Evolving Dispatching Rules with Greater Understandability for Dynamic Job Shop Scheduling

Rachel Hunt Rachel.Hunt@msor.vuw.ac.nz
School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, New Zealand

Mark Johnston Mark.Johnston@vuw.ac.nz
School of Mathematics, Statistics and Operations Research, Victoria University of Wellington, New Zealand

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

Abstract

Heuristic dispatching rules are one of the most popular and widely used methods of scheduling in dynamic job shop environments. The manual development of such dispatching rules is time consuming and requires substantial knowledge of the domain. There have been numerous works into using genetic programming (GP) as a framework for the automated generation of dispatching rules for job shop scheduling environments. However most existing works do not take into account the interpretability and semantic validity of the evolved dispatching rules. In this paper we propose the use of a grammar, implemented through strongly typed GP, to restrict the GP programs to only contain semantically meaningful expressions. Experimental results show that the dispatching rules evolved in the semantically constrained search space do not have (on average) performance that is as good as unconstrained. However the interpretability of evolved rules is substantially improved. This is the first work using GP for the automatic discovery of dispatching rules that has explored their interpretability in depth and considered it as an important trait of an effective dispatching rule.

Keywords

Genetic programming, Scheduling, Grammar

1 Introduction

Scheduling in dynamic job shop environments is a difficult task for which heuristic dispatching rules (DRs) are one of the most commonly used scheduling methods. DRs, also known as priority rules or scheduling rules, are popular for job shop scheduling (JSS) due to their ease of implementation, low computational cost and ability to cope with the dynamic nature of the task (Blazewicz et al., 1996). DRs are mathematical functions of attributes of the job shop, machines and jobs awaiting dispatch, used to determine which job should be processed next. When a machine becomes available to process a job, the DR is used to evaluate all jobs waiting in its queue, assigning each a priority value. The job with the highest priority is selected to begin processing. The development of more effective DRs for scheduling has the potential to increase

throughput and decrease costs, thus increasing profitability of the many job shops that there are world-wide (Jones and Rabelo, 1998).

The manual development of dispatching rules is a time consuming process reliant on domain knowledge. The automatic discovery of dispatching rules through evolutionary computation techniques has been studied in the academic literature throughout the past twenty years (Jones and Rabelo, 1998).

Genetic programming (GP) (Koza, 1992) is an evolutionary computation (EC) method which has been successfully applied to the JSS problem. GP automatically learns computer programs, which in the most common form of GP are represented as trees (Koza, 1992); internal nodes consist of functions and terminals are selected for the given problem domain. A randomly initialised population of computer programs is evolved using evolutionary operators (crossover, mutation and elitism). There has been a substantial body of research using GP for the automatic generation of heuristic dispatching rules for job shop scheduling (Hildebrandt et al., 2010; Hunt et al., 2014a; Jakobovic and Budin, 2006; Jakobovic et al., 2007; Miyashita, 2000; Nguyen et al., 2013b,a,c; Pickardt et al., 2010). GP cannot be easily used to represent a schedule of jobs, however tree-based GP naturally presents itself as an ideal candidate to the evolution of DRs, which as mathematical functions can very easily be represented as trees. GP is typically used to search the space of heuristics rather than searching the solution space of possible schedules directly (Ross, 2003).

A major issue with DRs evolved in this way is their interpretability. How well, or even whether, a human operator is able to understand and interpret how and why the DR works, is important if DRs evolved using EC techniques are to be used in real-world situations. One particular limitation is that many of the comparisons made in evolved DRs cannot be readily interpreted in terms of units, i.e., semantically most DRs evolved using GP are incorrect. Knowing how well a given rule will generalise across different distributions of arrivals and processing times, and different scales, is also necessary if they are to be used in practice.

Another factor in the interpretability of evolved DRs is their size. The allowable tree depth for most approaches using GP allows DRs which are much bigger than most DRs from the literature. The increased size allows more complex DRs, which may be able to discover more of the relationships and interactions amongst shop properties and therefore improve performance. However, the larger the DR, the more difficult it is to understand how it works and compare how a change in, e.g., processing time, would alter the priority of a given job.

Results in our previous work (Hunt et al., 2014a) suggested that DRs evolved using standard GP are not easily interpretable. This is due partly to the way that features are combined with mathematical operators, e.g., it makes more sense to multiply or divide by the job weight than to add or subtract it from the current time.

Jakobović and Marasović (2012), in their work evolving priority functions through GP within a meta-algorithm to form a scheduling heuristic, noted that "GP solutions are not 'analytically correct' ", further commenting that this is often true for all GP applications (Koza, 1992), and that solutions could be restricted so that only semantically correct expressions remain (e.g. not allowing processing time minus the number of unscheduled jobs). The authors state that preliminary results showed no statistically significant differences compared to standard tree-based GP. However it is not stated if comparisons were made only in terms of performance on test cases or also included consideration of the interpretability, reliability and generalisation ability (robustness) of the evolved rules, which we believe are important aspects that must be taken into

account when considering how 'good' a DR is.

Tree-based GP allows unrestricted composition of available functions and terminals, subject only to the tree-depth restriction, which is sufficient for many problem domains, but there is also a history of constraining the search space of GP. Koza (1992) introduced the use of "constrained syntactic structure" where programs were restricted by special "problem-specific syntactic rules of construction". The constrained syntactic structures were used, e.g., for symbolic multiple regression where programs were restricted to having a `LIST2` node at the top of the GP tree, to return a fixed number of component values instead of just a single value. The implementation of this required the initial population of programs to have the required syntactic structure, and, further, all of the genetic operators used in evolution were constrained so that the constrained syntactic structure of the program was preserved. The fitness evaluation method may also need to be modified to take into account the structure of the programs.

Montana (1995) introduced strongly typed genetic programming (STGP) as an enhanced version of GP. Standard GP is designed for a single data type. Under the "closure" assumption of GP (Koza, 1992), any function should be able to take as an argument any value returned from a terminal or other function. Strong typing enforces data type constraints, only generating trees which satisfy the constraints. Basic STGP is equivalent to Koza's constrained syntactic structures (Koza, 1992), except that rather than defining syntax by directly specifying which children each non-terminal can have, this is done indirectly in STGP by specifying the data types of each argument of each function as well as the data types returned by each terminal and function (Montana, 1995). Montana (1995) describes this difference as "relatively minor".

Keijzer and Babovic (1999) developed a system called dimensionally aware GP which uses the domain knowledge that is contained in units of measurement, in order to enhance the search efficiency and the interpretability of resulting output. The introduction of dimensionality to GP was expected to improve search efficiency. This is implemented by making GP nodes maintain knowledge of the units of the measurement it uses. The variable and constant terminals have exponents in their unit of measurement. In their experiments, dimensions of length, time and mass were used, with GP tasked with finding the formula of scientific laws from which data was generated. The results show that this form of GP which adds a semantic component is useful, producing output formulations that are "closer-to-correct and easier-to-interpret".

Another way of limiting the allowable expressions within an evolved dispatching rule is to use grammar-based GP (McKay et al., 2010). A grammar is used to describe the restrictions on expressions that can be used. A context-free grammar is given by (S, N, Σ, P) notation, where S is the start symbol, N is a set of non-terminals, Σ is the set of terminals, and P is the set of production rules of the grammar (Whigham, 1995). Grammar-based GP first appeared in the mid-1990s (McKay et al., 2010; Whigham, 1995). A major benefit of using a grammar to define program structure in GP is that it is a relatively easy means of restricting the search space and encoding knowledge of the problem (McKay et al., 2010). We can use the grammar to generate a population of programs defined by a language and ensure that the programs in the population continue to follow the rules of the language throughout evolution (Whigham, 1995).

Whigham (1995) used a context-free grammar to "define the structure of the language manipulated by a genetic program". The grammar was used in the generation of the program population and to ensure closure was still satisfied with the genetic operators. Crossover was restricted so it can only occur when nonterminals of the same type are selected as crossover points. Mutation replaces a nonterminal with a new sub-tree

generated by the grammar with the nonterminal as root node of the sub-tree. In both cases the program is restricted by the maximum depth parameter.

Existing GP approaches to discovering new dispatching rules for the JSS problem do not include domain knowledge beyond the use of properties of the jobs, machines and system as a whole. The use of grammar-based GP methods offers the benefits of including the domain knowledge of the types of properties that exist, e.g., counts and times. Further, the search space at the heuristic level which GP operates in is very large, and the restriction of using a grammar-based approach will reduce the search space. These benefits have the potential to improve human understanding of the evolved rules. For the JSS problem we want to better understand what makes a DR perform well, and of almost equal importance, what makes a rule perform badly. The aim of using a grammar-based or strongly typed GP system is to evolve rules which are more interpretable. It is desirable to know *why* a rule works – not just if it does – as this will help us to learn about the shop system and if the rule would transfer to another shop environment.

There is very little work using grammar-based GP approaches for the JSS problem. One of the three GP representations, R_1 investigated by Nguyen et al. (2013a) uses a grammar to "make the rules more readable and explainable". The grammar aims to "provide the adaptive dispatching rules the ability to apply different simple dispatching rules based on machine attributes." The two other GP representations used were an arithmetic representation, R_2, and a "mixed representation", R_3, combining the arithmetic and grammar based approaches. Experimental results showed that the evolved rules performed competitively with hybrid GAs from the literature, and performed effectively in dynamic environments (Nguyen et al., 2013a). Further, the evolved rules provided better interpretability of how the decisions on sequencing jobs should be made than conventional meta-heuristic approaches. Representation R_3 produced rules with better fitness (total weighted tardiness) than representations R_1 and R_2 (Nguyen et al., 2013a).

The aim of this work is to use a grammar-based GP approach (McKay et al., 2010) implemented through the STGP capabilities of ECJ20 (Luke, 2013) to evolve DRs for the multi-machine dynamic job shop environment. By using STGP to implement restrictions described by a grammar we constrain the search space, aiming to develop evolved rules with greater interpretability, enabling better understanding of what information effective rules are capturing from the shop system. In particular we aim to answer the following research questions.

- How does the performance of rules evolved under the constraint of a grammar compare to the performance of rules evolved without?
- Is the interpretability of evolved rules improved?
- Can we quantify interpretability?
- Can we justify the use of STGP if performance is worse as an acceptable compromise for improvement in interpretability?
- What does the restriction of the search space reveal about what are good components or combinations of DRs?
- Can we further restrict or alter the terminals we include in our terminal set based on the results?

Table 1: Job shop properties for a job j.

Property	Notation
Given Properties	
Number of operations	N_j
Set of operations	O_j
ith operation	$\sigma_{j,i}$
Processing time of operation i	$p(\sigma_{j,i})$
Machine required to process operation	$m(\sigma_{j,i})$
Weight	w_j
Due date	d_j
Release time	r_j
Calculated Properties	
Completion time	C_j
Flow time	$F_j = C_j - r_j$
Tardiness	$T_j = \max\{0, C_j - d_j\}$

The remainder of this paper is organised as follows. Section 2 gives some background on JSS and existing work using GP for JSS. Section 3 explains the proposed approach. Sections 4 and 5 present the results and analysis. Section 6 further discusses the results. Section 7 presents some conclusions and recommendations for future work.

2 Background

In this section we introduce the JSS problem, and survey the literature of automatic generation of scheduling heuristics.

2.1 Job Shop Scheduling Problem

In JSS problems there is a set of machines and a set of jobs which must be processed. JSS involves determining a production schedule for processing the jobs on each machine to optimise a measure of delivery speed or customer satisfaction.

Each job consists of a sequence of one or more operations, in a specified processing order, where each operation has a specified processing time on a certain machine (Blazewicz et al., 1996). The notation of the job properties are given in Table 1. In a dynamic job shop, jobs arrive according to a stochastic process, and no information about a job's properties are known until it has arrived in the shop. Job j arrives at time r_j and joins the queue at the first machine on its route ($m(\sigma_{1,j})$). Every time a machine becomes available (if any jobs are available), a DR is used to select the next job to begin processing on that machine. The DR evaluates all the jobs waiting in the machine's queue, assigning each a priority value. The job with the highest priority value is selected. When an operation σ finishes its prescribed processing time $p(\sigma)$ on machine $m(\sigma)$, the job is moved to the next machine on its route, or if the operation was the final operation of the job, then the job exits the system (at its completion time C_j) and is delivered to the customer. In this paper we focus on the ten machine dynamic job shop, with the objective of minimising the total weighted tardiness,

$$TWT = \sum w_j T_j. \tag{1}$$

JSS has been widely studied over the past 50 years (Potts and Strusevich, 2009; Hart et al., 2005), and there are many existing DRs for dynamic JSS in the literature.

The following DRs are some of the most commonly used. Vepsalainen and Morton (1987) investigated the performance of six scheduling rules for dynamic job shops with TWT objective. Their results showed that the weighted apparent tardiness cost (ATC) and weighted cost over time (COVERT) rules performed better than first-come first-served (FCFS), earliest due date (EDD), slack per remaining processing time (S/RPT) and weighted shortest processing time (WSPT). The ATC rule assigns job j priority

$$\frac{w_j}{p(\sigma_{ji})} \exp\left(-\left[\frac{d_j - t - p(\sigma_{ji}) - \sum_{q=i+1}^{N_j} p(\sigma_{jq})(b+1)}{kp_{avg}}\right]^+\right)$$

Here $bp(\sigma_{ji})$ approximates the waiting time at operation i, with lead time estimation parameter $b = 2$. Lookahead parameter k is set at 3, p_{avg} is the average processing time of the waiting jobs at the current machine, and $[A]^+$ is $\max\{0, A\}$. The weighted COVERT rule assigns job j priority

$$\frac{w_j}{p(\sigma_{ji})}\left[1 - \frac{(d_j - t - \sum_{q=i}^{N_j} p(\sigma_{jq}))^+}{k\sum_{q=1}^{N_j} bp(\sigma_{jq})}\right]^+$$

The lookahead parameter is set as $k = 2$ and lead time estimation parameter $b = 2$.

2.2 Genetic Programming based approaches for JSS

There has been a large body of work using GP systems to automatically evolve DRs for JSS environments. These cover a range of shop environments, including parallel machines (Jakobovic et al., 2007), static job shop (Hunt et al., 2014b), dynamic job shop environment with 10 machines (Hildebrandt et al., 2010; Hunt et al., 2014a), flexible job shop (Tay and Ho, 2008) and a complex semi-conductor manufacturing environment (Pickardt et al., 2013) (where 36 machines form 31 work centres). Work also focuses on different objective functions: mean flow time (Hildebrandt et al., 2010), makespan (Jakobović and Marasović, 2012; Jakobovic et al., 2007), and total weighted tardiness Jakobović and Marasović (2012). Research to improve the effectiveness of evolved DRs has been approached through many ways, including changing the training instances through evolution (Hildebrandt et al., 2010), encouraging less-myopic behaviour (Hunt et al., 2014a), using GP within a meta-algorithm (Jakobović and Marasović, 2012), use of automatically defined functions (Tay and Ho, 2008) and using a decision tree to decide which of two DRs should be used for each dispatching decision (Jakobovic and Budin, 2006). Rules evolved by GP are frequently reported to perform competitively with rules from the literature (Nguyen et al., 2012; Jakobovic et al., 2007).

2.3 Grammar-based GP for Scheduling

There is very limited work in the literature using semantically constrained GP for scheduling problems. The two works already mentioned, Jakobović and Marasović (2012) and Nguyen et al. (2013a), do not compare the interpretability of the DRs evolved with those evolved without semantic constraint. Tree-based GP is the ideal representation for discovery of dispatching rules. However, existing DRs discovered using GP are generally not semantically correct. The few works which have used semantically constrained GP have not shown a detailed performance comparison, let alone any analysis

of whether the interpretability is improved. In using GP to discover new DRs, GP is rediscovering domain knowledge. DRs which have greater interpretability will reveal more about the important interactions within job shop environments than DRs that we are unable to understand.

3 Method and Experimental Design

This section describes the GP system used for the evolution of DRs and the JSS scenarios used for training and testing of the DRs.

3.1 GP System for Automation Generation of DRs

We implement a GP system using ECJ20 (Luke, 2013) for evolving DRs. The fitness of a DR (individual) in the current GP population is evaluated using discrete-event simulations of problem instances of a job shop. For each problem instance the objective of interest, TWT, is calculated, normalised (by dividing by the average expected utilisation across all machines) and the mean over all problem instances is set as the fitness of the DR. At the end of the evolutionary process, the best-of-run DR is tested on independent test and extreme test problem instances.

3.2 Job Shop Scenarios

We randomly create problem instances with ten machines for all training and testing scenarios. In each problem instance, jobs arrive stochastically according to a Poisson process with rate λ and the average processing time for machines has mean μ (distribution dependent on the scenario, see Table 2). Equation (2) is used to set λ so that the machines have a desired expected utilisation (ρ). The utilisation is the proportion of time that a machine is busy (and $(1-\rho)$ is the proportion of time a machine is idle).

$$\lambda = \frac{\rho}{\mu \times p_M} \tag{2}$$

Here p_M is the probability of a job visiting a machine. For example, with ten machines, if each job has two operations,

$$\lambda = \frac{\rho}{\mu \times (2/10)}. \tag{3}$$

Due dates are set using Equation (4) (Baker, 1984),

$$d_j = r_j + h \times \sum_{l=1}^{N_j} p(\sigma_{j,l}), \tag{4}$$

where h is a due date tightness parameter, randomly chosen from the choices available for each job.

A warm up period of 500 jobs is used, (Holthaus and Rajendran (1997) finds this to be sufficient for the system to reach steady state), and we collect data from the next 2000 jobs to arrive ($N = 2000$), however new jobs keep arriving in the system until the 2500th job is completed.

Training Four job shop problem instances are used for training the population of GP individuals (DRs), one from each scenario in Table 2. At each generation, each rule is used to dispatch jobs for the four scenarios. The average normalised TWT from these scenarios is used as the fitness for the GP individual. The training scenarios are given in Table 2. For all four scenarios the processing times follow a Uniform(1,49) distribution

($\mu = 25$). The four scenarios give two utilisation levels, 85% and 95%, and either "full" (one operation at each machine, giving ten operations per job) or "missing" operations (the number of operations per job is uniformly distributed on $\{2, \ldots, 10\}$). Jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$. This setting has been previously used (Nguyen et al., 2012) and originates from research by Pinedo and Singer (1999) that showed that approximately 20% of jobs are of low importance, 60% are of average importance and the final 20% are very important, so weights of 1, 2 and 4 are assigned to jobs following these probabilities. The due date tightness parameter, h, for each job is chosen from $\{3,5,7\}$ with equal probability.

Testing We use the ten machine job shop of Rajendran and Holthaus (1999) for testing. As in training, the processing time distribution is Uniform(1,49) and jobs are given weight 1, 2 or 4, with probability $(0.2, 0.6, 0.2)$ (Nguyen et al., 2012). There are three due date tightness parameters (4, 6, and 8), four machine utilisations (80%, 85%, 90%, and 95%) and either "full" or "missing" operations. This gives a total of 24 test scenarios which are given in Table 2.

Extreme Testing In order to further test the generalisation ability of the best-of-run evolved dispatching rules, we have an additional set of extreme test instances. We change the distribution of processing times to a geometric distribution with mean $\mu = 25$ (parameter $p = 0.04$). The geometric distribution is chosen as it is a discrete distribution different to the distribution which was used in training. We also change the weights given to jobs, including an additional weight for *extremely* important jobs; jobs are now given weight 1, 2, 4 or 8, with probability $(0.2, 0.5, 0.2, 0.1)$. If jobs with extremely high importance are not correctly scheduled, the penalty is much higher, so this checks the ability of rules to take job weight into account. The due date tightness parameter is equally likely from $\{4,6,8\}$, $\{3,5,7\}$, or $\{3,4,5\}$. This includes some tighter due dates than in training. There are a total of 24 extreme test scenarios which are given in Table 2.

3.3 The GP System

The attributes of the jobs, machines and shop that are used as terminals in the GP system are shown in Table 3. `RJ` is the release time of the job into the system, or the time its last completed operation finished. The properties relating to the job's next operation, `NPR`, `NNQ` and `NQW` will return zero if the job does not have an operation after the current operation. `NQW` and `AQW` return 0 if no jobs have visited the machine yet, and if less than five jobs have visited then the average wait time of the jobs which have visited is returned.

The function set consists of a subset of the following functions $\{+, -, \times, \%$, `if>0`, `max`, `min`, `ifPS`, `ifOD`, `ifLO`$\}$, dependent on which grammar is being used. The arithmetic operators take two arguments. The first three arithmetic operators, $+$, $-$ and $\times$, have their usual meanings. The $\%$ operator is protected division, returning one if dividing by zero. The `max` and `min` functions take two arguments and return the maximum and minimum of their arguments respectively. The `if>0` function takes three arguments; if the first argument is greater than 0 then it returns the second, else the third is returned. The remaining three `if` operators are introduced below.

We will consider using "standard" GP where there is no restriction on which terminals can be arguments for functions, and a version of GP which is restricted by a grammar, implemented using the Strongly Typed GP (Montana, 1995) capabilities of ECJ20 (Luke, 2013). We refer to these methods as GP and STGP respectively. Both

Table 2: Simulation properties (processing time distribution, expected utilisation, due date tightness and operation setting) for training and testing.

		Distribution	ρ	h	Operations
Training	TR1	Uniform(1,49)	0.85	{3,5,7}	full
	TR2	Uniform(1,49)	0.95	{3,5,7}	full
	TR3	Uniform(1,49)	0.85	{3,5,7}	missing
	TR4	Uniform(1,49)	0.95	{3,5,7}	missing
Testing	P1	Uniform(1,49)	0.80	4	full
	P2	Uniform(1,49)	0.80	6	full
	P3	Uniform(1,49)	0.80	8	full
	P4	Uniform(1,49)	0.85	4	full
	P5	Uniform(1,49)	0.85	6	full
	P6	Uniform(1,49)	0.85	8	full
	P7	Uniform(1,49)	0.90	4	full
	P8	Uniform(1,49)	0.90	6	full
	P9	Uniform(1,49)	0.90	8	full
	P10	Uniform(1,49)	0.95	4	full
	P11	Uniform(1,49)	0.95	6	full
	P12	Uniform(1,49)	0.95	8	full
	P13	Uniform(1,49)	0.80	4	missing
	P14	Uniform(1,49)	0.80	6	missing
	P15	Uniform(1,49)	0.80	8	missing
	P16	Uniform(1,49)	0.85	4	missing
	P17	Uniform(1,49)	0.85	6	missing
	P18	Uniform(1,49)	0.85	8	missing
	P19	Uniform(1,49)	0.90	4	missing
	P20	Uniform(1,49)	0.90	6	missing
	P21	Uniform(1,49)	0.90	8	missing
	P22	Uniform(1,49)	0.95	4	missing
	P23	Uniform(1,49)	0.95	6	missing
	P24	Uniform(1,49)	0.95	8	missing
Extra Testing	XT1	Geometric(0.04)	0.80	{4,6,8}	full
	XT2	Geometric(0.04)	0.80	{3,5,7}	full
	XT3	Geometric(0.04)	0.80	{3,4,5}	full
	XT4	Geometric(0.04)	0.85	{4,6,8}	full
	XT5	Geometric(0.04)	0.85	{3,5,7}	full
	XT6	Geometric(0.04)	0.85	{3,4,5}	full
	XT7	Geometric(0.04)	0.90	{4,6,8}	full
	XT8	Geometric(0.04)	0.90	{3,5,7}	full
	XT9	Geometric(0.04)	0.90	{3,4,5}	full
	XT10	Geometric(0.04)	0.95	{4,6,8}	full
	XT11	Geometric(0.04)	0.95	{3,5,7}	full
	XT12	Geometric(0.04)	0.95	{3,4,5}	full
	XT13	Geometric(0.04)	0.80	{4,6,8}	missing
	XT14	Geometric(0.04)	0.80	{3,5,7}	missing
	XT15	Geometric(0.04)	0.80	{3,4,5}	missing
	XT16	Geometric(0.04)	0.85	{4,6,8}	missing
	XT17	Geometric(0.04)	0.85	{3,5,7}	missing
	XT18	Geometric(0.04)	0.85	{3,4,5}	missing
	XT19	Geometric(0.04)	0.90	{4,6,8}	missing
	XT20	Geometric(0.04)	0.90	{3,5,7}	missing
	XT21	Geometric(0.04)	0.90	{3,4,5}	missing
	XT22	Geometric(0.04)	0.95	{4,6,8}	missing
	XT23	Geometric(0.04)	0.95	{3,5,7}	missing
	XT24	Geometric(0.04)	0.95	{3,4,5}	missing

Table 3: Terminal set used in GP system.

Feature	Symbol
Job Properties	
Processing time of operation	PR
Remaining processing time of job	RT
Remaining number of operations	RO
Ready time of jobs	RJ
Due date of job	DD
Weight of jobs	W
Next operation's processing time	NPR
Number of jobs in queue at the next machine job visits	NNQ
Average waiting time of last five jobs processed at the next machine job visits	NQW
Machine Properties	
Ready time of machine	RM
Number of jobs in queue	NQ
Average waiting time of last five jobs processed	QW
Shop Properties	
Current time	CT
Average waiting time of last five jobs processed across all machines in the shop	AQW

methods use the same set of terminals (presented in Table 3) and functions.

We have used different grammars based on splitting the terminal set into four different type categories. One type represents counts: NQ, RO, NNQ. The second type consists of the job weight: W. The third type represents time durations: PR, RT QW, NPR, AQW, NQW. The final type represents absolute (clock) times: DD, RM, RJ and CT.

The basis for creating these grammars has been to prohibit some interactions. The allowable and restricted interactions are shown in Table 4. Black table cells show that the interaction is not allowed, and for interactions that are allowed the resulting type is given. Some of the interactions we restrict include adding or subtracting terminals that are a weight or a count to a time or a duration which does not make sense (or certainly does not make *as much sense* as multiplying or dividing). We have been very harsh in limiting what time terminals can be divided and multiplied by. The if>0 function can take any first argument, but the second and third arguments must be of the same type (and this is the type returned).

3.4 GP Parameter Settings

The initial population is generated using the ramped-half-and-half method (Koza, 1992) and has an initial maximum depth of five. The population size is 1024 and evolution is for 50 generations. GP trees have a maximum tree depth of six. This is a smaller maximum tree depth than is often used, as with the increase in tree size interpretability decreases. For the genetic operators crossover, mutation and elitism, we use rates of 85%, 10% and 5% respectively. Tournament selection with a tournament size of seven is used to select individuals for genetic operators. These are common parameter settings that have been previously used (Nguyen et al., 2013a).

Table 4: Allowable interactions for each operator in $\{\times, \%, +, -, \max, \min\}$. Where interaction is allowed the type of the result is given, and the cell is coloured black if the interaction is not allowed. Terminals are split into four different type categories: C represents counts, X represents job weight (and other ratios), D represents time durations, T represents absolute (clock) times, and I represents inverse time durations.

$\times$	X	C	D	T	I
X	X	X	D		I
C	X	C	D		I
D	D	D			X
T					X
I	I	I	X	X	

%	X	C	D	T	I
X	X	X	I		D
C	X	X	I		D
D	D	D	X	X	
T			X	X	
I	I	I			X

$+$	X	C	D	T	I
X	X				
C					
D			D	T	
T			T		
I					I

$-$	X	C	D	T	I
X	X				
C					
D			D	T	
T			T	D	
I					I

min	X	C	D	T	I
X	X				
C		C			
D			D		
T				T	
I					I

max	X	C	D	T	I
X	X				
C		C			
D			D		
T				T	
I					I

Table 5: Grammar 1 for STGP

S =	<A>		
N =	{A, X, C, T, D}		
Σ =	{×, %, −, +, PR, RT, RO, RJ, DD, W, RM, NQ,		
	QW, CT, NPR, NNQ, AQW, NQW}		
P =	{<A>	::=	<X> | <C> | <D> | <T>
	<X>	::=	(W) | (+ <X> <X>) | (− <X> <X>)
			| (× <X> <X>) | (% <X> <X>)
			| (% <C> <C>) | (% <C> <X>)
			| (% <X> <C>)| (× <C> <X>)
			| (× < *X* > <C>) | (% <D> <D>)
			| (% <T> <D>) | (% <T> <T>)
			| (% <D> <T>)
	<C>	::=	(RO) | (NQ) | (NNQ) | (× <C> <C>)
	<D>	::=	(PR) | (RT) | (QW) | (NPR) | (NQW) | (AQW)
			| (+ <D> <D>) | (− <D> <D>)
			| (− <T> <T>)
			| (× <D> <X>) | (× <X> <D>)
			| (× <C> <D>) | (× <D> <C>)
			| (% <D> <C>) | (% <D> <X>)
	<T>	::=	(DD) | (CT) | (RJ) | (RM)
			| (+ <D> <T>) | (+ <T> <D>)
			| (− <T> <D>) | (− <D> <T>) }

4 Experimental Results

Here we present the results of using GP and STGP, working through iterations of additions to the function and terminal sets. For all methods we perform 50 independent evolutionary runs, using 50 common random seeds, in order that we can perform statistical testing. Tables 6 and 7 give the mean and standard deviation of TWT for one problem instance per scenario. We use only one problem instance for each scenario as within a single instance a DR is called many times.

The performance of the 50 best-of-run evolved DRs is shown in Figures 1 and 2. Figure 1 presents test scenarios T1 to T12, which have the full number of operations in each job, and test scenarios T13 to T24, which have the "missing" number of operations in each job. Figure 2 presents test scenarios XT1 to XT12, which have the full number of operations in each job, and test scenarios XT13 to XT24, which have the "missing" number of operations in each job. For each method there is a histogram. Bars of histograms of different methods have the same height scale. The width of each bar represents the number of DRs which achieved TWT between the values on the vertical axis. The vertical axis is the TWT in 1000s, and the scale of the axis is different for each scenario's plot.

Table 6: Mean and standard deviation of total weighted tardiness on test and extreme test problem instances (lower values are better).

	Benchmark DRs		GP1	ST1	ST1A	ST1B
	wCOVERT	ATC	Mean±Stdev	Mean±Stdev	Mean±Stdev	Mean±Stdev
T1	74607.81	22613.24	18620.4 ± 19703	26531.7 ± 16482.1	32871.4 ± 20068.6	26830.4 ± 16839.7
T2	0	0	94.5 ± 406	105.3 ± 223.6	117.1 ± 185.6	62 ± 137.5
T3	0	0	35.6 ± 208.1	45.5 ± 107.2	64.9 ± 188.4	25.1 ± 77
T4	121130.95	38719.92	40592.2 ± 31286.7	53044 ± 28063.6	63303.5 ± 31973	57686.6 ± 34853.3
T5	10343.88	0	1027.1 ± 1399.6	2038 ± 2241.2	2571.8 ± 3035.3	1932.6 ± 2226.8
T6	938.99	0	483 ± 872.1	1158.1 ± 1545.2	2098.4 ± 3859.9	1095.9 ± 1242.2
T7	259278.68	155171.72	77953.9 ± 56282.9	128149.8 ± 77893.8	154828.3 ± 86266.8	143618.3 ± 104276.2
T8	72425.9	7782.92	4911.5 ± 5414	10150.5 ± 8825.8	14127.7 ± 14980	10617.5 ± 10107.6
T9	903.63	0	379.4 ± 679.3	1086.5 ± 1175	1869.6 ± 3701.1	1218.5 ± 1698.4
T10	1840851.25	1981855.19	4858750.6 ± 1757378.7	5967156.7 ± 2515125	6757291.9 ± 4247212.8	7188202.9 ± 4445614.6
T11	208240.82	123375.34	13927.2 ± 16400.7	27067.2 ± 27683.3	44833.5 ± 49257.9	40448.3 ± 44902.6
T12	189378.95	86525.72	9898.2 ± 17421.7	10008.1 ± 13762.9	25239 ± 39673.6	21625.3 ± 27940.5
T13	76278.61	20238.67	19170.5 ± 14379.1	30442.6 ± 20745.2	35971.2 ± 21358.5	30854.2 ± 21790.8
T14	1159.52	54.54	499.9 ± 780.9	1393.9 ± 1604.7	1352.3 ± 1497.7	863.6 ± 899.2
T15	0.75	3.55	78.9 ± 209.7	319.5 ± 555.9	345.3 ± 539.7	304.2 ± 526.8
T16	77436.51	20683.97	17528 ± 14731.9	28248.2 ± 19595.9	34451.5 ± 20482.5	28905.9 ± 21859
T17	100027.62	78622.32	30245.6 ± 23444.2	44064.9 ± 27911.2	60446.8 ± 40782.4	58269.9 ± 41473.5
T18	82.82	0	144 ± 321.1	698.5 ± 1112.2	698.5 ± 1185	455.5 ± 605.4
T19	306873.71	264216.57	211110.5 ± 34468.8	280645.5 ± 103866.7	306162.1 ± 108975.8	330419.2 ± 153391.4
T20	148573.57	102920.42	34813.2 ± 21809.5	55209.7 ± 35560.7	67531.3 ± 41671.7	72998.3 ± 56150
T21	148.12	26.71	702.4 ± 1070.4	1956.4 ± 2279.6	2877.8 ± 4722.3	1856.1 ± 1836
T22	970487.89	1022018.86	2131306.6 ± 742620.5	2498801.9 ± 943851.3	2504233.1 ± 951527.7	2554825.1 ± 937928.6
T23	1019714.74	901866.67	1067595.6 ± 211617.1	1432053.8 ± 629594.9	1551080.4 ± 734545.9	1673677 ± 891556.3
T24	47578.26	12591.08	10328.3 ± 15270.5	18193 ± 18026.2	28139.5 ± 35835.6	25178.8 ± 32389.9
XT1	1495.05	0	839.6 ± 1992.2	2626.7 ± 3402.2	2020.3 ± 2519.2	1533.3 ± 2187.1
XT2	41436.18	15270.89	3845.9 ± 6936.7	12454.5 ± 13604.4	10579.7 ± 11039.8	8109.6 ± 8335.9
XT3	1625.45	0	44208.1 ± 13438.2	77340.6 ± 55876	83068.1 ± 56806.9	65983 ± 48828
XT4	48998.52	5117.16	1837.4 ± 3792.7	10500.8 ± 12406.8	10482.9 ± 11787.8	7894.8 ± 8925.6
XT5	127840.53	60816.7	9236.6 ± 12572	37711 ± 38678	39982.9 ± 42297.6	31824.1 ± 38007.7
XT6	5302.12	42.63	70475.7 ± 41438.1	260283.5 ± 271169.8	257269.7 ± 240524.6	216565.6 ± 224909.7
XT7	113330.71	40819.43	6802.7 ± 9533.5	30570.1 ± 33198	34326.3 ± 39982.9	25104.5 ± 34560.9
XT8	298810.15	215212.49	37050.8 ± 35976.1	146640 ± 135460.9	157319.1 ± 149736.7	127298 ± 140196.5
XT9	778883.59	894735.09	186669.1 ± 94106.5	1325494.4 ± 3215605.2	852530.7 ± 816570.4	719792.5 ± 682277.8
XT10	916183.55	1035644.41	457585.2 ± 387358.4	2278321.6 ± 4219762.1	1815540.7 ± 1610245.4	1810411 ± 1357945
XT11	865992.09	745914.21	3426312.2 ± 2502422	6600602.5 ± 5267889.8	6745402.1 ± 5211364.3	6766461.7 ± 4729540.5
XT12	680.71	459.95	4845904.4 ± 3536695	7327446.6 ± 4749830.7	8247666.1 ± 5947853.9	9142446.6 ± 6333621.4
XT13	26150.96	15806.03	1727.7 ± 2285.2	6550.4 ± 6701.5	6679.5 ± 7141.9	5614.5 ± 6556
XT14	38192.13	13676.16	17684.9 ± 17318.6	56405.3 ± 50496.3	56424.3 ± 51620.8	48782.4 ± 51662.4
XT15	1824.9	1247.71	27490.9 ± 12945.8	69746.4 ± 56469.6	69237.7 ± 53841.9	61891.5 ± 53771.9
XT16	27222.08	14307.91	2929.1 ± 3178	13445 ± 16971.1	12571.4 ± 16056.8	10507.8 ± 11786.6
XT17	271583.74	263100.66	29152.5 ± 15765.6	61071.6 ± 45194	60208 ± 40680.3	53975.2 ± 42656.7
XT18	163517.5	80359.16	556743.8 ± 191343.2	1079218.6 ± 606650.9	1089129.9 ± 602294.1	1117038.2 ± 646134.5
XT19	91101.93	72758.91	79232.9 ± 33045.5	140976.3 ± 81261.6	151145.7 ± 88299.6	155978.2 ± 160220.8
XT20	133643.86	120459.95	66293.2 ± 34462.6	150761.6 ± 122068.2	166120.9 ± 136636.5	140002.9 ± 119349.1
XT21	570388.46	581547.71	275727.1 ± 112163.7	576309.2 ± 357192.5	614901 ± 376547.7	566211.1 ± 343913.8
XT22	454536.89	420731.17	2638620.4 ± 1292992.8	3278079.5 ± 1591076.1	3240274.4 ± 1510740.7	3286780.7 ± 1631151.2
XT23	428607.7	432883.83	1436336.5 ± 583534.5	2046697.6 ± 1514460.9	1955097 ± 891580.2	1909147 ± 945207.1
XT24	227787.75	205629.16	4854593.2 ± 2988638.3	4870916.5 ± 2763309.3	5017262.1 ± 3032322	5220491.3 ± 3556428

Table 7: Mean and standard deviation of total weighted tardiness on test and extreme test problem instances (lower values are better).

	GP2 Mean±Stdev	ST2 Mean±Stdev	ST2A Mean±Stdev	GP3 Mean±Stdev	ST3 Mean±Stdev
T1	24773.8 ± 22258.5	28579 ± 18402.5	29872 ± 18329.5	33612.2 ± 9796.4	30563.1 ± 14272.8
T2	38.3 ± 102.2	197.5 ± 405.7	201.1 ± 492	639.4 ± 544	431.5 ± 573.2
T3	40.4 ± 107.4	238 ± 873.7	68.3 ± 180.5	1350.5 ± 1140.8	740.2 ± 1131.3
T4	51095.2 ± 34739.9	56587 ± 30169.7	59488.9 ± 26846	60509.5 ± 15321.1	56746.7 ± 21450.8
T5	2029.8 ± 2354.8	2719.7 ± 5392.1	2411.9 ± 3349.8	10720.6 ± 6507.5	6238.7 ± 6728.5
T6	793.2 ± 1054.5	2602.2 ± 6596.4	1594.9 ± 2124.6	8360.9 ± 5643.2	5393.1 ± 6191.3
T7	110939.2 ± 75835.9	140111.1 ± 80244.7	162299.4 ± 97624	129015 ± 33633.5	140445.8 ± 58753.7
T8	10550.6 ± 10608.4	14481.7 ± 18783.8	13518.7 ± 12091.7	27949.5 ± 14274.9	22329.7 ± 18269.4
T9	722.8 ± 1120	3118.3 ± 7939	1291.7 ± 1611.2	12002.9 ± 8223.6	6457.9 ± 7847.9
T10	3575670.3 ± 1827093.6	5103150.3 ± 3042384.2	6727980.2 ± 5568178.8	3145499.8 ± 1834005.8	4175807.7 ± 2864281.2
T11	35386.5 ± 47386.6	48157.5 ± 56952.9	51122.1 ± 51552.4	67712 ± 30844.9	54664.5 ± 42606.2
T12	28618.4 ± 35851.2	29146.6 ± 46864	23050.3 ± 30567.2	49615.5 ± 27694	36433.9 ± 34880.3
T13	24442.2 ± 17160.1	31579.9 ± 21052.7	35856.3 ± 22507.5	33668.5 ± 8475.4	33196.3 ± 15442
T14	797.5 ± 1299.6	1711.5 ± 3055.3	1455.8 ± 1694.3	6293.6 ± 4010.8	4245.2 ± 4537.8
T15	78.9 ± 173.2	435.4 ± 935.9	325.7 ± 490.1	1788.6 ± 1390.1	1153.4 ± 1418
T16	22614.7 ± 16238.4	30470.5 ± 20548.8	34346.9 ± 22957.9	34090.4 ± 9549.4	33463.1 ± 16224
T17	43970.6 ± 34054.4	53878.1 ± 34819.1	60880.7 ± 35844.5	54939.4 ± 14678.5	56006.5 ± 25696.4
T18	296.8 ± 549.1	837.5 ± 1779.9	672.9 ± 826.7	4203.6 ± 2942.6	2547.7 ± 3022.8
T19	237696.1 ± 51878.6	278882.5 ± 88635	303171.7 ± 110099.7	228178.2 ± 43843.1	260078.5 ± 57227.4
T20	54789.4 ± 37518	67862.9 ± 43310.8	68184.1 ± 39831.8	69460.9 ± 20946.9	64913.1 ± 30238.7
T21	1413.8 ± 2022.3	3410.4 ± 6617.1	2014.1 ± 2115.7	12729.4 ± 8132.9	7981.6 ± 8478.1
T22	1670375.3 ± 583730.5	2207079.3 ± 995533.5	2738397.4 ± 2143717.9	1468687.6 ± 376933.1	1859420.2 ± 1023172.5
T23	1031699.8 ± 211353.9	1270494.5 ± 514338.2	1463156.2 ± 730838.7	832372.5 ± 170750.4	1064701.1 ± 406980.4
T24	20647.9 ± 23605.4	27585 ± 33560.2	24297.4 ± 25785.5	52003.2 ± 29316.8	38457.3 ± 33837.3
XT1	866.6 ± 1860.9	1871.3 ± 2484.3	2567.5 ± 4138.5	5585.8 ± 4176.8	6623.5 ± 8072.2
XT2	4460.5 ± 5256.9	9062.9 ± 10201.2	12898.6 ± 15092.6	12515.5 ± 7717.6	16122.3 ± 15636.4
XT3	41921 ± 23271.2	65256.8 ± 46639.2	81944.7 ± 56086.2	57053.4 ± 22683.8	75824.3 ± 35778.7
XT4	3600.2 ± 5955.2	8771 ± 11069.4	10513.6 ± 13761.6	17589 ± 12271.2	20002.2 ± 21374.1
XT5	15212.9 ± 15976.4	32438.1 ± 37828.2	47192.4 ± 51427.8	35661.2 ± 18954.8	50651.8 ± 44239.7
XT6	109770.3 ± 94304.6	212700.2 ± 185478.4	343919.3 ± 298896.8	162760.3 ± 83161.7	263476.9 ± 158978
XT7	10380.9 ± 12995.6	26378.9 ± 35020	38605.5 ± 48648.2	33639.8 ± 19850.7	46775.3 ± 48206.6
XT8	59740 ± 51070.6	133625.9 ± 148854	207873.4 ± 172959	97922 ± 49995.3	171796.5 ± 122011
XT9	296941 ± 254735.9	701654.7 ± 700422.3	1258956.7 ± 1473801.7	476989.7 ± 298486.3	789439.2 ± 495546.5
XT10	751358.5 ± 607389	1353870.9 ± 1271737.7	1941507.9 ± 1547631.3	751588.3 ± 354646.6	1373278.1 ± 936184.2
XT11	2600405.9 ± 1716016.2	4721098.5 ± 3876858.8	6456746.6 ± 6029924.6	2349615.7 ± 987453	3887599.7 ± 2592762.9
XT12	3397430 ± 2992225.2	6345487.2 ± 5065572.8	8202216 ± 7360155.4	2627453.8 ± 1218917.1	4407110.9 ± 3531301.5
XT13	2633.1 ± 3104.9	4956 ± 5486.3	7730.2 ± 9021.4	10151.6 ± 6776.2	10941.6 ± 10834
XT14	26547 ± 22032.3	45894.4 ± 43660.4	60088.8 ± 49727.9	43724.5 ± 22594.9	60368.3 ± 41481.6
XT15	33873.4 ± 20238.7	57418.5 ± 45119.5	77919.8 ± 57787.8	61714.7 ± 25193.8	71612.8 ± 38921.8
XT16	5107.1 ± 6867.4	11833.4 ± 14164.7	13400.9 ± 14203.6	18404.1 ± 12746	19899.3 ± 18441.9
XT17	37592.2 ± 23203	53070.5 ± 39649.6	61804.8 ± 38134.3	59779.3 ± 26957.9	69654.8 ± 40704.7
XT18	566907.1 ± 241945.3	969576.2 ± 608931.7	1218757.6 ± 838939.1	645855.6 ± 181925	893531.9 ± 409308.1
XT19	97910.1 ± 50899.1	132515.6 ± 84911.4	147089.5 ± 78345.2	127506.5 ± 43260.7	143898.7 ± 67458.7
XT20	79617.6 ± 44960	129630.6 ± 103966.3	170415.4 ± 117774	104074 ± 39349.7	147327 ± 73759.1
XT21	283775.9 ± 127774.4	470231.2 ± 321177.3	646679.5 ± 447791.3	355661.8 ± 120027.8	481870.7 ± 227198.4
XT22	1805194.1 ± 878782.8	2474629.4 ± 1419902	2918387.4 ± 1475730.7	1277834.8 ± 392185.2	1983138.2 ± 1032717.2
XT23	1079405.5 ± 564285.5	1480569 ± 861032.9	1872145.2 ± 1034348.1	799482.9 ± 245972.9	1270459.4 ± 644226.2
XT24	2415363.7 ± 1896732.4	3783461.9 ± 2714613.4	4383378.1 ± 3237748.4	1642736.1 ± 626408.3	2797480.5 ± 2113749.5

4.1 Stage One: Arithmetic GP and STGP

First we use the grammar in Table 5 to restrict STGP. This uses a reduced function set of only the four arithmetic operators $\{+, -, *, \%\}$. We compare rules evolved under this grammar to GP with the same arithmetic function set. We will call these two methods GP1 and ST1. We start with this arithmetic function set as DRs with these are most straightforward to understand.

Initial results of GP1 and ST1, shown in the first two vertical histograms of Figures 1 and 2 and in Table 6, show that DRs evolved under ST1 do not perform as well as those evolved under GP1. We performed a paired Mann Whitney U Test which showed that the difference of mean TWT is statistically significant (GP1 with lower) on 35/48 test and extreme test scenarios. We do not expect ST1 to attain the same level of performance as it is searching a heavily restricted search space. In general the results in Figures 1 and 2 show approximately the same shape for lower TWT values, but ST1 has a much longer tail (worse TWT performance) particularly on extreme test scenarios.

We wondered if this could be due to the grammar not allowing interactions which give inverse durations, e.g., $1/PR$ or W/PR, and squared durations, e.g., $RT \times PR$.

To investigate this premise we introduce a fifth type, I, to represent inverse-durations. The allowable interactions are shown in Table 4. Due to the implementation of strong typing in ECJ, to allow this additional type we need to define a terminal of type I. Firstly we add in a new terminal "invPR" which is $1/PR$ (we know PR is always > 0), and call this ST1A. Secondly, to the original terminal set we add in a new terminal "W/PR", which returns W/PR, since this, by itself, is a common rule, and is also a component of many of the successful dispatching rules from the literature for the dynamic TWT problem, in particular ATC and wCOVERT mentioned in Section 2. We call this ST1B. Of ST1, ST1A and ST1B, the best results are given by ST1A with statistically significantly lower means than ST1 on 3/24 test instances and 6/24 extreme test. GP1 has statistically significantly lower means than ST1A on 25/48, only two less than compared to ST1. There was no statistically significant difference of means at 5% between ST1 and ST1A, or between ST1 and ST1B. The performance of these two methods in comparison to ST1 did not convince us that it made a difference either in improving or hindering the ability of GP to find effective dispatching rules. Table 6 shows that the mean performance of GP1, ST1, ST1A and ST1B all have large standard deviations.

4.2 Stage Two: Inclusion of If, Max and Min

We introduce `if>0`, `max` and `min` into the function set of the second grammar. The allowable interactions are shown in Table 4. The first argument, the condition, of the `if>0` function can be a large or complex expression for which it is difficult to understand why different behaviour should happen based on the outcome of the given expression. Again we compare this to standard GP with the same function set. We will call these two methods GP2 and ST2.

Results from the best-of-run DRs from GP2 and ST2 are shown in the middle two vertical histograms of Figures 1 and 2 and in Table 7. These show that DRs evolved under ST2 do not perform as well as those evolved under GP2, with a greater number of DRs attaining high TWT values.

In the evolved best-of-run DRs from GP2 there are 81 `if>0` statements in 50 best-of-run evolved DRs (see Table 8). The evolved conditions (i.e. the first argument) of GP2 are frequently long, difficult to interpret, and/or it does not make sense to branch behaviour dependent on the condition value. Of the 81, 52 remain after simplification: 10 have complex first arguments, 21 have `NQW`, `NPR` or `NNQ`, 11 have `QW` or `AQW` (only

very rarely equal to zero), five NQ−NNQ, and five which are comparisons between the current time and the due date. This is a very high proportion of if>0 statements which are not playing an active part in the DR, and only 34/50 of the best-of-run evolved DRs contain if>0 statements. This caused us to question how useful the three argument if>0 is as a function, and if it could be better replaced by several if statements with predetermined condition statements.

As for GP1 and ST1, when we look at GP2 and ST2, there is a statistically significant difference of means at a significance level of 5% in favour of GP2 in 35/48 test instances. This decreased to 32/48 when we compared GP2 to ST2A (with the fifth inverse-duration type, and terminal 1/PR), and when comparing ST2 to ST2A, the mean TWT was improved by ST2A on 13 test instances. In this case the inclusion of the inverse-duration type seems to improve mean performance more than it did with arithmetic operators only.

4.3 Stage Three: Specialised Conditionals

The third grammar uses the arithmetic operators, the max and min operators and three new conditional operators: ifPS, ifOD and ifLO. The new conditional operators each have a fixed condition, and take two arguments. The ifPS operator returns the first argument if the slack is non-negative, i.e., it is possible for the job to be completed on or before its due date ($\mathtt{DD} - \mathtt{CT} - \mathtt{RT} \geq 0$) and returns the second argument otherwise. The ifOD operator tests if the job is already overdue, if the job is overdue ($\mathtt{DD} \leq \mathtt{CT}$) the first argument is returned, else the second argument is returned. The final new conditional operator, ifLO, returns the first argument if it is the last operation of the job, or the second argument if it is not. These methods will be referred to as GP3 and ST3.

We observe that GP3 and ST3 best-of-run DRs have much longer tails of high TWT than GP1/ST1 and GP2/ST2. Figures 1 and 2 show that on a number of test scenarios the ST3 vertical histogram shows lower TWT is achieved than under GP3. This is particularly obvious on, e.g., T7 and T13, and occurs less on extreme test instances.

Figures 1 and 2 show that the overall shape of the vertical histograms are similar for GP and STGP methods with the same function set. This shows that we are achieving a similar distribution of TWT from the best-of-run DRs with grammars restricting the allowable interactions. From Tables 6 and 7 it is clear that all methods have large standard deviations relative to the means.

5 Analysis of Evolved Rules

Table 8 shows the number of occurrences of the terminals and functions in the 50 best-of-run DRs, as well as the number of DRs the terminal/function appears in, for GP and STGP. This is constructed from unsimplified DRs. If we were to manually simplify the rules then the values in this table would change, as there are often fragments which are redundant or cancel out. However, simplification is sometimes subjective and is beyond the scope of this paper.

The most frequently used terminals across all methods are PR, DD, W, NQ and NNQ. The calculation of TWT requires both the job's weight and due date, so we expect both W and DD to appear in DRs for this objective function for effective performance. The least used terminals are NQW and NPR, the expected queue wait at the next machine the job visits and the processing time at that machine. RJ is also one of the least frequently used terminals, often CT will be approximately equal.

The introduction of three if statements with predetermined conditions has given an increase in the overall number of conditional statements used. Further, the most

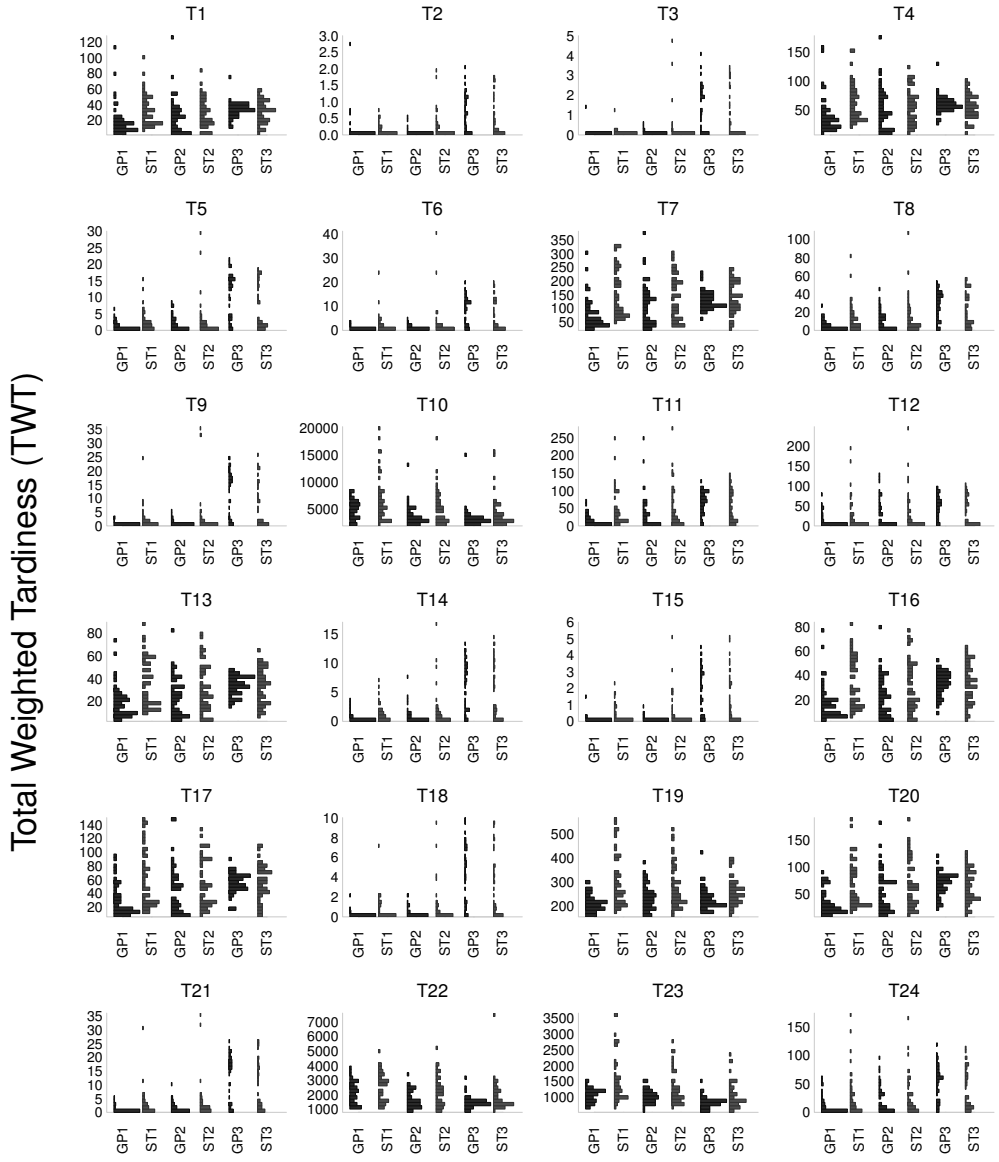

Figure 1: Histograms presenting results (TWT in 1000s) of test scenarios for GP and STGP. Note that the vertical axis scales are different in each subfigure.

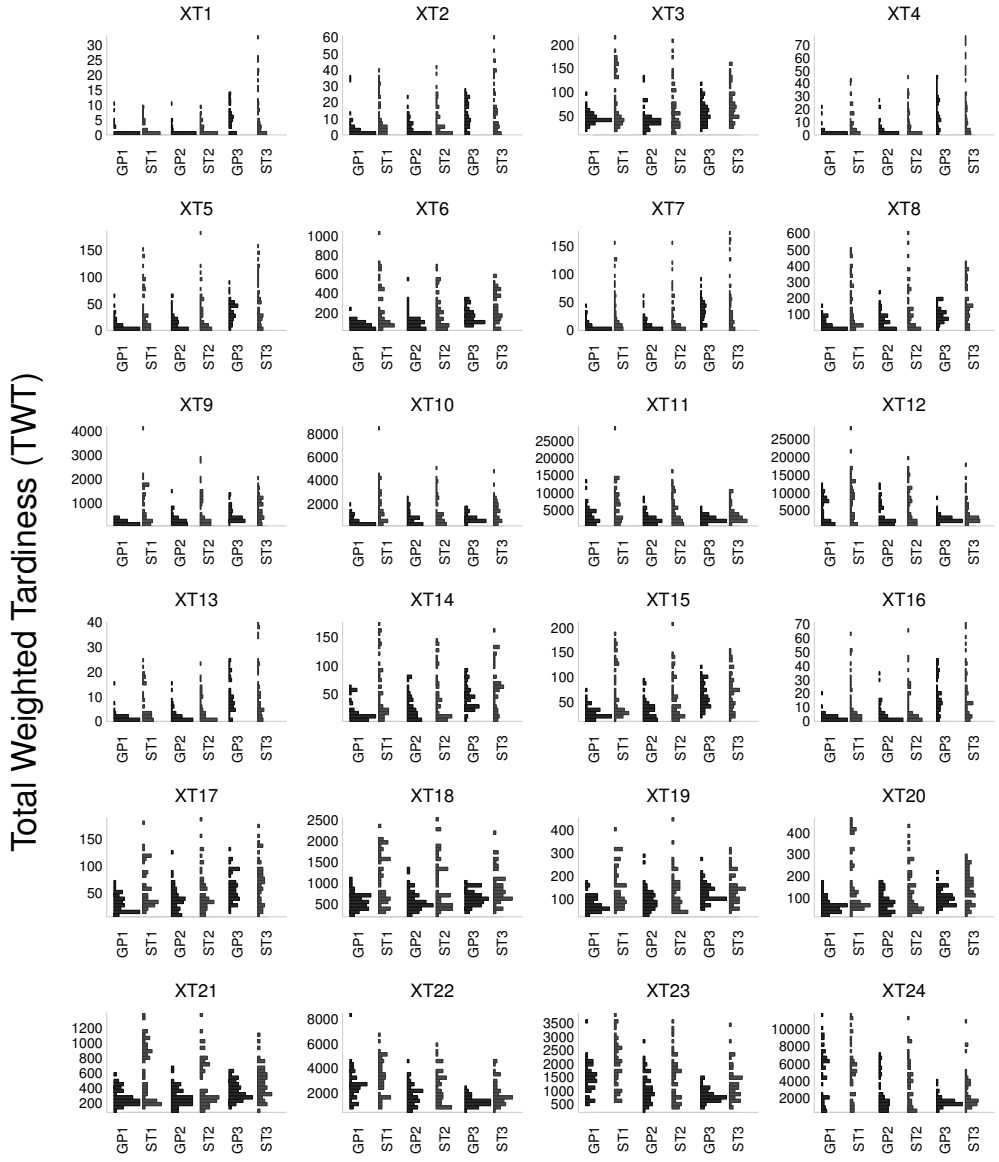

Figure 2: Histograms presenting results (TWT in 1000s) of extreme test scenarios for GP and STGP. Note that the vertical axis scales are different in each subfigure.

Table 8: Number of occurrences of terminals in rules and number of rules (out of 50) that terminals occur in.

	GP1		ST1		ST1A		ST1B	
	Total	Rules	Total	Rules	Total	Rules	Total	Rules
+	194	48	94	41	96	39	76	38
−	230	50	139	50	169	49	154	50
*	177	49	194	46	221	48	170	48
%	190	48	175	49	145	44	127	43
DD	146	50	83	50	88	50	71	50
PR	137	50	85	42	131	46	122	45
W	120	42	111	42	101	38	122	38
RT	93	42	56	34	52	31	53	29
RO	56	28	72	38	71	30	61	30
RJ	28	16	12	9	17	11	3	3
RM	50	29	46	28	45	32	39	28
NQ	131	48	145	44	129	44	129	44
QW	69	33	52	34	36	21	35	21
CT	90	38	53	36	51	33	44	29
NPR	11	8	14	11	24	20	23	17
NNQ	85	47	73	42	55	39	53	37
NQW	10	9	11	10	4	4	4	4
AQW	41	24	27	22	15	11	19	11
1/PR	-	-	-	-	73	30	-	-
W/PR	-	-	-	-	-	-	63	31
Total	1817		1442		1523		1368	
	GP2		ST2		GP3		ST3	
	Total	Rules	Total	Rules	Total	Rules	Total	Rules
+	143	42	76	34	81	38	72	30
−	195	50	144	50	90	37	103	41
*	145	46	154	47	75	36	141	48
%	136	48	144	47	130	40	144	41
max	79	36	73	32	61	31	51	30
min	93	33	53	29	64	29	39	21
if>0	81	34	66	32	-	-	-	-
ifPS	-	-	-	-	133	50	87	44
ifOD	-	-	-	-	54	28	39	25
ifLO	-	-	-	-	58	34	55	33
DD	146	50	103	49	51	29	74	42
PR	177	50	147	47	162	49	81	35
W	159	43	129	41	202	50	178	44
RT	88	33	61	33	28	17	28	14
RO	49	29	67	31	62	30	73	30
RJ	50	23	26	18	39	24	24	14
RM	85	36	53	33	54	28	54	27
NQ	180	48	190	48	178	50	216	48
QW	105	35	67	33	118	44	72	30
CT	103	40	62	37	43	26	68	32
NPR	36	18	22	19	17	10	20	11
NNQ	101	41	86	42	91	48	103	48
NQW	34	18	12	10	23	14	15	8
AQW	24	15	28	21	46	28	29	17
1/PR	-	-	86	37	-	-	-	-
W/PR	-	-	-	-	-	-	-	-
Total	2209		1849		1860		1766	

useful of the three, based upon overall frequency and number of rules it appears in, is `ifPS`, where the condition is the slack of the job. Every rule in GP3 uses the `ifPS` function, which is far more often than the `if>0` function is used in GP2 (34/50 rules). With the introduction of the specialised conditionals, the number of rules where the arithmetic operators are present is less. Also, although the terminal `DD` appears less frequently, in only 29 GP3 rules and 42 ST3 rules, it is present in the condition of `ifPS`.

5.1 Fragment Analysis

We have also analyzed which are the most common two-level fragments, i.e., one function with terminals as all its arguments. The most common fragments in GP1, ST1, GP2, ST2 and ST3 are `(- CT DD)` or `(- DD CT)`, the time left until the job is overdue. The most common fragment in ST1A and ST1B is similar, `(- RM DD)` which is also comparing the time to the due date.

It is interesting to compare the number of occurrences of these fragments in total to the number of rules the fragment appears in. For example, the fragments `(- CT DD)` or `(- DD CT)` occur 36 times but in only 23 distinct rules under GP1.

The most common fragments of GP3 are the most different to the other methods. The most common across all methods are differences between the current time or machine ready time and the job due date. Ratios of current time or machine ready time to job processing time appear in the top 10 of GP1, GP2 and GP3, yet not in the top 10 of any STGP methods (even though allowed).

5.2 Best Evolved DRs

We have selected some of the best evolved DRs from each method, based on comparing their performance across test and extreme test instances with the performance of ATC and WCOVERT. The DRs shown have not been simplified. The DRs evolved by ST methods are shorter and easier to interpret. Figures 3 and 4 show one of the best rules evolved under GP1 and ST1 respectively. In Figure 3 function nodes which take argument pairs that would not be allowed under the grammar are shaded. This affects nodes higher up the tree. This DR cannot easily be interpreted, but we can see a fragment that occurs twice in the tree is $\frac{\mathtt{PR}}{\mathtt{RT}}(\mathtt{DD}-\mathtt{CT})$. This is the ratio of the processing time of the current operation to the total remaining time, multiplied by the time until the job is due. The DR of Figure 4 performed similarly to the DR of Figure 3 in terms of performance relative to ATC and WCOVERT, and also contains the fragment $\frac{\mathtt{PR}}{\mathtt{RT}}(\mathtt{DD}-\mathtt{CT})$. The presence of this fragment in two DRs with very good performance suggests that this could make a useful higher level terminal.

Figures 5 and 6 show one of the best rules evolved under ST1A and ST1B respectively. The DR of Figure 5 shows that wherever the terminal `1/PR` appeared, it was multiplied by `W`. This is one reason `W/PR` was used as the only I type terminal for ST1B. With some simplification this rule becomes

$$\frac{\mathtt{PR}}{\mathtt{RT}}(\mathtt{RM}-\mathtt{DD})\left(\frac{\mathtt{RO}/\mathtt{W}+1}{\mathtt{RO}*\mathtt{RJ}}\right)$$

Note that this has a very similar fragment to that mentioned above, with $(\mathtt{RM}-\mathtt{DD})$ instead of $(\mathtt{DD}-\mathtt{CT})$. The ready machine time will often be the same as the current time, unless the machine has been idle and two jobs arrive at the machine queue at the same time. The DR of Figure 6 simplifies to an expression similar to that of Figure 5:

$$\frac{\mathtt{PR}}{\mathtt{RT}}\frac{(\mathtt{RM}-\mathtt{DD})}{\mathtt{DD}}$$

Table 9: Number of occurrences of fragments of depth two. We have combined (op A B) and (op B A) and present the top 10 most common for each method.

GP1	Total	Rules	ST1	Total	Rules
(− DD CT)	36	23	(− DD CT)	19	15
(− DD RM)	27	18	(− DD RM)	17	11
(− RT DD)	14	8	(∗ RO RO)	24	11
(% PR CT)	11	8	(% W NNQ)	11	10
(+ NQ NNQ)	7	6	(∗ RO NQ)	12	9
(∗ PR AQW)	7	6	(∗ NQ NNQ)	12	9
(% PR RT)	11	6	(∗ RT RO)	6	6
(% W NNQ)	7	5	(∗ NNQ NNQ)	14	6
(+ PR PR)	10	4	(∗ PR NQ)	6	5
(− PR RT)	4	4	(∗ PR NNQ)	6	5
ST1A	Total	Rules	ST1B	Total	Rules
(− DD RM)	21	16	(− DD RM)	17	15
(− RT DD)	14	12	(− DD CT)	17	13
(− DD CT)	15	10	(∗ RO NQ)	14	9
(∗ NQ NNQ)	12	9	(+ RT RM)	10	8
(∗ RO NQ)	9	7	(∗ NQ NQ)	18	7
(% W NNQ)	7	6	(∗ NQ NNQ)	7	7
(+ PR DD)	6	5	(∗ RO RO)	12	6
(∗ RO RO)	18	5	(∗ NNQ NNQ)	10	5
(∗ RO W)	7	5	(− RT DD)	4	4
(∗ NQ NQ)	16	5	(∗ PR NQ)	6	4
GP2	Total	Rules	ST2	Total	Rules
(− DD CT)	31	18	(− DD CT)	18	13
(− DD RM)	30	14	(− RT DD)	13	10
(% PR DD)	11	8	(− DD RM)	19	9
(∗ RT W)	6	5	(% W NNQ)	7	7
(% PR RM)	9	5	(max NQ NNQ)	8	6
(% PR CT)	6	5	(∗ NQ NNQ)	6	5
(+ RT CT)	6	4	(∗ RT W)	6	4
(∗ PR NQ)	7	4	(∗ RO NNQ)	4	4
(% PR RJ)	7	4	(max DD CT)	6	4
(+ PR W)	7	3	(+ CT AQW)	4	3
GP3	Total	Rules	ST3	Total	Rules
(% NQ NNQ)	8	6	(− DD CT)	15	11
(% W NNQ)	6	5	(% W NNQ)	17	11
(− PR W)	6	4	(− DD RM)	8	7
(− DD RM)	7	4	(∗ PR NNQ)	8	6
(− DD CT)	5	4	(ifOD RO NQ)	7	5
(% PR W)	7	4	(− PR CT)	6	4
(min RT NNQ)	4	4	(∗ RO NQ)	5	4
(∗ NNQ AQW)	3	3	(% DD CT)	7	4
(% PR RO)	9	3	(ifPS RO NQ)	6	4
(% PR RJ)	3	3	(+ W W)	8	3

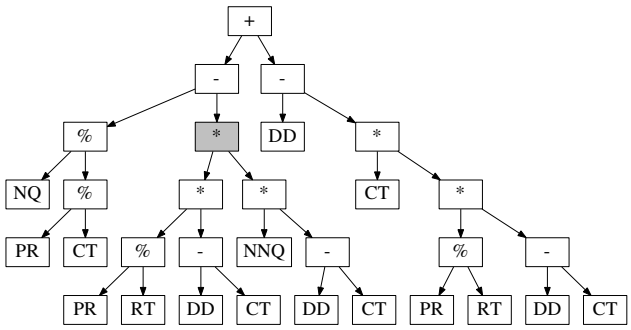

Figure 3: An evolved rule from *GP1* which outperformed ATC on 34/48 test instances and WCOVERT on 36/48 test instances.

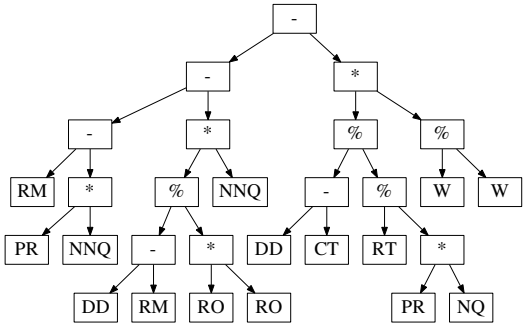

Figure 4: An evolved rule from *ST1* which outperformed ATC on 33/48 test instances and WCOVERT on 39/48 test instances.

Figures 7 and 8 show one of the best rules evolved under GP2 and ST2 respectively. Figure 7 shows one of the best performing DRs evolved by GP2. This rule simplifies to the DR in the right of Figure 7. This DR outperformed or equalled WCOVERT on 47/48 test and extreme test instances and equalled or outperformed ATC on 44/48 test and extreme test instances. However the shaded nodes show that this rule has an expression $\texttt{NNQ} + \texttt{W} - \texttt{RT} - \texttt{RM} + \texttt{DD}$ which adds or subtracts a weight, a count, a time duration and fixed points in time, i.e., the very interactions we are trying to prevent.

The DR from ST2 shown in Figure 8, shows an example of how the `if>0` operator can have a first argument that is always positive, namely `if>0 DD`, where `DD` is always positive. This rule assigns priority

$$\frac{-\texttt{PR}}{\texttt{DD} + \texttt{AQW}}$$

if the job is overdue, and

$$\left(\frac{-\texttt{PR}}{\texttt{DD} + \texttt{AQW}}\right)\left(\frac{\texttt{CT} * (\texttt{DD} - \texttt{CT} + \texttt{RT})}{\texttt{DD} * \texttt{RT}}\right)$$

if the job is not overdue.

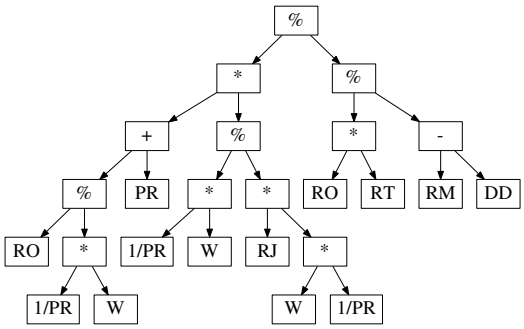

Figure 5: An evolved rule from *ST1A* which outperformed ATC on 25/48 test instances and WCOVERT on 32/48 test instances.

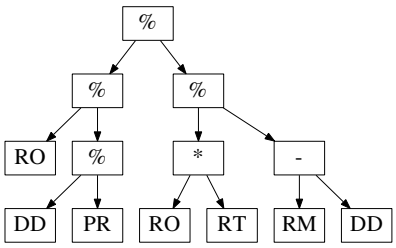

Figure 6: An evolved rule from *ST1B* which outperformed ATC on 34/48 test instances and WCOVERT on 38/48 test instances.

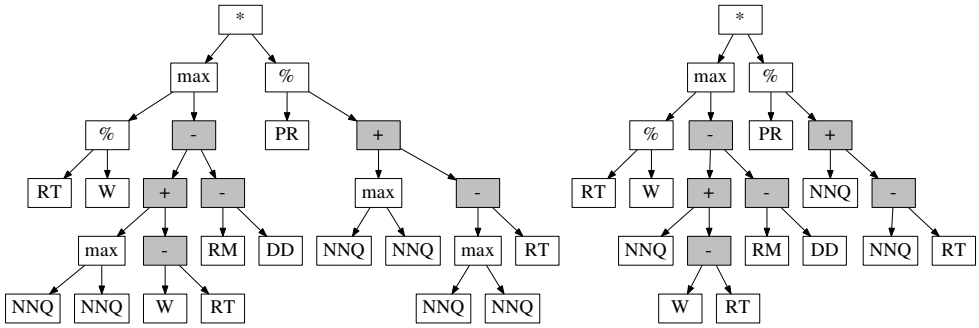

Figure 7: An evolved rule from *GP2*, and in simplified form on right, which outperformed ATC on 44/48 test instances and WCOVERT on 47/48 test instances.

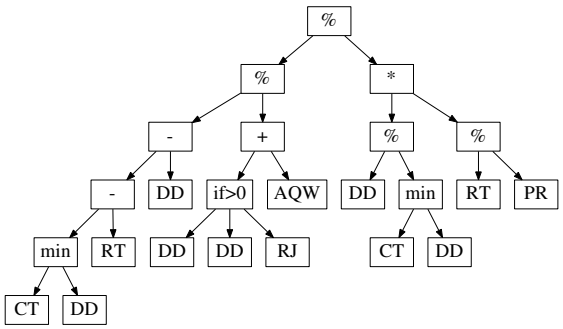

Figure 8: An evolved rule from *ST2* which outperformed ATC on 35/48 test instances and WCOVERT on 43/48 test instances.

Figure 9 shows one of the best performing DRs evolved by ST2A. This performance is not quite as good as that of the DR of Figure 7, however it simplifies to

$$\frac{(\min\{\mathtt{RM}, \max\{\mathtt{DD}, \mathtt{CT}\} - \mathtt{RT}\} - \max\{\mathtt{DD}, \mathtt{CT}\})\mathtt{PR}}{\mathtt{RT}}$$

which is relatively easy to interpret, and allows us to understand why it performs so well. There are three different cases: (1) if the job is overdue and has been waiting in the queue for the machine to become available, then the job is assigned priority 0; (2) if the job is not overdue but cannot complete on time ($\mathtt{DD}-\mathtt{RT}<\mathtt{RM}$), then the job is assigned priority proportional to $(-\mathtt{PR})$, which is always negative as processing time is always positive; and (3) if the job is not overdue and has positive slack, the job is assigned priority proportional to

$$\left(\frac{\mathtt{DD} - \mathtt{RM}}{\mathtt{RT}}\right) * (-\mathtt{PR}) .$$

This is the priority of situation (2) multiplied by the ratio of remaining time until overdue to remaining processing time, which, as we know the job has positive slack, must be greater than 1. The priority assigned in this case is also always negative, therefore for two jobs with the same PR, a job which had positive slack will be assigned a lower priority than one which does not have positive slack.

Figures 10 and 11 show one of the best rules evolved under GP3 and ST3 respectively. The shaded nodes in the DR in Figure 10 show that this rule has a fragment which subtracts RJ (a clock time) and NNQ (a count). The DR from GP3 shown in Figure 10 has each of the new conditional operators in it, however the arguments returned are almost always of different return types. Therefore even though the branching condition makes sense, the different outcomes dependent on the condition do not make so much sense. With so many conditional operators with these arguments, it is difficult to understand what the DR is doing. One feature of the grammar used is that all count terminals are of the same type, allowing fragments like $\max\{\mathtt{NNQ}, \mathtt{RO}\}$, which will occur in this DR if the job is overdue. Taking the maximum of the number of jobs in the queue and the number of remaining operations in the job is not the most sensible, and this is an area of potential future refinement in the grammar. There is a great difference in performance, in favour of ST3, compared to benchmark ATC and WCOVERT rules

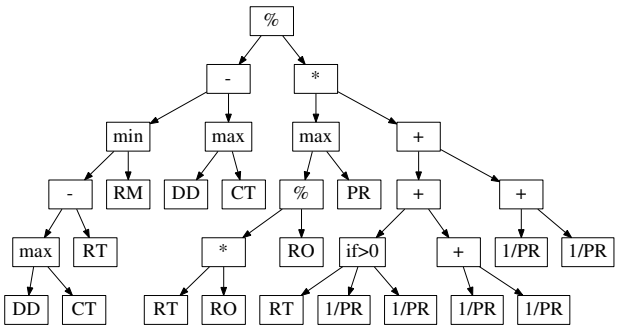

Figure 9: An evolved rule from *ST2A* which outperformed ATC on 39/48 test instances and WCOVERT on 43/48 test instances.

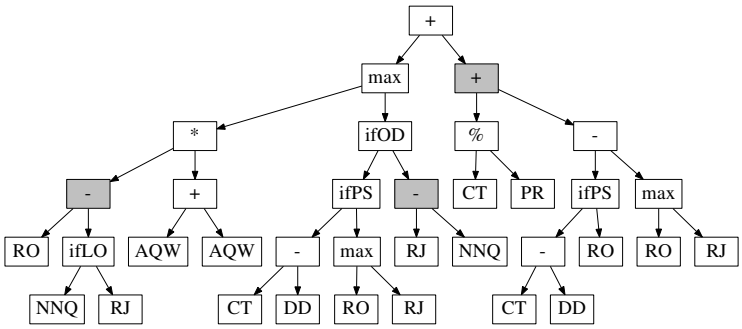

Figure 10: An evolved rule from *GP3* which outperformed ATC on 22/48 test instances and WCOVERT on 34/48 test instances.

of the DRs in Figures 10 and 11, the biggest difference observed between methods with the same function and terminal sets.

6 Further Discussion

Here we present further discussions of our results, examining the impact of tree depth, how performance of the best evolved rules compare on test and extreme test scenarios, and how the evolution and training time compares across all methods.

6.1 DR Size

In general, the larger a DR is, the more difficult it is to interpret. There is a clear trend that the best-of-run rules evolved under a grammar are smaller than their standard GP counterparts, and they also generally have smaller standard deviations. The mean number of nodes in DRs from GP1 is 37.2 compared to 28.8 from ST1, 30.5 from ST1A and 27.4 from ST1B. GP2 has mean number of nodes 44.2 compared to 37.0 from ST2, and GP3 has mean number of nodes 37.2 and ST3 has 35.3. The maximum number of nodes that a DR can evolve with a maximum depth of six is 63 (without `if>0` operator which takes three arguments), so the number of nodes in evolved rules is smaller than

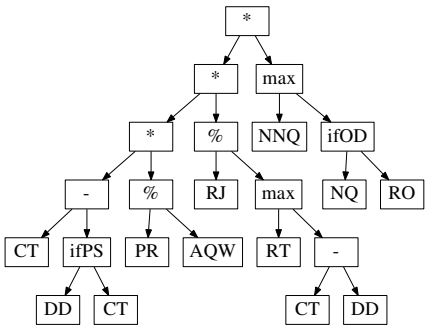

Figure 11: An evolved rule from *ST3*, simplified on the right which outperformed ATC on 36/48 test instances and WCOVERT on 38/48 test instances.

the potential largest size.

We have also performed a second set of 50 GP runs under GP2 and ST2 with a maximum depth of eight. Increasing the maximum depth from six to eight increases the maximum number of nodes to 255, a huge increase in the size of the search space. Although the mean TWT attained on each test and extreme test instance was decreased, it was not a statistically significant difference at the 5% significance level. The average size of the best-of-run DR increased to 71 nodes, twice the size of those evolved with an maximum depth of six. Examining rules evolved with this increase in maximum depth shows the ease of interpretability is definitely decreased. This is not to say that six is the optimal depth for DRs, especially as for the ATC and WCOVERT rules to be expressed in this form greater depth would be needed. However perhaps this shows that including the size of DRs as an objective to be minimised alongside a more traditional measure of shop performance is a sensible direction to pursue in future work. Parsimony pressure (Zhang and Mühlenbein, 1995) is one frequently used method of constraining the size of GP individuals to prevent bloat.

STGP by nature may need a deeper maximum depth to compensate for the restriction imposed by the strong typing, since for the same maximum depth STGP is searching a much smaller search space.

6.2 Comparison of Performance of Testing vs Extreme Testing

We evaluated the best-of-method rules shown in Figures 3 to 11 across 50 additional instances for each of the 24 test and 24 extreme test problem scenarios. Figure 12 shows how the rules performed in comparison to WCOVERT and ATC respectively, by plotting how many times the DR outperformed the benchmark rule out of 1200 ($= 50 \times 24$) possible test instances against how many times the DR outperformed the benchmark rule out of 1200 possible extreme test instances. Better performance is towards the top right corner of the plot. Figure 12 shows that the best DRs from GP2, ST2 and ST2A methods have the best performance, followed by GP1, ST1, ST1A and ST1B, and lastly GP3 and ST3. It is interesting that the performance of the corresponding ST methods is worse than GP1, similar for GP2, and much better for GP3. The best performing rules are the ones evolved by GP2 and ST2 (Figures 7 and 8 respectively).

Further we calculate Z-scores, $\frac{actual-mean}{standard\ deviation}$, to measure the relative performance of each rule across all solutions for the 50 problem instances. We average these

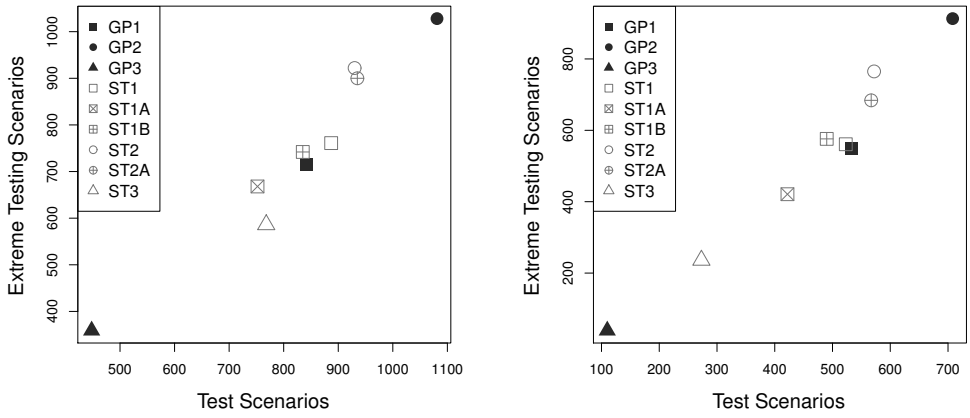

Figure 12: Number of instances DRs in Figures 3 to 11 that outperform WCOVERT (left) and ATC (right) on 50 problem instances from each test and extreme test scenario.

Table 10: Mean and standard deviation of evolution time and test time for GP and STGP.

	GP1 Mean±Stdev	ST1 Mean±Stdev	ST1A Mean±Stdev	ST1B Mean±Stdev
Evolution time (min)	17753.5±45558.3	7169.8±3774.9	5515.5±350.6	7010.1±2884.3
Testing time (ms)	1451.9±1269.3	404.9±263.2	281.7±48.6	394.8±225.5

Z-scores across the 48 problem scenarios, and plot these values for test instances vs extreme test instances in Figure 13. Better performance is now lower Z-score values. Figure 13 is less linear than those in Figure 12, and GP2, ST2 and ST2A are still the clearly better performers. However, the results of GP1, ST1, ST1A and ST1B appear more similar to those of GP3 and ST3 than in the comparisons of these rules with ATC and WCOVERT. The best rules from ST1, ST1B and ST3 have better mean Z-scores on both test and extreme test scenarios than GP1, SGP1A and GP3. The Z-scores have separated out the performance of the best rules more than the comparisons with ATC and WCOVERT. We can see that although GP1 and GP3 attain the same performance on the extreme testing scenarios, GP3 is much worse than GP1 on test scenarios.

6.3 Training and Testing Times

Tables 10 and 11 show the mean±standard deviation of evolution time (in minutes) and testing time (in milliseconds) for each method. Table 10 shows that ST1, ST1A and ST1B methods have approximately three times shorter mean evolution and between three and four times shorter mean testing time than GP1. Table 11 shows that ST2 and ST2A methods have longer mean evolution and shorter mean testing time than GP2,

Table 11: Mean and standard deviation of evolution time and test time for GP and STGP.

	GP2 Mean±Stdev	ST2 Mean±Stdev	ST2A Mean±Stdev	GP3 Mean±Stdev	ST3 Mean±Stdev
Evolution time (min)	5522.3±344.6	8418.2±3864.9	7143.6±3305.8	7206.6±3513.3	5747.5±2175.7
Testing time (ms)	569.9±33.9	477.4±253.2	382.4±198.6	731.3±378.7	324.7±152.8

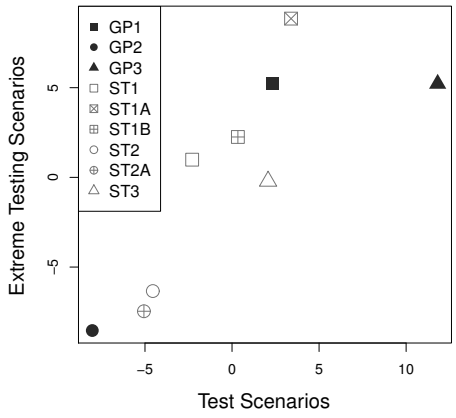

Figure 13: Mean Z-score of DRs in Figures 3 to 11 across 50 problem instances from each test and extreme test scenario.

and ST3 has shorter mean evolution and mean testing times than GP3 as well as smaller standard deviations.

7 Conclusions

The goal of this work was to use a grammar-based GP approach, using strongly typed GP, to semantically constrain the search space of possible heuristic dispatching rules, to evolve DRs with greater interpretability. This is the first work using GP for the automatic discovery of DRs that has explored the interpretability to this level. We investigated using three different function sets, and compared the best-of-run classifiers evolved with and without semantic constraint.

The mean performance of DRs that were evolved under the type constraints of STGP was (as expected) not as good as the mean performance of DRs evolved without semantic constraint. However there were still effective rules evolved under STGP. This highlights the need for multiple GP runs to establish performance of methods.

Our investigations led us to introduce new terminals, `1/PR` and `W/PR`, and new functions `ifOD`, `ifPS` and `ifLO`. The use of conditional operators with predetermined conditions made the rules easier to interpret, due to the condition being tested being known. The conditions evolved, particularly without the use of a grammar, are often very long and are unable to be interpreted easily, if at all. Of these three new conditionals `ifPS` was the most frequently used. However performance of evolved DRs was not as good as that of DRs evolved with the `if>0` function in the terminal set. This is another trade-off between interpretability and TWT performance.

We have found the interpretability of DRs evolved under STGP, and particularly the most effective one from the 50 best-of-run DRs, to be more easily understandable. Several of the rules are able to be understood as they break available jobs into categories based on if the job is overdue or not, and then different priority assignments are made based on this.

Interpretability is a difficult concept to quantify. The best we are able to do is manually inspect rules and see if, as human operators, we are able to explain a rule's effective or ineffective performance. Despite the small deterioration of mean TWT performance when the search space was semantically constrained, the best performing

rules were better able to be understood. We believe that this improvement justifies semantic constraint for the automatic generation of DRs. Further analysis of fragments appearing in the best-of-run rules revealed that the most common fragments are acceptable semantically even without semantic constraint. However most of the evolved DRs contained at least one interaction which was not allowed under the corresponding grammar. The use of a grammar decreases the mean program size, as well as the standard deviation of evolved program size. The disadvantage of the use of a grammar is that there is a larger number of best-of-run DRs which have a poor performance in terms of TWT.

This work has been a preliminary investigation, and the grammars used can be extended and refined, which may improve the effectiveness of evolved rules. In future work we would like to investigate further specific conditional operators and refine the grammars used to constrain STGP. We would also like to use the insight from fragment analysis to add popular fragments as terminals and investigate a multi-objective approach, with objectives such as DR effectiveness, compactness of DR, interpretability, generalisation ability and the number of distinct terminals featuring in the DR.

References

Baker, K. R. (1984). Sequencing rules and due-date assignments in a job shop. *Management Science*, 30(9):1093–1104.

Blazewicz, J., Domschke, W., and Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(1):1–33.

Hart, E., Ross, P., and Corne, D. (2005). Evolutionary scheduling: a review. *Genetic Programming and Evolvable Machines*, 6:191–220.

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

Holthaus, O. and Rajendran, C. (1997). Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105.

Hunt, R., Johnston, M., and Zhang, M. (2014a). Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 927–934.

Hunt, R., Johnston, M., and Zhang, M. (2014b). Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 618–625.

Jakobovic, D. and Budin, L. (2006). Dynamic scheduling with genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming*, pages 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*, pages 321–330.

Jakobović, D. and Marasović, K. (2012). Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing*, 12(9):2781–2789.

Jones, A. and Rabelo, L. C. (1998). Survey of job shop scheduling techniques. Technical report, National Institute of Standards and Technology, Gaithersberg.

Keijzer, M. and Babovic, V. (1999). Dimensionally aware genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1069–1076.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

Luke, S. (2013). *Essentials of Metaheuristics*. Lulu, second edition. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

McKay, R. I., Hoai, N. X., Whigham, P. A., Shan, Y., and O'Neill, M. (2010). Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3–4):365–396.

Miyashita, K. (2000). Job-shop scheduling with GP. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 505–512.

Montana, D. J. (1995). Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2012). A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1–8.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2013a). A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 17(5):621–639.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2013b). Dynamic multi-objective job shop scheduling: a genetic programming approach. In Uyar, A. S., Ozcan, E., and Urquhart, N., editors, *Automated Scheduling and Planning*, volume 505 of *Studies in Computational Intelligence*, pages 251–282. Springer Berlin Heidelberg.

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

Pickardt, C., Branke, J., Hildebrandt, T., Heger, J., and Scholz-Reiter, B. (2010). Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In *Proceedings of the Winter Simulation Conference*, pages 2504–2515.

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

Pinedo, M. and Singer, M. (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46(1):1–17.

Potts, C. and Strusevich, V. (2009). Fifty years of scheduling: a survey of milestones. *Journal of the Operational Research Society*, 60(S1):41–68.

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

Ross, P. (2003). Hyper-heuristics. In Burke, E. and Kendall, G., editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, volume 1, pages 529–556. Springer.

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

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

Whigham, P. (1995). Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41. Morgan Kaufmann.

Zhang, B.-T. and Mühlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38.