Evolving Time-Invariant Dispatching Rules in Job Shop Scheduling with Genetic Programming

No Author Given

No Institute Given

Abstract. Genetic Programming (GP) has achieved success in evolving dispatching rules for job shop scheduling problems, particularly in dynamic environment. However, there is still great potential to improving the performance of GP. One challenge that is yet to be addressed is the huge search space. In this paper, we propose a simple yet effective approach to improve the effectiveness and efficiency of GP. The new approach is based on a newly defined *time-invariance* property of dispatching rules, which is derived from the idea of translational invariance from machine learning. Then, we develop a new terminal selection scheme to guarantee the time-invariance throughout the GP process. The experimental studies show that by considering the time-invariance, GP can achieve much better rules in a much shorter time.

1 Introduction

Job Shop Scheduling (JSS) [20] is an important optimisation problem with a wide range of real-world applications in domains such as manufacturing [3, 13], project scheduling [24] and cloud computing. Given a set of machines and jobs, JSS is to process the jobs with the machines subject to certain constraints (e.g. each job must follow a specific routing among the machines, and each machine can process only one job at a time) and optimise some criteria such as makespan, flowtime and tardiness.

JSS can be either static or dynamic. In static JSS, all the jobs are available at the beginning of the scheduling horizon, and all the information is known in advance. In dynamic JSS, unpredicted job arrivals can occur in real time, and can affect the subsequent scheduling decisions. In this study, we focus on dynamic JSS, since it is closer to reality and more challenging than static JSS.

Dispatching rules (DRs) are promising decision making heuristics for solving dynamic JSS due to their low complexity, scalability and flexibility. Briefly speaking, a DR is a priority function of the job shop attributes such as the operation processing time and job due date. In each decision situation (e.g. a machine becomes idle and its queue is not empty), the DR is used to calculate the priority for each job/operation waiting in the queue, and the one with the best priority is selected to be processed next.

Recently, automatically designing/evolving DRs using Genetic Programming (GP) has achieved some success [2]. The DRs evolved by GP have shown to be much more effective than the DRs designed by human experts. However, a major

challenge for evolving DRs with GP is the huge search space caused by the large number of possible combinations involving the function and terminal sets. It is highly desired to develop intelligent guidances for the GP search process to achieve better effectiveness and efficiency.

1.1 Goals

In this paper, we propose a new simple yet effective method to improve the effectiveness of GP. The proposed method is based on a newly defined property called *time-invariance*. A DR is time-invariant if it generates the same schedule for the same *JSS pattern* (defined as a time window in the scheduling horizon, details in Section 3) regardless of when such pattern occurs. By restricting the search to only the time-invariant DRs, we expect to reduce the search space and improve the efficiency of the search. The goal of this paper is to investigate the effectiveness of considering time-invariance in GP. Specifically, we have the following objectives:

- Formally define the concept of time-invariance;
- Show that GP may generate DRs that are not time-invariant;
- Develop a new selection scheme of terminals so that GP always generates time-invariant DRs;
- Compare the resultant time-invariance-aware GP with the baseline GP to verify the proposed method.

1.2 Organisation

The rest of the paper is organised as follows: Section 2 gives the background introduction. Then, the concept of time-invariance is defined in Section 3. The new selection scheme of terminal set is developed in Section 4. Experimental studies are carried out in Section 5. Finally, Section 6 gives the conclusions and future work.

2 Background

2.1 Job Shop Scheduling

In JSS, a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ are given. Each job J_j has an arrival time $t_0(J_j)$ and a due date $\rho(J_j)$. It consists of a sequence of operations $[O_{1j} \rightarrow \cdots \rightarrow O_{l_j j}]$. Each operation O_{ij} must be processed by machine $\pi_{ij} \in \mathcal{M}$, and its processing time is δ_{ij}. Thus, it can be represented as a tuple $O_{ij} = \langle \pi_{ij}, \delta_{ij} \rangle$. An operation cannot be processed before the completion before its precedent operations. Each machine can process at most one operation at a time. Then, JSS is to find a feasible schedule to optimise some objective(s). The commonly considered JSS objectives include minimising the makespan ($C_{\max}$), total flowtime ($\sum C_j$), total weighted tardiness ($\sum w_j T_j$), number of tardy jobs, etc [20].

2.2 Automatic Design of Dispatching Rules

In dynamic JSS, unpredicted job arrivals occur in real time, and thus the scheduling process can be seen as a discrete event simulation. Once a machine becomes idle and its queue is not empty, a *dispatching rule* (DR) [1] is used to prioritise the operations waiting in the queue, and the most prior operation is processed next. For example, the Shortest Processing Time (SPT) rule selects the operation with the shortest processing time to be processed next.

So far, there have been a large number of DRs designed by human experts (e.g. [23, 8, 11, 21]), by considering a variety of job shop attributes such as operation processing time, job due date, work remaining and slack. However, the existing manually designed DRs are normally not effective enough, and restricted to the particular job shop scenario they are designed for.

Genetic Programming (GP) has been demonstrated to be powerful for automatically designing/evolving DRs in job shop scheduling. So far, there have been extensive studies [6, 10, 15, 7, 19, 9]) in this direction and successfully achieved much better results than the previously man-made rules. A comprehensive review can be found in [2].

For evolving DRs using GP, a major challenge is how to search effectively and efficiently in the huge search space. For example, in the conventional tree-based GP, an individual is represented as a GP tree. Then, the size of the search space depends on the maximal tree depth/size, the function set and the terminal set. The performance of GP can be dramatically improved by incorporating domain knowledge about job shop scheduling. Several works have been done to improve the search efficiency from different perspectives. For example, Nguyen et al. [15] investigated different representations and proposed a grammar-based representation to constrain the search space. Mei et al. [14] proposed to identify a subset of relevant features and remove the other redundant features from the terminal set to reduce the search space. They demonstrated that the more compact terminal set can lead to significantly better rules. Riley et al. [22] proposed a similar terminal selection idea. Durasević et al. [4] developed a dimensionally aware GP that considers the compatibility between the dimensions (e.g. time, weight, counting number) of the terminals and design initialisation and evolutionary operators in such a way that no semantically incorrect rule (e.g. adding time to weight) is generated. This way, the search is focused on only the semantically correct rules.

In this paper, we investigate a new way of incorporating domain knowledge by borrowing the idea of invariant features in machine learning.

3 Time-Invariant Dispatching Rule

Invariance is an important property that has been extensively studied in machine learning, particularly classification (e.g. [18]). Using invariant features tends to improve the generalisability and robustness of the classifier.

Evolving DRs using GP is similar to the training process of a classifier. In the GP process, the fitness of a DR is evaluated on a set of training JSS

instances/simulations. By providing a sufficient number of diversely distributed decision situations [5] in the training set, the learned DRs can make proper decisions for the decision situations in both the training instances and the unseen test instances.

As introduced in [5], a decision situation is represented as a feature table, where each row stands for an operation in the queue, and each column indicates a feature that may be considered by the DR. Therefore, how to select the features in the decision situation is an important issue. If there are too few features, some important information may be missed. If there are too many features, the limited training set may not be able to cover all the possible feature values, and the evolved DRs may not generalise well on the unseen test set which can be quite different from the training set. To achieve a good tradeoff between the loss of information and generalisation, we define a new property of time-invariance.

It is natural to assume that JSS instances contain some key patterns that are invariant under translation, i.e. shift on the time horizon. Here, a *JSS pattern* is defined as a time window of the entire scheduling horizon, including the *initial job shop state* (next idle time of each machine and the uncompleted jobs) when the time window starts and the *new job arrivals* during the time window.

Definition 1 (JSS pattern). *A JSS pattern for a window $[t_1, t_2]$ is defined as a tuple $\vartheta_{[t_1,t_2]} = \langle \boldsymbol{t}_\vartheta^{idle}, \mathcal{J}_\vartheta^{init}, \mathcal{J}_\vartheta^{new} \rangle$, where $\boldsymbol{t}_\vartheta^{idle}$ is the vector of the next idle time of the machines at t_1, $\mathcal{J}_\vartheta^{init}$ is the set of uncompleted jobs, and $\mathcal{J}_\vartheta^{new}$ is a sequence of new job arrivals.*

An example of a JSS pattern $\vartheta_{[0,8]}$ is given in Table 1 (top half). In this pattern, the two machines are both idle at time 0, and there is no uncompleted job. During the time window $[0, 8]$, there are two new job arrivals. The first job arrives at time 0 with a due date of 18, and the second arrives at time 2 with a due date of 20. Both jobs have two operations, as shown in the table. The job and operation ids are ignored.

Table 1. An example of a JSS pattern $\vartheta_{[0,8]}$ and the shifted pattern $h(\vartheta_{[0,8]}, 10)$. The pattern contains two jobs, each with two operations.

			$\mathcal{J}^{\text{new}}$		
	$\boldsymbol{t}^{\text{idle}}$	$\mathcal{J}^{\text{init}}$	t_0	ρ	sequence of operations
$\vartheta_{[0,8]}$	$(0, 0)$	$\emptyset$	0	18	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 4\rangle]$
			2	20	$[\langle M_2, 2\rangle \rightarrow \langle M_1, 4\rangle]$
$h(\vartheta_{[0,8]}, 10)$	$(\mathbf{10}, \mathbf{10})$	$\emptyset$	**10**	**28**	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 4\rangle]$
			12	**30**	$[\langle M_2, 2\rangle \rightarrow \langle M_1, 4\rangle]$

Obviously, a decision situation $\vartheta_t^{\text{ds}} = \vartheta_{[t,t]}$ is a special instantaneous JSS pattern. Then, we define the time-shift transformation for JSS patterns as follows.

Definition 2 (Time-shift transformation for JSS patterns). *Given a JSS pattern ϑ and a shift value Δt, the time-shift transformation $h(\cdot)$ generates a new pattern $h(\vartheta, \Delta t)$ by increasing the next idle times of the machines, and*

the arrival times and due dates of all the jobs by Δt*, i.e.* $\boldsymbol{t}_{\vartheta}^{idle} \rightarrow \boldsymbol{t}_{\vartheta}^{idle} + \Delta t$, $t_0(J) \rightarrow t_0(J) + \Delta t$, $\rho(J) \rightarrow \rho(J) + \Delta t$, $\forall J \in \mathcal{J}_{\vartheta}^{init} \cup \mathcal{J}_{\vartheta}^{new}$.

Table 1 gives an example of a shifted pattern $h(\vartheta_{[0,8]}, 10)$ at the bottom. The changed parts are highlighted in bold.

Two patterns ϑ_1 and ϑ_2 are said to be *equivalent under time-shift* if there exists a shift value Δt, so that $\vartheta_1 = h(\vartheta_2, \Delta t)$.

By applying a DR to a pattern, the output is a *schedule*, which can be represented as sequences of operations, each for a machine. For example, a schedule Φ (shown in Fig. 1) for the pattern ϑ given in Table 1 can be represented as follows:

$$\Phi = \begin{bmatrix} M_1 : O_{11} \rightarrow O_{22} \\ M_2 : O_{12} \rightarrow O_{21} \end{bmatrix} \tag{1}$$

where O_{ij} stands for the i^{th} operation of the job in the j^{th} row in the pattern. Note that the starting times of the operations are ignored. Since no delay is allowed, each operation will start as soon as it becomes ready (all its precedent operations in the same job and the same machine have been completed, and the machine is idle).

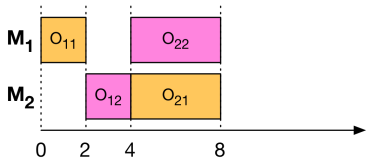

Fig. 1. A schedule for the JSS instance ϑ given in Table 1.

Then, the time-invariance property for DRs is defined as follows.

Definition 3 (Time-invariant dispatching rule). *A rule* Υ *is time-invariant if its generated schedule is invariant under the time-shift transformation, i.e.* $\Phi(\Upsilon, \vartheta) = \Phi(\Upsilon, g(\vartheta, \Delta t)), \forall \vartheta, \Delta t \in \mathbb{R}$.

For example, the SPT (shortest processing time) rule is time-invariant, as its decision making does not dependent on time.

Time-invariant DRs have the following promising properties.

- Time-invariant DRs generate the same schedule for the patterns that are equivalent under time-shift. Thus, they tend to show repetitive behaviours throughout the simulation horizon, which are also likely to generalise to the same patterns occur in the unseen test instances no matter when the patterns occur. Therefore, time-invariant DRs tend to have better generalisability.
- Time-invariant DRs are good at dealing with periodic job arrivals, e.g. the same order arrivals for weekly scheduling periods. In this case, time-invariant DRs generate the same schedule for each period (equivalent patterns under time-shift) without explicitly splitting the scheduling horizon into periods. This is particularly useful when the periodic pattern is not obvious.

3.1 An Example: Time-Invariance v.s. Time-Dependence

In a JSS instance, the same pattern may occur at different times. An example of a JSS instance is given in Table 2. When applying the SPT rule to this instance, the schedule for the first 6 jobs is shown in Fig. 2. Obviously, the two patterns $\vartheta_{[0,8]}$ and $\vartheta_{[16,24]}$ (described in Table 3) are equivalent under time-shift. It can be seen that $\vartheta_{[16,24]} = h(\vartheta_{[0,8]}, 16)$. As a result, the time-invariant SPT rule generates the same schedule in the two time windows.

Table 2. An example of a JSS instance.

j	$t_0(J_j)$	$\rho(J_j)$	sequence of operations
1	0	18	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 4\rangle]$
2	2	20	$[\langle M_2, 2\rangle \rightarrow \langle M_1, 4\rangle]$
3	9	21	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 2\rangle]$
4	10	28	$[\langle M_2, 4\rangle \rightarrow \langle M_1, 2\rangle]$
5	16	34	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 4\rangle]$
6	18	36	$[\langle M_2, 2\rangle \rightarrow \langle M_1, 4\rangle]$
		...	

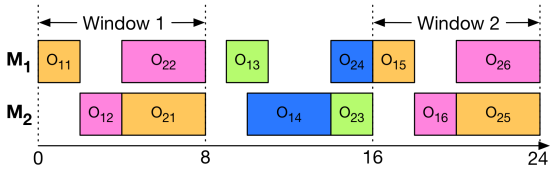

Fig. 2. A schedule obtained by the SPT rule for the JSS instance given in Table 2.

Table 3. The patterns occurring in time windows $[0, 8]$ and $[16, 24]$ of the instance shown in Table 2.

			$\mathcal{J}^{\text{new}}$		
Pattern	$\boldsymbol{t}^{\text{idle}}$	$\mathcal{J}^{\text{init}}$	t_0	ρ	sequence of operations
$\vartheta_{[0,8]}$	$(0, 0)$	$\emptyset$	0	18	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 4\rangle]$
			2	20	$[\langle M_2, 2\rangle \rightarrow \langle M_1, 4\rangle]$
$\vartheta_{[16,24]}$	$(16, 16)$	$\emptyset$	16	34	$[\langle M_1, 2\rangle \rightarrow \langle M_2, 4\rangle]$
			18	36	$[\langle M_2, 2\rangle \rightarrow \langle M_1, 4\rangle]$

However, GP cannot guarantee to always generate time-invariant DRs. For example, GP may generate a rule PT $\times$ (DD/t $-$ PT) during the evolutionary process. To show that the above rule is not time-invariant (or time-dependent), we examine its behaviour in the two decision situations at time 2 and time 18. In the former decision situation, O_{12} and O_{21} are waiting in the queue of M_2. In the latter one, O_{16} and O_{25} are waiting in the queue of M_2.

Decision situation 1. $\text{priority}(O_{12}) = 2 \times (20/2 - 2) = 16$,
$\text{priority}(O_{21}) = 4 \times (18/2 - 4) = 20$, select O_{12};
Decision situation 2. $\text{priority}(O_{16}) = 2 \times (36/18 - 2) = 0$,
$\text{priority}(O_{25}) = 4 \times (34/18 - 4) = -8.44$, select O_{25}.

As a result, the schedule obtained by the rule $\text{PT} \times (\text{DD}/t - \text{PT})$ is shown in Fig. 3. It can be seen that although the patterns $\vartheta_{[0,8]}$ and $\vartheta_{[16,24]}$ are equivalent under time-shift, i.e. $\vartheta_{[16,24]} = h(\vartheta_{[0,8]}, 16)$, the rule $\text{PT} \times (\text{DD}/t - \text{PT})$ generates different schedules in the two time windows.

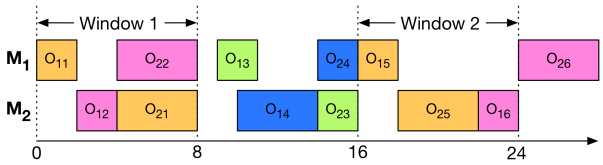

Fig. 3. A schedule obtained by the rule $\text{PT} \times (\text{DD}/t - \text{PT})$ for the JSS instance given in Table 2. It generates different schedules

3.2 Relationship Between Existing Rule Classifications

In [20], the DRs are classified into *static* and *dynamic* rules, or *local* and *global* rules. Static rules are time-independent, i.e. the priority values do not depend on time. In dynamic rules, the priority of jobs/operations change over time. Local rules only uses the local information pertaining to either the queue where the job is waiting or to the machine where the job is queued. Global rules may use information of other machines such as the work in the next queue.

The classification of *time-invariant* and *time-dependent* rules is different from both rule classifications in the following aspects.

- Static rules are time-invariant since they make time-independent decisions. Some dynamic rules can be time-invariant as well. For example, the dynamic minimum slack rule $\text{SL} = \max\{\text{DD} - \text{WKR} - t, 0\}$ (WKR is the work remaining) is time-invariant, since SL do not change under the time-shift transformation. For any operation O and decision situation ϑ^{ds}, $\text{SL}(O|h(\vartheta^{\text{ds}}, \Delta t)) = \max\{\text{DD}(O|\vartheta^{\text{ds}}) + \Delta t - \text{WKR}(O|\vartheta^{\text{ds}}) - (t + \Delta t), 0\} = \text{SL}(O|\vartheta^{\text{ds}})$.
- Both local and global rules can be time-invariant. For example, the global rule $\text{PT} + \text{WINQ}$ is time-invariant, since both PT and WINQ do not depend on time.

4 Selection of Terminals for Time-Invariance

In Section 3.1, we have shown that GP cannot guarantee to evolve time-invariant DRs. In this section, we propose a new selection scheme for terminals so that GP can always generate time-invariant DRs. Recalling that *a decision situation*

is special instantaneous JSS pattern. Therefore, the time-shift transformation of decision situations is the same as that of JSS patterns.

Then, we define a new concept called *time-invariant feature* as follows.

Definition 4 (Time-invariant feature). *A feature α is time-invariant if given any decision situation ϑ^{ds} and any operation O waiting in the queue, the value $a(O)$ is invariant under the time-shift transformation on ϑ^{ds}. That is,*

$$\alpha(O|\vartheta^{ds}) = \alpha(O|h(\vartheta^{ds}, \Delta t)), \forall \Delta t \in \mathbb{R}, \tag{2}$$

where $\alpha(O|\vartheta^{ds})$ is the value of feature α of operation O in ϑ^{ds}.

Based on the definition of the time-shift transformation (Definition 2), it is obvious that all the features that are independent of the idle time of the machine, the arrival time and due date of the jobs will be time-invariant. In addition, as a feature, the current time is obviously not time-invariant since its value changes along with the time-shift transformation.

Theorem 1. *If GP uses a terminal set consisting of only time-invariant features, then GP always generates time-invariant DRs.*

Proof. Given a terminal set $\mathcal{T}$, any DR Υ generated by GP is a priority function $f(\mathcal{T}(\cdot))$ of the terminals. For any decision situation ϑ^{ds} and operation O, the priority value of O is $f(\mathcal{T}(O|\vartheta^{\text{ds}}))$. Since all the terminals are time-invariant features, we have

$$\alpha(O|\vartheta^{\text{ds}}) = \alpha(O|h(\vartheta^{\text{ds}}, \Delta t)), \forall \alpha \in \mathcal{T}, \Delta t \in \mathbb{R}.$$

Therefore, $f(\mathcal{T}(O|\vartheta^{\text{ds}})) = f(\mathcal{T}(O|h(\vartheta^{\text{ds}}, \Delta t)), \forall \Delta t \in \mathbb{R}$. That is, the priority values calculated by Υ do not change under the time-shift transformation. In other words, Υ is a time-invariant DR.

Based on Theorem 1, we can guarantee the time-invariance of the GP-evolved rules simply by only including time-invariant features in the terminal set. To this end, we examined the time-invariance of the commonly used features in literature [14, 17], using the definition of the features and Definitions 2 and 4. If a feature is not time-invariant, then it is either removed from the terminal set, or replaced by a time-invariant counterpart. The detail is given in Table 4. It is easy to ensure the time-invariance of the time-invariant counterparts. For example, for any decision situation ϑ^{ds} and operation O, $\text{TIS}(O|h(\vartheta^{\text{ds}}, \Delta t)) = (t + \Delta t) - (\text{AT}(O|\vartheta^{\text{ds}}) + \Delta t) = \text{TIS}(O|\vartheta^{\text{ds}})$.

5 Experimental Studies

To evaluate the effectiveness of considering time-invariance, we compare between the baseline GP (BaselinGP) with the original terminals (the left part of Table 4) and the time-invariant GP (TivGP) with the time-invariant terminals (right part of Table 4).

Table 4. The terminals used in [14] and [17], and the time-invariant counterpart.

Original		**Time-invariant counterpart**	
Notation	Description	Notation	Description
t	Current time	-	-
NIQ	Number of operations In Queue	same	same
WIQ	Work In Queue	same	same
MRT	Machine Ready Time	MWT (t-MRT)	Machine Waiting Time
PT	Processing Time	same	same
NPT	Next Processing Time	same	same
ORT	Operation Ready Time	OWT (t-ORT)	Operation Waiting Time
NRT	Next Machine Ready Time	NWT (NRT-t)	Next Machine Waiting Time
WKR	Work Remaining	same	same
NOR	Number of Operations Remaining	same	same
WINQ	Work In Next Queue.	same	same
NINQ	Number of operations In Next Queue	same	same
FDD	Flow Due Date	rFDD (FDD-t)	Relative FDD
DD	Due Date	rDD (DD-t)	Relative DD
W	Weight	same	same
AT	Arrival Time	TIS (t-AT)	Time In System
SL	Slack	same	same

In the experiments, we consider three objectives: (1) the maximal tardiness (Tmax), (2) the mean tardiness (Tmean) and (3) the total weighted tardiness (TWT). For each objective, we consider utilisation levels of 0.85 and 0.95. Therefore, the experiments consist of $3 \times 2 = 6$ different job shop scenarios. The configuration parameters of the simulation model are given in Table 5. This simulation configuration has been used in previous studies [16, 5].

Table 5. The Dynamic JSS simulation system configuration.

Parameter	**Value**
#machines	10
#jobs	5000
#warmup jobs	1000
#operations per job	Random from 2 to 10
Job arrival process	Poisson process
Utilisation level	{0.85, 0.95}
Due date	4×total processing time
Eligible machine	Uniform discrete distribution
Processing time	Uniform discrete distribution between 1 and 99

For each scenario, we use the standard GP process to train DRs. The parameter setting of the GP is given in Table 6. Note that the only difference between BaselineGP and TivGP is the terminal set. This way, one can analyse how the use of time-invariant terminals affects the performance of GP.

During the training process, an individual is evaluated using a randomly generated simulation. To improve generalisation, the random seed for generating the training simulation changes per generation. In addition, the fitness is normalised

Table 6. The parameter setting of GP.

Parameter	Value
Terminal set	The original ones in Table 4 for BaselineGP The time-invariant ones in Table 4 for TivGP
Function set	$\{+, -, *, /, \min, \max\}$
Population size	1024
Maximal depth	8
Crossover rate	80%
Mutation rate	15%
Reproduction rate	5%
Parent selection	Tournament selection with size 7
Elitism	10 best individuals
Number of generations	51

by the objective value of the reference rule. The reference rule is set to EDD, ATC and WATC for Tmax, Tmean and TWT, respectively. Finally, the best individual in the last generation is selected as the best individual of the GP run.

For testing, a test set of 50 simulation replications is randomly generated for each scenario. The test fitness of a rule x is defined as the normalised total objective value over the test replications, i.e. $\Gamma(x, \Pi, F) = \frac{\sum_{\pi \in \Pi} F(x,\pi)}{\sum_{\pi \in \Pi} F(\text{RefRule(Obj)},\pi)}$, where $F \in \{\text{Tmax}, \text{Tmean}, \text{TWT}\}$.

In the experiments, both BaselineGP and TivGP were implemented in Java using the ECJ library [12]. The experiments were run on desktops with Intel(R) Core(TM) i7 CPU @3.60GHz. Both algorithms were run 30 times independently for each scenario.

5.1 Results and Discussions

Fig. 4 shows the curves of the test fitness of the compared algorithms in 30 runs. The ribbon around each curve is the standard error of the mean. From the figure, one can see that TivGP significantly outperformed BaselinGP in scenarios $\langle \text{Tmax}, 0.85, 4 \rangle$, $\langle \text{Tmax}, 0.95, 4 \rangle$ and $\langle \text{TWT}, 0.95, 4 \rangle$. For the two scenarios with objective Tmean, the two algorithms performed almost the same. TivGP was defeated by BaselineGP in only the scenario $\langle \text{TWT}, 0.85, 4 \rangle$. Overall, TivGP performed much better than BaselineGP (3 wins, 2 draws and 1 lose) over the 6 tested scenarios.

From Fig. 4, it can also be seen that in general, the curves of the test fitness is smoothly improving as the search continues. This indicates that it is reasonable to simply select the best rule in the last generation as the best rule of the run.

In addition to the test fitness, we investigate the growth of the tree size (number of nodes) of the best rules during the GP runs, which is shown in Fig. 5. From the figure, it is clear that TivGP led to significantly smaller tree sizes than BaselineGP in most of the scenarios. This implies that using the time-

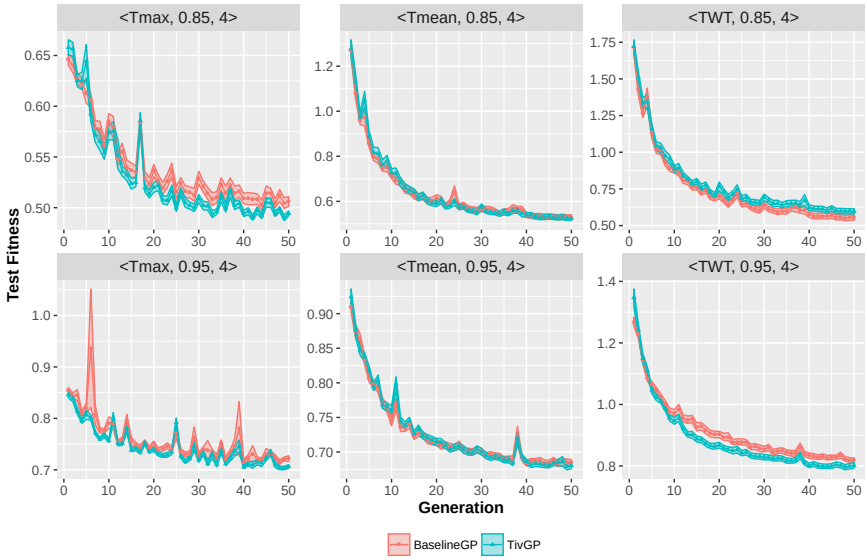

Fig. 4. The curves of the test fitness of the 30 runs of BaselineGP and TivGP.

invariant features in the terminal set can help evolving smaller (and possibly simpler) rules.

In order to analyse the usefulness of the terminals in GP, we investigate the growth of the number of unique terminals used in the GP tree over generations, which is shown in Fig. 6. From the figure, one can clearly see that TivGP uses significantly fewer unique terminals than BaselineGP. On average, the number of unique terminals in TivGP is one or two smaller than that in BaselineGP. This is partially because TivGP has a smaller terminal set than BaselineGP (the terminal t is included in BaselineGP but not in TivGP). This shows that the current time t can be safely removed from the terminal set, if all the other terminals are time-invariant features.

Fig. 7 shows the generational running time of the compared algorithms. One can clearly see that TivGP is much faster than BaselineGP in all the 6 scenarios. Since the computational effort of fitness evaluation largely depends on the tree size of the individuals, Fig. 7 indicates that the individuals in TivGP are generally much smaller than the individuals in BaselineGP.

5.2 Further Analysis

Since the test fitness depends on both the training fitness and generalisation, it is interesting to know the relationship between the training and test fitnesses to show the generalisation of the rules. To this end, we show the scatter plot of the training and test fitnesses of the best rules obtained by BaselineGP and TivGP for all the scenarios in Fig. 8.

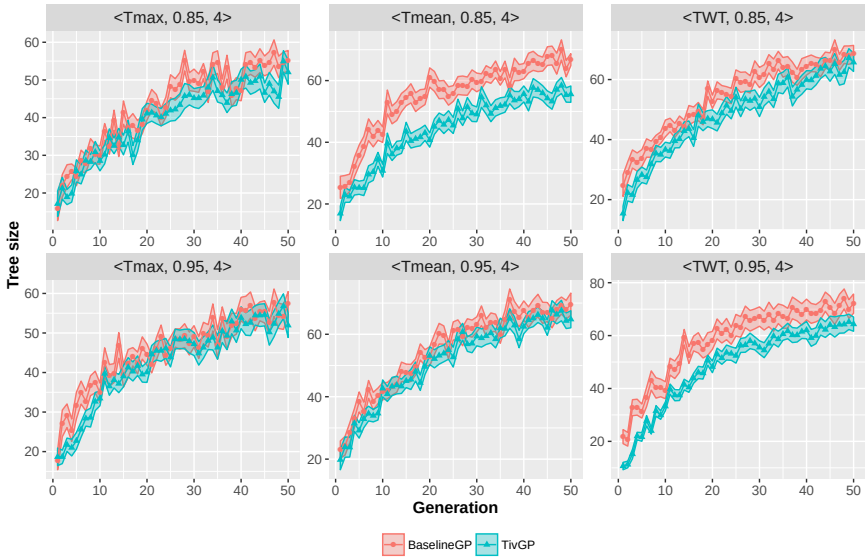

Fig. 5. The curves of the program size of the 30 runs of BaselineGP and TivGP.

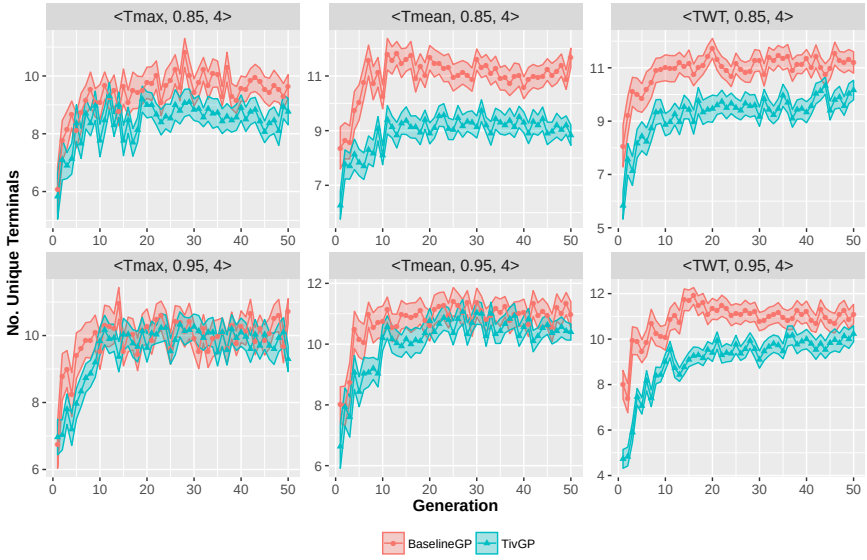

Fig. 6. The curves of the number of unique terminals of the 30 runs of BaselineGP and TivGP.

From the figure, one can see that the two algorithms have similar generalisations in all the scenarios. Briefly speaking, the training and test fitnesses

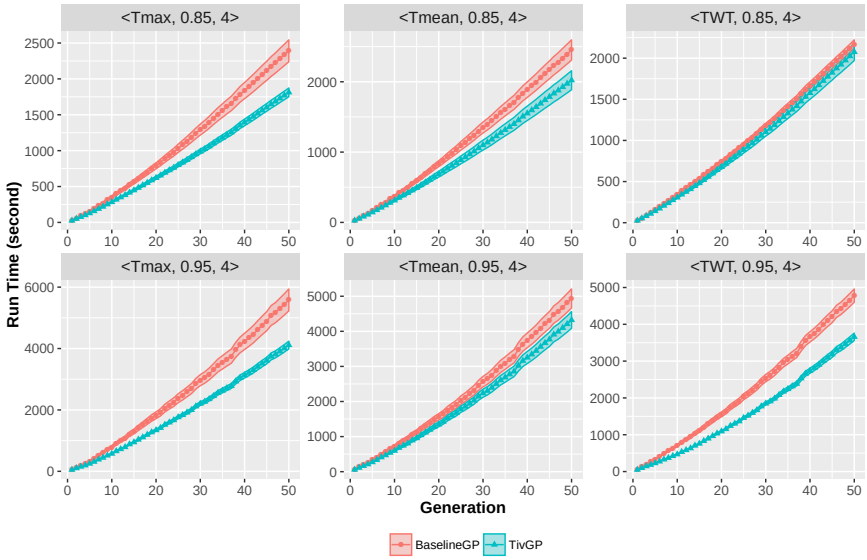

Fig. 7. The generational running time of the 30 runs of BaselineGP and TivGP.

are strongly correlated. If a rule has a better training fitness, then it is very likely to have a better test fitness, no matter which algorithm it is from. For the three scenarios where TivGP outperformed BaselineGP ($\langle \text{Tmax}, 0.85, 4\rangle$, $\langle \text{Tmax}, 0.95, 4\rangle$ and $\langle \text{TWT}, 0.95, 4\rangle$), the rules obtained by TivGP tend to have both better training and test fitnesses (more towards the bottom-left corner). In the scenario $\langle \text{Tmean}, 0.85, 4\rangle$, the two algorithms have similar distributions. In the scenario $\langle \text{Tmax}, 0.95, 4\rangle$, TivGP tends to have better generalisation (better test fitness for the same training fitness), but suffered from the two outliers in the top-right corner. In the scenario $\langle \text{TWT}, 0.85, 4\rangle$ where TivGP was outperformed by BaselineGP, the two algorithms seem to have very similar train-vs-test distributions. The only difference is that BaselineGP has more points towards the bottom-left corner (around the region $(0.42, 0.45)$).

5.3 Time-Invariance of the Evolved Rules

Obviously, all the rules evolved by TivGP are time-invariant. Then, we are interested in examining the time-invariance of the rules evolved by BaselineGP. To this end, we conduct a case study on the best rule obtained by BaselineGP for the scenario $\langle \text{TWT}, 0.85, 4\rangle$, where BaselineGP showed better performance than TivGP. The rule is shown below:

$$((B_1 + B_2) \times B_3 + \text{WIQ}) \times B_4 \times \min\{\text{PT}, \text{MRT}\}/B_5,$$
$$\text{where } B_1 = \max\{\text{NINQ}, \text{SL}\} \times \text{W}/(\text{WKR} - \text{NIQ}),$$

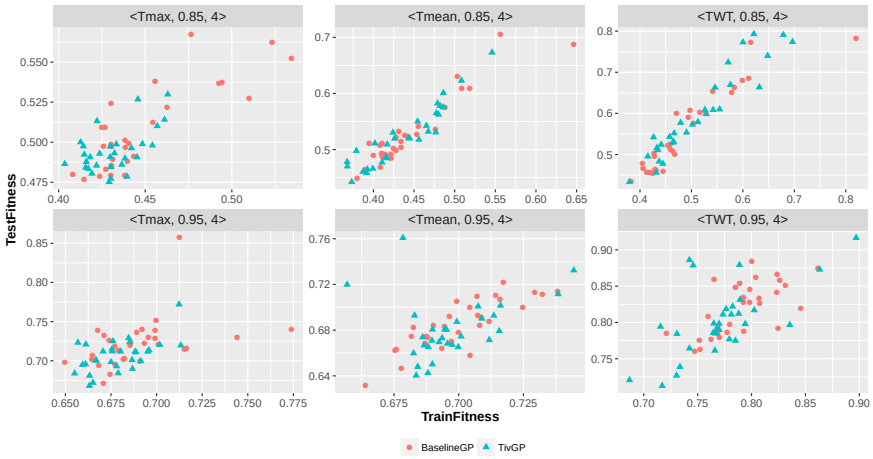

Fig. 8. The generational running time of the 30 runs of BaselineGP and TivGP.

$$B_2 = \max\{\text{NINQ}, \text{SL}\} + \text{NRT}/\text{PT},$$
$$B_3 = (t/\text{AT}) \times \text{WINQ} \times \text{PT}/\text{NRT},$$
$$B_4 = \max\{(\text{WKR} - \text{NIQ})/\text{W}, \text{SL}\},$$
$$B_5 = \text{WKR} - (\text{WKR} - \text{NIQ}) \times \text{PT}/\text{NRT}.$$

The rule consists of four features that are not time-invariant, i.e. NRT, t, AT and MRT. NRT occurs in the sub-trees NRT/PT and PT/NRT, which occur in three places. t and AT occur together as a pattern t/AT. MRT occurs in only one pattern $\min\{\text{PT}, \text{MRT}\}$. Therefore, the rule is not time-invariant.

However, based on the feature definitions, we know that the values of these features increase as the simulation continues, and thus tend to be large in most time of the simulation. Therefore, we can transform the above rule into a time-invariant rule as follows:

- Replace NRT with a large value (1000000 in this case);
- Replace t/AT with 1, which is the converged value when both t and AT approach infinity.
- Replace $\min\{\text{PT}, \text{MRT}\}$ with PT, since PT is almost always smaller than MRT.

After the transformation, we compare the test fitness of the resultant time-invariant rule to that of the original rule, and found that their test fitnesses are very close (0.476 versus 0.448).

We have examined several other rules evolved by BaselineGP, and found that they are either time-invariant, or can be easily transformed to time-variant rules with very close test fitnesses. This also validates the motivation of evolving time-invariant rules.

In summary, we have the following findings from the experimental studies:

– By using the time-invariant features as terminals, TivGP outperformed the BaselineGP counterpart in terms of test fitness, program size and efficiency.
– BaselineGP and TivGP have similar generalisability, and the test fitnesses of the evolved rules are strongly correlated with their training fitnesses.
– Although the rules evolved by BaselineGP are not necessarily time-invariant, they can be transformed into time-invariant rules with similar test performances.

6 Conclusions and Future Work

In this paper, to improve the effectiveness and efficiency of GP to search in the huge space, we proposed to consider only *time-invariant* DRs during the search process. The concept of time-invariance is borrowed from the translational invariance in machine learning. To this end, we defined the concepts of *JSS pattern*, which is a time window of the scheduling horizon, and the *time-shift transformation* for JSS patterns. To guarantee that GP always generates time-invariant DRs, we proposed to select only the *time-invariant* features in the terminal set, and the resultant GP is called the time-invariant GP (TivGP). The experimental studies showed that TivGP can achieve DRs with significantly better test performances and smaller sizes than the rules obtained by the baseline GP. TivGP also has a faster convergence than the baseline GP. Furthermore, although the baseline GP sometimes outperform TivGP, the resultant time-dependent DRs can be transformed into time-invariant DRs with similar performance. This demonstrates the efficacy of considering time-invariant DRs.

In the future, we will investigate more schemes that take the time-invariance into account, such as developing new fitness functions and search operators for GP.

References

1. J. H. Blackstone, D. T. Phillips, and G. L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *The International Journal of Production Research*, 20(1):27–45, 1982.
2. J. Branke, S. Nguyen, C. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, 2016.
3. J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A distance-based ranking model estimation of distribution algorithm for the flowshop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286–300, 2014.
4. M. Durasević, D. Jakobović, and K. Knežević. Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, 48:419–430, 2016.
5. T. Hildebrandt and J. Branke. On using surrogates with genetic programming. *Evolutionary computation*, 23(3):343–367, 2015.
6. T. Hildebrandt, J. Heger, and B. Scholz-Reiter. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 257–264. ACM, 2010.

7. N. Ho and J. Tay. Evolving dispatching rules for solving the flexible job-shop problem. In *IEEE Congress on Evolutionary Computation*, volume 3, pages 2848–2855. IEEE, 2005.
8. O. Holthaus and C. Rajendran. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105, 1997.
9. R. Hunt, M. Johnston, and M. Zhang. Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 927–934. ACM, 2014.
10. D. Jakobović and L. Budin. Dynamic scheduling with genetic programming. In *Genetic Programming*, pages 73–84. Springer, 2006.
11. M. Jayamohan and C. Rajendran. New dispatching rules for shop scheduling: a step forward. *International Journal of Production Research*, 38(3):563–586, 2000.
12. S. Luke et al. A java-based evolutionary computation research system. `https://cs.gmu.edu/~eclab/projects/ecj/`.
13. M. K. Marichelvam, T. Prabaharan, and X. S. Yang. A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *IEEE transactions on evolutionary computation*, 18(2):301–305, 2014.
14. Y. Mei, M. Zhang, and S. Nyugen. Feature selection in evolving job shop dispatching rules with genetic programming. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 365–372. ACM, 2016.
15. S. Nguyen, M. Zhang, M. Johnston, and K. Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 17(5):621–639, 2013.
16. S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Dynamic multi-objective job shop scheduling: A genetic programming approach. In *Automated Scheduling and Planning*, pages 251–282. Springer, 2013.
17. S. Nguyen, M. Zhang, and K. C. Tan. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. *IEEE Transactions on Cybernetics*, pages 1–15, DOI: 10.1109/TCYB.2016.2562674, 2016.
18. T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on pattern analysis and machine intelligence*, 24(7):971–987, 2002.
19. C. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter. Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. *International Journal of Production Economics*, 145(1):67–77, 2013.
20. M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.
21. C. Rajendran and O. Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1):156–170, 1999.
22. M. Riley, Y. Mei, and M. Zhang. Feature selection in evolving job shop dispatching rules with genetic programming. In *IEEE Congress on Evolutionary Computation*. IEEE, 2016.
23. V. Sels, N. Gheysen, and M. Vanhoucke. A comparison of priority rules for the job shop scheduling problem under different flow time-and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270, 2012.
24. J. Xiong, J. Liu, Y. Chen, and H. A. Abbass. A knowledge-based evolutionary multiobjective approach for stochastic extended resource investment project scheduling problems. *IEEE Transactions on Evolutionary Computation*, 18(5):742–763, 2014.