Sampling Heuristics for Multi-objective Dynamic Job Shop Scheduling Using Island Based Parallel 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 6140, New Zealand
{deepak.karunakaran,yi.mei,aaron.chen,mengjie.zhang}@ecs.vuw.ac.nz

Abstract. Dynamic job shop scheduling is a complex problem in production systems. Automated design of dispatching rules for these systems, particularly using the genetic programming based hyper-heuristics (GPHH) has been a promising approach in recent years. However, GPHH is a computationally intensive and time consuming approach. Parallel evolutionary algorithms are one of the key approaches to tackle this drawback. Furthermore when scheduling is performed under uncertain manufacturing environments while considering multiple conflicting objectives, evolving good rules requires large and diverse training instances. Under limited time and computational budget training on all instances is not possible. Therefore, we need an efficient way to decide which training samples are more suitable for training. We propose a method to sample those problem instances which have the potential to promote the evolution of good rules. In particular, a sampling heuristic which successively rejects clusters of problem instances in favour of those problem instances which show potential in improving the Pareto front for a dynamic multi-objective scheduling problem is developed. We exploit the efficient island model-based approaches to simultaneously consider multiple training instances for GPHH.

Keywords: Scheduling · Genetic programming · Parallel algorithms

1 Introduction

Job shop scheduling is a complex problem in manufacturing industries. In general, researchers consider deterministic scheduling problems in which once the information of a new job is obtained it remains constant. But in practice, due to events like machine breakdown, variation in raw material quality, operator availability, etc. uncertainty is ubiquitous in manufacturing environments. In fact, with increasing uncertainty, scheduling becomes more difficult [8]. Dispatching rules have been widely used for generating schedules for different objectives

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 347–359, 2018.
https://doi.org/10.1007/978-3-319-99259-4_28

and have shown good success for DJSS problems [12], particularly for scheduling under uncertainty [9]. But designing them requires considerable expertise and rigorous experimentation. Genetic programming based hyper-heuristic (GPHH) approach has been very successful in automatically evolving dispatching rules [12]. Due to the flexible representation and powerful search ability of GP, existing GPHH methods can evolve very good rules for diverse shop scenarios [7] while handling multiple objectives for DJSS problems [10].

GPHH in general demands high computational cost and time. To address this issue, surrogate models [5] have been proposed as an alternative but they suffer from poor accuracy among other drawbacks [10]. Another key technique to reduce computation time is to use parallel evolutionary algorithms [14]. There are two main categories of parallelization, *parallelizing an independent run* and *island models*. Island models are particularly interesting because of their ability to deal with local optima [14]. The island model uses a spatially structured network of subpopulations (on different processors) to exchange promising individuals among each other is an effective approach. For example, [15] use a specialized island model for multi-objective optimization. [1] is a recent work which demonstrates the effectiveness of asynchronous parallel evolution for hyper-heuristics.

For GPHH to be effective, it is important to use a large training set containing instances with diverse characteristics. This is more important when we consider uncertain manufacturing environments as they present varying shop scenarios [7]. Moreover, when we consider multiple objectives the importance of using diverse training set is compounded. For example, makespan and total tardiness are two frequently considered conflicting objectives. Minimizing the makespan results in high throughput where as minimizing tardiness requires jobs to be not very late. A conflicting scenario arises when a set of jobs with long processing times but shorter deadline compete with a set of jobs with shorter processing times and longer deadlines. For higher throughput the shorter jobs must be completed first as against the longer jobs which adversely affects the tardiness. For evolving good dispatching rules it is important to present the evolutionary system with training instances which capture scenarios highlighting all such conflicts amongst the objectives, under different shop scenarios.

Therefore, for evolving dispatching rules for practical environments we need a large and diverse training set. But this in turn will increase the already high computational cost of GPHH. To address this problem, we need a method which can effectively sample training instances to significantly improve the effectiveness of GPHH without exceeding the computational budget. The parallel island model has already been shown to be both efficient and effective for GPHH when dealing with multiple objectives [6]. Motivated by this success, we address our issue with the idea of associating the multiple islands with different training instances. Furthermore, we utilize the inherent potential of migration policies [13] to introduce a cooperative behavior among islands and collectively evolve better rules.

In this work, we consider multi-objective DJSS problems with a focus on uncertainty in processing times which affects all our objectives. Our primary goal is to develop a sampling heuristic for GPHH which selects good training

instances toward evolving a significantly better Pareto front. To this end, our specific objectives are: (1) Develop a feature extraction method toward clustering the training dataset into DJSS problem instances with different characteristics. (2) Develop a sampling heuristic which iteratively rejects the clusters (*successive reject heuristic*) in favor of those which have the potential to improve the Pareto front. (3) Determine the migration policies of the island model toward improving the efficacy of the proposed sampling heuristic.

2 Background

DJSS problems are characterized by continuous arrival of jobs from a Poisson process [12]. n_j operations constitute a job j with the constraint that they must be processed in a predefined route say $(o_{j,1} \rightarrow o_{j,2} \rightarrow, \ldots, o_{j,n_j})$. Each operation can be processed only on one machine in its route. Other conditions/assumptions which are followed in this work are no preemption, no recirculation of jobs, no machine failure and zero transit times. Total tardiness, makespan, total flow time are some of the objectives considered for DJSS problems. In this work, we consider two potentially conflicting objectives [12]: (1) total weighted tardiness (TWT) and (2) makespan (C_{max}).

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

where $C_j =$ completion time, $d_j =$ due date and w_j are weights.

$$C_{max} = \max(f_j),$$

where f_j is the flowtime of a job.

3 Proposed Method

3.1 Clustering of DJSS Problem Instances

We extract features from the DJSS problem instances based on the parameters: number of operations per job, processing time, due date factor and uncertainty in processing time. Firstly, the basic features for each job are extracted as described

Table 1. 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

in Table 1. Once the feature values are aggregated for all the jobs in an instance; the first, second and third quartiles of each aggregate are calculated to form a 12-dimensional feature vector characterizing each problem instance.

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

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

For each job the features described in Table 1 are calculated and aggregated e.g., for processing time the aggregated feature values are:

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

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

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

The objective of extraction of these features is to cluster the problem instances into classes with different characteristics. Considering our earlier example, assume we have one DJSS problem instance with high variability in number of operations per job and their processing times and another instance with low variability. It can be easily seen that our feature vectors corresponding to the two problem instances can be used to differentiate them.

$\mathcal{T}$ is the training set containing n DJSS problem instances. We extract features for all these problems and cluster them into $\mathcal{C}_1$ and $\mathcal{C}_2$ using K-means clustering. We apply K-means clustering again on each of these clusters to yield $\{\mathcal{C}_{11}, \mathcal{C}_{12}\}$ and $\{\mathcal{C}_{21}, \mathcal{C}_{22}\}$ respectively. This process can be repeated to obtain more sub-clusters $\{\{\mathcal{C}_{111}, \mathcal{C}_{112}\}, \{\mathcal{C}_{121}, \mathcal{C}_{122}\}\}$ and $\{\{\mathcal{C}_{211}, \mathcal{C}_{212}\}, \{\mathcal{C}_{221}, \mathcal{C}_{222}\}\}$ and so on.

3.2 Proposed Island Model

We use two classes of islands in our evolutionary system. The first class of islands, represented as G in Fig. 1(a) and (b), sample training instances from the set $\mathcal{T}$ throughout the evolutionary process. The second class of islands represented as A and B in Fig. 1(b) sample problem instances from the different clusters the choice of which varies with generations. The appropriate choice of the cluster is controlled by the *successive reject heuristic* (SRH).

We first describe the evolutionary process of island G in Algorithm 1. In every generation, a new training instance is sampled from $\mathcal{T}$. Unless otherwise mentioned, we use sample to denote a *simple random sample*. Due to our familiarity with NSGA-II [3] and the fact that we consider only two objectives in this work, we chose NSGA-II as our underlying evolutionary algorithm. In line 3, an iteration of NSGA-II is performed. After each generation, the migration policy determines (line 4) if there will be an exchange of individuals among the islands. The output is a set of dispatching rules which can generate a Pareto front. Note

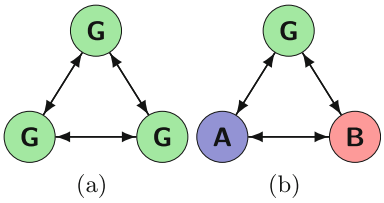

Fig. 1. (a) Standard island model, (b) Island model for successive reject heuristic

Algorithm 1. Island G

Input: $\mathcal{T}$
Output: $\{\omega_1, \omega_2, \ldots, \omega_p\}$
1 **for** $g \leftarrow 1 : N_G$ **do**
2 Sample an instance $\mathcal{I} \in \mathcal{T}$.
3 Run g^{th} iteration of NSGA-II using $\mathcal{I}$.
4 Receive/Send individuals using migration policies.
5 **end**
6 Collect the genetic programs corresponding to the Pareto front : $\{\omega_1, \omega_2, \ldots, \omega_p\}$.

Algorithm 2. Island Z ($Z \in \{A, B\}$)

Input: $\mathcal{C}_Z$,$\mathcal{N}_{SRH}$
Output: $\{\omega_1^z, \omega_2^z, \ldots, \omega_p^z\}$
1 **for** $g \leftarrow 1 : N_G$ **do**
2 Sample an instance $\mathcal{I} \in \mathcal{C}_Z$.
3 Run g^{th} iteration of NSGA-II using $\mathcal{I}$.
4 Receive/Send individuals using migration policies.
5 **if** $g \in \mathcal{N}_{SRH}$ **then**
6 $\mathcal{C}_Z \leftarrow SRH(P_k^A, P_k^B, \mathcal{C}_A, \mathcal{C}_B)$
7 **end**
8 Collect the genetic programs corresponding to the Pareto front : $\{\omega_1^z, \omega_2^z, \ldots, \omega_p^z\}$.

that the final output of the parallel evolutionary system is the combination of outputs from all individual islands (Table. 2).

In Algorithm 2, we describe the evolutionary process of the islands A and B (Fig. 1(b)). Both islands are similar, except that they sample their training instances from different clusters. Their migration policies are the same. The cluster $\mathcal{C}_Z$ is changed at discrete stages during the evolutionary process. The set $\mathcal{N}_{SRH}$ contains the generations at which the successive reject hypothesis is invoked to change $\mathcal{C}_Z$ (line 6). The rest of the procedure is the same as in Algorithm 1.

Table 2. Notation

Notation	Description
$\mathcal{T}$	Set of all DJSS problem instances for training
N_G	Total number of generations for evolutionary process
$\mathcal{C}_Z$	Cluster corresponding to island $Z \in \{A, B\}$
P_k^Z	Top k individuals from island $Z \in \{A, B\}$
$M_{\overrightarrow{X,Y}}$	Migration policy from island X to Y
P_{2k}	Combined list of top k individuals from islands A and B
TOP_k	List of top k individuals across island A and B
$TOP^{\mathcal{Z}_k}$	#individuals which are present both in $P_k^{\mathcal{Z}}$ and TOP_k, $\mathcal{Z} \in \{A, B\}$
$\mathcal{N}_{SRH}$	Set of generations at which SRH is invoked

Successive Reject Heuristic. The successive reject heuristic (SRH) is described in Algorithm 3. When the SRH is invoked by islands A and B at generation $g \in \mathcal{N}_{SRH}$, the top-k individuals from each island, P_k^A and P_k^B respectively, are sent to the island G on which the SRH algorithm is run. The two sets of individuals are then combined to get P_{2k} (line 3). A new set of DJSS problem instances,$\mathcal{I}$ is sampled from $\mathcal{T}$ with the purpose of assigning new fitness values to each individual in P_{2k} (lines 4–11).

After the fitness assignment, we use the NSGA-II fitness strategies to sort the individuals. NSGA-II first ranks the individuals based on dominance relation and then the individuals with same rank are ordered based on crowding distance [3]. We use the same approach to sort the individuals in P_{2k} (line 12). After sorting, the best k individuals are extracted from P_{2k} into the list TOP_k (line 13). Then we count the number of individuals corresponding to each island in the list TOP_k (lines 14–15). The cluster corresponding to the island with the lower number of individuals in TOP_k is rejected (line 17 or 20). The $\mathcal{C}_Z$ of the winning island is further clustered into two sub-clusters (lines 18 or 21). The new clusters which are the output of SRH algorithm are then randomly assigned to the islands. Till the next invocation of SRH, the evolution in the islands is continued using DJSS problem instances sampled from the new clusters.

Migration Policies. Migration policies play a major role in the performance of the island models [13]. A migration policy states the number of individuals to be sent to the destination island, frequency of migration and the generation from which the migration starts. For the standard island model shown in Fig. 1(a) designing a policy is straightforward. Due to symmetry, a single policy for all the islands will suffice [6]. Since we consider two classes of islands which are working together for a common goal, different migration policies must be designed for different classes islands.

Formally, a policy $\mathcal{M}_{\overrightarrow{I_1,I_2}}$ from island I_1 to I_2 is defined by a triplet <start generation, frequency, #individuals to send>. We consider the migration policy

Algorithm 3. Successive Reject Heuristic

Input: $P_k^A, P_k^B, \mathcal{C}_A, \mathcal{C}_B$
Output: $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\}$ to respective islands.
$\mathcal{T} \leftarrow$ set of all DJSS training instances.
$\mathbb{I} \leftarrow$sample from $\mathcal{T}$
$P_{2K} \leftarrow \{P_k^A, P_k^B\}$
foreach $p \in P_{2k}$ **do**
  $tot.fit. \leftarrow \overrightarrow{0}$
  foreach $\mathcal{I} \in \mathbb{I}$ **do**
    $obj.values \leftarrow$ Simulation for $(p, \mathcal{I})$.
    $tot.fit. \leftarrow tot.fit. + obj.values$
  end
  $fit(p) \leftarrow tot.fit.$
end
Sort P_{2k} using NSGA-II fitness startegies.
$TOP_k \leftarrow$ Extract top-k individuals from P_{2k}.
$TOP_k^{\mathcal{A}} \leftarrow |P_k^A \cap TOP_k|$
$TOP_k^{\mathcal{B}} \leftarrow |P_k^B \cap TOP_k|$
if $TOP_k^{\mathcal{A}} \geq TOP_k^{\mathcal{B}}$ **then**
  Reject $\mathcal{C}_B$.
  $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\} \leftarrow$ K-means cluster$(\mathcal{C}_A)$
else
  Reject $\mathcal{C}_A$.
  $\{\mathcal{C}_A^{new}\}$,$\{\mathcal{C}_B^{new}\} \leftarrow$ K-means cluster$(\mathcal{C}_B)$
end

$\mathcal{M}_{\overrightarrow{I_1,I_2}}$ to be different from $\mathcal{M}_{\overrightarrow{I_2,I_1}}$. The selection of individuals for migration is based on elitism, i.e., a proportion of fittest individual(s) are chosen from the population for migration.

For island G, it is more productive to receive individuals from A and B frequently as it will improve its diversity. This is because the evolved rules in A and B are exposed to training instances which are different from G. On the other hand a high frequency of migration between A and B will homogenize the islands, making SRH less effective. Moreover, the frequency of migration in $M_{\overrightarrow{AG}}$ is much higher than $M_{\overrightarrow{GA}}$ (similar for island B) for the same reasons. The same analysis holds for determining the number of individuals to be migrated between the two. Furthermore, the migration policy $M_{\overrightarrow{AB}}$ is restricted to exchanging individuals only and immediately after invocation of SRH.

4 Experiment Design

4.1 Simulation Model

We use a DES system (Jasima) [4] to generate DJSS problem instances. The job arrival follows a Poisson process with $\lambda = 0.85$ [7]. This assumption has been

used in large number of works [2,10,11]. For every run of the simulation, the first 500 jobs are considered as warm-up and the objective values are calculated for the next 2000 jobs.

The uncertainty in processing times is simulated using the model considered in [7]. Basically for an operation $o_{j,i}$ the relationship between the processing time with uncertainty $p'_{j,i}$ and processing time without uncertainty $p_{j,i}$ is:

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

θ follows exponential distribution [7]. In Table 3, the parameter β corresponds to the scale parameter of the exponential distribution.

In order to create problem instances with varying characteristics, DJSS problem instances are generated with many combinations of the simulation parameters shown in Table 3. The combination of these four pairs of parameters can simulate 16 types of jobs. For composing a training DJSS problem instance, *3* job types are considered at a time. On counting the unique combinations of *3* job types we find a total of 816 possible configurations (combinations without repetitions). Since we extract features from problem instances in order to perform clustering we create 20 DJSS problems for each configuration to build the training set $\mathcal{T}$. Our preliminary study showed that a larger training set would show no advantage but require more computational effort.

For testing, we create a new set (say $\mathcal{Y}$) of DJSS problems using the 816 possible configurations mentioned above. We sample 30 DJSS problem instances from $\mathcal{Y}$ to obtain our first test set. Due to large number of problem configurations it is not possible to test on each of them separately. Therefore, we create *four* more test sets by clustering $\mathcal{Y}$ and sampling 30 problem instances from each. These test sets are denoted by 3-$\mathcal{Y}$, 3-I, 3-II, 3-III and 3-IV, where 3 stands for number of job types.

We also want to observe the generalization ability of our methods over more complex configurations. Therefore, DJSS instances comprising of *4* job types are created. On counting, the total number of unique configurations in this case are as high as 3876 (combinations with repetitions). Performing the same procedure described above generates the following test sets: 4-$\mathcal{Y}$, 4-I, 4-II, 4-III and 4-IV.

Table 3. DJSS simulation parameters

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

Table 4. Migration policies

Island-pairs	Policies
$\overrightarrow{GG}$	<20, 20, 30>
$\overrightarrow{AB}$	<50, 50, 60>
$\overrightarrow{BA}$	<50, 50, 60>
$\overrightarrow{AG}$	<20, 20, 30>
$\overrightarrow{BG}$	<20, 20, 30>
$\overrightarrow{GB}$	<50, 25, 10>
$\overrightarrow{GA}$	<50, 25, 10>

Table 5. Terminal sets for GP.

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

4.2 Genetic Programming System

The terminal set for genetic programming is listed in Table 5. For all our islands we use a population size of 800 each. We also compare performance of our method with the standard NSGA-II for which the population size is set at 2500. With a tree depth of 6, the crossover and mutation are 0.85 and 0.1 respectively [10]. Each evolutionary algorithm is run for 150 generations.

4.3 Island Model

We use the SRH algorithm at generations 49 and 99, i.e., $\mathcal{N}_{SRH} = \{49, 99\}$. The SRH algorithm also requires the simulator to assign fitness to individuals. Furthermore, for GPHH to utilize the problem instances from a cluster, considerable number of generations are required. So frequently invoking SRH will not yield the desired outcome but only incur additional computational cost. Therefore the size of $\mathcal{N}_{SRH}$ is small and generations selected are far apart.

The migration policies are presented in Table 4. While deciding the frequency parameter of the migration policies involving islands A and B, $\mathcal{N}_{SRH}$ has been taken into account. The exchange of individuals starts after a delay as the evolved rules in the early generations are not good. For the TOP_k individuals the value $k = 30$ was chosen, after experimental evaluation.

5 Results and Discussion

In this Section, we present the results from our experiments. We compare the performance our method with standard NSGA-II algorithm and standard island model approach. The hypervolume ratio (HV), inverted generational distance (IGD) and spread (SPREAD) indicators are considered for comparison as they are frequently used in the literature [12] to compare the generated Pareto fronts. In order to approximate the true Pareto front as required by performance indicators the individuals from all the methods across all runs are combined. For

each method the solutions are compared over 30 problem instances from a test set. 30 independent runs produce 30 sets of dispatching rules for each method. The Wilcoxon-rank-sum test is used to compare the performance. We consider a significance level of 0.05.

The results are summarized in Tables 6, 7 and 8. Each cell in the tables consists of a triplet which represents $[win - draw - lose]$. For example, in Table 6 the comparison between standard island model and NSGA-II approach is summarized. For the training set 3-$\mathcal{Y}$, if we consider hypervolume indicator, then island model has significantly outperformed NSGA-II in 18 problem instances and there is no significant difference observed for 12 problem instances.

Table 6. Island-Model versus NSGA-II

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[18-12-0]	[11-19-0]	[18-12-0]	[19-11-0]	[17-13-0]	[14-16-0]	[15-15-0]	[18-12-0]	[16-13-0]	[12-18-0]
IGD	[24-6-0]	[21-9-0]	[25-5-0]	[29-1-0]	[27-3-0]	[24-6-0]	[21-9-0]	[22-8-0]	[22-8-0]	[19-11-0]
SPREAD	[3-22-5]	[0-18-12]	[4-23-0]	[5-25-0]	[2-28-0]	[3-21-6]	[6-22-2]	[4-23-3]	[5-20-5]	[2-24-2]

In Table 6, we compare NSGA-II with standard island model. As expected, the performance of island model is much better, which is line with the observations made in [6]. For HV and IGD performance indicators, the performance is very good, but for SPREAD indicator there is no clear winner. This significant difference in performance is consistent across all the test sets including 4-job type configurations.

Table 7. SRH-Island model versus NSGA-II

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[23-7-0]	[17-3-0]	[23-7-0]	[25-5-0]	[24-6-0]	[20-10-0]	[24-6-0]	[22-8-0]	[24-6-0]	[21-9-0]
IGD	[30-0-0]	[30-0-0]	[27-3-0]	[29-1-0]	[30-0-0]	[30-0-0]	[27-3-0]	[30-0-0]	[28-2-0]	[29-1-0]
SPREAD	[1-23-6]	[0-25-5]	[1-27-2]	[3-26-1]	[0-28-2]	[4-21-5]	[1-27-2]	[1-27-2]	[2-22-6]	[0-25-5]

In Table 7, we compare the performance of NSGA-II and SRH-based island model (SRH). Across all the test sets the proposed method has done well. Particularly for HV indicator, the SRH method has significantly done better in more than 20 problem instances for almost every test set. Similar performance is observed for IGD as well. Though once gain, with respect to SPREAD, there is no verifiable difference. This is because the obtained Pareto fronts are sparse for all the algorithms.

Finally we compare, the SRH approach with the standard island model. Once again the SRH approach performs significantly better on an average of 10 problem instances from each test set. This confirms that SRH approach was able to associate useful training instances through the successive rejection of clusters of training instances.

Table 8. SRH-Island model versus island model

	3-$\mathcal{Y}$	3-I	3-II	3-III	3-IV	4-$\mathcal{Y}$	4-I	4-II	4-III	4-IV
HV	[10-20-0]	[5-24-1]	[6-22-2]	[9-21-0]	[14-16-0]	[10-20-0]	[10-20-0]	[8-21-1]	[9-21-0]	[11-19-0]
IGD	[18-12-0]	[20-10-0]	[19-11-0]	[19-11-0]	[20-10-0]	[15-15-0]	[14-15-1]	[13-17-0]	[15-15-0]	[18-20-0]
SPREAD	[5-18-7]	[10-20-0]	[3-24-3]	[1-25-4]	[0-23-7]	[5-20-5]	[1-22-7]	[2-20-8]	[4-17-9]	[3-24-3]

5.1 Analysis

A frequently observed path taken by the successive reject heuristic is represented below.

$$\mathcal{T} \rightarrow \{\cancel{\mathcal{C}_1}, \mathcal{C}_2\} \rightarrow \{\mathcal{C}_{21}, \cancel{\mathcal{C}_{22}}\} \rightarrow \{\mathcal{C}_{211}, \mathcal{C}_{212}\}$$

In **retrospect**, we analyze the clusters which showed potential to guide the GPHH toward evolving better rules. In order to further validate the ability of SRH, we took the clusters represented by $\mathcal{C}_{21}$ and $\mathcal{C}_{22}$ as training sets. We performed 30 independent runs of NSGA-II algorithm on each. We observed that the cluster rejected by SRH ($\mathcal{C}_{22}$) performed significantly poor on both HV and IGD indicators. Figure 2 shows a box plot for HV indicator on a test problem instance from the set 3-$\mathcal{Y}$.

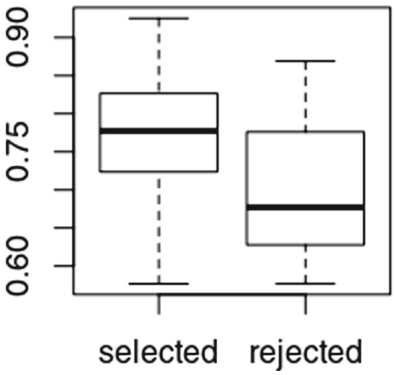

Fig. 2. Comparing $\mathcal{C}_{21}$(selected) and $\mathcal{C}_{22}$ (rejected) using HV.

Furthermore, we also analyzed the problem configurations associated with cluster $\mathcal{C}_{21}$. One of the reasons for analyzing $\mathcal{C}_{21}$ and not $\mathcal{C}_2$ is its smaller size and also the fact that out of 30 independent runs, this path was chosen by SRH for 20 of the runs. We observed that the DJSS instances whose job types were pertaining to equal proportion of high and low level of uncertainty were in high numbers. Also, DJSS instances comprising jobs with low and high number of operations per job were found in large numbers. In other words, SRH is biased towards instances with high variability in their jobs. A high variability in the training instances has more potential to present the GPHH with difficult and conflicting scenarios, as explained in a previous example.

6 Conclusions

Most of the research works in evolutionary scheduling focus on improving only the different aspects of algorithms. But it is also important to develop methods to select appropriate training instances for the evolutionary algorithms to produce desired outcome. We have successfully taken a step toward this direction by demonstrating that a simple sampling heuristic using basic features extracted from the problem instances could improve the effectiveness of the evolutionary process. By exploiting the potential of island model approach we obtained significantly better results while maintaining computational efficiency. We demonstrated the efficacy of our approach using just two objectives and in future, we would extend our work to tackle many-objective scheduling problems.

References

1. Bertels, A.R., Tauritz, D.R.: Why asynchronous parallel evolution is the future of hyper-heuristics: A CDCL SAT solver case study. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 1359–1365. ACM (2016)
2. Branke, J., Nguyen, S., Pickardt, C., Zhang, M.: Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. **20**, 110–124 (2016)
3. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
4. Hildebrandt, T.: Jasima - an efficient java simulator for manufacturing and logistics. Last Accessed **16** (2012)
5. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Evol. Comput. **23**(3), 343–367 (2015)
6. Karunakaran, D., Chen, G., Zhang, M.: Parallel multi-objective job shop scheduling using genetic programming. In: Ray, T., Sarker, R., Li, X. (eds.) ACALCI 2016. LNCS (LNAI), vol. 9592, pp. 234–245. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-28270-1_20
7. Karunakaran, D., Mei, Y., Chen, G., Zhang, M.: Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 282–289. ACM (2017)
8. Kouvelis, P., Yu, G.: Robust Discrete Optimization and its Applications, vol. 14. Springer, Heidelberg (2013). https://doi.org/10.1007/978-1-4757-2620-6
9. Lawrence, S.R., Sewell, E.C.: Heuristic, optimal, static, and dynamic schedules when processing times are uncertain. J. Oper. Manag. **15**(1), 71–82 (1997)
10. Nguyen, S., Mei, Y., Zhang, M.: Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. **3**(1), 41–66 (2017)
11. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Dynamic multi-objective job shop scheduling: a genetic programming approach. In: Uyar, A., Ozcan, E., Urquhart, N. (eds.) Automated Scheduling and Planning. SCI, vol. 505, pp. 251–282. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39304-4_10
12. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. IEEE Trans. Evol. Comput. **18**(2), 193–208 (2014)

13. Nowak, K., Izzo, D., Hennes, D.: Injection, saturation and feedback in metaheuristic interactions. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 1167–1174. ACM (2015)
14. Sudholt, D.: Parallel evolutionary algorithms. In: Kacprzyk, J., Pedrycz, W. (eds.) Springer Handbook of Computational Intelligence, pp. 929–959. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-43505-2_46
15. Xiao, N., Armstrong, M.P.: A specialized island model and its application in multiobjective optimization. In: Cantú-Paz, E. (ed.) GECCO 2003. LNCS, vol. 2724, pp. 1530–1540. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-45110-2_24