the most popular topics in the scheduling literature due to its complexity and applicability in real-world situations. A large number of studies on JSS have focused on sequencing decisions that determine the order in which waiting jobs are processed on a set of machines (or resources) in a manufacturing system (shop). However, sequencing is only one of several steps in the scheduling process (Ahmed and Fisher, 1992). One of the other important activities in JSS is due-date assignment (DDA), sometimes referred to as estimation of job flowtimes (EJF). This activity arises when a manager need to promise a delivery date to a customer (Bookbinder and Noor, 1985). The objective of DDA is to determine the due dates for arriving jobs by estimating the job flowtimes (the time from the arrival until the completion of the job), and therefore DDA strongly influences the delivery performance, that is, the ability to meet promised delivery dates, of a job shop (Cheng and Gupta, 1989). In practice, both

Manuscript received: February 6, 2012; revised: July 13, 2012, February 22, 2013, and April 11, 2013; accepted: April 22, 2013.

early and tardy jobs are undesirable (Cheng and Jiang, 1998) since early jobs increase the inventory costs (e.g., storage, insurance) while tardy jobs result in penalties, such as loss of customer goodwill and damaged reputation (Hino et al., 2005). In addition, accurate flowtime estimates (Sabuncuoglu and Comlekci, 2002) are needed for better management of shop floor activities, evaluation of the shop performance, and leadtime comparison, etc.

Many due-date assignment models (DDAMs) have been proposed in the job shop literature. The traditional DDAMs focus on exploiting the shop and job information to make a good flowtime estimation. Most of the early DDAMs are based on linear combinations of different terms (variables) and the coefficients of the models are then determined based on simulation results. Regression (linear and nonlinear) has been used very often in order to help find the best coefficients for the models employed (Ragatz and Mabert, 1984; Fry et al., 1989; Vig and Dooley, 1993; Veral, 2001; Sabuncuoglu and Comlekci, 2002; Sha et al., 2007; Joseph and Sridharan, 2011). Since the early 1990s, artificial intelligence methods have also been applied to deal with due-date assignment problems, for example, neural networks (Philipoom et al., 1994; Sha and Hsu, 2004; Patil, 2008), decision trees (Ozturk et al., 2006), regression trees (Sha and Liu, 2005), and a regression-based method with case-based tuning (Sha et al., 2007).

Even though experimental results with these DDAMs are promising, some limitations are still present. First, since a job can include several operations representing the processing steps of that job at particular machines, the operation-based flowtime estimation (OFE) method (Sabuncuoglu and Comlekci, 2002), which utilizes the detailed job, shop, and route information for operations of jobs, can help improve the quality of the prediction. However, this OFE method depends strongly on the determination of a large number of coefficients, which is not an easy task. Thus, there is a need to create a dynamic OFE method similar to dynamic total work content (DTWK), dynamic processing plus waiting (DPPW; Cheng and Jiang, 1998), and ADRES (Baykasoglu et al., 2008) to overcome this problem by replacing the coefficients with more general aggregate terms (job characteristics and states of the system). Second, there are no studies on the reusability of the DDAMs in the JSS literature, so it is questionable whether the models can be applied when there are changes in the shop without major revisions. Finally, various relevant factors need to be considered in order to make a good estimation of flowtime, which makes the design of a new DDAM a time-consuming and complicated task.

Genetic programming (GP; Koza, 1992) is an evolutionary computation method and it has been applied to evolve/train programs that are able to solve difficult computational problems. We see that GP is also a good candidate approach to helping overcome the three limitations discussed above because (1) the DDAMs can be easily represented by GP, (2) DDAMs can be automatically evolved/trained on different shop environments to provide some generality for the evolved DDAMs, and (3) the DDAMs evolved by GP can be interpreted.

This paper aims to develop a new approach that employs GP to evolve dynamic DDAMs for job shop environments. We expect the evolved DDAMs to outperform the existing models in terms of mean absolute percentage error and to be reusable for new (unseen) job shop simulation scenarios. Two types of DDAMs considered in this study are aggregate due-date assignment models (ADDAMs) and operation-based due-date assignment models (ODDAMs). The difference between these two models is that ADDAMs employ the aggregate information from jobs, machines, and the shop to

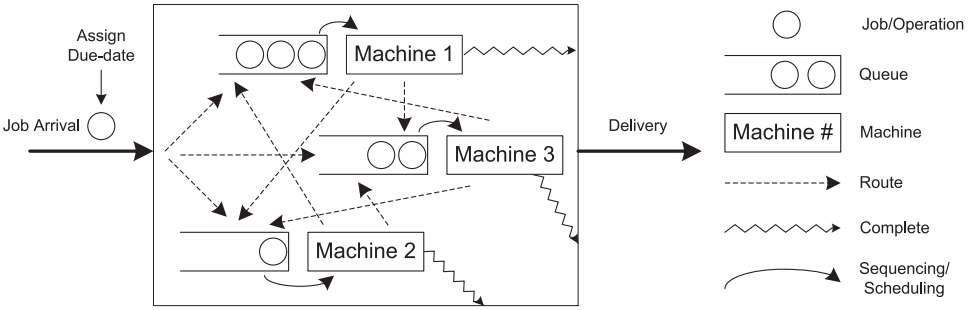


Figure 1: Job shop scheduling (shop with three machines).

predict the due date, while ODDAMs indirectly predict the due date by estimating the flowtime of each operation. The objectives for this study are:

1. Developing GP methods to automatically evolve reusable ADDAMs and ODDAMs for the job shop environment.
2. Comparing the evolved DDAMs obtained from the two GP methods with existing DDAMs.
3. Analyzing the proposed GP methods and the evolved DDAMs to understand how these models can estimate flowtime accurately.

The rest of this paper is organized as follows. In the next section, the background on JSS and due-date assignment methods are given. Automatic heuristic design methods are also reviewed. The methodology is developed in Section 3 and the experimental setting is presented in Section 4. The experimental results and the comparison of DDAMs are provided in Section 5. Analysis of the proposed algorithm and evolved DDAMs is presented in Section 6. Further investigation into sophisticated dispatching rules is shown in Section 7. Finally, Section 8 gives some conclusions from this research and directions for future studies.

2 Literature Review

This section introduces the terminology of JSS used in this study and traditional methods for due-date assignment. Since the focus of this study is on automatic design of DDAMs, a brief literature review of hyper-heuristics for heuristic generation is included.

2.1 Job Shop Scheduling

In the JSS problem, a number of jobs are to be processed, each including one or more operations to be performed in a specified sequence on specified machines and requiring certain amounts of time (Ramasesh, 1990). In practical situations, jobs can arrive at random over time and the processing times of these jobs are not known prior to their arrivals. There are many related decisions that need to be made for jobs and machines in the shops such as due-date assignment, job order release, and job scheduling. In this study, we focus on due-date assignment and job scheduling decisions; job release is simplified by immediately releasing jobs to the shop upon arrival. An example of a job shop is shown in Figure 1. In this figure, the due-date will be assigned to a newly arriving job by some DDAM. Then, the job will be released to the shop and

Table 1: Performance measures of DDAMs.

Mean absolute percentage error	$\texttt{MAPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{\lvert e_j\rvert}{f_j}$
Mean percentage error	$\texttt{MPE} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}\frac{e_j}{f_j}$
Mean absolute error	$\texttt{MAE} = \frac{\sum_{j\in\mathbb{T}}\lvert e_j\rvert}{\lvert\mathbb{C}\rvert}$
Standard deviation of lateness	$\texttt{STDL} = \sqrt{\frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}}(e_j-\bar{e})^2}$
Percent tardiness	$\%\texttt{T} = 100\times\frac{\lvert\mathbb{T}\rvert}{\lvert\mathbb{C}\rvert}$
Mean flowtime	$\texttt{MF} = \frac{\sum_{j\in\mathbb{C}} f_j}{\lvert\mathbb{C}\rvert}$

processed at the predetermined machines. If a job is transferred to a machine when it is busy, that job will have to wait in the corresponding queue. Meanwhile, when a machine completes a job (or operation), the next job in the queue will be selected based on some sequencing/scheduling (dispatching) rule (refer to Panwalkar and Iskander, 1977; Pinedo, 2008, for comprehensive reviews of dispatching rules).

2.2 Due-Date Assignment

Due-date assignment decisions are made whenever jobs (customer orders) are received from customers. Good due-date assignments are needed in order to maintain high delivery performance (delivery speed and delivery reliability). Generally, due dates can be set: (1) exogenously, or (2) endogenously (Cheng and Gupta, 1989; Ramasesh, 1990). In the former case, due dates are decided by independent agencies (sellers, buyers). In this study, we only focus on the second case, in which the due dates are internally set based on the characteristics of the jobs and shop (Ramasesh, 1990), to improve the delivery performance of job shops. Basically, the due date of a new job is calculated as:

$$d_j = r_j + \hat{f}_j \tag{1}$$

where d_j is the due date, r_j is the release time of the job (in our study, the release time is the arrival time of the job since the job is released to the shop immediately), and $\hat{f}_j$ is the estimated (predicted) flowtime of job j. The task of a DDAM is to calculate a value for $\hat{f}_j$. In the ideal case, we want the calculated due date d_j to be equal to the completion time of the job C_j. The performance (with respect to missing the due date) is normally measured by the error (lateness) between the completion time and due date $e_j = C_j - d_j = f_j - \hat{f}_j$, where f_j is the actual flowtime.

Some criteria to evaluate the performance of DDAMs (Cheng and Jiang, 1998; Baykasoglu et al., 2008) in the JSS literature are shown in Table 1. In this table, $\mathbb{C}$ is the set of jobs collected from the simulation runs to calculate the performance measures, e_j is the lateness of job j, $\bar{e}$ is the mean lateness, and $\mathbb{T}$ is the set of tardy jobs ($C_j - d_j > 0$). MAPE and MAE measure the accuracy of the flowtime estimation. Smaller MAPEs or MAEs indicate that the DDAM can make better predictions. MPE measures the bias of the DDAM. If the DDAM results in a negative (positive) MPE, it means that the DDAM tends to overestimate (underestimate) the due date. STDL measures the delivery reliability of the DDAM. Smaller STDL indicates that the estimated due dates are more reliable. Another delivery performance measure is %T, which shows the percentage of jobs that fail to meet the due date. Finally, MF measures the delivery speed of the scheduling system.

Many DDAMs have been proposed in the JSS literature. The DDAMs in early studies are mainly based on creating a simple model that employs aggregate information

from the new job and the shop. Examples of these methods are total work content (TWK) where $d_j = r_j + kp_j$, number of operations (NOP) where $d_j = r_j + km_j$, and processing plus waiting (PPW) where $d_j = r_j + p_j + km_j$. In these methods, p_j and m_j are the total processing time and number of operations of job j, and k is a coefficient that needs to be determined. Other more sophisticated models have also been proposed that incorporate more information of jobs and the shop to make better predictions of flowtimes. These include job in queue (JIQ), work in queue (WIQ), total work and number of operations (TWK + NOP), response mapping rule (RMR), and operations-based flowtime estimation (OFE). Comparisons of these DDAMs (Ragatz and Mabert, 1984; Fry et al., 1989; Philipoom et al., 1994; Chang, 1996; Cheng and Jiang, 1998; Sabuncuoglu and Comlekci, 2002) show that the DDAMs which employ more useful information can lead to better performance. However, the main drawback of these methods is that they depend strongly on the determination of the corresponding coefficients for factors used in the prediction models. The most popular method to determine the coefficients is using linear regression models.

Because of the complexity and stochastic nature of dynamic job shops, nonlinear models will be needed (Philipoom et al., 1994), which make it computationally more expensive to solve for regression methods. For this reason, many artificial intelligence methods have been applied to solve this problem. Philipoom et al. (1994) proposed a neural network (NN) method for due-date assignment and showed that their NN method can outperform conventional methods and nonlinear models. Also in this direction, Sha and Hsu (2004) developed an NN method for due-date assignment in a wafer fabrication system that showed very good results. Patil (2008) enhanced the NN method by using ensemble learning and bagging/boosting concepts. A genetic algorithm (GA) was also employed to search for NN architectures that develop a parsimonious model of flowtime prediction. The computational results showed that the enhanced NN method outperformed other simple NN methods. Although different shop environments were considered, the paper only focused on training and testing the NNs on the same shop environments, and the reusability of the obtained NNs on unseen different shop environments have not been examined. Baykasoglu and Gocken (2009) applied gene expression programming (GEP) to evolve a symbolic regression model for DDA in a specific multistage job shop. The results showed that the evolved DDAM was better than the previously proposed DDAMs. However, only aggregate information from the shop was employed to estimate the job flowtimes and detailed information of operations was not considered. Also, similar to Patil (2008), there is no analysis on the reusability of the evolved DDAMs. Other data-mining methods such as decision trees (Ozturk et al., 2006), regression trees (Sha and Liu, 2005), and a regression-based method with case-based tuning (Sha et al., 2007) have also been proposed, showing very promising results.

Although the DDAMs described above have shown good results in simulation studies, determining good model coefficients is not an easy task, especially with the dynamic changes in the shop floor. To overcome this problem, some dynamic DDAMs have been proposed, in which the coefficients are adjusted based on the information of the new job and states of the systems. Cheng and Jiang (1998) proposed dynamic total work content (DTWK) and dynamic processing plus waiting (DPPW) by applying Little's law (Little, 1961) from queueing theory:

- DTWK:

$$d_j = r_j + \max\left\{1, \frac{N_{s_t}}{\lambda \mu_p \mu_g}\right\} \sum_{i=1}^{m_j} p_{ji} \tag{2}$$

- DPPW:

$$d_j = r_j + \sum_{i=1}^{m_j} p_{ji} + \frac{N_{q_t} m_j}{\lambda \mu_g} \tag{3}$$

where N_{s_t} is the number of jobs in the shop at the moment a new job arrives, λ is the average arrival rate of jobs, μ_p and μ_g are respectively the average processing time and average number of operations, p_{ji} is the processing time of the ith operation of job j, and N_{q_t} is the total number of jobs in the queue of each machine.

In another study, Baykasoglu et al. (2008) developed ADRES, a new dynamic DDAM, which uses a simple smoothing method to estimate the wait time of the next job. In this model, the due date can be calculated as follows (assuming zero transportation times):

$$d_j = r_j + \sum_{i=1}^{m_j} w'_{ji} + \bar{p}_j \tag{4}$$

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

Previous research has shown that DTWK, DPPW, and ADRES are very effective DDAMs as compared to static regression-based DDAMs. Another advantage of these DDAMs is that no preliminary runs to obtain the parameter estimations are necessary. Therefore, they have been used as good candidates for comparison purposes (Sha and Liu, 2005; Baykasoglu et al., 2008; Baykasoglu and Gocken, 2009, 2011; Vinod and Sridharan, 2011).

One of the problems with the dynamic DDAMs is that they are still mainly based on the aggregate information of jobs and the shop to make the prediction and ignore the detailed operation information, while it is shown that this information can help improve the quality of the prediction (Sabuncuoglu and Comlekci, 2002). However, development of such operation-based DDAMs would be very difficult, since these models involve many different factors (variables). Thus, there is a need to have an automatic method to facilitate the design of such models. Also, there is no previous study on the reusability of the proposed models in the JSS literature, so it is questionable whether the proposed/evolved models can be applied, without major revisions, when there are changes in the shop.

2.3 Automatic Design of Heuristics

Hyper-heuristics (HH; for heuristic generation) are a new methodology which aims at automating the design and tuning of heuristic methods to solve hard computational search problems (Burke et al., 2010b). In order to generate a new heuristic, the HH framework must be able to combine various small components (normally common statistics or operators used in preexisting heuristics), and these heuristics are trained on a training set and evolved to become more effective. Because genetic programming (GP) is able to represent and evolve complex programs or rules, GP has become popular in the field of HH and the subfield is known as genetic programming-based hyper-heuristics

(GP-HH; Burke et al., 2009). Many GP-HH methods have been proposed in the literature. Fukunaga (2002, 2008) used GP to evolve variable selection heuristics in each application of a local search algorithm for the satisfiability (SAT) problem. The experimental results showed that the evolved heuristics are very competitive when compared with other standard heuristics. Burke et al. (2007a, 2007b, 2011) proposed a GP-HH framework to evolve construction heuristics for online bin packing. The results suggested that human designed heuristics can be rediscovered by GP. Keller and Poli (2007a, 2007b) proposed an effective grammar-based linear genetic programming method to solve the traveling salesman problem. Search performance and resource utilization are also investigated in their paper. Bader-El-Den et al. (2009) introduced a sophisticated grammar-based GP for evolving time-tabling heuristics. Their GP-HH is based on a grammar derived from a collection of graph coloring heuristics that have previously been shown effective in constructing time tables. Even though the proposed GP-HH produced competitive results when compared with some existing search methods in the literature, it was not shown whether the evolved heuristics can be reused on new problem instances. More recently, Burke et al. (2010a) proposed a GP-HH method to learn construction heuristics for two-dimensional strip packing problems. The evolved heuristics were very promising when they were compared with the best-fit heuristic and some meta-heuristic methods. They also provided some interesting insights on the generality of the evolved heuristics. A thorough review of applications of GP-HH is given in Burke et al. (2009).

Recently, GP-HH has been applied in several studies to evolve dispatching rules for JSS problems. Jakobovic and Budin (2006) applied GP to evolve dispatching rules for both single machine and job shop environments. The results for the single machine environment showed that the evolved rules were better than existing rules. For the job shop environment, a meta-algorithm is developed to show how the evolved rules are used to construct a schedule. This study also proposed an interesting way to provide adaptive behaviors for the evolved rules. They proposed a GP-3 system that evolves three components: a discriminant function and two dispatching rules. The discriminant function aims at identifying whether the considered machine to be scheduled is a bottleneck or not. This function serves as a binary classifier. Based on the classification decision obtained from the discriminant function, one of two dispatching rules is selected to sequence jobs in the queue of that machine. The results show that this GP-3 system performed better than traditional GP with a single tree representing a dispatching rule. An extension of this work was done in Jakobovic et al. (2007) to deal with parallel machines with different speeds and also showed very good results. Geiger et al. (2006) presented a learning system that combines GP with a simulation model of an industrial facility. Both static and dynamic environments are investigated in this study and the results showed that the evolved rules are very promising. The paper also proposed a method to learn dispatching rules for multiple machine problems in which GP evolves multiple trees simultaneously with modified crossover and mutation operators. Comparison with the optimal rule in a simple two-machine environment showed that the evolved rules are rather competitive.

Tay and Ho (2008) proposed a GP system to evolve dispatching rules for a multi-objective job shop environment. The multi-objective problem was converted into a single objective problem by linearly combining all objectives. The proposed GP program can be considered as a priority function which is used to calculate the priority of operations in the queue of a machine based on a set of static and dynamic variables. The set of instances was randomly generated and it was shown that the evolved dispatching

rules outperform other simple dispatching rules. Hildebrandt et al. (2010) reexamined this system under different dynamic job shop scenarios and showed that the rules evolved by Tay and Ho (2008) are only slightly better than the earliest release date (ERD) rule and quite far away from the performance of the shortest processing time (SPT) rule. They explained that the poor performance of these rules is caused by the use of the linear combination of different objectives and the fact that the randomly generated instances cannot effectively represent the situations that happened in a long term simulation. For that reason, Hildebrandt et al. (2010) evolved dispatching rules by training them on four simulation scenarios (10 machines with two utilization levels and two job types) and only aimed at minimizing mean flowtime. The experimental results indicated that the evolved rules were quite complicated but effective when compared with other existing rules. Moreover, these evolved rules were also robust when tested with another environment (50 machines and different processing time distributions).

It is noted that most works on GP for scheduling problems only aim to evolve dispatching rules and simplify DDA by applying simple DDAMs (e.g., TWK). However, because DDA strongly influences the delivery performance, it needs to be considered seriously in order to ensure an effective scheduling system. The objective of this study is to focus on the use of GP for automatic design of DDAMs. Baykasoglu and Gocken (2009) can be considered to be the most relevant work in this research direction. However, their study only focused on a special shop, and the reusability (under different shop conditions) of the evolved models was not considered. In addition, they did not take advantage of the detailed job, shop, and route information for operations of jobs to enhance the accuracy of flowtime estimates. The operation-based flowtime estimation will be the key issue to be investigated in our work. We also examine the reusability of evolved DDAMs to see whether the evolved DDAMs are able to perform well on unseen scenarios. In the next section, the proposed GP methods are described to show how GP can be used to evolve ADDAMs and ODDAMs for job shop environments.

3 GP for Evolving DDAMs

In this section, we describe two GP methods, GP-ADDAM and GP-ODDAM (Nguyen et al., 2012), to evolve ADDAMs and ODDAMs, respectively. First, the representation and evaluation scheme are discussed. Then, a fitness function is provided to measure the performance of the evolved DDAMs. Finally, the proposed GP procedure to evolve DDAMs is described.

3.1 Representation

The purpose of the proposed GP-ADDAM and GP-ODDAM is to evolve dynamic ADDAMs and ODDAMs that estimate job flowtimes (i.e., due dates by using Equation (1)) by employing information from jobs and the shop similar to DTWK and DPPW. In this case, we use tree-based GP (Koza, 1992) to create mathematical combinations of these pieces of information in each GP individual. For this reason, the function set will consist of standard mathematical operators $+$, $-$, $\times$, and protected division %, along with a conditional function `If` to allow GP to evolve sophisticated DDAMs. The protected division function % returns a value of 1 when division by 0 is attempted. Function `If` includes three arguments, and if the value from the first argument is greater than or equal to zero, `If` will return the value from the second argument; otherwise `If` will return the value from the third argument. Since ADDAMs and ODDAMs need different types of information, GP-ADDAM and GP-ODDAM will use different terminal sets, as shown in Table 2. In this table, the first five terminals are the same for the two

Table 2: Terminal sets for GP-ADDAM and GP-ODDAM (ψ is the new job, ϕ is the considered operation in GP-ODDAM, and δ is the machine that will process ϕ).

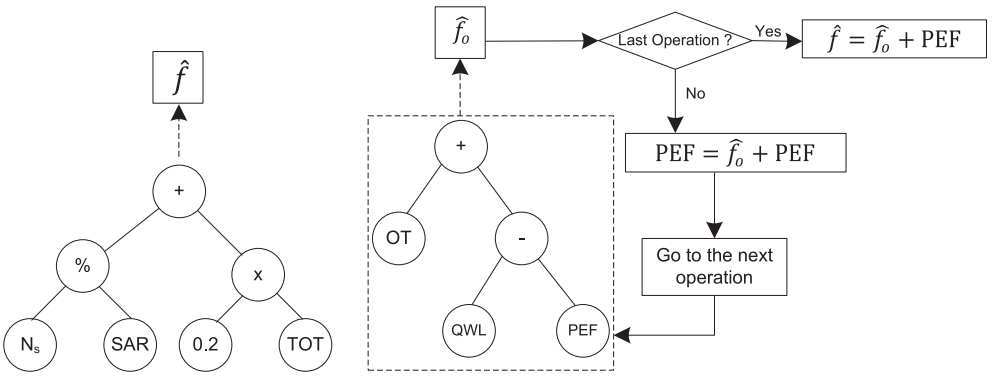
GP-ADDAM		GP-ODDAM	
N		Number of jobs in the shop	
SAR		Sampled arrival rate	
NO		Number of operations of job ψ	
M		Number of machines	
#		Random number from 0 to 1	
TAPR	Total average processing time of job in queues of machines that ψ will visit	APR	Average processing times of jobs in the queue of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

proposed GP methods. The next eight terminals are variables that characterize the state of operations/machines for GP-ODDAM and their aggregate counterparts for GP-ADDAM. The last terminal of each method provides extra information to estimate the flowtime. While SL provides the information of previous arriving jobs, PEF helps estimate the changes of the system through the period of time the new job spends in the system (more details are given in Section 3.2). SOTR and SAPR are calculated based on the 20 previous jobs processed at machine δ. On the other hand, SAR and SL are calculated based on the arrivals of the last 100 jobs and 20 jobs, respectively.

3.2 Evaluation

An example of how an individual in GP-ADDAM is evaluated is shown in Figure 2(a). In this method, a GP individual represents a mathematical function and the output of this function is the estimated flowtime $\hat{f}$ of the new job. The information used in this function is extracted from the new job and machines in the shop.

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



(a) GP-ADDAM (b) GP-ODDAM

Figure 2: DDAM evaluation scheme.

flowtime $\hat{f}_o$ of each operation of the new job, starting from the first operation. When $\hat{f}_o$ is obtained, a condition is checked to see whether the operation being considered is the last operation. If it is not the last operation of the new job, $\hat{f}_o$ will be used to update the partial estimated flowtime (PEF), which will also be used as a terminal in the GP individual. Then, the GP individual is applied to estimate the flowtime for the next operation. In the case that the flowtime of the last operation has been estimated, $\hat{f}_o$ will be added to the current PEF to obtain the estimated flowtime $\hat{f}$. The evaluation scheme for GP-ODDAM is shown in Figure 2(b) (noting that only the tree in the figure is evolved by the GP). The use of PEF (initially zero for the first operation) in the terminal set of GP-ODDAM also provides DDAMs a chance to predict the changes of the system, assuming that the partial estimated flowtime is predicted well.

3.3 Genetic Operators

Traditional genetic operators are employed by the proposed GP methods. For crossover, the GP system uses subtree crossover (Koza, 1992), which creates new individuals for the next generation by randomly recombining subtrees from two selected parents. Meanwhile, mutation is performed by subtree mutation (Koza, 1992), which randomly selects a node of a chosen individual and replaces the subtree rooted at that node by a new randomly generated subtree. For reproduction, an individual is selected from the population by the selection mechanism (e.g., tournament selection) and copied to the population of the next generation. In each generation of GP, the applications of each genetic operator are governed by the probability assigned to each type of genetic operator and referred to as crossover rate, mutation rate, and reproduction rate, respectively.

3.4 Fitness Function

As discussed in Section 2.2, the performance of a DDAM can be measured in many different ways, which indicate the delivery accuracy and delivery reliability. In this study, we will use `MAPE` to measure the quality of evolved DDAMs because it is a good indicator for both delivery accuracy and delivery reliability. A discrete-event simulation model of a job shop was implemented in order to evaluate the evolved DDAMs. In this

model, the interarrival times of jobs, the processing times, and route information of jobs will follow some specified probability distributions. Upon the arrival of a job j, the DDAM will be applied to estimate the flowtime $\hat{f}_j$ of that job. The error e_j of this estimation is recorded when job j leaves the system and the errors of all recorded jobs will be used to calculate MAPE as shown in Table 1. Since we want to evolve DDAMs that can be reused on unseen conditions, the quality of the evolved DDAMs will be measured based on their performance on a number of simulation scenarios $\mathbb{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$, which represent different shop characteristics. For a simulation scenario $\mathcal{S}_k$, the quality of a DDAM p_i is indicated by $\texttt{MAPE}_{p_i}^{\mathcal{S}_k}$. The fitness value of p_i is calculated as follows:

$$\text{fitness}(p_i) = \frac{1}{K}\sum_{k=1}^{K} \texttt{MAPE}_{p_i}^{\mathcal{S}_k}. \tag{5}$$

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

3.5 Evolution of DDAMs

Algorithm 1 shows how GP is used to evolve DDAMs in both GP-ADDAM and GP-ODDAM. A variety of simulation scenarios will be employed in this algorithm to provide the evolved (trained) DDAMs better generality, but it should be noted that a large number of scenarios also increases the computation time of the GP systems. The evolutionary process will be terminated when the maximum generation is reached and the algorithm will return the best found DDAM p^*. It should be noted that GP-ADDAM and GP-ODDAM use the same algorithm, but the terminals used by these two methods are different, since they have a different focus (as mentioned in Sections 3.1 and 3.2).

4 Experimental Setting

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

4.1 Job Shop Simulation Environment

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

- Number of machines
- Utilization
- Arrival process
- Distribution of processing time
- Distribution of number of operations (route length)

The number of machines will be the main factor that shows the scale of the shop; this may also influence the complexity of the JSS decisions. Utilization, on the other hand, indicates the congestion level of machines (and the shop). The performance of the JSS decisions under different utilization levels are of interest in most research in

Algorithm 1 General GP algorithm for GP-ADDAM and GP-ODDAM

load simulation scenarios $\mathbb{S} \leftarrow \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_K\}$;

randomly initialize the population $\mathbb{P} \leftarrow \{p_1, p_2, \ldots, p_{\text{popsize}}\}$;

$p^* \leftarrow \text{null}$ and $\text{fitness}(p^*) = +\infty$;

$generation \leftarrow 0$;

while $generation \leq maxGeneration$ **do**

  foreach $p_i \in \mathbb{P}$ **do**

    foreach $\mathcal{S}_k \in \mathbb{S}$ **do**

      calculate $\texttt{MAPE}_{p_i}^{\mathcal{S}_k}$

    end

    evaluate $\text{fitness}(p_i)$ by using equation (5) ;

    if $\text{fitness}(p_i) < \text{fitness}(p^*)$ **then**

      $p^* \leftarrow p_i$;

      $\text{fitness}(p^*) \leftarrow \text{fitness}(p_i)$;

    end

  end

  $\mathbb{P} \leftarrow$ apply genetic operators to $\mathbb{P}$;

  $generation \leftarrow generation + 1$

end

return p^*;

the JSS literature. The arrival process, distribution of processing times, and number of operations are factors that directly influence the difficulty of JSS decisions.

In our experiments, we employ a symmetrical (balanced) job shop model in which each operation of a job has equal probability to be processed at any machine in the shop (a job visits each machine at most once). Therefore, machines in the shop expect to have the same level of congestion in long simulation runs. This model has been used very often in the JSS literature (Chang, 1996; Cheng and Jiang, 1998; Sabuncuoglu and Comlekci, 2002; Land, 2004; Hildebrandt et al., 2010). Based on the discussion above, the scenarios for training and testing of DDAMs are designed and shown in Table 3.

In these experiments, without loss of generality, the processing times are randomly generated based on a specific distribution with mean equal to 1, and the arrival of jobs will follow a Poisson process with the arrival rates adjusted based on the utilization

Table 3: Training and testing scenarios.

Factor	Training	Testing
Number of machines	4, 6	4, 5, 6, 10, 20
Utilization	70%, 80%, 90%	60%, 70%, 80%, 90%, 95%
Distribution of processing time	Exponential	Exponential, Erlang-2, uniform
Distribution of number of operations	Missing	Missing, full

Table 4: Parameters of the proposed GP systems.

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

level. For the distribution of number of operations, the *missing* setting is used to indicate that the number of operations will follow a discrete uniform distribution from 1 to the number of machines. Meanwhile, the *full* setting indicates the case that each job will have its number of operations equal to the number of machines in the shop. In each replication of a simulation scenario, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the thousandth job is considered as the warmup time and the information collected from the next 5,000 completed jobs (set $\mathbb{C}$ in Section 2.2) is used to evaluate the performance of DDAMs.

In the training stage, since the simulation is very time-consuming, we only perform one replication for each scenario. There are $(2 \times 3 \times 1 \times 1) = 6$ simulation scenarios used to evaluate the performance of the evolved DDAMs. For testing, the best DDAM p^* obtained from a run of GP is applied to $(5 \times 5 \times 3 \times 2) = 150$ simulation scenarios and 30 simulation replications are performed for each scenario; therefore, we need $150 \times 30 = 4{,}500$ simulation replications to test the performance of p^*. The use of a large number of scenarios and replications in the testing stage will help us confirm the quality and reusability of the evolved DDAMs. For the shop floor level, first in first out (FIFO) is used as the dispatching rule to sequence jobs in queues of machines. By using FIFO, the earliest job that joins the queue of the machine will be processed first. We adopt FIFO in this study because it is one of the the most popular dispatching rules in the scheduling literature.

4.2 GP Parameters

The GP system for evolving DDAMs is developed based on the ECJ20 library (Luke, 2009). The parameter settings of the GP system used in the rest of this study are shown in Table 4. The initial GP population is created using the ramped-half-and-half method (Koza, 1992). Tournament selection is used to select individuals for genetic operators. Normally, a tournament selection size from four to seven is used and we use a tournament size of five in this study in order to maintain a balance between diversity and the convergence of the proposed GP methods (Koza, 1992; Banzhaf et al., 1998). Parameters

in Table 4 are similar to those in other applications of GP (Koza, 1992; Banzhaf et al., 1998). Since the terminal set includes many different terminals, the mutation rate is set to 15% to provide sufficient genetic material through the evolution process of the proposed GP methods.

5 Results

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

5.1 Comparison of DDAMs

For each GP method, 30 independent runs are performed and the best ADDAMs and ODDAMs obtained from each run are recorded and compared with existing dynamic DDAMs (DTWK, DPPW, and ADRES). Tables 5 and 6 show the comparison between evolved DDAMs and other DDAMs on 150 testing scenarios. In these tables, $\langle \cdot, \cdot, \cdot \rangle$ is the number of evolved DDAMs that are significantly better (by t-test with a significance level of 0.05) than DTWK, DPPW, and ADRES, respectively. It is easy to see that the evolved DDAMs dominate other DDAMs in most scenarios. These experimental results indicate the effectiveness of the proposed GP methods for evolving DDAMs. It is also interesting to see that the evolved DDAMs have very good reusability, since the evolved DDAMs can provide superior performance even on unseen scenarios (e.g., with the *full* setting).

Among the five factors discussed in Section 4.1, the distribution of processing times seems to have less impact on the reusability of the evolved DDAMs. The use of processing times drawn from an exponential distribution provides a wide range of jobs that can also reflect the cases when processing times follow other distributions. For some scenarios with high utilization level, the *full* setting, and a large number of machines, the number of evolved DDAMs that can beat DPPW is smaller, especially when the processing times follow an exponential or an Erlang-2 distribution. Most evolved DDAMs cannot show superior performance compared to DPPW for these scenarios. This may be because DPPW is based on the steady-state performance of the system and can perform better in shops with less diverse jobs when the *full* setting is used and with high levels of utilization. Another reason is that the evolved DDAMs have not been trained on these extreme scenarios (this issue will be investigated in Section 6.2). In general, it should be noted that the existing dynamic DDAMs can only perform well with less diverse jobs, which is not always the case for job shops in real-world applications. On the other hand, the evolved DDAMs not only show their superiority compared to the existing dynamic DDAMs, but also show good reusability, which makes them more robust when applied to complicated job shops in real-world situations.

5.2 GP-ADDAM versus GP-ODDAM

Table 7 shows the p values of t-tests of the average $\mathtt{MAPE}_{p^*}^{\mathcal{S}_k}$ (from 30 simulation replications) over the testing scenarios between GP-ADDAM and GP-ODDAM. In this table, the highlighted values indicate that GP-ADDAM is not significantly different from GP-ODDAM, and other values show that GP-ODDAM is significantly better than GP-ADDAM (with p value smaller than .05). The results show that GP-ODDAM is significantly better than GP-ADDAM on most simulation scenarios, especially in the case where the *missing* setting for the number of operations is used. In the case that the *full* setting for the distribution of number of operations is used, GP-ODDAM is also significantly better than GP-ADDAM in most cases except for the scenarios with a high

Table 5: Comparing the evolved ADDAM with existing DDAMs.

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

Table 6: Comparing the evolved ODDAM with existing DDAMs.

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

Table 7: GP-ADDAM versus GP-ODDAM (p Values from t-Tests).

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

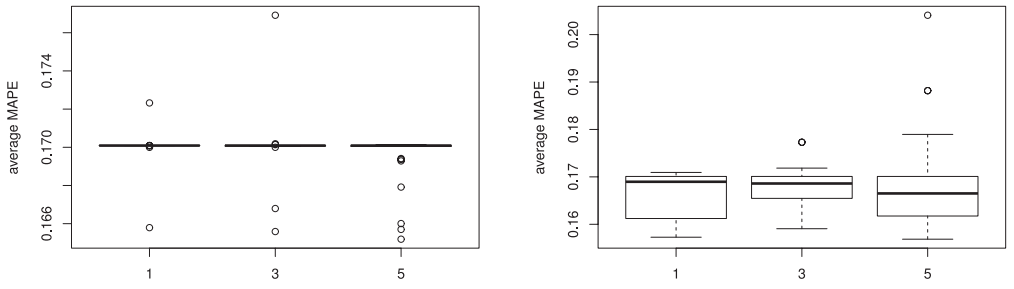
level of utilization (e.g., 95%) and large numbers of machines. These results suggest that the aggregate information of jobs used in ADDAMs is usually sufficient for estimating the flowtime of jobs when the shop is at a high congestion level and has a large number of machines. Also, ODDAMs may have difficulty estimating the operation flowtimes of later operations of jobs with a large number of operations. This observation, together with that observed in Section 5.1 for DPPW, indicates the importance of aggregate information for flowtime estimation in the cases of high utilization levels, less diverse jobs, and large numbers of machines. It suggests that systematic incorporation of information between ADDAM and ODDAM could enhance the accuracy of the flowtime estimation.

6 Analysis

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

6.1 Number of Simulation Replications

The number of simulation replications is normally an important parameter when we try to measure the performance of a stochastic system through simulation. Even though large numbers of simulation replications are essential for accurately measuring the performance of the stochastic systems, they will significantly increase the computational time of the proposed GP methods. Figure 3 shows the average $\texttt{MAPE}_{p^*}^{\mathcal{S}_k}$ from all testing scenarios obtained by the evolved DDAMs when different numbers of simulation replications are used for evaluating the evolved DDAMs. This figure shows that the use of more replications for training DDAMs does not appear to improve the



(a) GP-ADDAM (b) GP-ODDAM

Figure 3: Influence of the number of simulation replications on the performance of the proposed GP methods (the horizontal axis shows the number of simulation replications).

quality of the evolved DDAMs. An explanation for these results is that one replication is quite enough for training DDAMs, because the performance measure of the evolved DDAMs has been applied thousands of times in each simulation replication, which can characterize various situations in the dynamic systems. Also, accurately determining the performance measures of evolved DDAMs in the proposed GP methods is not as important as that in common simulation applications, because what we need is to find out which evolved DDAMs are better than the others (for individual selection in GP) and the performance measures from a single replication appear to be good enough for this purpose.

6.2 The Choice of Simulation Scenarios for Training DDAMs

As shown in Section 5.1, the evolved DDAMs have trouble beating DPPW in the scenarios with the *full* jobs. This problem may come from the fact that the evolved DDAMs have not been trained on these scenarios; therefore, they cannot handle these scenarios as well as the scenarios with the *missing* jobs. This raises the issue of how to choose suitable training scenarios in order to maximize the reusability of the evolved DDAMs without significantly increasing the computational time of the proposed GP methods. Tables 8 and 9 show the comparison between existing dynamic DDAMs and the evolved DDAMs trained on the same simulation scenarios in Table 3 but the *full* setting is used instead of the *missing* setting. The results in these tables suggest that the evolved DDAMs can only show their superiority on the scenarios with full setting and processing times following exponential and Erlang-2 distributions. This indicates that the evolved DDAMs do not have as good reusability as those trained with the *missing* jobs as shown in Tables 5 and 6. However, these evolved DDAMs can dominate DPPW in some scenarios with a small number of machines, utilization of 95% and *full* jobs, which cannot be achieved by the evolved DDAMs trained with *missing* jobs. Tables 10 and 11 show the performance of evolved DDAMs trained with both *full* and *missing* jobs (a total of 12 simulation scenarios used for training). These tables show that the evolved DDAMs in this case have better reusability than those evolved with *full* or *missing* alone. However, it is noted that these results are obtained by doubling the number of simulation scenarios, and therefore it also doubles the computational effort of the proposed GP methods. Figure 4 shows the average $\texttt{MAPE}_{p^*}^{\mathcal{S}_k}$ from all testing scenarios obtained by the evolved DDAMs when different training sets are employed. It is obvious that evolved DDAMs trained with full

Table 8: Performance of evolved ADDAMs trained with the *full* setting.

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

Table 9: Performance of evolved ODDAMs trained with the *full* setting.

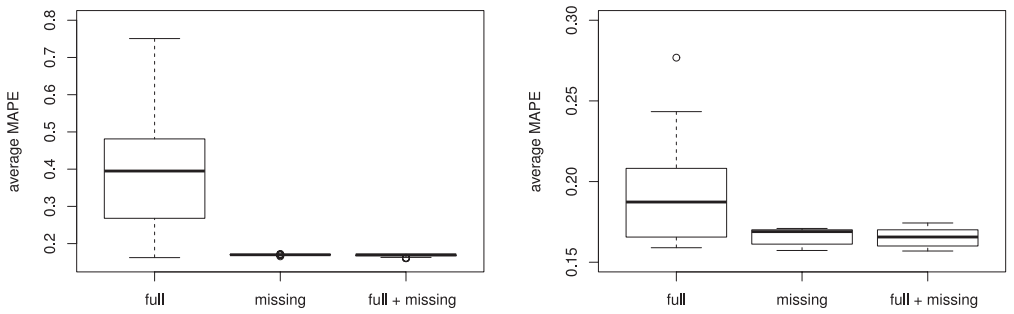
		Setting									
		Missing					Full				
		60%	70%	80%	90%	95%	60%	70%	80%	90%	95%
Exponential	4	⟨21, 25, 28⟩	⟨21, 0, 0⟩	⟨28, 29, 27⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩	⟨30, 28, 30⟩
	5	⟨22, 28, 30⟩	⟨25, 28, 30⟩	⟨28, 30, 30⟩	⟨27, 0, 0⟩	⟨30, 30, 30⟩	⟨30, 30, 30⟩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

Table 10: Performance of evolved ADDAMs trained with the *full + missing* setting.

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

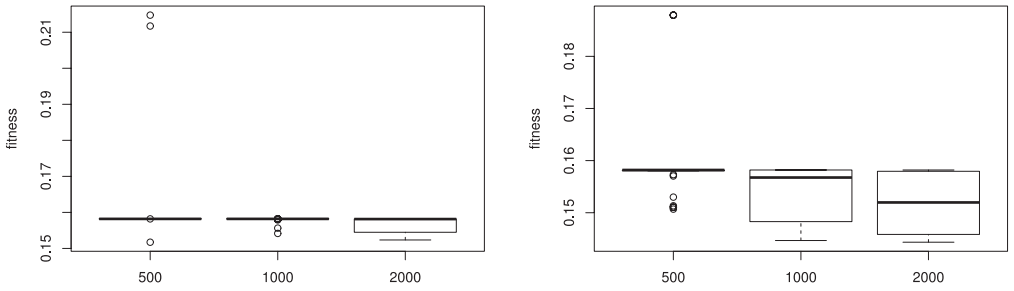
Table 11: Performance of evolved ODDAMs trained with the *full + missing* setting.

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



(a) GP-ADDAM (b) GP-ODDAM

Figure 4: Influence of training set on the performance of evolved DDAMs.



(a) GP-ADDAM (b) GP-ODDAM

Figure 5: Influence of population size on the performance of the proposed GP methods.

jobs are not as good as those trained with *missing* or *full + missing* jobs. It is also noted that the evolved DDAMs trained with *full + missing* jobs are not significantly better than those trained with *missing* jobs. The results in this section suggest that using missing jobs may be sufficient for evolving DDAMs with good reusability for these problems.

6.3 Population Size of the Proposed GP Methods

This section examines the influence of population size on the performance of the proposed GP methods. Figure 5 shows that a population size of 1,000 would be sufficient for the two proposed GP methods to explore the search space of DDAMs since population sizes more than 1,000 do not show significant improvement on the performance of the GP methods. Also, increasing the population size from 500 to 1,000 does not show significant improvements for GP-ADDAM, but it can significantly improve the performance of GP-ODDAM. This observation suggests that the search space of ODDAMs is more complicated than that of ADDAMs and requires a larger population size to ensure the diversity in the population.

6.4 Performance of Evolved DDAMs on Shops with a Bottleneck

Previous experiments have shown that the evolved DDAMs show very good results on symmetrical job shops. In this section, we examine the reusability of the evolved DDAMs in unbalanced shops. Table 12 and Table 13 show the performance of the

Table 12: Performance of evolved ADDAMs on shops with a bottleneck.

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

Table 13: Performance of evolved ODDAMs on shops with a bottleneck.

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

```
(+
  TOT
  (If(−N(+TLOT TQWL))
     (+TLOT TQWL)
     (+TQWL(∗TLOT 0.6615654)))
)
```

(a) $p^{\mathtt{ADDAM}}$

```
(If(∗(+(−SAR(−OT LOT))(−(+LOT OT) PEF))
(If(−(+PEF OT)(∗PEF(∗SAPR QWL)))(/(/OT N)(∗(+APR(−(∗SAPR LOT)PEF))(+OT QWL)))(+(∗QWL 0.2542787)SAR)))
  (If(∗(+(∗(+(−(/OT N)(−OT LOT))
  (−(∗SAPR LOT)PEF))(+(−SAR(−OT LOT))(−(+LOT OT)PEF)))(−LOT PEF))(∗PEF(∗(∗SAPR LOT)QWL)))
     (+(+OT QWL) LOT)
     (+OT QWL))
  (+OT QWL))
```

(b) $p^{\mathtt{ODDAM}}$

Figure 6: Best evolved DDAMs.

previous DDAMs (trained from symmetrical job shops) on the shop with a single bottleneck machine. The utilization values shown in these tables are the utilization values of the bottleneck machine. In these testing scenarios, the utilizations of non-bottleneck machines are 5% less than that of the bottleneck machine. In general, these results are similar to those in Table 5 and Table 6. The evolved DDAMs still provide superior performance as compared to DTWK, DPPW, and ADRES. Moreover, the performance of evolved ODDAMs on some scenarios with high utilization level and the *full* setting is better than that in Table 6 (more superior evolved ODDAMs). Perhaps, training DDAMs in different simulation scenarios has made the evolved DDAMs more robust than the existing DDAMs such as DTWK and DPPW, which depend on the variation of job flowtimes. This also suggests that operation-based DDAMs are more effective in these cases because they take into account detailed information of the machine that processes each operation of a job instead of using the aggregate information in ADDAMs.

6.5 Evolved DDAMs

In this section, we further examine the evolved DDAMs to explore useful patterns for the development of more effective DDAMs. The best evolved ADDAM $p^{\mathtt{ADDAM}}$ and ODDAM $p^{\mathtt{ODDAM}}$ in a set of evolved DDAMs obtained from 30 independent GP runs are shown in Figure 6. Both evolved DDAMs include the total processing times of jobs in queues and the processing time of the new job, that is, QWL + OT for ODDAM and

TQWL + TOT for ADDAM (when expanding the If expression in Figure 6(a)). This term is actually a good estimate of flowtime for a job with a small number of operations (for ADDAM) or for the first operation of a new job (for ODDAM). The main difference between these two evolved DDAMs is the use of conditional terms to decide which extra terms should be included in the estimation. For ODDAM, PEF, QWL, and LOT are used in the conditional term of function If. This DDAM shows that PEF is an important term to provide better flowtime estimations.

Table 14 shows $\text{MAPE}_{p^{\text{ODDAM}}}^{\mathcal{S}_k}$ and the t-test results between the p^{ODDAM} in Figure 6(b) and other DDAMs over 150 testing scenarios. In Table 14, the a, b, c, and d superscript notes are the indices to represent p^{ADDAM}, DTWK, DPPW, and ADRES, respectively. A superscript of a result in these tables shows the DDAMs that are not significantly different (by using t-tests) from the p^{ODDAM}. Meanwhile, a subscript shows the DDAMs that are significantly better than p^{ODDAM}, respectively. If an index is neither shown in the superscript nor the subscript, it means that p^{ODDAM} is significantly better than all other DDAMs. The tests are considered significant when the obtained p value is less than .05. For example, in the scenario with six machines, the *full* setting, utilization of 80%, and processing times follow exponential, 0.2025^{a} shows that there is no significant difference between p^{ODDAM} and p^{ADDAM}, and p^{ODDAM} is significantly better than DTWK, DPPW, and ADRES. It is easy to see that the evolved DDAMs dominate other DDAMs in most scenarios. In a few specific cases, p^{ADDAM}, DPPW, and ADRES are competitive with the evolved DDAMs. p^{ADDAM} and DPPW can also beat the evolved DDAMs in some scenarios with high utilization level (95%), the *full* setting and large numbers of machines. These results are quite consistent with what has been observed in Section 5.1. It is also noted that $\text{MAPE}_{p^*}^{\mathcal{S}_k}$ is better when the utilization increases, which is similar to that observed in Sabuncuoglu and Comlekci (2002). When the number of machines increases, it is also interesting to see that the performance of the evolved DDAMs deteriorates if the *missing* setting is used, but the performance of these DDAMs improves if the *full* setting is used.

Tables 15 and 16 show the detailed results obtained by the evolved and the existing DDAMs for two particular scenarios. The mean and the corresponding standard deviation of each performance measure are shown in these tables to provide a general evaluation of each DDAM. The MAPE, MAE, and STDL of the evolved DDAMs are better (smaller) than those obtained by the existing DDAM. This indicates that the evolved DDAMs provide better delivery accuracy and delivery reliability than existing DDAMs. It is also interesting that the MPE of p^{ODDAM} is positive while those of other DDAMs are negative. This means that the other DDAMs tend to overestimate the job flowtimes while p^{ODDAM} tends to underestimate the job flowtimes, but the estimations made by p^{ODDAM} are better because its MAPEs are smaller compared to those of existing DDAMs. Since the existing DDAMs overestimate flowtimes, some of %T values of those DDAMs are smaller than those of evolved DDAMs. However, with the current emphasis on the just in time (JIT; Cheng and Podolsky, 1993) production concept where both earliness and tardiness are undesirable and meeting the target job due date would be of significance for the practice of JIT philosophy, smaller MAPE and STDL would be more attractive than smaller %T.

7 Further Investigation with Due-Date-Based Dispatching Rules

It has been shown that the evolved ODDAMs have performed well when tested under different shop conditions. However, it is still questionable whether these evolved ODDAMs are still effective when other dispatching rules, rather than FIFO, are applied. To

Table 14: MAPE values obtained by the best evolved DDAMs.

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

Table 15: Performance of DDAMs (90% utilization, *missing* jobs, six machines, processing times follow Erlang-2 distribution).

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

Table 16: Performance of DDAMs (90% utilization, *full* jobs, six machines, processing times follow Erlang-2 distribution).

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

examine this issue, we test the performance of the evolved DDAMs with the popular critial ratio plus shortest processing time (CR+SPT) rule, which usually provides good due-date related performance (Cheng and Jiang, 1998). Different from the case, when FIFO is used as the dispatching rule, it is noted that estimating job flowtimes here is more complicated. The reason is that FIFO is not influenced by the estimated due dates from the DDAM, while the sequencing decisions from CR+SPT are strongly affected by the estimated due dates. For example, the priority (with CR+SPT) of waiting jobs can be calculated as

$$Z_{ji} = p_{ji} \times \max\left\{\frac{d_j - t}{\sum_{r=i}^{m_j} p_{jr}}, 1\right\} \tag{6}$$

where p_{ji} is the processing time of the operation (j, i), which is the ith operation of job j. After all jobs in the queue have been assigned priorities determined by Equation (6), the job with the smallest priority value will be processed next. While the value of p_{ji} is available when the job arrives, d_j and t depend on the DDAM and the moment the sequencing decision is made. Therefore, we do not know the exact priorities of the new job at the machines it is planned to visit. Unfortunately, in order to make an accurate due-date estimate of a new job, it is important to consider the priorities of that job determined by the dispatching rule (when waiting in the queues) because they will influence the job's waiting times (Sabuncuoglu and Comlekci, 2002). To tackle this issue with our proposed GP-ODDAM method, we include in the terminal set a new term called the estimated priority ratio (EPR), which can be determined by Equation (7).

$$\text{EPR}_{ji} = \frac{\sum_{(l,k)\in Q'_m} p_{lk}}{\sum_{(l,k)\in Q_m} p_{lk}} \tag{7}$$

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

$$\begin{aligned}\hat{Z}_{ji} &= p_{ji} \times \max\left\{\frac{(r_j + f'_j) - (r_j + \mathrm{PEF})}{\sum_{r=i}^{m_j} p_{jr}}, 1\right\} \\ &= p_{ji} \times \max\left\{\frac{f'_j - \mathrm{PEF}}{\sum_{r=i}^{m_j} p_{jr}}, 1\right\} \end{aligned} \tag{8}$$

$$\hat{Z}'_{lk} = p_{lk} \times \max\left\{\frac{d_l - (r_l + \mathrm{PEF})}{\sum_{r=k}^{m_l} p_{lr}}, 1\right\} \tag{9}$$

where $r_j + f'_j$ is a rough estimate of the due date and $r_j + \mathrm{PEF}$ is the estimate of the decision moment t. Since DPPW was shown to be a competitive DDAM in our previous experiment, it will be used to calculate $r_j + f'_j$. The role of EPR_{ji} is to represent the dominance relation regarding the priorities between the new job and jobs currently in the shop. Table 17 shows the performance of the evolved DDAMs with CR+SPT as the dispatching rule based on the training and test scenarios in Table 3. We use the population size of 2,000 in this experiment in order to ensure enough diversity to create more sophisticated DDAMs to deal with this situation. It can be seen that the evolved ODDAMs still perform better than the existing DDAMs in most scenarios, especially the cases with the *missing* setting. For the scenarios with the *full* setting, there are fewer superior evolved ODDAMs that can beat DTWK and DPPW when the utilization increases. These results suggest that DTWK and DPPW again significantly enhance their performance with less diverse jobs and high utilization levels. Meanwhile, the evolved ODDAMs still show their superiority in more complicated scenarios when tested with CR+SPT.

8 Conclusions

In this paper, two proposed GP methods have been proposed for evolving due-date assignment models. This is the first work that examines the reusability of the obtained DDAMs. The experimental results show that the evolved DDAMs can outperform the existing dynamic DDAMs with `MAPE` as the performance measure. Comparisons using other performance measures also confirm the effectiveness of the evolved DDAMs. From the performance of the evolved DDAMs on a number of simulation scenarios, it can also be concluded that the evolved DDAMs have good reusability and they are able to make good job flowtime estimates for unseen scenarios with different processing time distributions, utilization, job settings, and numbers of machines. When comparing the two proposed GP methods, it has been shown that GP-ODDAM is significantly better than GP-ADDAM in most testing scenarios. Typical examples of the evolved DDAMs show that these GP evolved DDAMs are understandable.

This study has shown the effectiveness of GP for evolving DDAMs in the case where FIFO, or a due-date-based dispatching rule CR+SPT, is used as the dispatching rule. In future study, we would like to investigate the use of these systems for automatic design of DDAMs for job shops employing other sophisticated dispatching rules. Studies on more complicated job shops would also be very interesting to see how the evolved DDAMs perform in special environments.

Table 17: Performance of evolved ODDAMs with CR+SPT as the dispatching rule.

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

References

Ahmed, I., and Fisher, W. W. (1992). Due date assignment, job order release, and sequencing interaction in job shop scheduling. *Decision Sciences*, 23(3):633–647.

Bader-El-Den, M. B., Poli, R., and Fatima, S. (2009). Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205–219.

Banzhaf, W., Nordin, P., Keller, R., and Francone, F. (1998). *Genetic programming: An introduction*. San Mateo, CA: Morgan Kaufmann.

Baykasoglu, A., and Gocken, M. (2009). Gene expression programming based due date assignment in a simulated job shop. *Expert Systems with Applications*, 36(10):12143–12150.

Baykasoglu, A., and Gocken, M. (2011). A simulation based approach to analyse the effects of job release on the performance of a multi-stage job-shop with processing flexibility. *International Journal of Production Research*, 49(2):585–610.

Baykasoglu, A., Gocken, M., and Unutmaz, Z. D. (2008). New approaches to due date assignment in job shops. *European Journal of Operational Research*, 187(1):31–45.

Bookbinder, J. H., and Noor, A. I. (1985). Setting job-shop due-dates with service-level constraints. *Journal of the Operational Research Society*, 36(11):1017–1026.

Burke, E., Hyde, M., Kendall, G., and Woodward, J. (2010a). A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., and Qu, R. (2010b). Hyper-heuristics: A survey of the state of the art. School of Computer Science and Information Technology, University of Nottingham. (Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747).

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 C. L. Munford and L. C. Jain (Eds.), *Computational intelligence: Collaboration, fusion and emergence* (pp. 177–201). Berlin: Springer-Verlag.

Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2007a). Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07*, pp. 1559–1565.

Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2007b). The scalability of evolved online bin packing heuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC '07*, pp. 2530–2537.

Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. (2011). Automating the packing heuristic design process with genetic programming. *Evolutionary Computation*, 20(1):63–89.

Chang, F.-C. R. (1996). A study of due-date assignment rules with constrained tightness in a dynamic job shop. *Computers and Industrial Engineering*, 31(1–2):205–208.

Cheng, T. C. E., and Gupta, M. C. (1989). Survey of scheduling research involving due date determination decisions. *European Journal of Operational Research*, 38(2):156–166.

Cheng, T. C. E., and Jiang, J. (1998). Job shop scheduling for missed due-date performance. *Computers and Industrial Engineering*, 34(2):297–307.

Cheng, T. C. E., and Podolsky, S. (1993). *Just-in-time manufacturing: An introduction*. London: Chapman and Hall.

Fry, T. D., Philipoom, P. R., and Markland, R. E. (1989). Due date assignment in a multistage job shop. *IIE Transactions*, 21(2):153–161.

Fukunaga, A. (2002). Automated discovery of composite SAT variable-selection heuristics. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 641–648.

Fukunaga, A. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):21–61.

Geiger, C. D., Uzsoy, R., and Aytug, H. (2006). Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Heuristics*, 9(1):7–34.

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, GECCO '10*, pp. 257–264.

Hino, C. M., Ronconi, D. P., and Mendes, A. B. (2005). Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. *European Journal of Operational Research*, 160(1):190–201.

Jakobovic, D., and Budin, L. (2006). Dynamic scheduling with genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming, EuroGP '06*, pp. 73–84.

Jakobovic, D., Jelenkovic, L., and Budin, L. (2007). Genetic programming heuristics for multiple machine scheduling. In *Proceedings of the 10th European Conference on Genetic Programming, EuroGP'07*, pp. 321–330.

Joseph, O., and Sridharan, R. (2011). Analysis of dynamic due-date assignment models in a flexible manufacturing system. *Journal of Manufacturing Systems*, 30(1):28–40.

Keller, R., and Poli, R. (2007a). Cost-benefit investigation of a genetic-programming hyperheuristic. In *Proceedings of the 8th International Conference on Artificial Evolution*, pp. 13–24.

Keller, R., and Poli, R. (2007b). Linear genetic programming of parsimonious metaheuristics. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC'07*, pp. 4508–4515.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.

Land, M. J. (2004). *Workload control in job shops, grasping the tap*. Ph.D. thesis, University of Groningen, The Netherlands.

Little, J. D. C. (1961). A proof for the queuing formula: $L = \lambda W$. *Operations Research*, 9(3):383–387.

Luke, S. (2009). *Essentials of metaheuristics*. Raleigh, NC: Lulu Press. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2012). Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming. In *Proceedings of the 15th European Conference on Genetic Programming, EuroGP'12*, pp. 121–133.

Ozturk, A., Kayaligil, S., and Ozdemirel, N. E. (2006). Manufacturing lead time estimation using data mining. *European Journal of Operational Research*, 173(2):683–700.

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

Patil, R. J. (2008). Using ensemble and metaheuristics learning principles with artificial neural networks to improve due date prediction performance. *International Journal of Production Research*, 46(21):6009–6027.

Philipoom, P. R., Rees, L. P., and Wiegmann, L. (1994). Using neural networks to determine internally-set due-date assignments for shop scheduling. *Decision Sciences*, 25(5–6):825–851.

Pinedo, M. L. (2008). *Scheduling: Theory, algorithms, and systems*, 3rd ed. Berlin: Springer.

Ragatz, G. L., and Mabert, V. A. (1984). A simulation analysis of due date assignment rules. *Journal of Operations Management*, 5(1):27–39.

Ramasesh, R. (1990). Dynamic job shop scheduling: A survey of simulation research. *Omega*, 18(1):43–57.

Sabuncuoglu, I., and Comlekci, A. (2002). Operation-based flowtime estimation in a dynamic job shop. *Omega*, 30(6):423–442.

Sha, D., and Hsu, S. (2004). Due-date assignment in wafer fabrication using artificial neural networks. *The International Journal of Advanced Manufacturing Technology*, 23(9–10):768–775.

Sha, D., and Liu, C.-H. (2005). Using data mining for due date assignment in a dynamic job shop environment. *The International Journal of Advanced Manufacturing Technology*, 25(11–12):1164–1174.

Sha, D. Y., Storch, R. L., and Liu, C. H. (2007). Development of a regression-based method with case-based tuning to solve the due date assignment problem. *International Journal of Production Research*, 45(1):65–82.

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

Veral, E. A. (2001). Computer simulation of due-date setting in multi-machine job shops. *Computers and Industrial Engineering*, 41(1):77–94.

Vig, M. M., and Dooley, K. J. (1993). Mixing static and dynamic flowtime estimates for due-date assignment. *Journal of Operations Management*, 11(1):67–79.

Vinod, V., and Sridharan, R. (2011). Simulation modeling and analysis of due-date assignment methods and scheduling decision rules in a dynamic job shop production system. *International Journal of Production Economics*, 129(1):127–146.