Active Sampling for Dynamic Job Shop Scheduling using Genetic Programming

Deepak Karunakaran, Yi Mei, Gang Chen and Mengjie Zhang
School of Engineering and Computer Science,
Victoria University of Wellington, PO Box 600, Wellington, New Zealand.
Email: {deepak.karunakaran, yi.mei, aaron.chen, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Dynamic job shop scheduling is an important but difficult problem in manufacturing systems which becomes complex particularly in uncertain environments with varying shop scenarios. Genetic programming based hyper-heuristics (GPHH) have been a successful approach for dynamic job shop scheduling (DJSS) problems by enabling the automated design of dispatching rules for DJSS problems. GPHH is a computationally intensive and time consuming approach. Furthermore, when complex shop scenarios are considered, it requires a large number of training instances. When faced with multiple shop scenarios and a large number of problem instances, identifying good training instances to evolve dispatching rules which perform well over diverse scenarios is of vital importance though challenging. Essentially this requires the tackling of exploration versus exploitation trade-off. To address this challenge, we propose a new framework for GPHH which incorporates active sampling of good training instances during evolutionary process. We propose a sampling algorithm based on the $\epsilon-$greedy method to evolve a set of dispatching rules. Through our experiments, we demonstrate the ability of our framework to efficiently identify useful training instances toward evolving dispatching rules which outperform the existing training methods.**

***Index Terms*—scheduling, active learning, genetic programming, dispatching rules.**

I. Introduction

Job shop scheduling (JSS) is an important activity in manufacturing industries. JSS is basically an assignment of the order of jobs to a set of machines to optimize the desired goal(s), e.g., makespan, tardiness, etc. When the jobs are continuously arriving at the shop, the problem becomes that of dynamic job shop scheduling (DJSS). In most of the existing works, the parameters associated with the jobs viz., processing times, due dates, etc., are considered as constants. However, in practice, due to events like machine breakdown, operator's error, etc., these parameters are uncertain. In fact, with increase in uncertainty the JSS problem becomes more difficult [1].

Dispatching rules are widely used for JSS problems [2]. Especially, under uncertain environments (e.g. jobs with uncertain processing times) they have shown good success [3]. However, designing dispatching rules requires expertise and rigorous experimentation. Recently, genetic programming based hyper-heuristic (GPHH) techniques have been proposed to automate the design of these rules [4]. GP is characterized by flexible representations and good search ability which are leveraged by GPHH techniques to evolve rules for complex shop conditions [5] toward optimizing multiple objectives [6].

Many existing works aim to design a single general rule. However, it is difficult to design a single general rule for all shop conditions, particularly under uncertain environment, with events like machine breakdown etc., when the shop scenarios become even more complex [7].

For example, in printing industry, more broadly also known as document management services [8], a large number of shop scenarios arise due to the dynamic environment characterized by events like break downs, operator mistakes, spurt in arrival of high priority jobs, etc. The resources in a printing shop are printers, cutters, shrink wrapper, binder, etc. In a dynamic printing shop, different jobs have different patterns, e.g. short jobs with short deadlines, or recurring jobs with specific characteristics like marketing pamphlets [8]. These get disrupted when jobs with a different pattern arrive at the same time, e.g., a large number of jobs with long patterns (jobs consisting of operations with long processing times) when shorter jobs are nearing deadline. This leads to uncertainty in the shop and adversely affects the scheduling objective(s). Rai et al. [8] proposes using dynamic cells (a group of associated jobs and resources) which essentially change the structure of printing shop as it groups together similar jobs and also groups together the required equipment (machines) dynamically, according to changing shop patterns (scenarios). The idea of dynamic cells is born out of the need to recognize and deal with a large number of shop scenarios arising in the dynamic environment. We can clearly see that this example highlights the importance of recognizing the large number of shop scenarios in the dynamic shop and developing methods to deal with each scenario separately.

Designing a universal dispatching rule which can deal with all such scenarios is not practical. Instead we need dispatching rules which are designed for specific shop scenarios. For that reason some recent works developed methods to evolve a *set* of scenario specific dispatching rules [7], [9] which are expected to address specific scenarios. For complex shop environments, automatically designing scenario-specific rules using GPHH is more difficult because even though GPHH has been successful in automating the design of dispatching rules, the associated computational cost (hardware) and time are high.

Furthermore, to evolve good rules for the high number of shop scenarios, a large training set must be considered. It is impractical for GPHH to consider each and every training instance during the evolutionary process. Moreover, not all the training instances are equally useful towards evolving good rules. For example, a training instance with *tight* due dates may be more useful for evolving rules towards solving JSS problems of minimizing tardiness because it is a "*harder*" problem instance. This raises the question as to how to balance between the exploring of the instance space for more useful training instances and exploiting the identified instances.

Active learning [10] is a concept based on the idea that a machine learning algorithm will perform better if it has the choice to select the training instances to learn from. Over the recent years this idea has inspired many algorithms in semi-supervised learning. Recently, in a related work [11], Sharma et al., presented algorithms based on multi-armed bandits which use active learning principles to achieve multi-task learning by sampling the harder tasks more often than the easier ones. Their goal is to develop agents for playing Atari games. Recently, evolutionary multi-tasking [12] has emerged as a promising area of research for solving optimization problems. Evolutionary multitasking attempts to solve multiple optimization problems simultaneously by exploiting the implicit parallelism of population-based search. In the context of evolving multiple scenario-specific rules the techniques presented in multi-tasking learning are thus highly useful.

Inspired by the concepts of evolutionary multitasking and the success of active learning techniques, we are motivated to solve the issues with GPHH which we described above. Integrating active learning methods into GPHH has some important challenges. Firstly, the current GPHH framework has no facility to actively sample training instances while evolving dispatching rules. Secondly, there is no existing method to determine the ability of training instances in promoting the evolution of scenario-specific rule or to evaluate how good a training instance represents a complex shop scenario. Without addressing these two challenges, the techniques from active learning cannot be employed. Furthermore, since GPHH is already computationally intensive, the integration of these techniques should consider aspects of computational efficiency as well.

A. Goals

The goal of this work is to develop a framework for GPHH which incorporates active sampling of potentially useful DJSS problem instances from a large instance space in order to evolve multiple scenario-specific dispatching rules. We present an algorithm based on the ϵ-greedy method which tackles the exploration versus exploitation trade-off while evolving a set of dispatching rules.

More specifically, the goal of this work is to develop an active sampling algorithm to sample useful training instances from the instance space toward evolving a set of scenario-specific dispatching rules. To this end, we intend to achieve following sub-goals:(1) A GPHH framework which incorporates active sampling of potentially more useful DJSS instances. (2) An ϵ-greedy method for GPHH for sampling potentially useful DJSS problem instances toward evolving a set of dispatching rules during the training process, while tackling the exploration versus exploitation trade-off along with the computational issues. (3) A procedure to identify a set of dispatching rules which are each good for shop scenarios pertaining to specific conditions.

II. Related Work

GPHH has been shown to be a good hyper-heuristic method and has been shown to outperform other approaches [13]. The idea of using multi-population systems to evolve genetic programs with diverse characteristics has already been presented in [2]. Jakobovic et al. [4] considered evolving a pair of dispatching rules for bottleneck machines and non-bottleneck machines. Similarly, in [7], Karunakaran et al., classified the machines in the shop based on their bottleneck levels arising due to varying uncertainty in processing times and evolved a pair of dispatching rules using cooperative co-evolution. The rules are associated with the machines with distinct characteristics and are jointly utilized for high-performing dynamic job shop scheduling. Thus both the rules are employed together for solving every problem instance.

Evolutionary multitasking [12] attempts to solve multiple optimization problems simultaneously by exploiting the implicit parallelism of population-based search. By utilizing the similarities and differences across the tasks, they are solved simultaneously. Multitasking facilitates the implicit knowledge transfer between diverse tasks and helps achieving optimization across the problem domains. In hindsight, these ideas essentially highlight the research directions which focus on dividing a global problem into multiple tasks and use more specific task oriented information to solve them. In our context, this further reinforces the importance of considering a large number of shop scenarios and designing scenario-specific rules.

Aimed at properly balancing exploration and exploitation, the CBCC3 algorithm by [14] adopted a co-evolution framework for large-scale optimization. The algorithm explicitly explored the advantage of adaptively allocating resources to different components of a modular optimization problem and achieved prominent success. Although the problems considered in their work and the proposed methods are quite different, their goal is closely related to our research.

In another related work [15], Karunakaran et al., developed sampling heuristics based on the Pareto fitness for a multi-objective DJSS problem using an island model. They addressed issues that are similar to the main focus of this paper, but in a clearly different context with multiple conflicting objectives. However, their ideas for feature extraction from the problem instances have been adopted in our proposed work.

III. Proposed Approach

In order to achieve the goals in this paper, we need to incorporate some new facets into the current GPHH framework to enable the active sampling methods. In particular, the GPHH approach must be enabled to evaluate the DJSS problem instances which were sampled to train a scenario-specific dispatching rule.

Before we describe our new framework, we need a method to associate the DJSS training instances with the shop scenarios which they represent. Since our aim is to enable the GPHH to evolve dispatching rules each specific to different scenarios, we should first be able to map the DJSS instances to the complex shop scenarios. Consequently, for the GPHH system to train, it should be provided with groups or clusters of DJSS problem instances where each cluster of problem instances corresponds to specific shop scenarios. Therefore, firstly we describe a feature extraction and clustering method for the DJSS problem instances using the procedure described in [15].

A. Clustering of DJSS Problem Instances

Referring back to our previous example of dynamic job shops from printing industry, the shop scenarios are defined mainly by the characteristics of the arriving jobs. These characteristics pertain to #operations, processing time of these operations, their due date, etc. The other factors which influence a shop scenario are dynamic events like machine break downs, variability in set-up times, etc., which lead to uncertainty in shop parameters. Since processing time is an important job parameter influencing most of the objectives, we consider the uncertainty in the processing times, which in essence captures the effect of aforementioned dynamic events.

TABLE I
JOB FEATURES

Feature	Description
#operations	number of operations per job.
p	estimated processing time of the job.
Δ^p	$\frac{p'}{p}$, p' is the actual processing time with uncertainty.
due date factor (ddf)	$\frac{(\delta_{duedate}-\delta_{reldate})}{p'}$; where $\delta_{duedate}$ is the due date and $\delta_{reldate}$ is the release date

Now, with this background, we extract features from the DJSS problem instances. Firstly, the basic features for each job are extracted as described in Table I. These features, as explained above, are closely related to the shop scenarios. We would also like to mention that similar feature extraction methods have been employed by [16] though for static scheduling problems and also their work is not related to GPHH. The estimated processing time p is the expected processing time which is used by the dispatching rule to make sequencing decisions. The realized processing time p' is the *actual* processing time obtained after the job is completed on the machines. Their ratio Δ^p, in the Table I is the delay ratio. The number of operations, due date factor and the processing time are the parameters which help in defining the characteristics of a job which in turn influences a shop scenario.

Once the basic features for each of the jobs in a DJSS problem instance are extracted, we need to create a feature vector for the dynamic JSS problem. In order to do that, firstly each of the basic features for each of the jobs is aggregated. Then, the first, second and third quartiles of each aggregate are calculated to form a 12-dimensional feature vector characterizing each problem instance. We illustrate this feature extraction methodology using an example below.

Consider an example of a DJSS problem instance with just 10 jobs

$$\{j_1, j_2, j_3, \dots j_{10}\}$$

For each job, the features described in Table I are calculated and aggregated e.g., for processing time the aggregated feature values are:

$$\{p_1, p_2, p_3, \dots p_{10}\}$$

Then for each feature aggregate, the quartiles are calculated. The feature vector of the DJSS instance is of the form

$$\{\#ops_{Q1}, \#ops_{Q2}, \#ops_{Q3}, p_{Q1}, p_{Q2}, p_{Q3}, \\ \Delta^p_{Q1}, \Delta^p_{Q2}, \Delta^p_{Q3}, ddf_{Q1}, ddf_{Q2}, ddf_{Q3}\}$$

After extracting the feature vectors from each of the DJSS training instances, they are clustered to form groups of similar problem instances. Since we started extracting features from each job and then aggregated them for the jobs arriving at a shop, we expect that the combination of job characteristics arriving at the shop is reflected in our aggregated features. Therefore, after clustering the DJSS instances, we should expect that the problem instances corresponding to a cluster pertain to similar shop scenarios.

Now we propose our new GPHH framework.

B. GPHH Framework Using Active Sampling

Before delving into the details of our proposed framework, we briefly outline the commonly used GPHH approach. In the most popular GPHH approach, the GP uses the same set of DJSS training instances all the time. The total number of the DJSS training instances used is quite low. If G is the total number of generations, then usually it is just a small multiple of G. For example, Nguyen et al. [2] considers four DJSS training instances per generation. At the end of the algorithm, it is common for the best dispatching rule to be considered as the final result e.g. [17], [18]. Both these aspects are different from the proposed framework for GPHH.

The proposed GPHH framework is explained using the flowchart in Fig. 1. The framework consists of three main components, namely: training, testing and validation. These are shown using blue boxes in the flowchart.

Before the GPHH procedure starts, we divide the set of problem instances ($\mathcal{X}$) evenly into *three* sets: training set $\mathcal{S}$, validation set $\mathcal{V}$ and test set $\mathcal{T}$. The notation is defined in Table II. Using the methodology described above, we extract features from each of the problem instances from the three sets and cluster them. These three sets are used for training, validation and testing respectively, as shown in the flowchart.

The training step consists of many evolutionary (training) epochs. In the parlance of evolutionary computation, an epoch, γ, consists of a fixed number of generations. Thus if G is the total number of generations, then the number of epochs $\mathcal{E}$ is

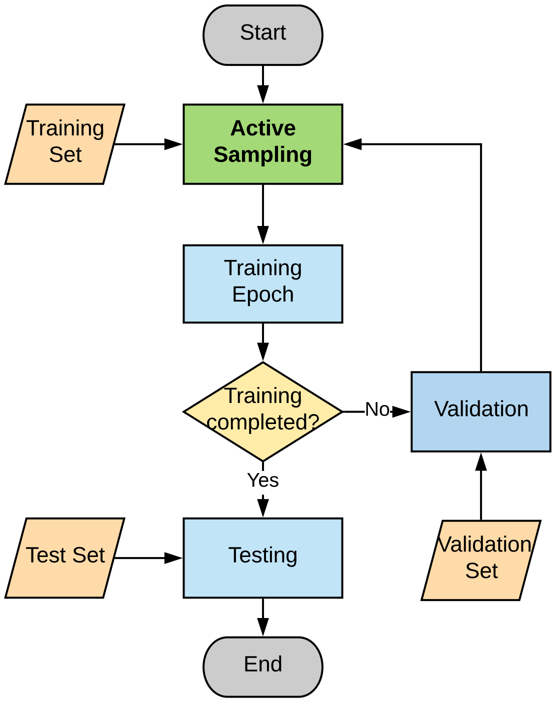

Fig. 1. Proposed GPHH framework using active sampling.

TABLE II
NOTATION

Symbol	Definition
$\mathcal{X}$	set of all DJSS problem instances
$\mathcal{S}$,$\mathcal{V}$,$\mathcal{T}$	training, validation and test sets respectively.
$\mathcal{S}_c$,$\mathcal{V}_c$,$\mathcal{T}_c$	denote clustered sets.
N_v	number of instances in each cluster of $\mathcal{V}$
$p(V_k, i)$	ith problem instance from the cluster v_k. similarly for $\mathcal{S}$ and $\mathcal{T}$.
G	total number of generations
γ	number of generations for which a subpopulation is evolved
$\mathcal{E}$	number of epochs $\mathcal{E} = G\%\gamma$
N_g	number of gens. for which all subpops. are evolved initially.
$\mathcal{R}_k$	rank order of all dispatching rules for a cluster v_k
$\bar{r}_{\omega_i}$	average rank of a dispatching rule over all clusters

$\lfloor G/\gamma \rfloor$, where $\lfloor\,\rfloor$ is the floor function. At the end of each epoch, a set of dispatching rules is obtained.

As seen in the flowchart, a training epoch is preceded by the active sampling procedure. The goal of active sampling is to effectively sample those DJSS instances which represent the shop scenarios which have the ability to promote evolution of good rules. For the purpose of quantifying this ability of the DJSS instances, the validation step is considered. After the completion of an epoch, which considers the specific set of sampled DJSS instances for training, new dispatching rules can be obtained corresponding to specific DJSS training instances. By evaluating these dispatching rules on the validation set, the usefulness of (quality of) DJSS instances can be indirectly evaluated. The validation procedure depends on the active sampling algorithm, therefore, we explain it in detail later. For now, consider that the validation step is able to quantify the quality of DJSS instances. The active sampling uses this information to sample better DJSS instances for the next epoch.

After the completion of all the training epochs, the final set of dispatching rules is obtained. Now the question is how to determine which dispatching rule is to be used when a test DJSS instance is presented. The testing component of the framework, introduced in the next section, is designed to address this question. Having outlined our GPHH framework with a flowchart, we explain in more detail our active sampling method.

Algorithm 1: GPHH with Active Sampling based on ϵ-*greedy* approach

```
Input: $\mathcal{S}_c$,$\mathcal{V}_c$
Output: Set of dispatching rules associated with clusters in $\mathcal{S}_c$
1  Create $n$ subpopulations, where $n$ is the number of clusters in $\mathcal{S}_c$ .
2  for $k \leftarrow 1 : n$ do
3      for $g \leftarrow 1 : N_G$ do
4          Sample an instance $\mathcal{I} \in s_k$.
5          Run $g^{th}$ iteration of GPHH using $\mathcal{I}$.
6      end
7  end
8  Collect the best dispatching rule from each subpopulation: $\{\omega_1, \omega_2, \ldots, \omega_n\}$.
9  for $epoch \leftarrow 1 : \mathcal{E}$ do
10     Randomly select a number $eps \in [0, 1]$
11     if $eps \leq \epsilon$ then
12         Rank the dispatching rules $\{\omega_1, \omega_2, \ldots, \omega_n\}$ using Algorithm 2, which outputs avg. rank $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$
13         Determine subpopulation $k_{best}$ whose dispatching rule has best average rank in $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$.
14     end
15     else
16         $k_{best}$ is selected randomly.
17     end
18     for $g \leftarrow 1 : \gamma$ do
19         Sample an instance $\mathcal{I} \in s_{k_{best}}$
20         Run $g^{th}$ generation of GPHH using $\mathcal{I}$ for the $K_{best}$th subpopulation.
21     end
22     Increment $\epsilon \leftarrow \epsilon + \delta_\epsilon$
23 end
24 Collect the best dispatching rule from each subpopulation: $\{\omega_1, \omega_2, \ldots, \omega_n\}$.
25 Output the final pairs of dispatching rules and associated clusters. $\{(\omega_1, s_1), (\omega_2, s_2), \ldots, (\omega_n, s_n)\}$
```

C. GPHH with Active Sampling using ϵ-greedy strategy

The purpose of the active sampling method is to tackle the exploration vs. exploitation dilemma. This dilemma has been extensively studied in the multi-armed bandit (MAB) [19] framework. In a typical multi-armed bandit problem, an agent is modeled which attempts to acquire new knowledge (explores) while simultaneously optimizing the decisions based on existing knowledge (exploits). The agent tries to balance these competing tasks in order to maximize the payoff over the considered period of time. In the MAB framework, the agent chooses an *action* from a discrete set of actions (arms) for which it gets a *reward*. In a sequence of *trials*, the agent performs actions and gets rewards. In order to quantify the performance of agent, a notion of *regret* is used which is the difference between the collective rewards of the agent and the reward of an optimal strategy.

The ϵ-greedy [20] is a widely used heuristic in the MAB framework. It is very simple to use and in many cases it outperforms more sophisticated methods [20]. It is not only efficient but outperforms more advanced methods like UCB (Upper confidence bound) based algorithms on most bandit problem instances [20]. Considering the fact that efficiency of our training approach is an important criterion, the simple and efficient ϵ-greedy heuristic is as a sound choice for our active sampling method.

ϵ-greedy method is used for sequential decision problems. In each round of decision making, it chooses the arm with the highest empirical mean reward with a probability of ϵ and a random arm with a probability of $1-\epsilon$. Usually, ϵ increases with every round. With this background, we explain the active sampling method based on the ϵ-greedy method using Algorithm 1.

1) Training: In lines 1-7 of the Algorithm 1, for each cluster in $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$, a sub-population is created (using the usual initialization methods used for GP) and after GP evolution for N_G generations, n dispatching rules, one pertaining to each cluster are obtained, $\{\omega_1, \omega_2, \ldots, \omega_n\}$ (line 8).

The rest of the GPHH procedure comprises of epochs. For each epoch, consisting of γ generations, the ϵ-greedy strategy either picks the 'best' sub population for evolution with a probability of ϵ or chooses a random sub-population with a probability of $1-\epsilon$ (line 11). By *best* sub population, we actually mean the cluster of DJSS instances which shows the maximum promise in evolution of an effective dispatching rule. This is identified using the validation step in line 12.

Using the ϵ-$greedy$ approach, the most promising sub population is given a chance for evolution with a higher probability (ϵ). This is shown in lines 10-17. We rely on the validation set to quantify the quality of a sub population. Algorithm 2 is employed for this purpose (line 12) which is explained below. The validation procedure ranks the dispatching rules according to their performance on the validation set. The sub-population k_{best} (line 16) which corresponds to the evolved dispatching rule with the highest rank assigned by validation is used for evolution (line 18-21). This sub population using DJSS training instances from the associated cluster. The value of ϵ is incremented at the end of every epoch till it reaches the value of 1 in the end (line 22). We can observe that our active sampling method identifies those clusters that serve as better candidates for training the dispatching rules and expends comparatively more computational resources while using them for training.

Algorithm 2: Validation for Active Sampling using ϵ-greedy

```
Input: $\mathcal{V}_c = \{v_1, v_2, \ldots, v_m\}, \mathcal{D} = \{\omega_1, \omega_2, \ldots, \omega_n\}$
Output: Avg. rank of dispatching rules $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$
1  for $k \leftarrow 1 : m$ do
2      $\mathcal{R}_k \leftarrow \emptyset$
3      $\Sigma_k \leftarrow \emptyset$
4      for each dispatching rule $\omega \in \mathcal{D}$ do
5          $Sum_w^k \leftarrow 0$
6          for each problem instance $p(v_k, i) \in v_k$ do
7              $Sum_w^k \leftarrow$ Sum + DJSSsimulate($p(v_k, i), \omega$)
8          end
9          $\Sigma_k \leftarrow \{\Sigma_k, Sum_w^k\}$
10     end
11     $\mathcal{R}_k \leftarrow$ Sorting($\Sigma_k$)
12 end
13 for $z \leftarrow 1 : n$ do
14     $r \leftarrow 0$
15     for $k \leftarrow 1 : n$ do
16         $r \leftarrow r$ + Rank of $\omega_k$ in $\mathcal{R}_z$
17     end
18     $\omega_z^{\bar{r}} \leftarrow r/|\mathcal{V}_c|$
19 end
20 return $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$
```

2) Validation: The aim of the validation step is to evaluate the efficacy of the training cluster in evolution of effective dispatching rules. But the efficacy can be measured only after a dispatching rule has been evolved using that cluster. Since the requirement is to compare the clusters among each other, we use a separate validation set which also consists of clusters of DJSS problem instances. By ranking the evolved dispatching rules on each of the clusters, and determining the average rank, we are therefore able to quantitatively compare the efficacy of training clusters. The validation step of our GPHH framework is explained in Algorithm 2.

The input to this algorithm is the clustered set $\mathcal{V}_c = \{v_1, v_2, \ldots, v_m\}$ and the set of best dispatching rules $\{\omega_1, \omega_2, \ldots, \omega_n\}$, from each sub population. In order to rank these dispatching rules, each of them is evaluated on a set of problem instances from the clusters. In lines 1-11 of the algorithm, on each cluster, for each dispatching rule the sum of the objective values resulting from the evaluation is determined (line 7). $p(v_k, i)$ denotes a DJSS problem instance in the cluster v_k. Thus for each cluster, the dispatching rules can be sorted based on the calculated summation values (line 11). Thus each dispatching rule has a rank on each cluster. In lines 13-19 of the algorithm, the average rank of each dispatching rule across the all the clusters is obtained as $\{\omega_1^{\bar{r}} \ldots, \omega_n^{\bar{r}}\}$.

At the completion of training step of the GPHH procedure, the output comprises of a set of dispatching rules each associated with a cluster from the training set $\{(\omega_1, s_1), (\omega_2, s_2), \ldots, (\omega_n, s_n)\}$, line 25 of Algorithm 1. Now we explain the testing component of our proposed framework using Algorithm 3.

```
Algorithm 3: Testing
   Input: Test Instance $\mathcal{I}_t$ and
          $\{(\omega_1, s_1), (\omega_2, s_2), \ldots, (\omega_n, s_n)\}$
   Output: Objective Value : $TWT$
 1 Schedule the first $N_j$ jobs in $\mathcal{I}_t$ using the dispatching
     rule $SPT$
 2 Determine the feature vector $\hat{f}_t$ for $N_j$ jobs in $\mathcal{I}_t$
 3 $d_{min} \leftarrow \infty$
 4 $\omega_t \leftarrow \emptyset$
 5 for $k \leftarrow 1 : n$ do
 6     $f_k \leftarrow$ feature vector of centroid of $s_k$
 7     if distance$(f_k, \hat{f}_t) < d_{min}$ then
 8         $\omega_t \leftarrow \omega_k$
 9         $d_{min} \leftarrow$ distance$(f_k, \hat{f}_t)$
10     end
11 end
12 Use the dispatching rule $\omega_t$ for remaining jobs in $\mathcal{I}_t$.
13 Evaluate $TWT$
14 Return $TWT$
```

3) Testing: Our evolved solution is a set of dispatching rules associated with specific clusters of DJSS instances. When a new test instance $\mathcal{I}_t$ is presented, we must select the most suitable dispatching rule for scheduling. Essentially, we want to determine the association between the shop scenario and corresponding test instance.

We propose to calculate the feature vector of the test instance. This is described in Algorithm 3. The first few jobs in a problem instance are usually ignored for calculation of the scheduling objective, as it is considered as the warm-up period. So we schedule this set of jobs using a standard rule (SPT i.e., shortest processing time) to determine the feature vector of the test instance. This is shown in the lines 1-2 of the algorithm. The feature vectors are extracted using the procedure developed in Section III-A.

After the feature extraction, the Euclidean distance of this feature vector from feature vectors of the centroids of the clusters in $\mathcal{S}_c$ is evaluated. Then the dispathching rule corresponding to the cluster with minimum distances is chosen. This is shown in the lines 5-11 of the algorithm.

If we consider the computational complexity of GPHH, then it is determined by the number of simulations of the DJSS environment required to determine the fitness of each individual in the population, which is equivalent to

$$N_S \times G$$

where N_S is the population size and G is the number of generations.

For the proposed approach, the number of simulations are incremented by the those required in the validation stage. Therefore, the computational cost has three components. Firstly, the simulations required as in lines 2-7 of Algorithm 1, secondly the simulations corresponding to the lines 18-21 pertaining to the evaluations within an epoch and the third component is that of validation (line 12). The summation of the three components leads to

$$(N \times N_G) + (\mathcal{E} \times \gamma \times \frac{N}{n}) + (\mathcal{E} \times \sum_k |v_k| + \mathcal{D} \times \sum_k |v_k|)$$

where $|v_k|$ is the number of problem instances in the validation set v_k and N is the population size. Moreover, $(\mathcal{E} \times \sum_k |v_k|)$ is arrived at by considering the fact that only one dispatching rule in $\mathcal{D}$ (see Algorithm 2) is changed at the end of an epoch.

It is important that the computational resource utilized by our proposed approach does not exceed that of the standard GPHH. This can be easily achieved by tuning the values of $\mathcal{E}$ and N.

IV. Experiment Design

Following several recent research works [17], [21], in our dynamic job shop scheduling experiments we consider the jobs to be arriving continuously to the shop from a Poisson process ($\lambda = 0.85$) [2]. The n_j operations of job j follow the constraint that their order of processing must follow a predefined route e.g. $(o_{j,1} \rightarrow o_{j,2} \rightarrow, \ldots, o_{j,n_j})$. No preemption, no recirculation of jobs, no machine failure and zero transit times are the conditions and assumptions of the problems considered. Furthermore, there is a one-to-one mapping between operations and machines.

TABLE III
DJSS SIMULATION PARAMETERS

Simulation paramter	Values
Processing time range	[0,49],[20,69]
Uncertainty scale (β)	$\{0.2, 0.4\}$
Due date tightness	$\{1.5, 2.5\}$
# operations per job	$\{8, 10\}$

We consider total weighted tardiness, which is frequently considered in literature [2], as the scheduling objective.

$$TWT = \Sigma w_j \times \max(C_j - d_j, 0),$$

where C_j is the completion time, d_j is the due date and w_j are the weights of a job j.

In order to simulate the jobs, we use a discrete event simulation system (Jasima) [22]. For each simulation, 500 jobs in the beginning are considered as warm-up and the objective value is determined for the following 2000 jobs which is in line with other research works like [17], [23]. Consequently, $N_j = 500$, is used in Algorithm 3.

The estimated processing time of an operation, i.e. $p_{j,i}$, is usually different from the actual processing time $(p'_{j,i})$ due to uncertainty. $p'_{j,i}$ is known only at the time of realization on the machine. Also $p'_{j,i} > p_{j,i}$ is a practical assumption which is supported by the relationship below [7]:

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

TABLE IV
TERMINAL SET FOR GPHH

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
DD	Due date
W	Job weight
ERC	Ephemeral Random constant

θ follows exponential distribution which is defined by the parameter β.

DJSS problem instances are generated using the parameters specified in Table III. The four pairs of parameters can be used to simulate 16 types of jobs. A problem instance is composed of 3 types of jobs at a time. We did not choose a larger number of job types, which will require much higher computational cost without providing additional insights. Counting their unique combinations with repetition (given by $\binom{r+n-1}{r}$) results in 816 possible configurations. For each of this configuration, 60 DJSS problem instances are created which are then evenly divided into training, validation and test sets. Our preliminary study showed that a larger set of instances do not show any advantage.

TABLE V
FUNCTION SET FOR GP.

Function Set	Meaning
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Protected Division
Max	Maximum
Min	Minimum

A. Genetic Programming System

The set of terminals which we considered is listed in Table IV and the function set is presented in Table V. The mutation, crossover and maximum tree depth are 0.1, 0.85 and 6 respectively [17]. For subpopulations we use a population size of 600 each. The performance is compared with standard GP which uses a population size of 1500. Accordingly the number of generation for standard GP is 200 while it is 100 for the proposed GPHH approach. We have ensured that the computational cost of our new approach does not exceed that of standard GP. The parameter values are presented in Table VI.

TABLE VI
PARAMETER VALUES

		GPHH	ϵ-greedy
1	# sub population	-	4
2	population size	1500	600 $\times$ 4
3	#generations	200	100
4	#clusters validation	-	20
5	#instances used per validation cluster	-	20
6	#clusters training	-	4
7	#generations per epoch	-	10

V. RESULTS

In this section, we describe our results. In order to perform comparisons, 30 independent runs of a method are used to produce 30 sets of solutions. We obtained 30 test subsets from $\mathcal{T}_c$ which is obtained using feature extraction and clustering explained in Section III-A. Each subset consists of 30 DJSS problem instances. For each method, the evolved solutions are compared over each of the 30 problem instances in a test subset. A significance level of 0.05 Wilcoxon-rank-sum test is used to compare the performance. The results are summarized in Tables VII-IX.

A cell in the table consists of a triplet which must be read as $[win\text{-}draw\text{-}lose]$. As an example, the first cell in Table VII is [7-23-0]. This means that for the training set $A-I$, out of the 30 problem instances, the method ϵ-greedy significantly outperformed standard GP in 7 instances (win) and there is no significant difference in 23 (draw). On none of the test instances standard GP outperforms ϵ-greedy approach.

TABLE VII
ϵ-GREEDY VS. GPHH (IMPROVED PERFORMANCE)

A-I	A-II	A-III	A-IV	A-V	A-VI
[7-23-0]	[4-25-1]	[4-26-0]	[4-26-0]	[5-25-0]	[3-26-1]
A-VII	**A-VIII**	**A-IX**	**A-X**	**A-XI**	**A-XII**
[5-24-1]	[5-23-2]	[6-24-0]	[4-26-0]	[3-26-1]	[5-25-0]
A-XIII	**A-XIV**	**A-XV**	**A-XVI**	**A-XVII**	
[5-24-1]	[5-25-0]	[4-25-1]	[15-15-0]	[6-24-0]	

We divided the test subsets based on the performance of our proposed approach. In Table VII we present the results from those test subsets for which our proposed algorithm showed improved performance. Out of the 30 test subsets, the ϵ-greedy based algorithm shown improvement in 17 sets. This improvement is quite prominent for the sets A-I, A-XVI and A-XVII.

In Table VIII we present the results from those test subsets for which the performance of the proposed approach is neither better nor worse. There are 6 such subsets which show these characteristics.

TABLE VIII
ϵ-GREEDY VS. GPHH (NO IMPROVEMENT)

B-I	B-II	B-III	B-IV	B-V	B-VI
[1-27-2]	[9-14-7]	[4-18-8]	[14-5-11]	[6-17-7]	[3-25-2]

In Table IX we present the results from those test subsets for which the performance of ϵ-greedy based approach is poorer than standard GPHH. There are 7 such test sets.

TABLE IX
ϵ-GREEDY VS. GPHH (POOR PERFORMANCE)

C-I	C-II	C-III	C-IV	C-V	C-VI	C-VII
[2-10-18]	[1-3-26]	[0-8-22]	[1-5-24]	[0-10-20]	[2-2-26]	[1-1-28]

The reason for the poor performance of ϵ-greedy in some of the test sets could be attributed to the fact that in order to evolve more specific rules, some of the characteristics on other clusters are not taken into account. This variation in performance is indicative of the fact that scenario-specific rules, which are trained using specific clusters of DJSS training instances, do work well on those scenarios but are outperformed by a general rule which is trained over all the scenarios. We can hypothesize that the evolved rules suffer from overfitting on DJSS instances corresponding to some shop scenarios.

VI. CONCLUSIONS

GPHH approach for job shop scheduling has seen many works in recent years toward improving evolutionary mechanisms to obtain gains of the quality of solutions. In this work, we explored the research direction of improving the quality of solutions by sampling better training instances from the instance space. We have proposed a new framework for GPHH which incorporates active sampling mechanism to identify good training instances toward evolving rules for complex shop scenarios. The proposed algorithm succeeds in its attempts to provide a better balance between exploration and exploitation. Our results show that our algorithm does very well to evolve rules when a reasonable number of complex scenarios are considered. Thus, through our experiments we have demonstrated the importance of this framework.

The ϵ-greedy approach is simple and one of the first methods which is usually applied in the multi-armed bandits context. However, for larger problems this technique is inefficient. This is because for larger problems, the multi-armed bandit formulation results in large number of arms which ϵ-greedy is not efficient at dealing with, particularly because it remembers only the best arm. In the context of our case the number of possible workshop scenarios is actually infinite and therefore we should explore a large number of clusters of DJSS instances to evolve scenario-specific rules. Clearly, the ϵ-greedy method for active sampling is not enough for this task. Therefore, one of our future research directions is to develop new active sampling approaches to address the aforementioned limitations.

Moreover, we have extracted only numerical features from DJSS instances. To extract more information from shop scenarios and represent them better, it is important to extract non-numerical features of the shop e.g. the structure of the queued operations on the machines and the relationship between then. The representation of these features and their incorporation into active sampling framework is another of our future research directions.

REFERENCES

[1] P. Kouvelis and G. Yu, *Robust discrete optimization and its applications*. Springer Science & Business Media, 2013, vol. 14.

[2] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, “Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming,” *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 2, pp. 193–208, 2014.

[3] S. R. Lawrence and E. C. Sewell, “Heuristic, optimal, static, and dynamic schedules when processing times are uncertain,” *Journal of Operations Management*, vol. 15, no. 1, pp. 71–82, 1997.

[4] D. Jakobović and L. Budin, “Dynamic scheduling with genetic programming,” in *Genetic Programming*. Springer, 2006, pp. 73–84.

[5] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, “Dynamic job shop scheduling under uncertainty using genetic programming,” *Intelligent and Evolutionary Systems*, p. 195, 2016.

[6] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, “Dynamic multi-objective job shop scheduling: A genetic programming approach,” in *Automated Scheduling and Planning*. Springer, 2013, pp. 251–282.

[7] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, “Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty,” in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 282–289.

[8] S. Rai, A. V. Godambe, C. B. Duke, and G. H. Williams, “Printshop resource optimization via the use of autonomous cells,” Jul. 18 2006, uS Patent 7,079,266.

[9] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, “Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach,” in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 257–264.

[10] B. Settles, “Active learning literature survey. 2010,” *Computer Sciences Technical Report*, vol. 1648, 2014.

[11] S. Sharma, A. K. Jha, P. S. Hegde, and B. Ravindran, “Learning to multi-task by active sampling,” in *International Conference on Learning Representations*, 2018.

[12] A. Gupta, Y.-S. Ong, and L. Feng, “Multifactorial evolution: toward evolutionary multitasking,” *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2016.

[13] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, “Hyper-heuristic evolution of dispatching rules: A comparison of rule representations,” *Evolutionary computation*, vol. 23, no. 2, pp. 249–277, 2015.

[14] M. N. Omidvar, B. Kazimipour, X. Li, and X. Yao, “Cbcc3-a contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance.” in *CEC*, 2016, pp. 3541–3548.

[15] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, “Sampling heuristics for multi-objective dynamic job shop scheduling using island based parallel genetic programming,” in *International Conference on Parallel Problem Solving from Nature*. Springer, 2018, pp. 347–359.

[16] K. A. Smith-Miles, R. J. James, J. W. Giffin, and Y. Tu, “A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance,” in *International Conference on Learning and Intelligent Optimization*. Springer, 2009, pp. 89–103.

[17] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, “A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem,” *Evolutionary Computation, IEEE Transactions on*, vol. 17, no. 5, pp. 621–639, 2013.

[18] R. Hunt, M. Johnston, and M. Zhang, “Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming,” in *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2014, pp. 927–934.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time analysis of the multiarmed bandit problem,” *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[20] V. Kuleshov and D. Precup, “Algorithms for multi-armed bandit problems,” *arXiv preprint arXiv:1402.6028*, 2014.

[21] J. Branke and C. W. Pickardt, “Evolutionary search for difficult problem instances to support the design of job shop dispatching rules,” *European Journal of Operational Research*, vol. 212, no. 1, pp. 22–32, 2011.

[22] T. Hildebrandt, “Jasima; an efficient java simulator for manufacturing and logistics,” *Last accessed*, vol. 16, 2012.

[23] S. Nguyen, Y. Mei, and M. Zhang, “Genetic programming for production scheduling: a survey with a unified framework,” *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, 2017.