Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning

Cristiane Ferreira, Gonçalo Figueira, Pedro Amorim
INESC TEC, Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, s/n, 4600-001 Porto, Portugal

Abstract

The emergence of Industry 4.0 is making production systems more flexible and also more dynamic. In these settings, schedules often need to be adapted in real-time by dispatching rules. Although substantial progress was made until the ’90s, the performance of these rules is still rather limited. The machine learning literature is developing a variety of methods to improve them, but the resulting rules are difficult to interpret and do not generalise well for a wide range of settings. This paper is the first major attempt at combining machine learning with domain problem reasoning for scheduling. The idea consists of using the insights obtained with the latter to guide the empirical search of the former. Our hypothesis is that this guided empirical learning process should result in dispatching rules that are effective and interpretable and which generalise well to different instance classes. We test our approach in the classical dynamic job shop scheduling problem minimising tardiness, which is one of the most well-studied scheduling problems. Nonetheless, results suggest that our approach was able to find new state-of-the-art rules, which significantly outperform the existing literature in the vast majority of settings, from loose to tight due dates and from low utilisation conditions to congested shops. Overall, the average improvement is 19%. Moreover, the rules are compact, interpretable, and generalise well to extreme, unseen scenarios.

Keywords: Scheduling, Dynamic Job Shop, Dispatching Rules, Genetic Programming

1. Introduction

Industrial environments experience disruptive progress with the advent of Industry 4.0 (i4.0). Several technologies enable real-time interaction between physical and digital machines, such as additive manufacturing, Internet of Things, distributed ledgers, advanced robotics and artificial intelligence [Olsen and Tomlin 2020]. These advancements provide more flexible production systems capable of responding quickly to new demands, breakdowns, and other unforeseen events. However, this adds also more complexity to production scheduling. Processes are naturally more dynamic and subject to disruptions: new job arrivals, lot size changes, cancellations, production failures or machine

*Corresponding author

Email addresses: cristiane.ferreira@fe.up.pt (Cristiane Ferreira), goncalo.figueira@fe.up.pt (Gonçalo Figueira), pamorim@fe.up.pt (Pedro Amorim)

Preprint submitted to Elsevier September 9, 2021
breakdowns. The scheduling of operations must therefore be designed to respond in real-time to possible deviations (Parente et al., 2020). Thus, research is needed to leverage dynamic scheduling for the prompt and effective re-optimisation of operations’ sequence under unexpected events.

Different strategies are applied to manage uncertainty in dynamic scheduling. There are proactive approaches that generate robust schedules based on some prediction of the system behaviour. These methods have received some attention, but their performance highly depends on the predictive information obtained. In a hybrid predictive/reactive alternative, a baseline schedule is continuously adapted during execution. Finally, a completely reactive approach takes no decision in advance. Instead, immediate and local decisions are made in the presence of real-time events. This strategy usually adopts simple heuristics such as Dispatching Rules (DRs) to prioritise jobs waiting to be processed in a machine’s queue (Ouelhadj and Petrovic, 2008).

As simple sequencing procedures, DRs are flexible, easy to implement, and extremely fast to execute. However, their performance depends heavily on the context. Lawrence and Sewell (1997) compare the performance of DRs and exact methods in a dynamic scheduling context. They show that DRs’ performance converges to that of a dynamic and sophisticated heuristic for high uncertainty levels. Similarly, Jain and Foley (2016) conclude that under high load levels and long interruptions, the rules work better than adaptting a complete schedule. Effective DRs have been developed for different problems, such as the classical job shops (Chen and Matis, 2013), shops with batch release (Xiong et al., 2017) and parallel machines (Su et al., 2017). The applications are not limited to manufacturing. For instance, Xu et al. (2016) approach a problem faced by subcontractors in the construction industry. Jung et al. (2019) use scheduling rules for operating rooms in such a way that emergency surgeries are promptly attempted. In a real case study, Sweeney et al. (2019) investigate the development of rules for sequencing vessels in the Upper Mississippi River.

In this work, we aim at designing rules for the Job Shop Scheduling Problem (JSSP), one of the most studied problems in dynamic scheduling. The JSSP is defined on a set of jobs $J$ that require operations to be processed in a set of machines $M$. Each job $j \in J$ has a set operations $O_j$ that must be executed in a predefined sequence. An operation $i \in O_j$ has a specific processing time $p_{ij}$ and must be executed by a specific machine. Each machine can only process one operation at a time (Pinedo, 2008). In the dynamic version of the problem, the shop serves a continuous stream of jobs whose arrivals are usually unknown in advance. Thus, the operations’ sequence must be continuously re-optimised to adapt to unexpected deviations (Holthaus and Rajendran, 1997). The metric considered is the mean tardiness, which is challenging because the performance of rules highly depends on the shop load conditions, i.e., there is no rule with superior performance in both tight and light load conditions (Holthaus and Rajendran, 2000).

Dispatching rules have been traditionally derived by empirical or analytical studies, sustained by scheduling theory. To boost their effectiveness, machine learning algorithms appear as a promising alternative. For example, Shahzad and Mebarki (2012) use data-mining to analyse the behaviour of dispatching rules and generate decision trees to select them. Jun et al. (2019) use Random Forests with the same purpose. There are also Reinforcement Learning approaches, using Neural Networks either to select dispatching rules or to select directly the next operation (Wasch-
All these approaches make use of complex, black-box models, such as Neural Networks and tree ensembles, which cannot be easily interpreted. These models are designed and validated with little domain expertise. As a result, despite their good behaviour in some tested cases, they frequently do not generalise well to unseen scenarios.

Our approach aims at designing interpretable DRs and incorporating domain-specific knowledge. This is made possible by the use of Genetic Programming (GP), an evolutionary hyper-heuristic (Burke et al., 2009). GP evolves computer programs or mathematical expressions, which are executed to solve a problem. The expressions might be used in different ways, such as selecting heuristics (Drake et al., 2020) or computing directly the decisions, like a dispatching rule. One of its advantages compared to other algorithms is the possibility of generating reasonably sized, interpretable-by-design models. By evolving models with GP, researchers might be able to understand their behaviour and even increase knowledge on the problem (Kronberger, 2010).

In the last decade, there has been increasing attention to the use of GP for production scheduling with promising results (Nguyen et al., 2017a). However, most papers focus on algorithmic efficiency and produce extremely long rules. Moreover, these rules work well in particular instances, but none of them performs well in a wide range of settings. We propose a guided empirical learning process, whereby insights derived from problem reasoning and existing rules are used to adjust the algorithm search space, and thus find better rules. We aim at deriving effective, compact and meaningful rules with good performance in general. To assess the generalisability of the evolved rules, we perform comprehensive computational experiments, varying due date tightness, shop utilisation and number of machines, in a wide spectrum of values. We also test the rules in settings with stochastic processing times. Our hypothesis is that by incorporating problem expertise in the algorithm configuration, it should find well-performing and more general rules, which hopefully will also be interpretable.

The remainder of this paper is organised as follows. A literature review is provided in Section 2, which first describes the studies on dispatching rules and then reports the main works on GP applied to JSSP. In Section 3, we describe our guided empirical learning approach and the algorithm implemented. We detail the learning process and the four GP variants in Section 4. In Section 5, the evolved rules and their performance comparison are found. We conclude the paper in Section 6 with the overall achievements and some final considerations.

2. Literature Review

This section describes the related work on dynamic job shop scheduling. At first, we describe the best dispatching rules manually designed with focus on the mean tardiness metric. Then we review the main achievements on the automated generation of rules using GP.

2.1. Dispatching rules for dynamic job shop scheduling

Dispatching rules are used to prioritise jobs awaiting to be processed in a queue. They assign scores to each job based on its properties or the system status. Research on the performance of these rules has a long tradition,
particularly in dynamic job shops. Haupt (1989) and Ramasesh (1990) are relevant works on the topic, providing thorough explanations on the behaviour and classifications of dispatching rules. The most common performance metrics may be grouped into flowtime-related and due-date-related measures. Next, we discuss the most relevant findings on the use of dispatching rules for job shops.

The flowtime is the time the job spends in the shop from release to its completion time. In general, the SPT rule, which selects the job with the shortest operation processing time, effectively minimises mean flowtime. Although the SPT is myopic as it may select jobs that could find long queues when visiting the subsequent machines. For this reason, in addition to the current operation processing time, some rules also include look-ahead information. Holthaus and Rajendran (1997) were the first to add the workload in the next machine queue (WINQ) to the SPT, devising the rule PT+WINQ. Later on, Rajendran and Holthaus (1999) included the processing time of the job’s subsequent operation (NPT) and proposed the rule 2PT+WINQ+NPT, which remains the best for minimising the mean flowtime of jobs.

Tardiness and percentage of tardy jobs are the most common due date related measures. Past research has shown that if due dates are loose or the shop utilisation is low, the dispatching rule should consider some due date information of jobs. Otherwise, in highly loaded shops, processing-time bases rules work better (Blackstone et al., 1982; Haupt, 1989; Ramasesh, 1990). Optimising due date measures is challenging as there is no simple rule for all load conditions. However, much work has been developed in that direction. For instance, simulation studies show that prioritising operations due date instead of jobs due date is more effective. Operations due date are usually calculated by distributing the allowance (remaining time until due date) of jobs per remaining work (the sum of all subsequent operations processing time).

Anderson and Nyirenda (1990) proposed the rules CR+SPT and S/RPT+SPT, that prioritise operations according to their due dates at dispatching time. To calculate due dates, while the former considers the Critical Ratio (CR), the latter uses the job slack divided by the remaining work. The CR of a job is the ratio between its allowance and its remaining work. Slack is calculated by subtracting the remaining work from the job allowance. The idea behind those rules is that the operation due date should be as short as the proportional job allowance for that operation. In practice, they apply the SPT rule if the due date is close or has passed and operations due date in the opposite case. In Raghu and Rajendran (1993), the authors use shop utilisation as a parameter to balance between operations processing time (PT) and due date (PT/S/RPT). They also improve the calculation of the WINQ parameter by proposing a better estimation of the next machine’s queue. The resulting rule (RR) is $PT \cdot \exp(u) + PT \cdot S/RPT \cdot \exp(-u) + WINQ$, where $u$ is the mean shop utilisation. Although this rule presents good results in general, under very tight load conditions, $2PT+WINQ+NPT$ performs better (Holthaus and Rajendran, 2000).

Instead of pursuing a unique dispatching rule, recent works focus on selecting rules according to the conditions of the production system. Very different approaches have been proposed in that direction. Romero-Silva et al. (2018) study a flowshop with multi-class jobs. The objective is to analyse if using different dispatching rules in each queue leads to improved mean flowtime and flowtime variance. They concluded that a better performance is obtained when applying distinct rules for each machine. In Amin and El-Bouri (2018), a preference voting model is employed
to select rules based on multiple criteria. The authors designed a Linear Programming Model for classifying rules according to their relative performance in each criterion. Pergher et al. (2020) develop a framework for due date assignment, order release and dispatching rules according to the decision-maker preferences and multiple performance criteria. A decision support system performs an interactive process where the user inputs trade-offs judgements.

The recent literature on dispatching rules for the JSSP has focused on specific issues, such as arrivals in batch (Xiong et al., 2017) and precedence between jobs (Fan et al., 2021). For the classical JSSP minimising the mean tardiness, the rules generated in the ’90s remain the state-of-the-art. In the last ten years, the automated design of rules has received much attention, mainly with Genetic Programming. In the following, we describe the main achievements obtained with that method.

2.2. GP for the automated design of dispatching rules

Genetic programming has been applied in several studies on dispatching rules, with promising results in several scheduling problems. We briefly describe the main achievements for dynamic job shops in the following. For comprehensive surveys on the use of GP for production scheduling, see Branke et al. (2016), and Nguyen et al. (2017a).

Hildebrandt et al. (2010) and Branke et al. (2015) develop GP algorithms to minimise mean flowtime and show that their results outperform classical dispatching rules. In Nguyen et al. (2015) and Nguyen et al. (2019), the authors find effective rules for several performance metrics - including mean tardiness - but they train specific rules for each load condition. To our knowledge, Karunakaran et al. (2017) and Ferreira et al. (2020) are the only authors focusing on shops with stochastic processing times. However, the former do not benchmark the evolved rules with existing methods, and the latter consider deterministic job arrivals.

Some studies show that GP finds effective ensembles of heuristics for job shops. In Hart and Sim (2016), the authors use GP on the design of a hyper-heuristic for the static version of the JSSP. Their heuristic applies several rules in turn and outperforms single dispatching rules. Park et al. (2018) compares different combination schemes for ensembles. They show that using a linear combination of the rules’ scores works better than applying single dispatching rules or ensembles with majority voting. GP may also be combined with other heuristic approaches such as Iterated Local Search (Nguyen et al., 2015), or more standard evolutionary algorithms (Pickardt et al., 2012).

Different authors studied methods for improving the algorithm efficiency. Surrogate models, for instance, provide faster fitness calculation. They are used to avoid employing high computational effort in the evaluation of non-promising rules. Running these models is usually much efficient than performing a complete simulation for accessing the fitness of individuals (Nguyen et al., 2014; Hildebrandt and Branke, 2015; Nguyen et al., 2017b). Early studies have shown that the tree-based representation is the best approach to evolve powerful scheduling heuristics (Nguyen et al., 2013). However, as they tend to grow during the evolutionary process, alternative representation structures have also been tried (Nguyen et al., 2013; Branke et al., 2015; Nguyen et al., 2019). Besides a good representation, an important decision consists of the terminal set, i.e., the set of the jobs and system attributes that compose the rules. Branke et al. (2016) presents a detailed survey on that, and Mei et al. (2016) proposes a mechanism for selecting
relevant terminals.

The research on GP for the JSSP extends to variants, as the Flexible JSSP. Tay and Ho (2008) were the first to evolve rules for the problem. Their work was followed by Nie et al. (2013) with a Gene Expression Programming (GEP) approach. Differently from GP, GEP-based algorithms make use of a fixed-length representation structure. We also highlight the work from Zhou et al. (2019), which develops a co-evolutionary GP algorithm for due date assignment and dispatching rules. A multi-objective JSSP was tackled by Nguyen et al. (2013) and Nguyen et al. (2014). The authors evolve rules that are non-dominated for the percentage of tardy jobs, mean and maximum tardiness, mean and maximum flowtime. Recently, Fan et al. (2021) evolve dispatching rules for the JSSP with extended precedence constraints.

Despite the extensive literature on the automated development of rules for job shops, most works do not benchmark their results with state-of-the-art rules. Others train distinct rules for load condition (Nguyen et al., 2013; Nguyen et al., 2015; Nguyen et al., 2019). None of the previous works presents a single rule for minimising the mean tardiness that outperforms existing ones. Moreover, there is limited discussion on the size of the rules. A simplification technique is proposed in Nguyen et al. (2017b), but their results are still too complex and difficult to interpret. This work aims at overcoming these gaps and find compact rules that outperform the existing ones in distinct job shops settings and load conditions.

3. Methodology

Despite the good performance of machine learning algorithms, such as Neural Networks and ensembles, most of them generate complex models. They are usually composed of constants and weights that cannot be easily interpreted. On the other hand, the optimal solutions of some optimisation problems consist of simple symbolic expressions (e.g. the Economic Order Quantity, which can be easily found by GP – Lopes et al. (2020)). In scheduling, Xu et al. (2016) show that under varying and stochastic processing costs, jobs are optimally prioritised by the Minimum Slack and Longest Remaining Work policy. Our hypothesis is that on more complex problems, if appropriately guided, GP would still find reasonably sized and effective (if not optimal) symbolic expressions.

In order to guide GP, we use problem knowledge in the learning process. Some initiatives with similar purpose are found in previous works, particularly in the study of hyper-heuristics. Burke et al. (2009) describes a methodology for the use of GP with contribution from domain experts. There are six steps, starting from the analysis of existing heuristics for the problem. However, human participation is limited to the initial decisions when the algorithm and function set are defined. There is no feedback from GP solutions to improve the algorithmic decisions. A more iterative process is followed by Lopes et al. (2020). The authors apply a cooperative co-evolutionary GP algorithm that evolves simultaneously two expressions. Each expression is assigned a specific function set, with some symbol engineering, defined based on the analysis of existing heuristics. The proposed GP finds the EOQ rule for deterministic demand scenarios and meaningful and effective expressions in the stochastic case.
In this paper, we go beyond those studies and formalise the iterative process, as illustrated in Figure 1. Analytical reasoning on the problem (left-hand side) provides the necessary insights to specify data, parameters and constraints to boost the computational experiments (on the right). The left-hand side describes the process often found in traditional studies of dispatching rules (Section 2.1) while the right-hand side consists of the typical methodology from the GP literature (Section 2.2). The aim here is to cross-fertilise these two approaches into a more holistic methodology. This has resulted in the six steps of Figure 1. Precedence relations exist between some, but not all, steps - many of them may be conducted in parallel.

Figure 1: Proposed methodology, an iterative approach where reasoning on the problem and results conduct to inputs for the computational experiments. The algorithm search is combined with human intelligence.

In step 1, the problem is studied. Theoretical foundation and existing rules are analysed. This study provides conclusions on the behaviour of rules and specific terminals (step 3). There are many possible outcomes from step 3 regarding the best learning configuration, terminals to be considered, GP parameters, or constraints that must be incorporated. They may be used in step 4 to define new terminals and operators (or remove existing ones) for GP, refine the training set, or even tune the algorithm parameters. Depending on the results of step 3, one may fix a specific structure for the evolved rules and guide the learning algorithm through a restricted search space. In step 6, GP runs, generates new rules to be analysed in step 1, and a new iteration begins. Step 2 refers to the developments of the GP and the simulator, which is performed only once.

Next, in Section 3.1 we describe the proposed GP algorithm and explain the fitness calculation, which is the average mean tardiness produced by each rule relative to a benchmark. The same metric used for training (in Section 4) is applied for the validation of the results, in Section 5. The simulation model is detailed in Section 3.2 and the design of the experiments are explained in Section 3.3. We conclude with the parameters tuning in Section 3.4.
3.1. Learning algorithm

Genetic Programming (GP) evolves a population of mathematical expressions interpreted or executed to solve a problem [Koza 1994]. The aim is to find the highest fit individual in a search space, where each individual’s fitness corresponds to how well the respective expression solves the problem. This approach uses a flexible representation scheme that, combined with a powerful search mechanism, can explore both the structure and parameters of individuals. A generic GP algorithm consists of creating an initial population that is evolved over many generations. At each generation, all individuals from the population are evaluated by the fitness function. The best ones are reproduced and combined for the next generations. The process continues until the algorithm reaches a stop criteria.

Symbolic expression trees are the most common representation of individuals in GP. They are composed of a function set: terminals and operators appropriate to the specific problem. The former are the leaf nodes and include parameters of the problem operated by the latter. Figure 2 illustrates an individual and the corresponding dispatching rule. In this example, the terminals are 2, \( PT \) (processing time of current operation), \( W \) (work in the next queue) and \( NPT \) (processing time of the next operation). The mathematical operators + and ∗ compose the operators set. Details on the specific implementation of GP in this work are given next.

![Figure 2: A GP individual tree example and its corresponding dispatching rule](image)

The pseudo-code of the proposed GP is provided in Algorithm 1. Each individual \( A \) represents a dispatching rule whose terminals are parameters from the jobs and the shopfloor. The proposed GP runs a generational evolution where a whole population \( P_g \) of individuals is evaluated and updated at each generation \( g \). The generation of the initial population in line 3 uses the algorithm Ramped Half-and-Half [Koza 1994], which creates random trees by combining two methods, "full" and "grow", with equal probability. The minimum and maximum depth for trees must be predefined. In trees created by the "full" method, the path length from the root to every terminal is equal to the specified maximum depth. The "grow" method instead generates trees with variable shapes and sizes.

The fitness computation requires running a job shop simulator, which is the most expensive procedure in the whole algorithm. Simulation studies of dynamic job shops usually apply several replicas of the same instance - each with thousands of job completions. In lines 6-10, the fitness of individual \( a \) is calculated. The simulator returns
Algorithm 1: Genetic Programming algorithm

**Input:** Training set \( I \), benchmark values \( \overline{T}_{Bi} \)

**Result:** Dispatching rule \( a^* \)

1. \( a^* \leftarrow 0; \)
2. \( g \leftarrow 1; \)
3. Initialise population \( P_0; \)
4. for \( g = 1 \) to \( \text{max}_g \) do
   5. for \( e = 1 \) to \( 2 \) do
      6. foreach individual \( a \in P_g \) do
         7. foreach instance \( i \in I \) do
            8. \( \overline{T}_{ai} \leftarrow \frac{1}{n_1} \sum_{k=1}^{n_1} \text{simulate}(a, i); \)
            9. \( \tau_a = \prod_{i \in I} \left( \frac{\overline{T}_{ai}}{\overline{T}_{Bi}} \right)^{1/|I|}; \)
           10. \( f\text{itness}(a, g) = \tau_a + |a| \cdot g \cdot \rho; \)
      11. \( P_g \leftarrow \text{Best 50\% individuals of } P_g; \)
      12. \( a^* \leftarrow \arg \min_{a \in P_g \cup a^*} f\text{itness}(a, g); \)
      13. \( P_{g+1} \leftarrow \text{Crossover and mutation in } P_g; \)

An individual evaluation combines the tardiness performance and the tree size. The tardiness of a job is the difference between its completion time \( c_j \) and due date \( d_j \) if the job is delayed. Given a set of jobs \( J \), we calculate the mean tardiness \( \overline{T} \) as follows.

\[
\overline{T} = \frac{\sum_{j \in J} \max(0, c_j - d_j)}{|J|}
\]

The fitness function computes the mean tardiness performance of each individual \( a \) on a training set of instances. The simulation procedure applies rule \( a \) in \( n_e \) replications of instance \( i \), and returns the average mean tardiness \( \overline{T}_{ai} \) (line 8 in Algorithm 1). We define set \( R = \{ \text{RR, 2PTWINQNPT, CR+SPT, S/RPT+SPT, ...} \} \), composed of benchmark rules mentioned in seminar works (Anderson and Nyirenda 1990; Raghu and Rajendran 1993; Rajendran and Holthaus 1999; Holthaus and Rajendran 1997 2000). \( \overline{T}_{ri} \) is the mean tardiness found when running the simulator with rule \( r \in R \) for a given instance \( i \). The overall performance of \( a \) is estimated by \( \tau_a \), the mean relative performance in all
training instances - the smaller, the better. The relative performance of \( a \) for \( i \) is calculated as follows.

\[
\tau_{ai} = \frac{\max(T_{ai}, \epsilon)}{\max(\min_{r \in R} T_{ri}, \epsilon)}
\]

Thus, the relative performance of \( a \) is the ratio between the mean tardiness found when running this rule and the best mean tardiness found using existing rules. To avoid divisions by zero, we use a a small positive number (\( \epsilon = 10^{-4} \)) when the value of \( \min_{r \in R} T_{ri} \) is null – the same applies to \( T_{ai} \) for a fair comparison on their values. We calculate \( \tau_{a} \) by the geometric mean (see line 9 in Algorithm 1), as it handles ratios more consistently than the arithmetic mean, i.e., given two rules \( a \) and \( b \), the geometric mean of \( T_{ai}/T_{ri} \) and \( T_{ri}/T_{ai} \) is 1, while the arithmetic mean depends on the values of \( T_{ai} \) and \( T_{ri} \). Moreover, as the training set represents very different settings, the ratio \( T_{ai}/T_{ri} \) may scale to distinct magnitude orders. Thus, the geometric mean smooths large deviations from 1.

In addition to relative performance, the fitness function also considers a strategy to avoid large trees. GP trees tend to grow during the search and result in complex rules - an effect known as bloat. This is a drawback of the symbolic tree representation, as complex trees are computationally inefficient and cannot be easily interpreted. Bloat control strategies are usually applied to avoid this behaviour and pursue compact trees. Our approach is to penalise the tree size in the fitness function using a bloat control factor \( \rho \). The penalty depends on the number of nodes \( |a| \), the number of generations completed \( g \) and \( \rho \). Considering the twofold objective of getting effective and small rules, the fitness of individual \( a \) in generation \( g \) \( f(a, g) \) is calculated as follows (line 10 of Algorithm 1):

\[
f(a, g) = \tau_{a} + |a| \cdot g \cdot \rho
\]

In line 18, the population of the next generation is created from the current population \( P_{g} \). There are two breeding pipelines, crossover and mutation, and each one is applied according to a probability. The crossover pipeline selects two individuals as parents to produce two offspring. This operation selects a node at random in each tree and swaps the two sub-trees rooted by those nodes. Mutation replaces a sub-tree with a randomly generated one, generating a new individual. Both pipelines use tournament for parent selection, returning the best of \( t \) individuals selected at random (\( t \) is said tournament size). The breeding process picks certain kinds of nodes with different probabilities. For example, one can state to pick non-terminals 90% of the time and terminals 10% of the time. The algorithm runs until it reaches a maximum number of generations \( \text{max}_g \) and returns the best individual found \( a^* \).

3.2. Simulation model

The simulation model follows the literature except for the number of machines \( |M| \). Most researchers consider a fixed number of ten machines (Holthaus and Rajendran [1997]), but we simulate instances with distinct values. The number of operations of each job is sampled from a discrete uniform distribution whose limits depend on \( |M| \). Machines are also randomly assigned to operations via a uniform distribution. Re-circulation is allowed but not in subsequent operations. We assume that operations are not preemutable, and setup times are included in operations processing time.
The simulation starts with an empty setting, and a warm-up time is considered for the system to reach a steady state. Following the typical procedure, jobs are numbered in increase sequence upon arrival. Metrics are calculated only for jobs 501 to 2500 (a total of 2000 completed jobs). The arrival of jobs follows a Poisson distribution which rate $\lambda$ is based on the utilisation level of the shop ($u$). Given the mean number of operations per job $\bar{n}$ and the mean processing time of operations $\bar{p}$, the arrival rate is calculated as follows:

$$\lambda = u \cdot |M| \cdot \bar{n} \cdot \bar{p}$$

Jobs are released to the shop when they arrive and are pushed into the first operation machines’ queue. Once an operation finishes processing, the job moves to the next machine. Every time a machine is available, the dispatching rule runs and assigns a score to each job in the queue. The job with the least score is selected for execution. Given the release time $r_j$ of job $j$, its due date $d_j$ depend on the allowance factor $a$ and is calculated via Total Work Content (Baker, 1984):

$$d_j = r_j + a \cdot \sum_{i \in O_j} p_{ij}$$

Lawrence and Sewell (1997) studied the performance of dispatching rules in job shops with uncertain processing times. Based on their work, we introduced stochastic operation times in our simulation model. Given the processing time of an operation $p_{ij}$ and a coefficient of variation ($cv$), the actual processing times are found by a Gamma distribution with mean $E[p_{ij}]$ and standard deviation $\sigma_{ij} = E[p_{ij}] \times cv$. The expected processing times $E[p_{ij}]$ are the same as in the deterministic version. A pseudo-code of the simulator algorithm is found in Section A from Appendix. The next section describes the instance sets.

### 3.3. Instance sets

An instance is defined by a tuple $<|M|, \bar{p}, a, u, cv>$ whose parameters are described in Table 1. As mentioned in Section 2, the rules’ performance depends on the load conditions, which are defined by the due date allowance ($a$) and machines utilisation ($u$). In order to consider both tight and light conditions, job shop simulation studies typically use two or three values for $a$ and $u$, in most cases varying from 3 to 8 and 85% to 95%, respectively (Holthaus and Rajendran 1997, Rajendran and Holthaus 1999, Dominic et al. 2004, Nguyen et al. 2019). The coefficient of variation $cv$ determines the uncertainty level in operations processing times (usually deterministic, $cv = 0$). We consider three different instance sets: training, validation and test. The training set is used during GP runs, while the validation set is used to analyse the outputs of GP and compare different versions. We use the test set to evaluate the final selected rules.

The training set is sufficiently diversified to assess rules in distinct load conditions but does not include extreme cases or stochastic times. This allows us to assess whether the rules can generalise, and even extrapolate, to scenarios never seen before. We simulate ten machines, and frequently used values for $\bar{p}$, $a$, and $u$. Therefore, the size of the
Table 1: Training and validation instance sets

| Parameter | Description               | Training | Validation | Test          |
|-----------|---------------------------|----------|------------|---------------|
| $|M|$ | Number of machines        | 10       | 10         | 10, 20, 50, 100 |
| $\bar{p}$ | Mean proc. time           | 25, 50   | 25, 50, 100| 25, 50, 100, 250, 500 |
| $a$      | Due dates allowance factor| 3, 4, 6  | 2, 3, 4, 6, 8 | 1.3, 2, 3, 4, 6, 8, 10 |
| $u$      | Shop utilisation level    | .85, .90, .97 | .80, .85, .90, .95, .97 | .70, .80, .85, .90, .95, .97, .99 |
| $cv$     | Uncertainty in proc. times| 0        | 0          | 0.0, 0.2, 0.4, 0.6, 0.8, 1.0 |
| # Instances |                | 18       | 75         | 897           |

The training instance set is $1 \cdot 2 \cdot 3 \cdot 3 \cdot 1 = 18$. As described in Algorithm 1, there are two evaluation phases. The simulator runs two replications for each training instance in the first evaluation phase, and ten in the second ($n_1 = 2, n_2 = 10$).

To represent a larger range of scenarios, the validation set is composed of five values for $a$ and $u$, resulting in 25 settings in total. As in the training set, a full-factorial combination of all values is used to compose the instance set. These values enable simulating loose due dates, where the best rule (RR, from Raghu and Rajendran (1993)) produces no tardiness and high congested shops. The shop utilisation also includes the extreme values found in previous works. For validating the rules, we ran 100 simulation replications for each instance. The number of replications is adequate to ensure that the validation of the results is statistically significant for a confidence level of 95%. The same number of simulation replications is applied to the test set, which is explained next.

The test instance set includes, in addition to all the values of the validation set, extreme values of allowance factor (1.3 and 10) and shop utilisation (0.70 and 0.99). We aim to test the evolved rules in all load conditions found in the literature. Moreover, processing times are sampled with $cv$ at different levels, similarly to Lawrence and Sewell (1997). Previous studies show that the shop size does not affect the relative performance of the rules (Blackstone et al., 1982). For that reason, almost all studies on dispatching rules for the JSSP consider ten machines or less in the simulation model. However, that may not be true for some recent and more sophisticated parameters used in certain rules. Some of them, such as WOR, the workload in the next machines’ queue, from Nguyen et al. (2019), may be influenced by the number of machines in the shop. Therefore, and to attest the generalisation of our rules, we consider larger values of $|M|$ in our test set. The number of operations is selected from U(2$f$, 14$f$), where $f = |M|/10$. For $|M| = 10$, the number of operations follows Holthaus and Rajendran (2000) and is selected from U(2,14). Given the large set of parameters, we do not apply a full-factorial combination on all their values on the test set. The extreme load conditions (where $a = 1.3$ or 10, $u = 0.7$ or 0.99) are tested only for $|M| = 10$, $\bar{p} = \{25, 50, 100\}$ and $cv = 0$.

3.4. Algorithm parameters

Preliminary experiments were conducted to tune the algorithm parameters. We observed that GP is not very sensitive to small changes in the crossover and mutation probabilities. Thus, we selected two extreme configurations,
as shown in Table 2, and performed 30 independent runs for each combination of crossover/mutation probability and ρ (the bloat control factor). The table shows the average performance (τ) of the best evolved rules, considering the validation set. We also report the average and maximum rules size, i.e., the number of nodes in the corresponding trees. Notice that, in general, the highest mutation probability leads to better performance, regardless of the bloat control factor. Moreover, for ρ = 0, the smaller crossover probability produces a smaller average size. The crossover operation seems to promote unnecessarily long trees.

| ρ     | Crossover = 0.8/ Mutation = 0.2 | Crossover = 0.2/Mutation = 0.8 |
|-------|--------------------------------|--------------------------------|
|       | Avg. τ | Avg. size | Max. size | Avg. τ | Avg. size | Max. size |
| 10^-4 | 2.46   | 12.7      | 27        | 2.43   | 13.6      | 27        |
| 10^-5 | 2.31   | 39.1      | 101       | 2.06   | 40.2      | 73        |
| 0     | 2.39   | 197.8     | 389       | 2.05   | 98.6      | 207       |

The GP parameters used in the final experiments are found in Table 3. Limiting the tree depth to small values prematurely prunes good branches, so we decided to set the maximum depth of trees to a large number commonly used (17). As the bloat control factor ρ impacts both the size of the rules and their performance, it received special attention. Section 4 details the results obtained for different values of ρ. We observed that the larger the population size, the better the average results obtained, although for values above 200, the improvement in solution quality is negligible. The number of generations (max g) is limited to 50, as in preliminary runs, no improvement in the fitness function was observed beyond generation 35. As the population is usually too homogeneous after some generations, the tournament size was set to the smaller possible value (2) to promote diversity. We set the values for the probability of selecting a terminal and a non-terminal (0.1 and 0.9, respectively) and the parameters of the Ramped Half-and-Half algorithm (minimum tree depth of 2 and maximum 6). All algorithms were coded in Java, and the GP procedure uses the evolutionary computation library ECJ [Luke 2017].

| GP Parameters Setting          |
|--------------------------------|
| Population size                | 200 |
| Number of generations (max g)  | 50  |
| Tournament size (t)            | 2   |
| Maximum trees depth            | 17  |
| Crossover probability          | 0.2 |
| Mutation probability           | 0.8 |
4. Learning iterations

We performed four iterations on the procedure described in Figure[1] which have resulted in four GP variants: GPLit, GPBasic, GPImp and GPStruct. The first two variants are based on the set of terminals used by the GP and scheduling literature, respectively. In the last two iterations, we improve the function set and propose a specific structure for the rules based on the analysis of the problem and the results obtained. The following sections describe each variant and the results obtained with them.

We present beforehand the list of terminals of GPBasic, GPImp and GPStruct in Table 4. The second column of the table shows the size associated with each terminal, i.e., the number of atomic parameters and operators. CR, for instance, has size three, corresponding to AJ/RPT (“AJ”, “/”, “RPT”) and PT-CR has size five, corresponding to “PT”, “/”, “AJ”, “/” and “RPT”. To calculate WINQ (the work in the next machines’ queue), we use the procedure described in [Raghu and Rajendran (1993)]. As the terminal set of GPLit is too extensive, it is presented in Section B of Appendix.

Table 4: Terminal sets considered in GPBasic, GPImp and GPStruct

| Terminal   | Size | Description                              | GPBasic | GPImp | GPStruct |
|------------|------|------------------------------------------|---------|-------|----------|
| Slack      | 0    |                                          | x       | x     | x        |
| Slack ≤ 0  |      |                                          | x       | x     |          |
| PT         | 1    | Processing time of current operation     | x       | x     | x        |
| WINQ       | 1    | Work in the next queue                   | x       | x     | x        |
| NPT        | 1    | Processing time of next operation        | x       |       |          |
| U          | 1    | Shop utilisation                         | x       |       |          |
| PT-S/RPT   | 5    | Operation due date (using slack)         |         | x     |          |
| PT-CR      | 5    | Operation due date (using allowance)     | x       |       |          |
| S⁺         | 1    | Slack if positive                        | x       | x     |          |
| AJ⁺        | 1    | Allowance if positive                    | x       | x     |          |
| S⁺/RPT     | 3    | Relative slack if positive               | x       | x     |          |
| CR⁺        | 3    | Critical ratio if positive               | x       | x     |          |
| PT-S⁺/RPT  | 5    | Operation due date (using slack) if positive | x       | x     |          |
| PT-CR⁺     | 5    | Operation due date (using allowance) if positive | x       | x     |          |
| RPT        | 1    | Remaining processing time                | x       | x     |          |

4.1. Using terminals from the literature

In the first two iterations, the function sets follow the literature in two different ways. While the first (GPLit) reproduces previous GP experiments, the second (GPBasic) is based on the classical dispatching rules. Next, we detail both sets and the average results obtained with them.
We provide two terminal sets for GPLit, namely GPLita from Nguyen et al. (2013) and GPLitb based on Nguyen et al. (2019). In both works, GP finds well-performing rules for specific job shop settings. Ephemeral random constants are not included in our experiments to promote interpretability and generalisability. In the operators set, we use the basic mathematical operations (+, -, ·, /), max and min, as in the works mentioned. GPLitb also contains an If operator, as in Nguyen et al. (2019). In all experiments the protected division applies, i.e., \( a/b = 1 \) if \( b = 0 \).

In the second iteration, we aim at providing GP with a smaller set of selected terminals and operators. The earlier studies on dynamic job shop scheduling show that look-ahead parameters should be incorporated into the SPT rule. Moreover, due date information (slack or allowance) must be considered to prioritise the most urgent operations. These parameters are found in the best existing rules, such as RR, 2PT+WiNQ+NPT, CR+SPT and S/RPT+SPT. Thus, the terminal set of GPBasic contains only the parameters present in those rules, including the processing time of current operation (PT), due date information (S/RPT and CR), next operation processing time (NPT) and work in the next queue (WINQ). The parameters \( \exp(u) \) and \( \exp(-u) \) from RR are omitted since there is no justification for using the exponential function. GPBasic and the next two variants use only the basic four mathematical operations (+, -, ·, /) in the operators set.

We describe the results of these two variants in the following. The rules’ performance is calculated using the metric for training described in Section 3.1. We set four distinct values for the bloat control factor \( \rho = \{10^{-3}, 10^{-4}, 10^{-5}, 0\} \) to explore the trade-off between rules size and performance. Following Nguyen et al. (2019), 30 independent executions were performed for each GP variant and value of \( \rho \). When \( \rho = 10^{-3} \), both versions of GPLit get stuck in tiny rules with poor performance (\( \tau \sim 3 \)). Similarly, GPBasic produces PT+WINQ, which is not effective for mean tardiness (\( \tau = 2.36 \)) as it does not consider due dates. The for the other values of \( \rho \) is presented in Figure 3. The average performance of existing rules is also indicated by horizontal dashed lines.

The overall performances of GPLita and GPLitb are similar. In 90 runs, GPLita resulted in four good rules (\( \tau < 1 \)) and GPLitb in only two. Between both versions, GPLitb found the best rule (\( \tau = 0.93 \)) and its average performance is slightly better, 2.24 against 2.34 from GPLita. The best rules are found when no bloat control is applied or using light bloat control. Regarding GPBasic, when \( \rho = 10^{-4} \), the algorithm generates PT+WINQ in 17 out of 30 runs, resulting in poor average performance. However, for \( \rho > 10^{-4} \), this version performs better than GPLita and GPLitb. Nine rules are better than RR, and the best performance is 0.86. From these results, we conclude that with the same parameters, GP was able to find a better combination than manually devised rules. Moreover, we suspect that the function sets of GPLita and GPLitb are probably unnecessarily large. As a consequence, there is no dimensional consistency in the rules devised with them - some terminals refer to the number of operations, others to the remaining work or even the number of machines yet to be visited. Combining these terminals is thus an additional challenge for GP. It is also interesting to observe that terminal U, which is used to adapt rules to the shop utilisation in the rule RR, is not present in the best evolved rules with GPBasic.

Figure 4 presents the size of all rules evolved in these two first iterations and the size of RR for reference. We observe that our bloat control approach is effective to provide smaller rules, although the average performance is
similar for $\rho = 10^{-5}$ and $\rho = 0$ (see Figure 3). The good performance of GPBasic obtained at the cost of considerably longer rules. Indeed, there is no significant difference in the size obtained with the different GP variants. These first experiments demonstrate that GP can produce rules that perform well on average, but there is room for improvement. The well-performing rules are too long and cannot be easily interpreted. In the next section, we detail how the terminal set of GPBasic is improved in the last two iterations.

4.2. Improving the terminal set

Highly loaded settings and tight due dates lead to negative values for allowance and slack, which introduces bias in the rules’ behaviour [Haupt 1989, Anderson and Nyirenda 1990]. The term CR·PT prioritise operations with long instead of short processing times if CR is negative. Moreover, these parameters are not relevant in tight load conditions, even for optimising due date measures [Ramasesh 1990]. Thus, in GPImp, the slack and allowance of jobs are considered only when positive; we use 0 otherwise. Terminal S (slack) is substituted by $S^+$ (slack if positive), and CR becomes $CR^+$ (critical ratio if positive). Hence, GPImp uses $PT·S^+/RPT$ instead of $PT·S/RPT$ and $PT·CR^+$ instead of $PT·CR$ as well. We also decompose some terminals into their atomic parameters to verify if GP finds better combinations. For instance, in addition to $PT·S^+/RPT$, we also include $PT$, $S^+$, $RPT$ and $S^+/RPT$. Terminals $NPT$ (next operation processing time) and $U$ (shop utilisation) are discarded, as they do not appear in the best rules evolved with GPBasic.

The last learning iteration designs a preliminary structure for the rules. If job slack is negative, it is not possible to meet the due date. From the good performance of the rules generated with GPImp, we conclude that it is better to not consider any due date information in such cases. Thus, in GPStruct, two expressions are used to calculate the score of each operation, depending on job slack. The idea is similar to $CR+SPT$ and $S/RPT+SPT$, which apply operations...
Figure 4: Size of rules evolved with GPLita, GPLitb and GPBasic

Figure 5: Fixed structure proposed for rules evolved

Figure 6 presents the rules’ performance by GP variant - GPLita and GPLitb are not included given their poor performance. GPImp provides much better rules than GPBasic, and is in turn outperformed by GPStruct. GPStruct provides the best performing rule ($\tau = 0.80$) and the best average performance. Moreover, GPStruct consistently generates good rules, regardless the value of $\rho - \tau \leq 0.86$ for 79% of the rules evolved. For instance, the rule $If (slack > 0) PT \cdot CR + 2WINQ Else 2PT + WINQ \ (\tau = 0.84)$ was generated in 56% of the runs with $\rho = 10^{-4}$. It is interesting that when PT-CR is not used, GP finds the same expression from the combination of the nodes \{PT, ”·”, CR\}. 

17
Table 5 summarises the average performance and the best rule of GPLit (joint GPLit_1 and GPLit_2), GPBasic, GPImp and GPStruct. The improvement in performance provided by each iteration is clear, regardless of the bloat control factor (ρ). On average, the rules evolved with GPBasic present a performance of 1.61 while GPImp and GPStruct find rules with 1.05 and 0.89, respectively. We also report the average size of the rules. As the rules evolved with GPStruct combine two expressions, they are slightly greater than the others.

Table 5: Performance and size of rules evolved using preliminary versions of GP.

| ρ  | GPLit | GPBasic | GPImp | GPStruct |
|----|-------|---------|-------|----------|
|    | r | r_min | r_max | r | r_min | r_max | r | r_min | r_max | r | r_min | r_max | r | r_min | r_max |
| 10^{-4} | 2.40 | 1.00 | 13.6 | 2.07 | 1.24 | 8.3 | 1.23 | 0.83 | 12.0 | 0.87 | 0.83 | 15.7 |
| 10^{-5} | 2.20 | 0.97 | 39.0 | 1.39 | 0.82 | 45.7 | 0.97 | 0.81 | 47.0 | 0.89 | 0.80 | 34.8 |
| 0     | 2.11 | 0.93 | 91.7 | 1.36 | 0.86 | 76.3 | 0.95 | 0.81 | 76.3 | 0.91 | 0.80 | 114.3 |
| Avg.  | 2.24 | 0.97 | 48.1 | 1.61 | 1.00 | 43.4 | 1.05 | 0.81 | 45.1 | 0.89 | 0.81 | 54.9 |

5. Evolved Rules

In this section, we exhibit and analyse the best evolved rules considering a two-criteria objective, finding effective and interpretable rules. These objectives are measured by the rules’ relative performance in the test set and their size, respectively. For presenting the detailed results in the test set, we select non-dominated rules in both criteria. Next, we explain this selection.
5.1. Rules selection

The chart in Figure 7 presents the performance on the validation set (y-axis) and size (x-axis) of all evolved rules with $\tau \leq 1$. Both axes are on a logarithmic scale. Observe that the rules generated with GPLit (GPLit_a and GPLit_b) are all dominated. The chart highlights three non-dominated rules (R1, R2, R3), evolved with GPImp and GPStruct. Among them, R1 is the shortest and least performing (0.83). Rule R3 is the largest and best performing (0.80), although R2 presents a very similar performance (of around 0.80).

![Figure 7: Size and relative performance of the evolved rules.](image)

Table 6 presents the mathematical expression of each non-dominated rule. R2 was simplified by applying mathematical operations, e.g., $a + a + a$ is converted to $3a$, and $a/a$ is 1. Column “Size” shows the size of the expression after these modifications and the original size in brackets. R3 is too large and cannot be easily interpreted. Thus we do not present it. We detail R1 and R2 next.

Table 6: Evolved rules

| Rule | Mathematical expression | Size | $\tau$ | Algorithm |
|------|--------------------------|------|--------|-----------|
| R1   | $2PT + PT \cdot S/rPT + WINQ$ | 11   | 0.828  | GPImp     |
|      | *If (Slack > 0) Then*       |      |        |           |
|      | $PT + PT \cdot CR + WINQ$   | 19 (31) | 0.805  | GPStruct  |
|      | *Else*                      |      |        |           |
|      | $3PT + WINQ - (WINQ/PT)$    |      |        |           |
| R3   | (large expression)         | 77   | 0.797  | GPStruct  |

R1 has a structure similar to RR from [Raghu and Rajendran (1993)](#). Both rules are composed of three terms, PT
(operations processing time), $\text{PT-S/RPT}$ (operations due date) and WINQ (work in the next machines’ queue). The difference is in the balance between PT and PT-S/RPT. The rule RR use weights $\exp(u)$ and $\exp(-u)$ for PT and PT-S/RPT, respectively, adapting the influence of those terms to the shop utilisation $u$. R1, instead, omits the term PT-S/RPT for delayed jobs and applies weight 2 for PT (in RR, the weight of PT varies from 2.0 to 2.7). For negative or null slacks, R1 behaves just like $2\text{PT} + \text{WINQ}$, which is an expression frequently found by GPStruct for the branch of non-positive slacks.

The rule R2 is equivalent to R1 if the job slack is positive. The terminal CR corresponds to AJ/RPT. As slack (S) is $\text{AJ - RPT}$, CR = $(\text{S+RPT})/\text{RPT} = 1 + \text{S/RPT}$. Thus, PT-$(2 + \text{S/RPT})$, from R1, corresponds to PT-$(1 + \text{CR})$, from R2. The behaviour of both rules is different only when the slack is negative or zero. In that case, R2 assigns weight 3 for PT and subtract WINQ/PT, favouring jobs with short processing times. The balance between operations processing time (PT) and work in the next queue (WINQ) in R2 is 3:1, similar to $2\text{PT} + \text{WINQ} + \text{NPT}$ - the best existing rule for extreme tight load settings. Except when PT is considerably small, the term WINQ/PT is much smaller than the other terms in the expression, with almost no influence on the resulting score. Thus, R2 emphasise the use of the shortest processing time for jobs in congested shops.

5.2. Performance under deterministic processing times

This section details the results obtained with R1 and R2 in our test set for deterministic processing times. We recall that the performance $\tau_a$ is the mean relative performance of rule $a$ against the best existing rules (described in Section 2.1). For each instance $i$, we calculate $\bar{T}_{ai}/\bar{T}_{ri}$, where $r$ is the best rule for $i$ among all of the benchmarked rules. Thus, if $\tau_a < 1$, rule $a$ outperforms, on average, all existing rules combined.

Table 7 presents the value of $\tau$ for each combination of shop utilisation and allowance factor. In these experiments we use the most common values for processing times ($\bar{p} = \{25, 50, 100\}$) and number of machines ($|M| = 10$). We performed statistical tests to compare the result of each evolved rule to the best existing rules. We obtained significant results at the 0.05 level for almost all settings – the exceptions are marked with †. In these cells, we consider that there is a tie between the evolved rule and the benchmark as in cells with $\tau = 1.00$.

The cells in the top right corner correspond to light load settings - large allowance factor and low shop utilisation. In these settings, the (evolved and benchmark) rules usually produce no tardiness, resulting in $\tau = 1.00$ (tie). Considering all settings, R1 and R2 perform better than the benchmark. R1 outperforms existing rules in 33 out of 49 settings and is worse in 7. R2 has better average performance than R1 and works better in a larger number of settings, 38 out of 49. The effectiveness of both rules is due to a better balance between operations processing time and due date, especially when job slack is negative. However, the evolved rules deteriorate under higher utilisation levels. Observe that $\tau > 1$ more frequently in cells where $u \geq 0.97$. As the average queue size increases with utilisation level, the value of WINQ becomes larger, but the weight of PT does not follow. We conclude that it could be desirable to train specific rules for these tight load settings.
Table 7: Performance under deterministic processing times and distinct values of shop utilisation and allowance factor.

| $u$  | $a$ | R1       |       | R2       |       |
|------|-----|----------|-------|----------|-------|
| 0.70 | 1.3 | 0.98     | 0.78  | 1.00     | 0.87  |
|      |     | 0.94     | 0.53  | 1.00     | 0.82  |
| 0.80 | 0.97| 0.97     | 0.85  | 0.56     | 0.73  |
| 0.90 | 1.00| 0.99     | 0.90  | 0.69     | 0.62  |
| 0.95 | 0.98| 0.98     | 0.86  | 0.91     | 0.92  |
| 0.97 | 0.99| 0.99     | 0.96  | 1.08     | 1.00  |
| 0.99 | 1.02| 1.01     | 1.10  | 1.21     | 1.10  |
| Avg  | 0.99| 0.97     | 0.91  | 0.82     | 0.81  |

Table 8 extends the analysis to a larger number of machines and different processing time distributions. Each cell shows the average result for $u = \{0.8, 0.85, 0.9, 0.95, 0.97\}$, and all results are significant at the 0.05 level. As the number of machines increases, R1 and R2 tend to perform better than the benchmark rules (the value of $\tau$ decreases). The exception is the settings with very tight due dates ($a = 2$), where both rules deteriorate performance. We observed that as $|M|$ increases, the average queue size also increases, from 4.5 (with ten machines) to 7 (with 100 machines). This system behaviour affects the balance between WINQ and PT and is likely the reason for the worse performance of the rules in such settings. Another exception is $a = 8$ and $|M| = 100$, where RR (the benchmark rule), R1, and R2 produce zero tardiness, resulting in $\tau = 1.00$. The processing time distribution has little effect on the performance of the rules.

Table 8: Performance for different number of machines and processing time distribution.

| $\bar{p}$ | $|M|$ | R1       |       | R2       |       |
|----------|------|----------|-------|----------|-------|
| 10       | 25 - 100 | 0.97     | 0.73  | 0.83     | 0.97  |
| 20       | 50    | 1.00     | 0.50  | 0.68     | 1.00  |
| 50       | 100   | 1.06     | 0.33  | 0.55     | 1.04  |
| 250      | 10    | 0.96     | 0.73  | 0.84     | 0.96  |
| 500      | 10    | 0.96     | 0.72  | 0.83     | 0.96  |

5.3. Performance under stochastic processing times

To assess the effectiveness of the evolved rules under uncertain processing times, we compare the mean tardiness generated by each rule for different $cv$ values. As explained in Section 3.3, $cv$ is the coefficient of variation for sampling the actual processing times, (larger values for $cv$ correspond to higher uncertainty levels; when $cv = 0$, 21
processing times are deterministic). In Table 9 we report the average value of \( \tau \) under stochastic processing times. Although the relative performance of both rules deteriorates, they are still better than the benchmark – notice that the training set is composed of instances with deterministic processing times. The average performance of R1 and R2 for each setting is presented in Section C of Appendix.

Table 9: Overall performance under stochastic processing times

| Rule | cv 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 | Avg  |
|------|--------|-----|-----|-----|-----|-----|------|
| R1   | 0.83   | 0.91| 0.92| 0.90| 0.92| 0.91| 0.90 |
| R2   | 0.81   | 0.85| 0.86| 0.87| 0.88| 0.89| 0.86 |

Our analysis finishes with the effect of uncertainty in the mean tardiness for distinct load conditions. The charts in Figure 8 present the mean tardiness \( \bar{T} \) obtained for rules RR, 2PT+WINQ+NPT, R1 and R2, considering all uncertainty levels. Notice that now we report absolute values (unlike the previous sections). These results refer to instances where \( \bar{p} = 50 \) and \( |M| = 10 \). In general, all dispatching rules produce higher tardiness as \( cv \) increases. Comparing the existing rules, RR and 2PT+WINQ+NPT are good in light and tight load conditions, respectively, but none of them works well in both cases. In medium conditions (Figure 8b), RR performs well if the \( cv \) is low; as uncertainty increases its performance degrades fast. R2 consistently produces lower tardiness than both, regardless of the setting, and R1 is good in medium conditions.

(a) Light load conditions, \( a = 8 \) and \( u = 0.89 \)  
(b) Medium load conditions, \( a = 4 \) and \( u = 0.90 \)  
(c) Tight load conditions, \( a = 2 \) and \( u = 0.97 \)

Figure 8: Mean tardiness generated by evolved and existing rules for stochastic processing times.
6. Conclusions and future work

This work focuses on the use of problem expertise to leverage the computational power of machine learning. Genetic Programming is applied as the learning approach on the design of dispatching rules for the Dynamic Job Shop Scheduling Problem. The objective is to find rules that minimise mean tardiness, one of the most challenging performance criteria in dynamic job shops. We propose a guided empirical learning procedure, where reasoning on the problem derives insights to guide the algorithmic search in an iterative way. At each iteration, modifications are applied to the terminal set or rules structure, and new results are generated and analysed. Unlike traditional approaches, GP is also provided with problem expertise from the existing literature.

We found that GP can combine parameters from existing rules and generate better ones. Another finding is that a small terminal set of selected parameters enables evolving more effective dispatching rules than working with large terminal sets. Moreover, we conclude that the overall effectiveness of the evolved rules is even better when providing a fixed good structure.

All rules generated were evaluated in two criteria, performance and size, and the non-dominated were selected for testing. Besides having an interpretable structure, these rules present an overall improvement of 19% compared to the best possible combination of the state-of-the-art rules. This superiority is achieved in almost all settings, from extreme light load conditions (where all jobs meet the due date) to congested ones. Among all tested settings, only for extremely tight conditions, where shop utilisation is around 99%, the average performance of the new rules does not outperform the literature. Furthermore, the good behaviour of the rules is confirmed in shops with a different number of machines and under stochastic processing times.

As future work, our guided empirical learning approach should be tested in different problems, such as the Flexible Job Shop Scheduling or Parallel Machine Scheduling. Also, as there are several performance criteria in all these scheduling problems, multi-objective optimisation becomes a relevant research topic.

Acknowledgments

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme, and by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project SAICTPAC/0034/2015-POCI-01-0145-FEDER-016418

References

Amin, G.R., El-Bouri, A., 2018. A minimax linear programming model for dispatching rule selection. Computers & Industrial Engineering 121, 27 – 35. doi:doi:https://doi.org/10.1016/j.cie.2018.05.021.

Anderson, E.J., Nyirenda, J.C., 1990. Two new rules to minimize tardiness in a job shop. International Journal of Production Research 28, 2277–2292. doi:doi:10.1080/00207549008942866
Johannes Kepler University, Linz, Austria.

Lawrence, S.R., Sewell, E.C., 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. Journal of Operations Management 15, 71 – 82. doi:https://doi.org/10.1016/S0272-6963(96)00090-3

Lopes, R.L., Figueira, G., Amorim, P., Almada-Lobo, B., 2020. Cooperative coevolution of expressions for (r,q) inventory management policies using genetic programming. International Journal of Production Research 58, 509–525. doi:https://doi.org/10.1080/00207543.2019.1597293

Luke, S., 2017. Ec then and now, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, Association for Computing Machinery, New York, NY, USA. p. 1223–1230. doi:https://doi.org/10.1145/3067695.3082467

Luo, S., 2020. Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Applied Soft Computing 91, 106208. doi:https://doi.org/10.1016/j.asoc.2020.106208

Mei, Y., Zhang, M., Nyugen, S., 2016. Feature selection in evolving job shop dispatching rules with genetic programming, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, Association for Computing Machinery, New York, NY, USA. p. 365–372. doi:https://doi.org/10.1145/2908812.2908822

Nguyen, S., Mei, Y., Xue, B., Zhang, M., 2019. A hybrid genetic programming algorithm for automated design of dispatching rules. Evolutionary Computation 27, 467–496. doi:https://doi.org/10.1162/evco_a_00230

Nguyen, S., Mei, Y., Zhang, M., 2017a. Genetic programming for production scheduling: a survey with a unified framework. Complex & Intelligent Systems 3, 41–66.

Nguyen, S., Zhang, M., Johnston, M., Tan, K., 2013. Learning iterative dispatching rules for job shop scheduling with genetic programming. The International Journal of Advanced Manufacturing Technology 67. doi:https://doi.org/10.1007/s00170-013-4756-9

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

Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2013. Dynamic Multi-objective Job Shop Scheduling: A Genetic Programming Approach. Springer Berlin Heidelberg, Berlin, Heidelberg. chapter 10. pp. 251–282. doi:https://doi.org/10.1007/978-3-642-39304-4_10

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

Nguyen, S., Zhang, M., Johnston, M., Tan, K.C., 2014. Selection schemes in surrogate-assisted genetic programming for job shop scheduling, in: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, I.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (Eds.), Simulated Evolution and Learning, Springer International Publishing, Cham. pp. 656–667.

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

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

Nie, L., Gao, L., Li, P., Li, X., 2013. A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. Journal of Intelligent Manufacturing 24, 763–774. doi:https://doi.org/10.1007/s10845-012-0626-9

Olsen, T.L., Tomlin, B., 2020. Industry 4.0: Opportunities and challenges for operations management. Manufacturing & Service Operations Management 22, 113–122. doi:https://doi.org/10.1287/msom.2019.0796

Ouelhadj, D., Petrovic, S., 2008. A survey of dynamic scheduling in manufacturing systems. Journal of Scheduling 12, 417. doi:https://doi.org/10.1007/s10951-008-0090-x

Parente, M., Figueira, G., Amorim, P., Marques, A., 2020. Production scheduling in the context of industry 4.0: review and trends. International Journal of Production Research 0, 1–31. doi:https://doi.org/10.1080/00207543.2020.1718794

Park, J., Mei, Y., Nguyen, S., Chen, G., Zhang, M., 2018. An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. Applied Soft Computing 63, 72 – 86. doi:https://doi.org/10.1016/j.asoc.2017.11.020

Pergher, I., Frej, E.A., Roselli, L.R.P., [de Almeida], A.T., 2020. Integrating simulation and fitradeo method for scheduling rules selection in job-shop production systems. International Journal of Production Economics 227, 107669. doi:https://doi.org/10.1016/j.ijpe.2020.107669
A. Job Shop Simulator

The pseudo-code of the job shop simulator is described in Algorithm 2. The inputs are instance parameters <\{M\}, \bar{p}, a, u, cv> and a dispatching rule. As described in Section 3.2, each job arrival \( r_j \) is sampled from a Poisson distribution which mean rate depends on the value of \( u \). The allowance factor \( a \) is considered to calculate jobs’ due dates. The parameter \( \bar{p} \) is the mean processing time, and \( cv \) is the coefficient of variation for sampling the actual processing times. Table 10 describes some of the variables defined in the algorithm and the respective initialisation.

The next operation of each job enters machines’ queue \( Q_m \) at time \( start_j \), when they are available for execution.
Table 10: Main variables considered by Algorithm 2

| Variable     | Description                               | Initialisation |
|--------------|-------------------------------------------|----------------|
| num_jobs     | Number of finished jobs                   | 0              |
| J’           | Jobs not completed                        | J              |
| start_j      | Minimum start time of job j next operation | r_j            |
| m_next_j     | Next machine of job j ∈ J                 | First machine of j |
| Q_m          | Queue in machine m                        | 0              |
| A_m          | Next available time of machine m          | 0              |

The simulation starts with an empty shopfloor and goes on until a number of jobs have been computed (max_jobs). The next operation of each job enters the machines’ queue (Q_m) at time start_j, when they are available for execution. In line 2, the algorithm verifies which is the earliest event between the availability of a new operation of a job (start_j) and the availability of machine (A_m). An operation becomes available upon job arrival or when the predecessor operation finishes. In both cases, the operation may enter the next machines’ queue (line 4), and the value of start_j is set to ∞. Once a machine whose queue is not empty becomes available (line 7), the next operation to be executed is selected. In lines 8 to 10, the dispatching rule runs and calculates a score for each job in the queue. The job j* with the least score is executed (in case of ties, the algorithm selects the job that arrived earlier to the queue). If j* has no operations left, it is completed, removed from J’ and the tardiness is calculated (lines 15-18). Otherwise, the algorithm updates the start time of the jobs’ next operation (line 20). We recall that jobs are numbered on arrival in a crescent sequence order, and the tardiness T_j is computed only for jobs numbered from min to max. Following the
values adopted in literature, we set \( \min = 501 \) and \( \max = 2500 \), \( \text{max\_jobs} = \max - \min = 2000 \).

**Algorithm 2: Job Shop Simulator**

**Input:** Instance \(<|M|, \bar{p}, a, u, cv>\) and DR

**Result:** Mean Tardiness \( \bar{T} \)

1. while \( \text{num\_jobs} < \text{max\_jobs} \) do
2. 
3.     if \( \min\{\text{start}_j\} < \min \{A_m\} \) then
4.         \( j^* \leftarrow \arg \min\{\text{start}_j\} \);
5.         \( Q_{m,\text{next},j} \leftarrow Q_{m,\text{next},j} + j^*; \)
6.         \( \text{start}_j \leftarrow \infty ; \)
7.     else
8.         \( m \leftarrow \arg \min\{A_m\}; \) \( m \in M, Q_m \neq \emptyset \)
9.         foreach job \( j \in Q_m \) do
10.            \( \text{score}_j \leftarrow \text{run\_DR on } j ; \)
11.            \( j^* \leftarrow \arg \min\{\text{score}_j\}; \)
12.       execute \( j^* \) in \( m \);
13.       \( A_m \leftarrow \text{finish\_time}_{j^*} ; \)
14.       if \( m = \text{last\_machine of } j^* \) then
15.           \( J' \leftarrow J' \setminus j^* ; \)
16.           if \( j^* \in \{\min...\max\} \) then
17.               \( \text{num\_jobs} \leftarrow \text{num\_jobs} + 1 ; \)
18.               \( T_j \leftarrow \max(0, d_j - \text{finish\_time}_{j^*} ) ; \)
19.               \( \bar{T} \leftarrow \bar{T} + T_j ; \)
20.           else
21.               \( \text{start}_j \leftarrow \text{finish\_time}_{j^*} ; \)
22.       \( \bar{T} \leftarrow \bar{T} / \text{max\_jobs}; \)

**B. Terminal sets considered in GP experiments**

Table 11 lists all terminals used in the first learning iteration, where two distinct sets are proposed. For calculating the evolved rules’ size, we assume that all terminals in these sets have size one.
### Table 11: Terminal sets for GPLit

| Description                                                                 | GPLit_a | GPLit_b |
|----------------------------------------------------------------------------|---------|---------|
| Average processing time of operations in current queue                     | x       | x       |
| Expected processing time of current operation                              | x       | x       |
| Job allowance                                                               |         | x       |
| Job arrival time                                                            |         | x       |
| Job due date                                                                |         | x       |
| Job queueing time                                                           |         | x       |
| Machine ready time                                                          | x       |         |
| Maximum due date time among jobs in queue                                   | x       |         |
| Maximum processing time among operations in queue                           | x       |         |
| Minimum due date time among jobs in queue                                   | x       |         |
| Minimum processing time among operations in queue                           | x       |         |
| Number of jobs in the system                                                | x       |         |
| Number of operations left                                                   | x       | x       |
| Operation ready time                                                        | x       |         |
| Processing time of the next operation of the job                            | x       | x       |
| Queue size in current machine                                               |         | x       |
| Queue size in the next machine                                              |         | x       |
| Remaining processing time of job                                            | x       |         |
| Remaining processing time of all jobs in the queue                          | x       |         |
| Remaining time of the successive operations                                 |         |         |
| Slack of job                                                                | x       | x       |
| Time in the system                                                          | x       | x       |
| Work in current machines’ queue                                             |         |         |
| Work in the next machines’ queue                                            |         | x       |
| Work in all successive machines’ queues                                     |         | x       |

#### C. Performance of evolved rules under stochastic processing times

Table 12 presents the performance $\tau$ of R1 and R2 for each combination of allowance factor and shop utilisation. Despite some variance in the results, the relative performance of both rules tends to be more homogeneous compared to settings with deterministic processing times. Moreover, both rules behaviour is better under light load conditions.
Table 12: Performance under stochastic processing times for each setting.

|      | R1  |          |      | R2  |          |
|------|-----|----------|------|-----|----------|
|      | 2   | 3        | 4    | 6   | 8        | Avg  |        | 2   | 3   | 4    | 6   | 8   | Avg  |
| 0.80 | 0.95| 0.89     | 0.83 | 0.78| 0.97     | **0.88** | 0.95 | 0.88 | 0.82 | 0.68 | 0.76 | **0.81** |
| 0.85 | 0.97| 0.89     | 0.85 | 0.77| 0.77     | **0.84** | 0.96 | 0.88 | 0.83 | 0.74 | 0.62 | **0.80** |
| 0.90 | 0.96| 0.93     | 0.88 | 0.82| 0.78     | **0.87** | 0.95 | 0.91 | 0.86 | 0.79 | 0.73 | **0.85** |
| 0.95 | 0.97| 0.97     | 0.93 | 0.89| 0.88     | **0.93** | 0.96 | 0.96 | 0.92 | 0.86 | 0.82 | **0.90** |
| 0.97 | 0.98| 0.97     | 0.97 | 0.96| 0.94     | **0.97** | 0.97 | 0.97 | 0.93 | 0.93 | 0.91 | **0.94** |
| Avg  | 0.97| 0.93     | 0.89 | 0.84| 0.86     | **0.90** | 0.96 | 0.92 | 0.87 | 0.80 | 0.76 | **0.86** |