

Designing Dispatching Rules to Minimize Total Tardiness

Joc Cing TAY* and Nhu Binh HO

Evolutionary and Complex Systems Programme,
School of Computer Engineering,
Nanyang Technological University, Singapore 639798
*asjctay@ntu.edu.sg

Abstract. We approximate optimal solutions to the Flexible Job-Shop Problem by using dispatching rules discovered through Genetic Programming. While Simple Priority Rules have been widely applied in practice, their efficacy remains poor due to lack of a global view. Composite Dispatching Rules have been shown to be more effective as they are constructed through human experience. In this work, we employ suitable parameter and operator spaces for evolving Composite Dispatching Rules using Genetic Programming, with an aim towards greater scalability and flexibility. Experimental results show that Composite Dispatching Rules generated by our Genetic Programming framework outperforms the Single and Composite Dispatching Rules selected from literature over large validation sets with respect to total tardiness. Further results on sensitivity to changes (in coefficient values and terminals) among the evolved rules indicate that their designs are optimal.

1 Introduction

In today's highly competitive marketplace, a high level of delivery performance has become necessary to satisfy customers. Due to market trends, product orders of low volume, high variety types have been increasing in demand. Hoitomt *et al.* [1] mentions that these products comprise between 50 to 75 % of all manufactured components, thereby making schedule optimization an indispensable step in the overall manufacturing process.

The Job-Shop Scheduling Problem (JSP) is one of the most popular manufacturing optimization models used in practice [2]. It has attracted many researchers due to its wide applicability and inherent difficulty [3-6]. It is also well known that the JSP is NP-hard [7], hence general, deterministic methods of search are inefficient as the problem size grows larger. The $n \times m$ classical JSP involves n jobs and m machines. Each job is to be processed on each machine in a pre-defined sequence, and each machine processes only one job at a time. In practice, the shop-floor setup typically consists of multiple copies of the most critical machines so that bottlenecks due to long operations or busy machines can be reduced. As such, an operation may be proc-

* Corresponding author

essed on more than one machine having the same function. This leads to a more complex problem known as the Flexible Job-Shop Scheduling Problem (FJSP). The extension involves two tasks; assignment of an operation to an appropriate machine and sequencing the operations on each machine. In addition, for complex manufacturing systems, a job can typically visit a machine more than once (known as recirculation). These three features of the FJSP significantly increase the complexity of finding even approximately optimal solutions [8].

The classical JSP and FJSP have been solved by many stochastic local search methods, such as Simulated Annealing [4], Tabu Search [5, 9, 10] and Genetic Algorithms [11-14]. The reported results of applying them show that good approximations of optimality can be found, albeit at the expense of a huge computational cost, particularly when the problem size is large. In practice, dispatching rules have been applied to avoid these costs [15-17]. Although the quality of solutions produced by dispatching rules are no better than the local search methods, they are the more frequently applied technique due to their ease of implementation and their low time complexity. Whenever a machine is available, a priority-based dispatching rule inspects the waiting jobs and selects the job with the highest priority to be processed next. Recently, the introduction of composite dispatching rules (CDR) have been increasingly investigated by the some researchers [18, 19], but typically only for classical JSPs. These rules are the heuristic combination of single dispatching rules that aim to inherit the advantages of the former. The results show that, with careful combination, the composite dispatching rules will perform better than the single ones with regards to the quality of schedules.

In this paper, we investigate the potential use of GP for evolving effective composite dispatching rules for solving the FJSP, with the objective of minimizing total tardiness. The purpose of this research is to find efficient composite dispatching rules that perform better than the dispatching rules presented in literature for solving the same problem. By using a wide training data set, we believe that the evolved CDRs can be applied directed in practice without any modifications. Furthermore, the results of these CDRs could be used as the input to other local search methods in solving FJSP problems, such as Genetic Algorithms [13, 14].

The remainder of this paper is organized as follows. Section 2 gives the formal definition of the FJSP. Section 3 reviews recent related works for solving the JSP and FJSP using dispatching rules and an overview of GP. Section 4 describes our proposed GP framework for evolving CDRs while Section 5 analyzes the performance results of the CDRs obtained with GP. Finally, Section 6 gives some concluding remarks and directions for future work.

2 Problem Definition

Similar to the classical JSP, solving the FJSP requires the optimal assignment of each operation of each job to a machine with known starting and ending times. However, the task is more challenging than the classical one because it requires a proper selection of a machine from a set of machines to process each operation of each job. Fur-

thermore, if a job is allowed to recirculate, this will significantly increase the complexity of the system [20]. The FJSP is formulated as follows:

- Let $J = \{J_i\}_{1 \leq i \leq n}$, indexed i , be a set of n jobs to be scheduled.
- Each job J_i consists of a predetermined sequence of operations. Let $O_{i,j}$ be operation j of J_i .
- Let $M = \{M_k\}_{1 \leq k \leq m}$, indexed k , be a set of m machines.
- Each machine can process only one operation at a time.
- Each operation $O_{i,j}$ can be processed without interruption on one of a set of machines M_k in a given set $\mu_{i,j} \subset M$ with $p_{i,j,k}$ time units.
- Let C_i and d_i be the completion time and due date of job J_i respectively. The tardiness of this job is calculated by the following formula:

$$T_i = \max\{0, C_i - d_i\}$$

- The objective function T of this problem is to find a schedule that minimizes the sum of tardiness of all jobs (total tardiness problem):

$$T = \min \sum_{i=1}^n T_i = \min \sum_{i=1}^n \max\{0, C_i - d_i\}$$

Total tardiness is one of the major objectives in production scheduling. A job that is late may penalize the company's reputation and reduce customer satisfaction. Hence, keeping the due dates of jobs under control is one of the most important tasks faced by companies [19].

The FJSP can also be considered to be a Multi Purpose Machine (*MPM*) job-shop [21]. Using the $\alpha|\beta|\gamma$ notation of [22], the problem we wish to solve can be denoted by

$$J \text{ MPM } | \text{prec } r_j \text{ } d_j | \sum_j T_j$$

where J denotes job-shop problem, *MPM* denotes multi purpose machine, *prec* represents a set of independent *chains* while r_j and d_j represents *release date* and *due date* given to each job respectively; finally, $\sum_j T_j$ represents total tardiness.

In this paper, we shall assume the following:

- All machines are available at time 0.
- Preemption of operations is not allowed.
- Each job has its own release date and due date.
- The order of operations for each job is predefined and cannot modified.

3 Previous Works

Dispatching rules have received much attention from researchers over the past decades [15-17]. In general, whenever a machine is freed, a job with the highest priority in the queue is selected to be processed on a machine or work center. A comprehensive survey on dispatching rules is by Panwalkar and Wafik [15] and Blackstone *et al.* [16]. Depending on the specification of each rule, it can be classified [15] into:

- Simple Priority Rules
- CDRs
- Weighted Priority Indexes
- Heuristic Scheduling Rules

Simple Priority Rules (SPR) are usually based on a single objective function. They usually involve only one model parameter, such as processing time, due date, number of operations or arrival time. The Shortest Processing Time (SPT) rule is an example of a SPR. It orders the jobs on the queue in the order of increasing processing times. When a machine is freed, the next job with the shortest time in the queue will be removed for processing. SPT has been found to be the best rule for minimizing the mean flow time and number of tardy jobs [17]. The Earliest Due Date (EDD) rule is another example of a SPR where the next job to be processed is the one with the earliest due date. Unfortunately, no SPR performs well across every performance measure such as tardiness or flow time [23]. To overcome this limitation, CDRs have been studied to combine good features from such SPRs.

There are two kinds of CDRs presented in literature; the first type involves deploying a select number of SPRs at different machines or work centers. Each machine or work center employs a single rule. When a job enters a specific machine or work center, it is processed by the SPR that is preselected for that machine or work center. For instance, Barman [23] applied three different SPRs to solve the flow shop problem corresponding to three work centers. Experimental results show that it obtains better results than a single SPR that is common to all three machines. However, this approach may not be suitable for a shop floor with large number of machines or work centers; and the best independent distribution of single SPRs is difficult to predetermine. Furthermore, it still has the limitation of a localized view. The second type involves applying the composition of several SPRs (otherwise known as a CDR) to evaluate the priorities of jobs on the queue [17]. The latter type is executed similarly to SPRs; when a machine is free, this CDR evaluates the queue and then selects the job with the highest priority. For example, Oliver and Chandrasekharan [17] present five CDRs for solving the JSP. Their results indicate that CDRs are more effective compared to individual SPRs. CDRs inherit the simplicity of SPRs while achieving some scalability as the number of machines increase. Moreover, if well designed, CDRs can solve realistic problems with multiple objectives [8]. However, the challenge is to find a good combination of SPRs to apply to all machines or work centers.

Weighted priority index rules are the linear combination of SPRs with computed weights [18, 19]. Depending on specific business domains, the importance of a job determines its weight. For instance, consider n jobs with different weights w , each job J_i is assigned weight w_i . The sum of the weighted tardiness as the objective function is given as follows:

$$T = \min \sum_{i=1}^n w_i T_i = \min \sum_{i=1}^n w_i \times \max\{0, C_i - d_i\}$$

In this paper, weighted priority rules are not considered as they are a generalization of our current formulation of total tardiness where we have assumed instead that all jobs have unit weights (or all jobs are equally important) (see Section 2).

Heuristic rules are rules that depend on the configuration of the system. These rules are usually used together with previous rules, such as SPRs, CDRs or weighted priority index rules. For instance, the heuristic rules may use the expertise of human experience, such as inserting an operation of a job into an idle time slot by visual inspection of a schedule [15].

The results from recent researchers [17, 23] show that CDRs outperform individual SPRs in improving the efficiency of the shop floor. In this work, we focus our attention on finding a computational method to build effective CDRs; one that is based on the composition of fundamental measures rather than on the algebraic combination of SPRs. However, this may be difficult to enumerate manually due to the large parameter and operator space, hence we employ a GP framework.

Genetic programming (GP) [24] belongs to a family of evolutionary computation methods. It is based on the Darwinian principle of reproduction and survival of the fittest. Given a set of functions and terminals and an initial population of randomly generated syntax trees (representing programs), the programs are evolved through genetic recombination and natural selection. GP has been applied to many different problems; from classical tasks, such as function fitting or pattern recognition, to non-trivial tasks that are competitive with significant human endeavours such as designing electrical circuits [25] or antennas [26].

The most important feature that makes GP different from the canonical GA is its ability to vary the logical structure and size of evolved computer programs dynamically. It can therefore solve more challenging problems that have eluded the canonical GA due to the latter's requirement of a fixed-length chromosome. However, GP has rarely been applied to manufacturing optimization [27]; this is due to the direct permutation property of scheduling where jobs and/or machines can be simply reordered (in the case of JSP) to improve optimality. For instance, the chromosomes presented in [10-14] have fixed lengths, which can be evolved easily by direct permutation. On the other hand, GP uses a tree-based encoding with dynamic length; making it difficult to encode the JSP (for that matter, a FJSP) into a tree-based chromosome. Unlike previous approaches [17-19, 23] where a predefined set of SPRs were combined in advance by human experience, we apply GP to find superior constructions of CDRs which are composed of fundamental terminals (see Table 1). These discovered rules are then used to solve the FJSP directly; the advantage being that the obtained CDRs can solve the FJSPs in shorter computational time as compared to genetic algorithms [10-14].

4 Design of the GP Framework

In GP, an individual (ie, computer program) is composed of terminals and functions. Therefore, when applying GP to solve a specific problem, they should be well designed to satisfy the requirements of the current problem. After evaluating many parameters related to the FJSP, the terminal set and the function set that are used in our algorithm are described as follows.

4.1 Terminal set

In job-shop scheduling, there are many parameters that can effect the quality of results; potentially, all of them can be considered to comprise a dispatching rule. However, in order to create a short and meaningful dispatching rule, only a small and sufficient number of parameters should be evaluated properly. They also help to reduce the search space and improve the efficiency of the algorithm. Based upon the dispatching rules involving due dates in [15-17] and our experimental works, the terminal set proposed in this study is given in Table 1.

Table 1. Terminal Set

Terminal	Meaning
ReleaseDate	Release date of a job (RD)
DueDate	Due date of a job (DD)
ProcessingTime	Processing time of each operation (PT)
CurrentTime	Current time (CT)
RemainingTime	Remaining processing time of each job (RT)
numOfOperations	Number of operations of each job (nOps)
avgTotalProcTime	Average total processing time of each job (aTPT)

In Table 1, *CurrentTime* represents the time when a particular machine is free and starts to select a job to process on its queue. *RemainingTime* corresponds to the elapsed time for the current job to finish. Some previous dispatching rules use total processing time of each job as one of their parameters. However, in FJSP, as an operation of each job can be processed on a set of machines, we normalize the average processing time of each operation with the following formula:

$$\bar{p}_{i,j} = \frac{\sum_{k \in n(\mu_{i,j})} p_{i,j,k}}{n(\mu_{i,j})}$$

where $p_{i,j,k}$ stands for processing time of operation $O_{i,j}$ on machine M_k and $n(\mu_{i,j})$ represents the number of machines that can process $O_{i,j}$.

4.2 Function set

Similar to other applications of GP [24-26] for solving optimization problems, we use four basic operators: addition, subtraction, multiplication, and division for creating a CDR. Furthermore, we employ a well-known Automatically Defined Function (ADF) (proposed by Koza [28]). The ADF is sub-tree which can be used as a function in the main tree. The size of the ADF is varied in the same manner as the main tree. It enables GP to define useful and reusable subroutines dynamically during its run. The results from [28] indicate that GP using ADF outperforms GP without ADF in solving the same optimization problem. The more parameters are used in ADF, the more changes will be needed for GP to evolve good subroutines. However, it can lead to a higher number of generations. We limit the ADF used in our approach to two param-

ters. The operators used in the ADF are also the four basic operators mentioned above. The operators of the function set in our approach are given in Table 2.

Table 2. Function Set

Function	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
ADF(x_1, x_2)	Automatically Defined Function

4.3 Encoding a CDR using a GP Chromosome

The obtained results from each generation of GP are a set of computer programs represented as trees. As mentioned in Section 2, the objective in our study is to minimize the total tardiness of the FJSPs. Therefore, we propose a method to form a CDR from the tree-based result of GP. This CDR is then combined with the *least waiting time assignment* [13] to evaluate the total tardiness of the FJSPs. These two processes are described in detail as follows.

To find a suitable machine to process an operation $O_{i,j}$, we apply the *least waiting time assignment* on the set of setting machines that can process $O_{i,j}$. This rule is intended to reduce the workloads of the machines by balancing operations to be assigned. It is calculated by *summing* all the subsequent operations in the waiting list *plus* the remaining processing time on each machine and the processing time of $O_{i,j}$. Therefore, it depends on the total time this operation has to wait to be processed in the worst case, not relying only on its own processing time.

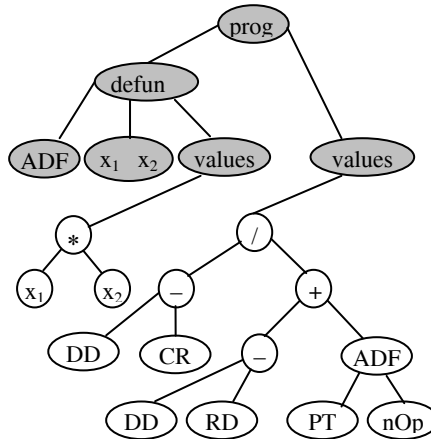


Fig. 1. Example of a GP tree with defined functions and terminals

In determining the proper order of operations on the queue of a particular machine, we use the CDR generated by GP. When a machine is freed, the generated rule is applied directly to the set of operations that are waiting in the queue of the machine. The operation with the highest priority is then selected to be processed on the machine. Figure 1 above gives an example of a dispatching rule tree generated by GP. It shows the overall structure of the generated tree that gives a possible CDR. The left child of *progn* shows the function-defining branch (containing the *defun*). In this case, the ADF function is defined by: $ADF(x_1, x_2) = x_1 * x_2$. The right child gives the result-producing branch. This CDR therefore represents the following formula:

$$\frac{(DD - CR)}{(DD - RD) + ADF(PT, nOps)}$$

Since $ADF(x_1, x_2) = x_1 * x_2$, we obtain:

$$\frac{(DD - CR)}{(DD - RD) + (PT * nOps)}$$

Any tree in the genomic population of GP that contains our defined functions and terminals can be interpreted as a CDR in the same way. The obtained CDR is then applied to solve a FJSP problem to evaluate its total tardiness.

4.4 GP Parameter Setting

Through experimentation, the set of parameters used in our GP framework is listed in Table 3.

Table 3. Choice of parameter values

Parameters	Value
Population Size	100
Number of Generations	200
Creation Type	Ramped half and half
Maximum depth for creation	7
Maximum depth for crossover	17
Crossover Probability	100%
Swap Mutation Probability	3%
Shrink Mutation Probability	3%
Number of best rules copy to new generation	5

We implemented *Ramped half and half* to generate the initial population of GP. This method was proposed by Koza [24] and it has been widely used by previous researchers. It divides the initial population into two parts; half of which contains the random generated trees with maximum depth (in this experiment, this value is 7) and the remaining half contains the random generated trees with depth values ranging from one to the maximum depth. In order to keep the best trees that may be destroyed

by GP's operators, we sort the current population and copy five of them to the next generation.

5 Experimental results

This section reports and analyses the results of our computational experiments. The system was implemented using C++, running on a 2 GHz PC with 512 MB RAM. We will describe how to generate the test cases that are used to evolve CDRs for minimizing total tardiness objective of FJSP problems. The performance results of the evolved dispatching rules will be compared to some commonly used dispatching rules in literature. Finally, the evolved dispatching rules' sensitive parameters will be discussed.

5.1 Test Case Generation

Various experiments were conducted to evaluate the efficiency of our proposed algorithms. We categorized these experiments into three classes: FJSP with 100% flexible (FJSP-100), FJSP with 50% of flexibility (FJSP-50), and FJSP with 20% of flexibility (FJSP-20). The FJSP with $c\%$ of flexibility means that less than or equal $c\%$ of total machines are selected to process an operation. The processing times of each operation was drawn out of $U((\text{number of machines})/2, (\text{number of machines})\times 2)$, where U refers to a uniform distribution. In practice, an operation can be processed on any of a group of machines that constitute a work center. The variance of these processing times is ideally zero or usually small. Therefore, in our test cases, we set the maximum difference between two operations to be 5 unit times. The release date of each job depends on the number of jobs in a particular test case. If the number of jobs is larger than 50, the release date is drawn out of $U[0,40]$, else it is taken from $U[0,20]$. Baker [29] proposed a formula to estimate the due date of a job using the TWK-method:

$$d_i = r_i + c \times \sum_{j=1}^{n_i} p_{ij}$$

where r_i and d_i denote release and due dates of job i respectively. p_{ij} presents the processing time of operation O_{ij} , and c denotes the tightness factor of the due date. The higher the value of c , the looser is the job's due date. We adapt this formula to generate due dates of jobs by replacing the parameter p_{ij} with \bar{p}_{iq} .

Depending on the tightness of the due date, we separate the samples of each class FJSP-100, FJSP-50, or FJSP-20 into tight, moderate, or loose due dates corresponding to values of $c = 1.2, 1.5, \text{ and } 2$. We also generate mixed samples where each sample contains 34% jobs with tight due dates, 33% of jobs with moderate due dates, and the remaining ones with loose due dates. Specifically, the class FJSP-100 holds 9 samples of tight due date, 9 samples of moderate due date, 9 samples of loose due date, and 9 samples of mix due date. Similarly for FJSP-50 and FJSP-20, with 36 samples each. Each training set contains three classes of 108 FJSP problems with different number

of jobs, machines and different tightness of jobs. Another five validation sets (with 108x5 FJSP problems) of similar compositions were generated.

In order to understand how our GP framework can adapt to the different conditions of the shop floors for evolving efficient dispatching rules, we divide the experiments into two test samples:

- Test sample 1: varying both the number of jobs and number of machines. Number of jobs and number of machines range from 10 to 200 and 5 to 15, respectively. This test sample contains 108 training FJSP problems and 108x5 validating FJSP problems.
- Test sample 2: varying the number of jobs and fixing the number of machines. Number of jobs ranges from 20 to 200 and number of machines is fixed at 10. This test sample contains 108 training FJSP problems and 108x5 validating FJSP problems.

The evolved dispatching rules obtained from the test sample 1 are aimed to solve the FJSP problems in the general case of a varying number of jobs and machines while the results of test sample 2 are aimed to solve FJSP problems where the number of machines is unchanged. After training the GP framework with the training set, five best rules were reported. The mean total tardiness of these evolved rules after 500 runs on the validation sets is then reported.

In order to compare the effectiveness of the evolved rules to the human-made rules presented in literature, five frequently used single and composite dispatching rules were selected as benchmarks:

- FIFO (First In First Out): select the job with the earliest coming. This rule is often used in practice since it is simple and easy to implement [16].
- SPT (Shortest Processing Time): select the job with the shortest processing time. This rule is commonly used as a benchmark for minimizing mean flow time and percent of tardy jobs [30].
- EDD (Earliest Due Date): select the job with the earliest due date. This rule is the most popular due date based rule. It is known to be used as a benchmark for reducing maximum tardiness and variance of tardiness [30].
- MDD (Modified Due Date) ($= \max\{CT+PT_i, DD_i\}$): process the jobs in non-decreasing order of MDD. This rule is aimed to minimize total tardiness of jobs [18].
- SL (Slack Time) ($= DD_i - CT - RT_i$): select the job with the minimum slack time. This rule is also used to reduce total tardiness of jobs [17].

Blackstone *et al.* [16] mentions that the study of job shops by analytic techniques, such as queuing theory, becomes extremely complex even for small problems. Therefore, the use of simulation for analyzing dispatching rules is unavoidable. Due to the same difficulty in examining the dispatching rules for solving FJSPs, we also rely on simulation to study the rules' effectiveness. As mentioned earlier in Section 4.3, the *least waiting time assignment* [13] is used to find a suitable machine to process an operation $O_{i,j}$. However, for the first assignment, all the machines are idle; there are no operations on the machine's queue waiting for processing. Therefore, we generate an array of random job orders and assign their first operations for the first assignment.

The remaining ones will be automatically assigned by the *least waiting time assignment rule* and the CDR. Depending on the positions of the first operations in the job order, the end results could be different. Due to the sensitivity of the random job orders that make small changes on the results, all the rules described in this paper are evaluated after 500 runs. For comparative studies of algorithms in constrained problems, we adopt the approach of [31] in using a one way Analysis of Variances (ANOVA) [32]. The function of ANOVA is based on the ratio of variations. It tests the differences between the means of two or more groups. In this paper, it is used to compare the sample mean of a particular objective for an evolved rule with other sample means (for other rules) that overlap with the former's confidence interval (CI). If an overlap exists, this implies some uncertainty concerning the existence of a performance differential. The values of 99% CI for each sample mean are calculated and presented.

5.2 Test sample 1

The best five dispatching rules that were selected from 5 runs times of GP on the training set of test sample 1 are given in Table 4; where possible, they were simplified algebraically.

Table 4. GP generated dispatching rules for test sample 1

Rule	Expression
<i>Rule_1</i>	$aTPT * (CT + RD + PT - 3) + (CT * PT + RD + nOps) - (nOps * PT + 2PT + CT + 1)$
<i>Rule_2</i>	$(PT + CT + RD + 2) * (RT + PT + aTPT)$
<i>Rule_3</i>	$CT * aTPT + 5nOps + 3^{RD}$
<i>Rule_4</i>	$DD * (RD + aTPT + RT + PT)$
<i>Rule_5</i>	$(aTPT + PT) * (CT + RD) + (DD - RD)$

Figure 2 below compares the results of the evolved rules in Table 4 and the five selected dispatching rules for solving different FJSP data. The x-axis represents the dispatching rules while the y-axis represents the average total tardiness of each rule after 500 runs on the five validation test sets. The bars on x-axis from left to right denote FIFO, SL, SPT, MDD, EDD, SL, and *Rule_1 to Rule_5*.

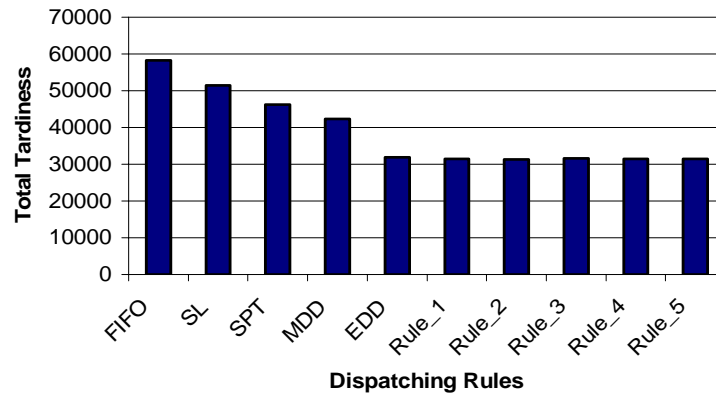


Fig. 2. Performance of dispatching rules on validation test sets in test sample 1

Results from Figure 2 show that the FIFO rule performs poorly in comparison to the others. This is because the due dates of jobs are ignored by FIFO, and therefore the rule does not focus on minimizing total tardiness. The composite dispatching rule SL can obtain slightly better results than FIFO but is still poor in comparison to the remaining rules. Figure 2 indicates that MDD outperforms SL. From the definition of MDD and SL described in Section 5.1, we observe that although these two composite rules contain similar parameters (DD and CT), the gap between the results of the two rules are quite large due to different algebraic combinations of the parameters. This emphasizes that the functions that combine the rules can significantly affect the results. EDD is the best among five rules selected from literature (FIFO, SPT, EDD, MDD, SL) for solving FJSP problems. This could be explained by the presence of the parameter DD in its formula. If the job on the queue is selected by EDD, it has more likely to finish on time since the job with the earliest due date will be selected. Therefore, the total tardiness can be minimized. Although the other rules such as SL or MDD also contain the parameter - due date (DD), EDD obtains almost 50% better results than these rules. This again demonstrates that if an *ineffective* composite dispatching rule is applied to specific problems, it may achieve worse results than the single ones.

We now compare the GP generated rules against the most effective rule (EDD). Figure 3 represents the data distribution of EDD and the five GP evolved dispatching rules after 500 runs. For each rule in Figure 3, the box represents the interquartile range which contains the 50% of values. A line across the box indicates the median. Two lines that extend from the box represent the highest and lowest values while the circles represent the outliers.

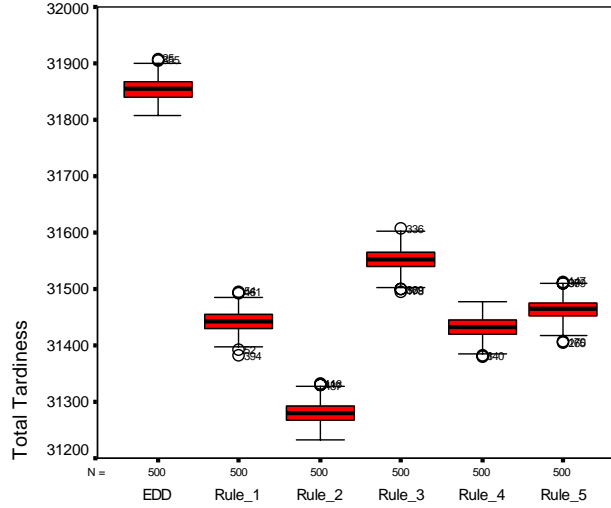


Fig. 3. Data distribution of EDD and *Rule_1* to *Rule_5* after 500 runs

Figure 4 shows in detail the mean total tardiness with 99% CI for each rule. For each rule, the small square in the middle denotes the mean value while two leaves in two sides denote the CI values. It indicates that the results of all evolved rules are much better than by the most effective human-made rule EDD. The best performing rule is the generated rule - *Rule_2* ($((PT+CT+RD+2) * (RT+PT+aTPT))$). This is statistically true since its CI does not overlap with the others. It can be considered as the best rule among them to solve total tardiness objective. Although the mean value of total tardiness of *Rule_4* (31432.42) is smaller than the one of *Rule_1* (31442.20), we cannot conclude that *Rule_4* is more effective than *Rule_1* as their CIs are overlapping. In order to verify if *Rule_4* is really better than *Rule_1* (or not), we apply ANOVA to analyze the data obtained by these rules. Since $F_{ratio} = 75.26$ is greater than $F_{critical} = 3.85$, we reject the null hypothesis that the samples are similar. Therefore, the difference between *Rule_4* and *Rule_1* is statistically significant. Jayamohan and Rajendran [30] mentions that the use of both due date information and processing time can lead to good results in minimizing total tardiness. Our five evolved rules present evidence for this conclusion as their formulation contains these parameters. Furthermore, we find that some parameters, such as the total number of operations (nOps) and total processing time of job (aTPT), are ignored or considered insignificant by previous researchers but according to our results, they *do* contribute to reducing mean tardiness. For example, the formula of *Rule_3* suggests that jobs with fewer number of operations have higher priority. In conclusion, the results from this test sample show that the evolved dispatching rules which are formed by the GP framework are very promising in solving the FJSP in general case.

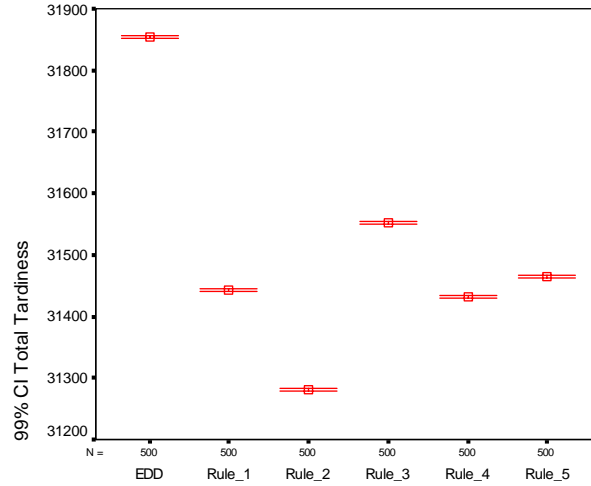


Fig. 4. Mean total tardiness with 99% confidence interval of EDD and *Rule_1* to *Rule_5* after 500 runs

5.3 Test sample 2

Table 5 below represents the best five dispatching rules that were selected from 5 runs times of GP on the training set of test sample 2; where possible, they were simplified algebraically.

Table 5. GP generated dispatching rules for test sample 2

Rule	Expression
<i>Rule_6</i>	$3aTPT + (PT/aTPT + 1) * (RD + RT)$
<i>Rule_7</i>	$6nOps + PT + CT * (PT + aTPT)$
<i>Rule_8</i>	$nOps + 9aTPT + 4PT$
<i>Rule_9</i>	$4aTPT - 2nOps + 3DD + 2PT$
<i>Rule_10</i>	$DD/aTPT + 2aTPT + PT + DD + RD$

Similar to Section 5.2, we compare these evolved rules in Table 5 to five selected rules from literature. The bars on the x-axis from left to right denote FIFO, SL, SPT, MDD, EDD, SL, and *Rule_6* to *Rule_10* while the y-axis represents total mean total tardiness after 500 runs. A visual inspection on Figure 5 again demonstrates that when the number of machines is fixed (to 10), FIFO obtains the worst results while EDD obtains the best. In this special case of FJSP customized for a particular shop floor with 10 machines, the order of the rules' performances selected from literature does not change (similar to the results in Figure 2). We conjecture that even with larger validation sets of the types described in Test sample 1 and Test sample 2, the performances of the rules selected from literature are the same regardless of varying number of machines in FJSP problems.

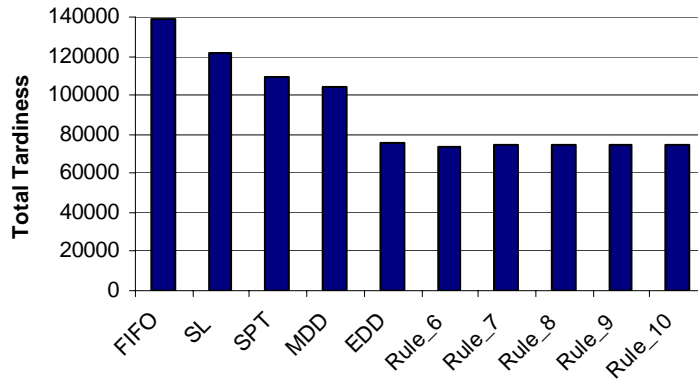


Fig. 5. Performance of dispatching rules on validation test sets in test sample 2

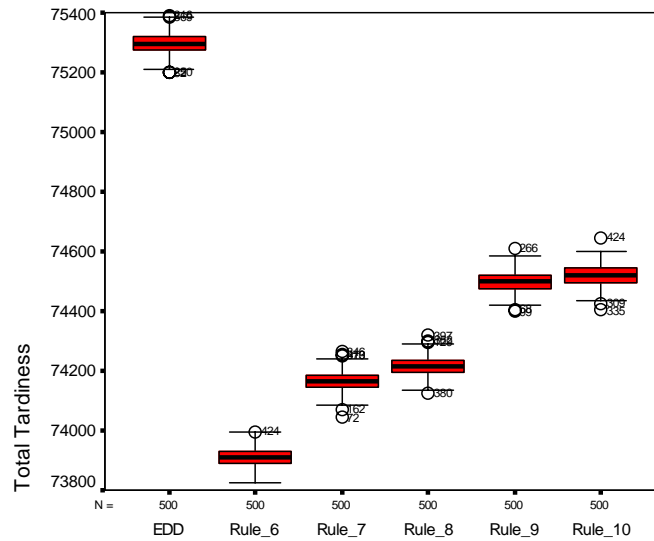


Fig. 6. Data distribution of EDD and *Rule_6* to *Rule_10* after 500 runs

The total tardiness values of the evolved dispatching rules fare better than EDD. Figure 6 and Figure 7 gives the data distribution and the mean total tardiness with 99% CI of EDD and five evolved dispatching rules in Table 5 after 500 runs respectively. The results in these two figures show that the evolved dispatching rules outperform the most effective rule (EDD) among the selected rules from literature. In Figure 7, since the CIs of all the rules do not overlap, we can conclude that *Rule_6* is the most effective rule. The order of the evolved rules' performances decreases from *Rule_6* to

Rule_10. Similar to the CDRs represented in Table 4, the CDRs in Table 5 are also combined with the same parameters (RD, DD, PT, CT, RT, aTPT, and nOps). The use of both due date information and processing time in their formulas could lead to the effectiveness of the rules [30]. Especially, we believe that the parameters nOps and aTPT contribute to the success of the CDRs as well.

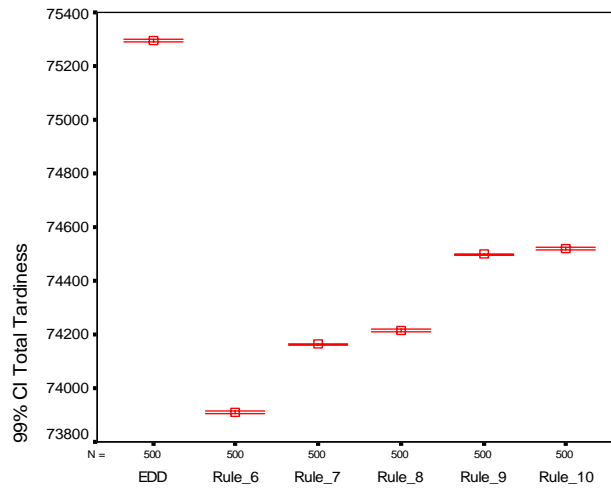


Fig. 7. Mean total tardiness with 99% confidence interval of EDD and *Rule_6* to *Rule_10* after 500 runs

5.4 Sensitivity of parameters

In order to understand why these evolved rules are effective in minimizing total tardiness, we now take a closer look at the combination of their parameters. While single rules consider only one parameter of the shop, the evolved rules employ almost all the important parameters. However, the combination of these parameters plays an essential role to the success of the rule. For instance, the composite rules SL and MDD combine the parameter DD with other parameters CT, PT, and RT but they fail to get better results than the EDD with just one parameter DD (see Figure 2). The parameters aTPT and RD could also be important for solving the problem. They are present in all the rules and contribute mainly to change the priority of one operation to be selected in a queue. For example, *Rule_2* $((PT + CT + RD + 2) * (RT + PT + aTPT))$ in Table 4 was constructed with these two terms. The first term operates in favor of release date RD and processing time PT while the second term runs in favor of average total processing time aTPT and remaining time RT. When the release date of a job is small, this means that the job is released early, the first term produces small results. Similarly, when the processing time of the operation is small, the second term produces a small result. Both parameters help to decrease the value of the ratio and

assign a high priority to the job. Another example, it is well known that the SPT rule is effective in minimizing the number of tardy jobs [17]. Two terms of this rule also contains PT and aTPT that are in favor of the SPT. Therefore, they also contribute to improve the efficacy of the rule.

For evaluating how good dispatching rules are evolved under the GP framework, we modify *Rule_2* by eliminating or changing slightly the coefficients of some parameters. The modifications are listed in Table 5 below.

Table 5. Modified Dispatching Rules from *Rule_2*

Rule	Expression	Modification(s) from <i>Rule_2</i>
<i>Rule_2</i>	$(PT + CT + RD + 2) * (RT + PT + aTPT)$	Original Version
<i>Rule_2_1</i>	$(PT + RD + 2) * (RT + PT + aTPT)$	Removed CT
<i>Rule_2_2</i>	$(PT + CT + RD + 2) * (PT + aTPT)$	Removed RT
<i>Rule_2_3</i>	$(PT + CT + 20 * RD + 2) * (RT + PT + aTPT)$	Changed RD's coefficient from 1 to 20
<i>Rule_2_4</i>	$(PT + CT + RD + 2) * (RT + PT + 20 * aTPT)$	Changed aTPT's coefficient from 1 to 20

In Table 5, *Rule_2_1* and *Rule_2_2* are obtained from *Rule_2* by eliminating CT and RT respectively. By changing the coefficient of RD in *Rule_2* from 1 to 20, we produce *Rule_2_3*. Similarly, *Rule_2_4* is constructed by changing the coefficient of aTPT in *Rule_2* from 1 to 20. They are then applied to solve the FJSP problems in test sample 1. Figure 8 below compares their mean total tardiness with 99% CIs to *Rule_2*'s results after 500 runs.

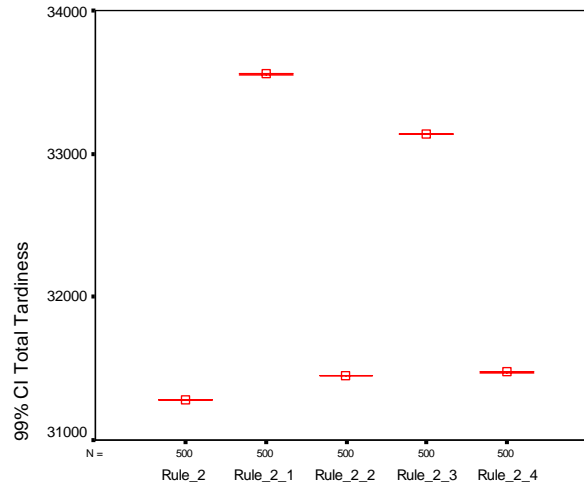


Fig. 8. Mean total tardiness with 99% confidence interval of *Rule_2* and its modified dispatching rules after 500 runs

Figure 8 indicates that although we made small modifications to a small number of parameters of an evolved rule, the results from the obtained rules are much worse than the original one. This implies that the evolved dispatching rules from the GP

framework are well designed. It also validates the importance of selecting proper parameters and of the proper algebraic combination of these parameters to construct efficient CDRs. Any changes on the evolved rules could lead to poorer results.

Generally, the overall experimental results indicate that the evolved rules from our GP framework are more effective than the frequently used dispatching rules in literature. Furthermore, two parameters $aTPT$ and $nOps$ that have received limited study from previous research were found to contribute to the success of evolved CDRs. However, while the importance of selecting proper parameters is one factor to consider when trying to design effective CDRs. We have also proven experimentally that the way to combine these parameters is also crucial. By investigating the potential use of GP for evolving effective CDRs, both parameters and their combination have been explored.

6 Conclusion and Future Works

In this paper, a GP-based approach for designing effective composite dispatching rules that minimizes total tardiness in the Flexible Job-Shop model has been presented and analyzed.

CDRs have been studied widely by previous researchers [15]-[17]. However, all of them were constructed based on the experience of a human scheduler. We employ a GP-framework to generate a CDR based on fundamental terminals that can effectively solve the FJSP (together with a machine assignment rule) by minimizing total tardiness. Two large test samples for training (under our GP framework) and validation were generated. Five evolved rules from each test sample that were most effective were selected to be tested on the validation sets. These rules are based on the combination of parameters such as processing time, release date, due date, current time, number of operations, and average total processing time of each job using basic arithmetical operators for combination. Five other popular rules selected from literature were used as performance benchmarks.

We observed that two composite dispatching rules MDD and SL contain similar parameters (DD and CT), but the performance differential between the results of the two rules were quite large due to use of different algebraic combinations of the parameters. Also, the single dispatching rule EDD contains only one parameter (EDD) but was significantly better than the other rules from literature. This implies that the way to combine the rules can significantly affect the optimality of the schedules; ineffective composite dispatching rules may achieve worse results than the single ones and hence the need for an automated design approach. The experimental results show that our evolved dispatching rules outperform the most effective human-made rule EDD. In particular, two parameters $aTPT$ and $nOps$ that have received limited study from previous research was found to contribute significantly to the effectiveness of evolved CDRs. We have also proven statistically that our evolved CDRs are sufficiently well-designed through the use of ANOVA (which analyzed the sensitivity to changes in the coefficient values and terminal parameters). Finally, by using a large

training data set, we believe that our evolved CDRs can be applied directly in practice without further modifications.

Several possible extensions of this study can be developed. Similar to other applications of GP where the parameters are sensitive, denser terminal sets and more varied ADRs should be investigated to improve the generated rules. The approach of this study can be applied to find the efficient composite dispatching rules for other similar problems, such as a flow shop or the classical job shop. The rules evolved from this GP framework are still quite complex in structure. Therefore, an algebraic simplification tool could be used to make the formula more meaningful. Consideration could even be given to including the number of parameters used as a measure for minimization.

Acknowledgments

This research was funded in part by Nanyang Technological University and CEI Contract Manufacturing Limited Company, Singapore.

References

1. Hoitomt, D.J., Luh, P.B., Pattipati, K.R.: A Practical Approach to Job-Shop Scheduling Problems. *Ieee T Robot Autom* **9** (1993) 1-13
2. Jain, A.S., Meeran, S.: Deterministic job-shop scheduling: Past, present and future. *Eur J Oper Res* **113** (1999) 390-434
3. Carlier, J., Pinson, E.: An Algorithm for Solving the Job-Shop Problem. *Manage Sci* **35** (1989) 164-176
4. Kolonko, M.: Some new results on simulated annealing applied to the job shop scheduling problem. *Eur J Oper Res* **113** (1999) 123-136
5. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. *Manage Sci* **42** (1996) 797-813
6. Yamada, T., Nakano, R.: A fusion of crossover and local search. *Proceedings of The IEEE International Conference on Industrial Technology* (1996) 426-430
7. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flow shop and job-shop scheduling. *Mathematics of Operations Research* **1** (1976) 117-129
8. Pinedo, M., Chao, X.: *Operations scheduling with applications in manufacturing and services*. McGraw-Hill chapter 3 (1999)
9. Brandimarte, P.: Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research (Historical Archive)* **41** (1993) 157-183
10. Mastrolilli, M., Gambardella, L.M.: Effective Neighborhood Functions for the Flexible Job Shop Problem. *J Sched* **3** (2000) 3-20
11. Kacem, I., Hammadi, S., Borne, P.: Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *Ieee T Syst Man Cy C* **32** (2002) 1-13
12. Kacem, I., Hammadi, S., Borne, P.: Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Math Comput Simulat* **60** (2002) 245-276

13. Ho, N.B., Tay, J.C.: GENACE: an efficient cultural algorithm for solving the flexible job-shop problem. *Proceedings of Congress on Evolutionary Computation*, Vol. 2 (2004) 1759-1766
14. Tay, J.C., Wibowo, D.: An effective chromosome representation for evolving flexible job shop schedules. *Proceedings of Genetic and Evolutionary Computation* **3103** (2004) 210-221
15. Panwalkar, S.S., Iskander, W.: A Survey of Scheduling Rules. *Oper Res* **25** (1977) 45-61
16. Blackstone, J.H., Phillips, D.T., Hogg, G.L.: A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *Int J Prod Res* **20** (1982) 27-45
17. Holthaus, O., Rajendran, C.: Efficient dispatching rules for scheduling in a job shop. *Int J Prod Econ* **48** (1997) 87-105
18. John, J.K., Xiaoming, L.: A Weighted Modified Due Date Rule for Sequencing to Minimize Weighted Tardiness. *J Sched* **7** (2004) 261-276
19. Jayamohan, M.S., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. *European Journal of Operational Research* **157** (2004) 307-321
20. Pinedo, M.: *Scheduling theory, algorithms, and systems*. Prentice Hall second edition, chapter 2 (2002)
21. Brucker, P., Jurisch, B., Krämer, A.: Complexity of scheduling problems with multi-purpose machines. *Ann Oper Res* **70** (1997) 57-73
22. Graham, R.L., Lawler, E.L., Lenstra, J.K., Kan, A.H.G.R.: Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics* **5** (1979) 287-236
23. Barman, S.: Simple Priority Rule Combinations: An Approach To Improve Both Flow Time And Tardiness. *Int J Prod Res* **35** (1997) 2857-2870
24. Koza, J.: *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA (1992)
25. Koza, J.R., Bennett, F.H., Andre, D., Keane, M.A., Dunlap, F.: Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming. *IEEE Transactions on Evolutionary Computation* **1** (1997) 109-128
26. Lohn, J.D., Hornby, G.S., Linden, D.S.: An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission. *Proceedings of Genetic Programming Theory Practice* (2004)
27. Dimopoulos, C., Zalzalá, A.M.S.: Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software* **32** (2001) 489-498
28. Koza, J.: *Genetic Programming II, Automatic Discovery of Resuable Programs*, Chapter 4. MIT Press (1994)
29. Baker, K.R.: Sequencing Rules and Due-Date Assignments in a Job Shop. *Manage Sci* **30** (1984) 1093-1104
30. Jayamohan, M.S., Rajendran, C.: New dispatching rules for shop scheduling: a step forward. *Int J Prod Res* **38** (2000) 563-586
31. Quek, H.C., Tay, J.C.: Issues in the Performance Measurement of Constraint Satisfaction Techniques. *Artificial Intelligence in Engineering* **14** (2000) 281-294
32. Johnson, R.A.: *Statistics: Principles and Methods*. John Wiley (2001)