Toward Evolving Dispatching Rules for Dynamic Job Shop Scheduling Under Uncertainty

Deepak Karunakaran
School of Engineering and Computer Science, Victoria University of Wellington
P.O. Box 600
Wellington, New Zealand
deepak.karunakaran@ecs.vuw.ac.nz

Yi Mei
School of Engineering and Computer Science, Victoria University of Wellington
P.O. Box 600
Wellington, New Zealand
yi.mei@ecs.vuw.ac.nz

Gang Chen
School of Engineering and Computer Science, Victoria University of Wellington
P.O. Box 600
Wellington, New Zealand
aaron.chen@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and Computer Science, Victoria University of Wellington
P.O. Box 600
Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

ABSTRACT

Dynamic job shop scheduling (DJSS) is a complex problem which is an important aspect of manufacturing systems. Even though the manufacturing environment is uncertain, most of the existing research works consider deterministic scheduling problems where the time required for processing any job is known in advance and never changes. In this work, we consider DJSS problems with varied uncertainty configurations of machines in terms of processing times and the total flow time as scheduling objective. With the varying levels of uncertainty, many machines become bottlenecks of the job shop. It is essential to identify these bottleneck machines and schedule the jobs to be performed by them carefully. Driven by this idea, we develop a new effective method to evolve pairs of dispatching rules each for a different bottleneck level of the machines. A clustering approach to classifying the bottleneck level of the machines arising in the system due to uncertain processing times is proposed. Then, a cooperative co-evolution technique to evolve pairs of dispatching rules which generalize well across different uncertainty configurations is presented. We perform empirical analysis to show its generalization characteristic over the different uncertainty configurations and show that the proposed method outperforms the current approaches.

CCS CONCEPTS

•Computing methodologies → Planning under uncertainty; Heuristic function construction;

KEYWORDS

job shop scheduling, uncertainty, genetic programming.

GECCO '17, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3071178.3071202

ACM Reference format:
Deepak Karunakaran, Yi Mei, Gang Chen, and Mengjie Zhang. 2017. Toward Evolving Dispatching Rules for Dynamic Job Shop Scheduling Under Uncertainty. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071202

1 INTRODUCTION

Job shop scheduling is an NP-hard problem [29] and is vital for industrial manufacturing. Generally, research works consider deterministic job shop scenarios; though in practice, uncertainty in manufacturing environments is ubiquitous. Machine breakdowns, rework, operator skill and availability, variation in raw material quality, variable due dates, etc. are some of the sources of uncertainty. Apparently scheduling becomes much more difficult with the increasing uncertainty in the shop floor [16].

Dynamic job shop scheduling (DJSS) problems are characterized by continuous arrival of new jobs to the shop and no prior information about them is known. In the deterministic case, once the information of a new job is known, it stays constant. However, in an uncertain scenario, the information varies at the time of realization of the schedule. For example, the processing time of a job varies when a schedule is realized and is different from its *expected* value. In this paper, we consider DJSS with processing time uncertainty because it affects most scheduling objectives.

Some of the methods used to describe the uncertainty [18] are: (1) bounded form, (2) probability description and (3) fuzzy description. The literature review [18] shows that probability description is more widely used. The different approaches used for optimization under uncertainty depend on the description method(s). Broadly, these approaches are classified into *preventive scheduling* and *reactive scheduling*. Preventive scheduling, which employs historical information to generate schedules robust under uncertain events, is further classified into stochastic scheduling [2], robust optimization [16], fuzzy programming [8] and sensitivity analysis [27]. Reactive scheduling methods modify the generated schedule in response to any uncertain events with the help of dispatching rules.

A dispatching rule is a function which assigns priority values to every operation queued on a machine. The operation with the lowest priority value is then processed on the machine. As the new jobs arrive, their operations get queued on the machines as defined in their routes. Dispatching rules are efficient in generating schedules for different scheduling objectives. For scheduling under uncertainty, the approaches using dispatching rules usually performed very well. For example, Lawrence et al. [17] compare the performance of their method using dispatching rules against branch-and-bound technique and show that with the increasing uncertainty in processing times, dispatching rules appeared to perform relatively better. Literature review [9, 14, 17, 21] shows that dispatching rules are highly successful for job shop scheduling under uncertainty.

As an efficient method to generate schedules with very little computational cost, dispatching rules are practical, but their design requires considerable effort due to the need of rigorous experimental analysis. In order to solve this problem, researchers have proposed genetic programming based hyper-heuristic (GPHH) approach to evolve dispatching rules [3, 13, 19, 20, 23, 24]. The approach is shown to be highly successful in various empirical studies [3, 21]. Moreover, it is possible to learn diverse dispatching rules for different shop characteristics by exploiting the flexible representation of genetic programs. Since GPHH approach makes it convenient to evolve multiple rules, it is feasible to evolve rules corresponding to different shop scenarios. For example, [22] proposes a cooperative co-evolutionary approach to design dispatching rules for different scheduling objectives with due-date assignment. Cooperative co-evolution [30] in particular has shown the ability to evolve solutions for problems with interacting subcomponents. Cooperative co-evolution shows itself as a key technique to be investigated when scheduling is considered in a job shop with different sets of machines pertaining to varying bottleneck levels, but affecting the system as a whole.

Genetic programming based hyper-heuristic (GPHH) approach to evolve dispatching rules for dynamic job shop scheduling is thus a promising area. Some recent works [14] have used genetic programming to evolve rules for dynamic job shop scheduling under uncertain processing times. [14] considers job specific uncertainty characteristics and propose an exponential moving average terminal for evolving better rules. To be specific, in a dynamic job shop machines different uncertainty configurations could result in varying bottleneck levels of machines. There are limited existing research works which consider identifying and managing the bottlenecks separately e.g. [12]. Our research therefore exploits this opportunity to investigate the efficacy of evolving specific dispatching rules for bottleneck machines. Exploring GPHH approaches to solve this problem holds good potential. Considering the potential of GPHH approach to evolve rules for different shop characteristics, particularly using multi-population algorithms like cooperative co-evolution, it is a promising research direction to develop methods to design dispatching rules which perform well under varied uncertainty characteristics of a dynamic job shop.

The main goal of this work is to develop a method to design dispatching rules for dynamic job shop scheduling under uncertain processing times, which perform well under varied uncertainty configurations of the machines. The specific objectives are: (1) Develop a new training process using specific uncertainty configurations pertaining to bottleneck and non-bottleneck scenarios. (2) Develop a new effective method to classify the different *bottleneck* levels arising in a dynamic job shop with uncertain processing times, which could then be associated with specific dispatching rules. (3) Develop a cooperative co-evolutionary method to evolve (pairs of) dispatching rules which are able to generalize well for the varying scenarios in the shop.

In the next section, we present the background to the job shop scheduling problem, the genetic programming based hyper-heuristic approach and cooperative co-evolution. Some related works are presented in Section 3. In Section 4, we describe our proposed methods. In Sections 5 and 6, we describe the experiment design and results respectively. Section 7 consists of conclusions and future work.

2 BACKGROUND

2.1 Job Shop Scheduling

We briefly describe the dynamic job shop scheduling (DJSS) problem. In a DJSS problem the jobs arrive at the shop continuously which are assumed to follow a Poisson distribution [21]. A job j has n_j operations which are processed in a predefined route, which can be defined in the form of $(o_{j,1} \rightarrow o_{j,2} \rightarrow, \ldots, o_{j,n_j})$, for a set of operations O_j in job j. Each operation must be processed on one particular machine in the route.

Each operation has a processing time, say $p_{j,i}$. In practice, the actual processing time which is realized in the shop, which we denote as $p'_{j,i}$ is often different from $p_{j,i}$. Typically, the sources of uncertainty are events like repair, rework, unscheduled maintenance, etc which will always delay the processing of operations on a machine. So $p'_{j,i} > p_{j,i}$ is a practical assumption.

Total tardiness, makespan, total flow time , etc. are some of the scheduling objectives considered in the literature. We consider total flow time as the objective in the DJSS problem. Total flow time is defined as

$$F = \sum_j (C_j - r_j)$$

r_j and C_j are the release times and completion times of the jobs.

The general assumptions for DJSS, such as no preemption, no recirculation of jobs, no machine failure, no alternate routing and zero transit times are also considered in this work.

2.2 Genetic Programming Based Hyper Heuristics (GPHHs)

For hard problems in combinatorial optimization like job shop scheduling, hyper-heuristic techniques [5] are useful as they search in the heuristic space rather than in the solution space. In other words, they are used to automate the selection of heuristics. The designed heuristics generate the final solution to the problem.

Genetic programming based hyper-heuristic approach [6] has shown good success [4]. Particularly when compared with other representations like neural network or linear, the flexible representation of a genetic program is usually considered more desirable [3]. The representation is conducive to incorporate the desired characteristics of a job shop into the dispatching rule. For example, Hunt

et al. [11] develop new terminals which generate "less myopic" schedules for DJSS problem. Similarly, in [21] new representations of genetic programming are presented for DJSS problem to evolve better solutions.

2.3 Cooperative Co-evolution

Cooperative co-evolution algorithms (CCEA) are characterized by two or more interacting subspaces within a search space such that the fitness of an individual is evaluated based on its interactions with other individuals (subspace) [33]. A problem is decomposed into subproblems and solutions to the subproblems are then evolved. These are then combined together to form the final solution. The subpopulations belong to different "ecological niches" [30].

CCEA have been employed before for different DJSS problems. Park et al. [25] propose a cooperative co-evolution based multi-level genetic programming approach to evolve ensembles of dispatching rules for DJSS. They had also developed a similar co-evolutionary approach to evolve ensembles of rules for static JSSP [26]. In a related work [15], a co-evolutionary algorithm is proposed to integrate the planning and scheduling activities in flexible manufacturing systems. The success of CCEAs to develop solutions to complex problems in manufacturing systems by considering its sub-components motivates us to explore their utility in DJSS problems under uncertainty leading to multi-bottleneck levels. To our knowledge, there is no previous work which considers cooperative co-evolutionary approach toward evolving dispatching rules over different scenarios in dynamic shop arising due to uncertainty in shop parameters.

3 RELATED WORKS

Adams et al. [1] develop a shifting bottleneck procedure for job shop scheduling to minimize makespan. This work has been modified to solve varied classes of problems [7, 28]. Moreover, bottleneck identification has been shown to be a useful step in order to provide additional computational resources to optimize the sequencing at that machine [32, 34].

Jakobovič et al. [12] propose a genetic programming based method for static job shop scheduling where they consider evolving separate rules for bottleneck and non-bottleneck machines. The machines are classified using a decision rule which is a genetic program with a different set of terminals. Since this work is closely related to our research we compare our methods with it. In the next section we present some more details about this work before describing our proposed methods.

4 PROPOSED METHODS

In this Section, we describe our proposed methods in detail. Firstly, we discuss the work by Jakobović et al. [12] which proposes an adaptive scheduling heuristic for bottleneck and non-bottleneck machines in a static job shop scheduling problem. We make some required modifications for it to work on DJSS. We call this method as GP3. We use this method and the standard GP approach as our benchmarks. Then we develop a new method (GP2-K) which uses unsupervised clustering of machines' states to classify the bottleneck and non-bottleneck machines. Unlike our benchmarks, in GP2-K we use two *specific* uncertainty configurations for evolving bottleneck and non-bottleneck machines. Finally, we present a cooperative co-evolutionary (CGP2-K) approach which aims to evolve dispatching rules with better generalization characteristics for varied levels of uncertainty.

Jakobović et al. [12] propose an adaptive scheduling heuristic, where they evolve a pair of dispatching rules, one for the bottleneck machine and the other for non-bottleneck machine. In order to classify a machine into the two types, they use a third rule, decision rule, which uses a different set of terminals. These terminals are shown in Table 1.

Table 1: Terminal Set: Jakobović-GP3 (Decision rule)

Terminal	Definition
MTWK	Total processing time of all operations on a machine
MTWKr	Processing time of all remaining operations on a machine
MTWKav	Average duration of all operations on a machine.
MNOPr	Number of remaining operations on a machine.
MNOPw	Number of waiting operations on a machine.
MUTL	Machine Utilization.

Compared to static job shop scheduling, in DJSS problems under uncertain processing times, the variation in the bottleneck characteristics of a machine is more prominent. Therefore, the terminals, in particular, the machine utilization (MUTL) terminal should represent the current state of machine. We determine machine utilization by using exponentially decreasing weights for older time periods, so that the recent load on the machine is better represented.

Algorithm 1: GP2-K [Training]

Input:
- $\mathcal{G}_\tau$, total number of generations.
- DJSS training instance ($\mathcal{P}_t$).
  - Simulation parameters
  - $\mathcal{U}_l$ uncertainty configuration (low)
  - $\mathcal{U}_h$ uncertainty configuration (high)

Output: Pair of dispatching rules : $\{DR_l, DR_h\}$

```
Initialize subpopulations S_1, S_2
Set g ← 0
while g ≤ G_τ do
    g ← g + 1
    foreach individual I ∈ S_1 do
        assign fitness to I using DJSS simulation with U_l config.
    end
    foreach individual in S_2 do
        assign fitness to I using DJSS simulation with U_h config.
    end
    Evolve individuals in S_1, S_2 using crossover and mutation.
end
```

4.1 GP2-K-means (GP2-K)

Although GP3 moves a step towards the scenario-dependent rule learning, it is difficult to evolve both the dispatching rules and the decision rule together. Specifically, the error made by the decision rule can potentially affect the dispatching rule learning, since the dispatching rule is applied to a wrong scenario. To address this issue, we propose a new GP training process, which is called GP2-K. GP2-K separates the dispatching rule learning from the decision rule learning. To this end, we select two training sets, one with high uncertainty, and the other with low uncertainty. Then we keep two sub-populations, one for DR_l and the other for DR_h. This way, we can guarantee that the dispatching rule is always applied to the correct scenario, and thus its performance can be evaluated more accurately.

The proposed training process is described in Algorithm 1. Note that difference between the uncertainty levels of the two configurations should not be too high, as the goal is to evolve solutions which work under subtle variation in uncertainty levels; which is also more practical. If the variation were stark, both the bottleneck classification and the design of dispatching rules would be easier. The two subpopulations are then evolved independently (lines 3-12). The pair of best evolved rules from subpopulations at the end of last generation is the final output. Note that the newly proposed training process does not include the decision rule learning. During the test process, we need to classify the current state to decide which dispatching rule to use. To this end, we propose a clustering approach to avoid the need of the decision rule learning.

The K-means clustering component of the method is explained using Algorithm 2. A machine state vector is constructed using the terminal set used in GP3 method (line 5). Initially the set of machine state vectors $\mathcal{H}$ is empty. As the simulation progresses, the machine-state vectors are stored in $\mathcal{H}$.

At the outset, when the simulation is just warming-up and the size of $\mathcal{H}$ is very small we use the following steps. Initially we start simply by using the queue lengths ($MNOPr$, number of remaining operations on a machine) to classify between bottleneck and non-bottleneck machines; higher value of $MNOPr$ implies high level of bottleneck. Once the number of state vectors is greater than 2 ($k = 2$), we are able to apply clustering method but we continue associating (labeling) the centroids to bottleneck and non-bottleneck machines using the feature $MNOPr$; this is done for a small number(10) of jobs during the warm-up period. Thereafter, when $\mathcal{H}$ becomes sufficiently large, for every new pair of centroids their distance is calculated from the pair of centroids obtained in the previous step which are already labeled as bottleneck (C_h) and non-bottleneck (C_l). Based on the distance values, the new centroids are then labeled (line 8).

Once the labeled centroids are obtained, (either using $MNOPr$ initially or using the preceding centroids labels) their distance from the current machine state vector is determined (lines 9-12) and those values are used to decide the dispatching rule from $\{DR_l, DR_h\}$, to be used for sequencing. The preliminary study showed that after the warm-up period of DJSS simulation, during which a fixed number of jobs are ignored for total flow time computation, there are sufficient number of machine-state vectors for the cluster centroids C_l and C_h to be distinct.

Algorithm 2: K-means-clustering approach: GP2-K & CGP2-K

Input:
- Pair of dispatching rules : $\{DR_l, DR_h\}$
- DJSS problem instance.
  - uncertainty configuration.
  - set of machines $\mathcal{M}$.
  - Simulation parameters.

Output: Total flow time :$\mathcal{T}$

1 Set of system state vectors: $\mathcal{H} \leftarrow \emptyset$.
2 Cluster Centroids: $\{C_l, C_h\} \leftarrow \emptyset$.
3 **while** *new jobs arrive* **do**
4 **foreach** $m \in \mathcal{M}$ **do**
5 $\mathcal{F}(m) = [MTWK, MTWKr, MTWKav, MNOPr,$
6 $MNOPw, MUTL]$.
7 **if** $size(\mathcal{H}) > 2$ **then**
8 $\{C_1, C_2\} \leftarrow KmeansCluster(\mathcal{H})$
9 $\{C_l, C_h\} \leftarrow associateClusters(\{C_1, C_2\})$
10 **if** $distance(C_l, \mathcal{F}(m) \leq distance(C_h, \mathcal{F}(m)$ **then**
11 Use DR_l for sequencing on machine m
12 **else**
13 Use DR_h for sequencing on machine m
14 **else**
15 Use DR_l for sequencing on machine m.
16 Add $\mathcal{F}(m)$ to $\mathcal{H}$.
17 **end**
18 Update $\mathcal{T}$.
19 **end**

4.2 Co-evolutionary GP2-K (CGP2-K)

GP2-K is designed to evolve a pair of DRs independently in separate subpopulations and then they are applied to a DJSS problem instance where they interact with each other through sequencing decisions. GP2-K method does not take into account the effect of this interaction between the two dispatching rules. Cooperative co-evolution is a technique which is applicable to a problem with interacting sub-components. Therefore we propose a cooperative co-evolutionary GP2-K i.e. CGP2-K.

In CGP2-K we divide the evolution into two stages as described in Algorithm 3. In the first stage, for some generations the dispatching rules are evolved in separate sub-populations. In this stage, each subpopulation is associated with a specific uncertainty configuration, similar to GP2-K method. In the second stage, which is the *co-evolutionary* stage the fitness is assigned using trials, in which each individual in a sub-population is paired with the best individual from the other subpopulation. In this stage, a single uncertainty configuration is used for both subpopulations which corresponds to lower uncertainty level.

In the lines 3-12 of Algorithm 3, the first stage of the method is presented. The evolution in the subpopulations happen separately using specific uncertainty configurations without any interaction between the two. The co-evolutionary stage is described in lines 13-24. For each generation, the best individuals from each subpopulation is determined (lines 15-16). For calculating the fitness of an individual, the best individual from the other subpopulation

Algorithm 3: Co-evolutionary method - CGP2-K

Input:
- $\mathcal{G}_c$, the generation after which co-evolutions starts.
- $\mathcal{G}_\tau$, total number of generations.
- DJSS training instance ($\mathcal{P}_t$).
  - Simulation parameters
  - $\mathcal{U}_l$ uncertainty configuration (low)
  - $\mathcal{U}_h$ uncertainty configuration (high)

Output: Pair of dispatching rules : $\{DR_l, DR_h\}$

Initialize subpopulations $\mathcal{S}_1, \mathcal{S}_2$
Set $g \leftarrow 0$
/* Stage: 1 */
while $g \leq \mathcal{G}_c$ **do**
  $g \leftarrow g + 1$
  foreach *individual* $\mathcal{I} \in \mathcal{S}_1$ **do**
    assign fitness to $\mathcal{I}$ using DJSS simulation with $\mathcal{U}_l$ config.
  end
  foreach *individual in* $\mathcal{S}_2$ **do**
    assign fitness to $\mathcal{I}$ using DJSS simulation with $\mathcal{U}_h$ config.
  end
  Evolve individuals in $\mathcal{S}_1, \mathcal{S}_2$ using crossover and mutation.
end
/* Stage: 2 - cooperative co-evolution starts */
while $\mathcal{G}_c < g \leq \mathcal{G}_\tau$ **do**
  $g \leftarrow g + 1$
  $\mathcal{I}_1 \leftarrow$ BestIndividual($\mathcal{S}_1$)
  $\mathcal{I}_2 \leftarrow$ BestIndividual($\mathcal{S}_2$)
  foreach *individual* $\mathcal{I} \in \mathcal{S}_1$ **do**
    assign fitness to the pair $\{\mathcal{I}\}$ using Algorithm 2 with $\mathcal{U}_l$ configuration and $\{DR_l, DR_h\} \leftarrow \{\mathcal{I}, \mathcal{I}_2\}$ for problem-instance $\mathcal{P}_t$.
  end
  foreach *individual* $\mathcal{I} \in \mathcal{S}_2$ **do**
    assign fitness to the pair $\{\mathcal{I}\}$ using Algorithm 2 with $\mathcal{U}_l$ configuration and $\{DR_l, DR_h\} \leftarrow \{\mathcal{I}_1, \mathcal{I}\}$ for problem-instance $\mathcal{P}_t$.
  end
  Evolve individuals in $\mathcal{S}_1, \mathcal{S}_2$ using crossover and mutation.
end

is paired with it and to assign the fitness value, this pair is evaluated using the procedure described in Algorithm 2; which is same as what was used for GP2-K. The configuration used for fitness evaluation is $\mathcal{U}_l$, which corresponds to the lower level of uncertainty. After all the generations are complete, the combination of best individuals from the last generation is returned as output. The procedure used for testing is similar to the one used with GP2-K.

5 EXPERIMENT DESIGN

5.1 Simulation Model

We assume that the uncertainty in processing times of the operations is machine specific. Each machine in the job shop is characterized with its own uncertainty distributions. A machine could be associated with more than one uncertainty distribution depending on other factors, e.g. maintenance schedule. This assumption is practical and is a characteristic of imperfect production systems [31]. Therefore, in our model we consider machines with one or more levels of uncertainty.

For an operation $o_{j,i}$ if the processing time without uncertainty is $p_{j,i}$, then the processing time with uncertainty $p'_{j,i}$ follows the relation

$$p'_{j,i} = (1 + \theta_{j,1})p_{j,i},\ \theta_{j,1} \geq 0.$$

Here $\theta_{j,i}$ is the delay ratio which is a measure of the severity of the disturbances. Based on [14], θ in our simulation model should follow exponential distributions.

We use a discrete event simulation system (see Jasima [10]) to generate DJSS problem instances under uncertainty. Similar to many previous works [11, 22] we consider 8 operations per job and 10 machines in each problem. The processing times of the operation are uniformly sampled from [1, 49]. This is a simulation configuration which has been followed in many other studies [11, 22]. The job arrival is assumed to be a Poisson process with $\lambda = 0.85$ [14]. We consider total flow time as the scheduling objective for all our experiments. For every run of the simulation first 500 jobs are ignored (warm-up period) and the total flow time is calculated only over the next 2000 jobs. We conduct 30 independent runs for each method on same training data.

Table 2: Machine uncertainty (scale parameter (β) values of exponential distributions)

	m0	m1	m2	m3	m4	m5	m6	m7	m8	m9
I	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
II	0.1	0.1	0.1	0.1	0.1	0.2	{0.35, 0.1}	{0.35, 0.1}	{0.2, 0.1}	{0.3, 0.1}
III	**0.1**	**0.1**	**0.1**	**0.1**	**0.1**	**0.3**	**{0.6, 0.3}**	**{0.6, 0.3}**	**{0.2, 0.1}**	**{0.3, 0.1}**
IV	0.3	0.3	0.3	0.3	0.3	0.5	{0.8, 0.5}	{0.8, 0.5}	{0.4, 0.3}	{0.5, 0.3}
V	0.3	0.3	0.3	0.3	0.3	0.9	{1.2, 0.75}	{1.2, 0.75}	{0.65, 0.4}	{0.75, 0.4}
VI	**0.1**	**0.1**	**0.1**	**0.35**	**0.35**	**0.35**	**0.65**	**0.65**	**0.65**	**0.65**

Uncertainty Configurations. We have considered *six* uncertainty configurations for the machines which are presented in Table 2. The columns correspond to the machines. Each machine is associated with one or more θ parameter settings which follow exponential distributions. The scale parameter (β) of the associated exponential distributions are given. In practice, a production environment is characterized by varying defect and rework rates [31] which is reflected through the variation in levels of uncertainty. Therefore, some of the machines are associated with a pair of parameter values, e.g. *m6* in configuration *III* is associated with two scale parameters, $\{0.6, 0.3\}$. The duration for which a machine is associated with a specific uncertainty level is sampled from $[1000, 1700]$. For our simulation, we found that this is a sufficient duration for the machine to move through transient to a steady bottleneck level.

We require two levels of uncertainty configurations during training for the methods GP2-K and CGP2-K. These two configurations are shown in bold in Table 2. The configurations *III* and *VI* correspond to the low and high levels respectively. As explained in Section 4.1, the difference between the uncertainty levels of two

configurations is not high. In the configuration VI, for the machines $m6 - m9$ associated with $\beta = 0.65$ the uncertainty level is marginally higher but consistent. Similarly for machines $m3 - m5$, $\beta = 0.35$ which is marginally higher. The different test and train configurations used are summarized in the Table 3. GP1(l) and GP1(h) are the standard GP methods.

Table 3: Training and Test Configurations

	Train	Test
GP1(l)	III	I, II, III, IV, V, VI
GP1(h)	VI	I, II, III, IV, V, VI
GP3	III, VI	I, II, III, IV, V, VI
GP2-K	III, VI	I, II, III, IV, V, VI
CGP2-K	III, VI	I, II, III, IV, V, VI

5.2 Genetic Programming System

For genetic programming, we list the terminal and function sets in Table 4. The function if has three arguments; if first is less than 0, it returns third argument and the second otherwise. For all our methods, we consider a population size of 1000 and the evolution is run over 50 generations. The maximal tree depth for a genetic program is set to 8, crossover rate is 0.85, elitism is 0.5 and mutation is 0.1 [22].

For CGP2-K method (we choose $\mathcal{G}_c = 30$, Algorithm 3) the first 30 generations are used to evolve dispatching rules in independent subpopulations and the next 20 generations are used for co-evolution.

Table 4: Function and Terminal Sets for GP.

Function Set	Meaning
+	Addition
−	Subtraction
∗	Multiplication
/	Protected Division
Max	Maximum
Min	Minimum
If	Conditional
Terminal Set	**Meaning**
PT	Processing time of operation
RO	Remaining operations for job
RJ	Ready time of job
RT	Remaining processing time of job
RM	Ready time of machine
ERC	Ephemeral Random constant

6 RESULTS & DISCUSSION

In this Section, we present the results from our experiments. We compare $five$ methods over six test configurations (Table 3). For each method the solutions are tested over 30 problem instances. The Wilcoxon-rank-sum-test is used to compare the performance of the methods. A significance level of 0.05 is considered.

The results are summarized in Tables 5-9 and Figure 1. Each cell in the tables consists of a triplet detailing the corresponding statistical test result. Consider the first cell of Table 5, $[25 - 5 - 0]$ which should be read as $[win - draw - lose]$. The Table 5 compares the GP1(l) method against the other 4 methods. The cell $[25 - 5 - 0]$ corresponds to the column of test configuration I and row of method GP1(h). It means that the GP1(l) performed significantly better in 25 problem instances, is similar in 5 instances and is significantly poor in 0 instances. Furthermore, for those cells which show significant difference in more than 5 problem instances, color shading is used. The 'green' color is used to show those cases where significant improvement is observed in more than 5 problem instances i.e. $win \geq max(5, loss)$. Similarly 'orange' denotes significantly worse performance, $lose \geq max(5, win)$. For each table, heading mentions the method name and associated training configurations.

In Table 5, we compare the standard GP method, GP1(l) which is trained on configuration III. For the test configuration I, which is characterized by low uncertainty level, it outperforms all methods. It is better than CGP2-K by a thin margin. However as the level of uncertainty increases, its generalization drops rapidly. For test configurations IV to VI, GP1(l) performed noticeably worse than other methods. Refer the cells marked in orange.

Table 5: GP1(l) (Configuration-III)

	I	II	III	IV	V	VI
GP1(h)	**[25-5-0]**	**[18-12-0]**	[0-26-4]	[0-5-25]	[0-0-30]	[0-1-29]
GP3	**[29-1-0]**	**[25-5-0]**	[0-27-3]	[0-8-22]	[0-1-29]	[0-4-26]
GP2-K	**[19-11-0]**	**[18-12-0]**	[0-26-4]	[0-7-23]	[0-0-30]	[0-1-29]
CGP2-K	**[5-25-0]**	[3-27-0]	[0-26-4]	[0-4-26]	[0-0-30]	[0-1-29]

In Table 6, the standard GP method trained on higher uncertainty level corresponding to configuration VI is compared. Its performance is significantly better than GP1(l) for configurations $IV - VI$, as evidenced by the cells marked in green. These cells correspond to configurations with relatively higher uncertainty levels. GP1(h) performs poorly for configurations $I - II$ for GP1(l) and CGP2-K. Refer the colored cells in the first two columns. Its is almost an exact draw with GP2-K across all configurations. For configurations with higher level of uncertainty it is significantly similar to CGP2-K for most test problems.

Table 6: GP1(h) (Configuration-VI)

	I	II	III	IV	V	VI
GP1(l)	[0-5-25]	[0-12-18]	[4-26-0]	**[25-5-0]**	**[30-0-0]**	**[29-1-0]**
GP3	[4-26-0]	[3-27-0]	[0-30-0]	[0-29-1]	[1-27-2]	[3-26-1]
GP2-K	[0-29-1]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]
CGP2-K	[0-21-9]	[0-22-8]	[0-27-3]	[1-27-2]	[4-26-0]	[2-28-0]

In Table 7, the performance of GP3 is presented. It outperforms GP1(l) on configurations $IV - VI$ but is significantly poor for most of the test problems on configuration $I - II$ (refer the orange cells). Apparently, the bottlenecks arising for higher level of uncertainty configuration (configuration VI) have had a dominating influence during training. Consequently, the generalization characteristic of GP3 is poor.

Table 7: GP3 (Configuration III & VI)

	I	II	III	IV	V	VI
GP1(l)	[0-1-29]	[0-5-25]	[3-27-0]	**[22-8-0]**	**[29-1-0]**	**[26-4-0]**
GP1(h)	[0-26-4]	[0-27-3]	[0-30-0]	[1-29-0]	[2-27-1]	[1-26-3]
GP2-K	[0-21-9]	[0-28-2]	[1-29-0]	[1-28-1]	[0-30-0]	[0-29-1]
CGP2-K	[0-3-27]	[0-12-18]	[0-25-5]	[1-29-0]	[1-29-0]	[0-29-1]

In Table 8, the results from GP2-K, which uses clustering method during testing, are shown to be similar to GP1(h), as mentioned earlier. It outperforms GP3 for test configuration *I*, even though they both use same configurations in training. This is because in GP2-K the dispatching rules for bottleneck and non-bottleneck scenarios are learned using specific training configuration and later a clustering method is used to choose the rules during testing against a non-linear GP classifier in the former.

Table 8: GP2-K (Configuration III & VI)

	I	II	III	IV	V	VI
GP1(l)	[0-11-19]	[0-12-18]	[4-26-0]	**[23-7-0]**	**[30-0-0]**	**[29-1-0]**
GP1(h)	[1-29-0]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]	[0-30-0]
GP3	**[9-21-0]**	[2-28-0]	[0-29-1]	[1-28-1]	[0-30-0]	[1-29-0]
CGP2-K	[0-26-4]	[0-21-9]	[0-27-3]	[0-28-2]	[3-27-0]	[2-28-0]

In Table 9, the performance of the co-evolutionary method is shown. Across all the test configurations this method is able to perform well. Though it is marginally poor in test configuration *I* with a very low uncertainty level. It outperforms GP1(h), GP3 and GP2-K on configurations $I - III$ as evidenced by green cells. In the case of test configurations $IV - VI$, the performance CGP2-K outperforms GP1(l) and is almost similar to other methods. This shows that the generalization characteristic of the proposed method is superior to all other methods considered in this work. The co-evolution process takes into account the interactions of the dispatching rules through their sequencing decisions in combination with a more effective clustering method to classify the bottleneck and non-bottleneck dispatching rules.

Table 9: CGP2-K (Configuration III & VI)

	I	II	III	IV	V	VI
GP1(l)	[0-25-5]	[0-27-3]	[4-26-0]	**[26-4-0]**	**[30-0-0]**	**[29-1-0]**
GP1(h)	**[9-21-0]**	**[8-22-0]**	[3-27-0]	[2-27-1]	[0-26-4]	[0-28-2]
GP3	**[27-3-0]**	**[18-12-0]**	**[5-25-0]**	[0-29-1]	[0-29-1]	[1-29-0]
GP2-K	[4-26-0]	**[9-21-0]**	[3-27-0]	[2-28-0]	[0-27-3]	[0-28-2]

In Figure 1, we present boxplots to compare generalization performance of the methods on individual instances. We picked 5 out of the total 30 problem instances under the two test configurations *III* and *VI*. The groups of 4 boxplots correspond to one problem instance each. A boxplot is marked in green if its median is lower than the medians corresponding to all other boxplots in the same group. The order of methods is same as mentioned in the caption. In a large number of the cases, the boxplot corresponding to CGP2-K enjoyed the smallest median value, with respect to these two configurations.

6.1 Analysis

Dispatching Rules. We compare the pair of dispatching rules evolved using CGP2-K. We determine the frequency of the terminals obtained from the 30 runs, which is shown in Figure 2. There is clear difference between the evolved rules with respect to their choice of terminals. The rules which were evolved using a configuration with higher level of uncertainty tend to use the terminals corresponding to the jobs more often. A higher level of uncertainty leads to bottleneck machines, leading to a larger queue length. This

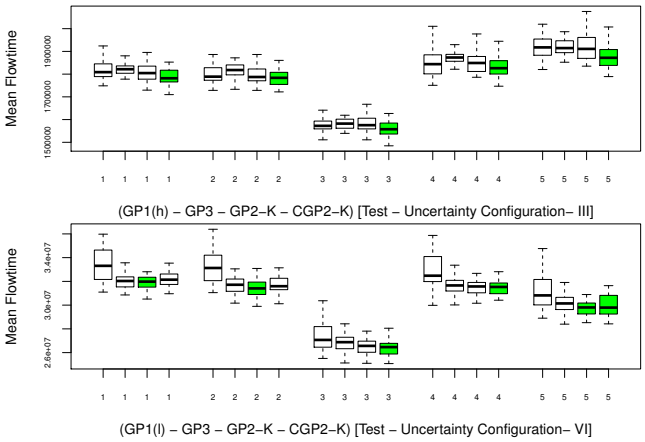

Figure 1: Boxplots: The order of boxplots is same as mentioned in the caption. The result with lowest medians are marked in green.

makes the problem harder. Therefore, the dispatching rule which is evolved on this configuration tends to use more of the terminals which correspond to the job characteristics.

In particular, the two terminals *RO* and *RJ* corresponding to remaining operations for job and ready time of job respectively. For a bottleneck machine, the priority value is expected to be very different for a job with many pending operations compared with a job with fewer pending operations, due to their higher impact on scheduling objective when compared with a non-bottleneck machine.

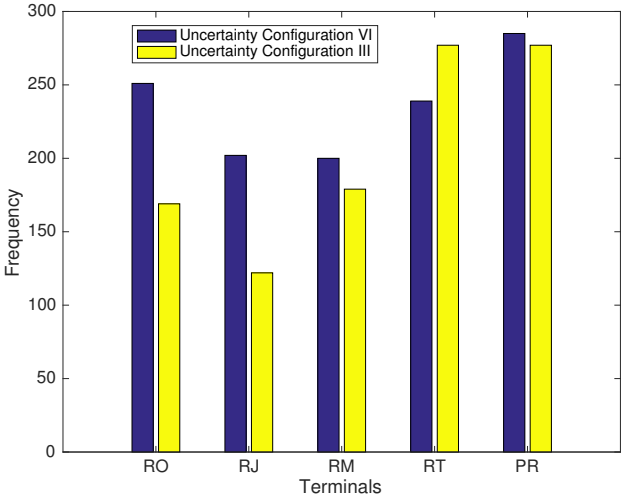

Figure 2: Histogram of Frequency of Terminals

Dispatching Rule 1: CGP2-K - (low)

```
(* (If (If (Min RJ RO) PR (If (- RM PR)
0.580 RM)) (+ (* (Min 0.522 RJ) (*RT PR))
(* PR PR)) RT) (- (Max (- (If RO RT PR)
(Min 0.0837 RM)) (Max (/ RJ PR) (Min
0.0597 RT))) RT))
```

Dispatching Rule 2: CGP2-K - (high)

```
(Min (Min (+ RO RO) (+ (- (- (/ RT RM) (+
(Min PR PR) (* RO PR))) (Max 0.250 RT))
(/ RM (* (* RO PR) (* (Min RJ RT) PR)))))
(- (/ (+ RO (Min RJ RT)) (/ PR RO)) (* (Min
PR PR) (* (Min RJ RT) PR))))
```

An example of one of the best pairs of dispatching rules evolved using CGP2-K is given above. The terminals *RO* and *RJ* are shown in bold.

7 CONCLUSIONS

In this work, we present genetic programming based hyper-heuristic approaches for dynamic job shop scheduling under machine-specific uncertain processing times. We develop methods to classify a machine as bottleneck or non-bottleneck in a dynamic and uncertain environment. Then we develop methods to evolve pair of dispatching rules to minimize total flow time in a DJSS problem under varying levels of uncertainty. The different configurations which cover both low and high levels of uncertainty in processing times were considered. A cooperative co-evolutionary method to evolve pairs of DRs which are able to generalize well across different uncertainty configurations is proposed. We empirically show that the proposed co-evolutionary approach performs well across all the scenarios when compared to standard GP and GP3 [12] methods. Finally, we analyze the proposed methods to get more insights into our results and show that features of the evolved rules correlate with their expected role in sequencing the jobs.

In our future work, we will consider multiple scheduling objectives and different types of uncertainties e.g. rush arrival of jobs, variation in due dates etc. We would also like to combine job specific processing time uncertainty in our current simulation model.

8 ACKNOWLEDGEMENT

This work is supported in part by the Marsden Fund of New Zealand (VUW1209, VUW1509 and VUW1614), administrated by Royal Society of New Zealand.

REFERENCES

[1] Joseph Adams, Egon Balas, and Daniel Zawack. 1988. The shifting bottleneck procedure for job shop scheduling. *Management science* 34, 3 (1988), 391–401.
[2] J Balasubramanian and IE Grossmann. 2004. Approximation to multistage stochastic optimization in multiperiod batch plant scheduling under demand uncertainty. *Industrial & engineering chemistry research* 43, 14 (2004), 3695–3713.
[3] Jürgen Branke, Torsten Hildebrandt, and Bernd Scholz-Reiter. 2015. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary computation* 23, 2 (2015), 249–277.
[4] Juergen Branke, Su Nguyen, Christoph W Pickardt, and Mengjie Zhang. 2016. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation* 20, 1 (2016), 110–124.
[5] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. 2003. Hyper-heuristics: An emerging direction in modern search technology. *International series in operations research and management science* (2003), 457–474.
[6] Edmund K Burke, Mathew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan, and John R Woodward. 2009. Exploring hyper-heuristic methodologies with genetic programming. In *Computational intelligence*. Springer, 177–201.
[7] Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering* 36, 2 (1999), 343–364.
[8] Philippe Fortemps. 1997. Jobshop scheduling with imprecise durations: a fuzzy approach. *IEEE Transactions on Fuzzy Systems* 5, 4 (1997), 557–569.
[9] Kai Zhou Gao, Ponnuthurai Nagaratnam Suganthan, Mehmet Fatih Tasgetiren, Quan Ke Pan, and Qiang Qiang Sun. 2015. Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion. *Computers & Industrial Engineering* 90 (2015), 107–117.
[10] T Hildebrandt. 2012. Jasima – An Efficient Java Simulator for Manufacturing and Logistics. *http://code.google.com/p/jasima* (2012).
[11] Rachel Hunt, Mark Johnston, and Mengjie Zhang. 2014. Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 927–934.
[12] Domagoj Jakobović and Leo Budin. 2006. Dynamic scheduling with genetic programming. In *Genetic Programming*. Springer, 73–84.
[13] Domagoj Jakobović, Leonardo Jelenković, and Leo Budin. 2007. Genetic programming heuristics for multiple machine scheduling. In *Genetic Programming*. Springer, 321–330.
[14] Deepak Karunakaran, Yi Mei, Gang Chen, and Mengjie Zhang. 2016. Dynamic Job Shop Scheduling Under Uncertainty Using Genetic Programming. *Intelligent and Evolutionary Systems* (2016), 195.
[15] Yeo Keun Kim, Kitae Park, and Jesuk Ko. 2003. A symbiotic evolutionary algorithm for the integration of process planning and job shop scheduling. *Computers & operations research* 30, 8 (2003), 1151–1171.
[16] Panos Kouvelis and Gang Yu. 2013. *Robust discrete optimization and its applications*. Vol. 14. Springer Science & Business Media.
[17] Stephen R Lawrence and Edward C Sewell. 1997. Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. *Journal of Operations Management* 15, 1 (1997), 71–82.
[18] Zukui Li and Marianthi Ierapetritou. 2008. Process scheduling under uncertainty: Review and challenges. *Computers & Chemical Engineering* 32, 4 (2008), 715–727.
[19] Yi Mei, Su Nguyen, and Mengjie Zhang. 2017. Evolving Time-Invariant Dispatching Rules in Job Shop Scheduling with Genetic Programming. In *European Conference on Genetic Programming*. Springer, 147–163.
[20] Su Nguyen, Yi Mei, and Mengjie Zhang. 2017. Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems* (2017), 1–26.
[21] Su Nguyen, Mengjie Zhang, Michael Johnston, and Kay Chen Tan. 2013. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *Evolutionary Computation, IEEE Transactions on* 17, 5 (2013), 621–639.
[22] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. 2014. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation* 18, 2 (2014), 193–208.
[23] Su Nguyen, Mengjie Zhang, Mark Johnston, and Kay Chen Tan. 2015. Automatic programming via iterated local search for dynamic job shop scheduling. *IEEE transactions on cybernetics* 45, 1 (2015), 1–14.
[24] Su Nguyen, Mengjie Zhang, and Kay Chen Tan. 2016. Surrogate-Assisted Genetic Programming With Simplified Models for Automated Design of Dispatching Rules. *IEEE Transactions on Cybernetics* (2016).
[25] John Park, Yi Mei, Su Nguyen, Gang Chen, Mark Johnston, and Mengjie Zhang. 2016. *Genetic Programming Based Hyper-heuristics for Dynamic Job Shop Scheduling: Cooperative Coevolutionary Approaches*. Springer International Publishing, Cham, 115–132. DOI: http://dx.doi.org/10.1007/978-3-319-30668-1_8
[26] John Park, Su Nguyen, Mengjie Zhang, and Mark Johnston. 2015. Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In *European Conference on Genetic Programming*. Springer, 92–104.
[27] Bernard Penz, Christophe Rapine, and Denis Trystram. 2001. Sensitivity analysis of scheduling algorithms. *European Journal of Operational Research* 134, 3 (2001), 606–615.
[28] Michael Pinedo and Marcos Singer. 1999. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics* 46, 1 (1999), 1–17.
[29] Michael L Pinedo. 2012. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media.
[30] Mitchell A Potter and Kenneth A De Jong. 2000. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary computation* 8, 1 (2000), 1–29.
[31] Meir J Rosenblatt and Hau L Lee. 1986. Economic production cycles with imperfect production processes. *IIE transactions* 18, 1 (1986), 48–55.
[32] Jun-Qiang Wang, Jian Chen, Yingqian Zhang, and George Q Huang. 2016. Schedule-based execution bottleneck identification in a job shop. *Computers & Industrial Engineering* 98 (2016), 308–322.
[33] R Paul Wiegand. 2003. *An analysis of cooperative coevolutionary algorithms*. Ph.D. Dissertation. George Mason University.
[34] Rui Zhang and Cheng Wu. 2009. Bottleneck identification procedures for the job shop scheduling problem with applications to genetic algorithms. *The International Journal of Advanced Manufacturing Technology* 42, 11-12 (2009), 1153–1164.