People-centric Evolutionary System for Dynamic Production Scheduling

Su Nguyen, *Member, IEEE* and Mengjie Zhang, *Fellow, IEEE* and Damminda Alahakoon, and Kay Chen Tan, *Fellow, IEEE*

***Abstract*—Evolving production scheduling heuristics is a challenging task because of the dynamic and complex production environments and the interdependency of multiple scheduling decisions. Different genetic programming methods have been developed for this task and achieved very encouraging results. However, these methods usually have trouble in discovering powerful and compact heuristics, especially for difficult problems. Moreover, there is no systematic approach for the decision makers to intervene and embed their knowledge and preferences in the evolutionary process. This paper develops a novel people-centric evolutionary system for dynamic production scheduling. The two key components of the system are a new mapping technique to incrementally monitor the evolutionary process and a new adaptive surrogate model to improve the efficiency of genetic programming. The experimental results with dynamic flexible job shop scheduling show that the proposed system outperforms the existing algorithms for evolving scheduling heuristics in terms of scheduling performance and heuristic sizes. The new system also allows the decision makers to interact on the fly and guide the evolution towards desired solutions.**

***Index Terms*—genetic programming, flexible job shop scheduling, diversity, visualisation.**

I. Introduction

THERE have been an increasing number of studies on machine learning (ML) and artificial intelligence (AI) in the context of smart factories and Industry 4.0 [1]. Most of the studies emphasise the applications of ML and AI to advanced process control (APC) such as real-time equipment monitoring, fault detection and classification, quality control, and preventive maintenance. Scheduling is a critical function in APC to make sure that manufacturing resources are optimally utilised. However, the scheduling methods proposed for smart factories have been still based on traditional approaches (e.g. dispatching rules) with the support of computer simulation and real-time data [2], [3]. There have been several studies on adaptive scheduling methods based on ML and online optimisation but these methods are restricted by many assumptions and not flexible enough to cope with the complexity of smart factories [4], [5], [6]. Big data analytics solutions developed for scheduling purposes only provide real-time monitoring of orders and inventories, with only little emphases on making optimal planning and scheduling decisions.

Advances in computing power and optimisation have led to the developments of new solution methods for production scheduling. Meta-heuristics [7], [8], [9] is a popular approach to find near optimal solutions for a wide range of production scheduling problems. However, manual designing a good meta-heuristic algorithm is not a trivial task and it can be very time consuming and requires a lot of problem domain knowledge. In addition, there is no guarantee that these algorithms may work in stochastic and dynamic production environments without major modifications. In recent years, genetic programming (GP) has been adopted to solve a wide range of production and supply chain management problems [10], [11], [12]. Due to its flexibility and powerful search mechanisms, GP can be easily customised to deal with different planning and scheduling decisions in complex production and logistics systems, which cannot be easily handled by other ML and optimisation methods. One of the most popular applications of GP in this domain is the automated design of production scheduling heuristics [13]. Previous studies have shown that GP can successfully evolve superior scheduling heuristics compared with those manually designed by researchers and practitioners and other ML methods [14], [15], [16], [17]. Moreover, heuristics evolved by GP have some useful properties for practical applications such as simplicity, scalability, reactiveness, and interpretability (in some cases), which are very difficult to achieve with conventional optimisation methods. As an evolutionary computation (EC) methods, GP can also take advantage of advanced techniques such as evolutionary multi-objective optimisation and surrogate models to improve its efficiency and to tackle real-world challenges. These properties make GP an attractive ML solution for dynamic and complex production systems such as smart factories.

Although the past experiments have demonstrated that GP can discover very powerful heuristics for different scheduling applications, searching for such heuristics is not trivial. When the production environments are complex and involve multiple planning and scheduling decisions [14], [16], [18], a large number of features (from jobs and machines) need to be taken into consideration, which results in a very large heuristic search space. Therefore, it is computationally expensive and difficult to find the most effective heuristics. To overcome this limitation, researchers have investigated and developed new representations [17], [19], surrogate models [20], [21], feature selection methods [22], and learning techniques [23], [24], [25]. For instance, Hildebrandt and Branke [20] proposed a surrogate-assisted GP which relies on a simple surrogate model based on phenotypic characteristics of evolved scheduling heuristics for dynamic job shop scheduling. They showed that the surrogate model can improve GP convergence by efficiently generate new heuristics. Nguyen et al. [21] extended this surrogate model to deal with dynamic flexible job shop scheduling and make a modification to enhance population diversity. The experiments showed that surrogate-assisted GP

is significantly better than the simple GP method in terms of fitnesses. On the other hand, Mei et al. [22] proposed a new method to measure the importance of each feature and developed a two-phase learning algorithm for GP. They showed that it is possible to improve the effectiveness of GP if the optimal feature subset is selected. Some researchers focused on ensemble learning strategies [24], [25] and they showed that combining multiple heuristics can improve scheduling decisions. Hart and Sim [23] developed a sophisticated algorithm based on ensemble learning to deal with a diverse set of problem instances and showed very promising results.

The above developments undoubtedly enhance the performance of evolved scheduling heuristics. However, there are key drawbacks with these existing methods. *First*, the existing methods cannot efficiently monitor the evolutionary process, particularly the areas in the search space that GP has explored. Because of this reason, GP can waste its time examining bad heuristics or revisiting the same favourable heuristics without making significant improvements. In some extreme cases, GP can double the sizes of its heuristics just to achieve marginal improvements in terms of fitnesses. Given that the fitness evaluations can be very computationally expensive (e.g. using discrete-event simulation to measure heuristic performance), GP needs to be effective in generating potential heuristics when applying genetic operations.

Second, the sizes of evolved heuristics increase dramatically during the evolutionary process, especially with advanced GP methods mentioned above. As a consequence, it is very difficult to explain how the scheduling decisions are made. It is undesirable because interpretability is one of the attractive properties of GP. Although some techniques including simplification [26] and visualisation [17] have been proposed to gain more insights about the evolved heuristics, they are restricted to simple scheduling problems and a low number of features. Interpretable ML [27] and people-centric AI [28] are the current trend in the AI and ML research community to gain the values of these technologies by interpreting the representations learned and decisions made by these AL/ML models, and building new technologies to enhance human interaction with AI/ML.

Finally, GP evolution is currently a black-box optimisation process. In previous studies, researchers and decision makers can only provide some inputs (e.g. features, heuristic structures) and observe the final outputs. They have little or no knowledge about how or which heuristics are generated. Therefore, it is impossible to embed their insights and preferences to speed up the evolutionary process or to guide the search towards their most favourite solutions.

This paper aims to develop a new people-centric evolutionary system (PES) that can efficiently utilise the historical search data to adaptively and interactively guide the search towards effective and interpretable production heuristics. The main contributions of this paper are:

1) A new *mapping technique* that can efficiently capture the topological relations of evolved heuristics,
2) A new *adaptive surrogate model* that can integrate diversity control and bloat control to evolve powerful and compact scheduling heuristics,
3) A new *surrogate-assisted algorithm* that can evolve superior and compact production scheduling heuristics,
4) A new *human-computer interaction* component that allows decision makers to monitor and to guide the evolutionary process on the fly.

Different from existing approaches, the proposed system provides users and decision makers a *"people-centric"* way to monitor the evolutionary process, control the evolutionary behaviours, integrate preferences, and generate compact solutions. To demonstrate the effectiveness of the new approach, the proposed PES algorithm is applied to evolve scheduling heuristics composed of priority dispatching rules and routing rules for dynamic flexible job shop scheduling problems. This problem is investigated as it is a good example of a complex production environment in which multiple scheduling decisions, uncertainty, and dynamic changes have to be taken into consideration. Extensive experiments with different simulation scenarios are conducted to compare the performance and generalisation of PES and other GP algorithms.

The rest of this paper is organised as follows. Section II describes the dynamic flexible job shop scheduling problems and scenarios to be investigated in this study. Section III provides details of the proposed algorithm. Section IV and Section V present the experiment design and the results respectively. Further analyses of the proposed algorithm are presented in Section VI. Finally, conclusions and future research directions are shown in Section VII.

II. Dynamic Flexible Job Shop Scheduling

This paper investigates the dynamic flexible job shop scheduling (DFJSS) problem, in which a job j (e.g. a customer order) has a number of operations $O_j = \{o_{j1}, o_{j2}, \ldots, o_{jN_j}\}$ to be processed by a set of available machines $\mathcal{M}$. Different from the traditional job shop scheduling problem in which each operation is pre-assigned to a specific machine $m \in \mathcal{M}$, an operation o_{ji} can be handled by a subset of machines $\mathcal{M}_{ji} \subset \mathcal{M}$. The processing time of the operation o_{ji} is $p(o_{ji}, m)$ if it is routed to a machine $m \in \mathcal{M}_{ji}$. The goal of DFJSS in this study is to allocate machines to jobs to optimise the total weighted tardiness. Two types of decisions in DFJSS are routing decisions and sequencing decisions. Routing decisions specify which machine to process an operation. Meanwhile, sequencing decisions govern the priorities of jobs or operations waiting in queues.

The flexible job shop in this study is similar to the simulation model used in [7] and [16], but its goal is to determine long term performance of scheduling heuristics rather than a scheduling solution for a particular instance. The shop will have $|\mathcal{M}| = 10$ machines as this is sufficient to present the complexity and situations in large job shop production systems. In our investigated scenarios, new random jobs will arrive over time and each job j will have the number of operations N_j following a discrete uniform distribution $\mathcal{U}\{1, |\mathcal{M}|\}$. In DFJSS, each operation o_{ji} can be processed by a set of alternative machines $\mathcal{M}_{ji}$ and the size of $\mathcal{M}_{ji}$ is governed by flexibility $f\%$ [16]. Basically, a flexible job shop with $f\%$ implies that at most $f\%$ of the total number of

machines in the shop are available to process an operation. In our shop, if the flexibility $f\% = 30\%$, the size of $\mathcal{M}_{ji}$ will be randomly generated by the discrete uniform distribution $\mathcal{U}\{1, 3\}$ as $f \times |\mathcal{M}| = 3$. The processing time of the operation o_{ji} if it is handled by a machine $m \in \mathcal{M}_{ji}$ is $p(o_{ji}, m)$ which follows an exponential distribution $Exp(1/\mu)$ where $1/\mu = 1$ is the mean processing time.

Job arrivals will follow Poisson process with the arrival rate λ [29], [21] calculated as follow:

$$\lambda = \rho \frac{\mu |\mathcal{M}|}{\bar{n}} \tag{1}$$

where ρ is the utilisation of machines and $\bar{n}$ is the average number of operations per job which is $\frac{1}{2}(1 + |\mathcal{M}|)$.

As a new job arrive at the system, a due date d_j will be assigned to the job based on the total work content (TWK) rule [30], i.e. $d_j = r_j + h \times \sum_{o_{ji} \in O_j} p_{avg}(o_{ji})$ where h is the allowance or tightness factor [31], [7] and $p_{avg}(o_{ji})$ is the average processing time of operation o_{ji} (across all alternative machines). For the weights w_j of jobs, the 4:2:1 rule is employed [31], [7]. This rule is inspired by Pinedo and Singer [32] when their research showed that 20% of the customers are very important ($w_j = 4$), 60% are of average importance ($w_j = 2$), and the remaining 20% are of less importance ($w_j = 1$).

In simulation, we start with an empty shop and jobs will randomly arrive over time. Because our study focuses on routing rules and dispatching rules, the job release policy will be simplified and jobs will be immediately released to the shop upon their arrivals (r_j is the arrival time). When a job j is about to visit the next machine to process the current operation o_{ji}, the routing rule is activated to determine the priority $\mathcal{P}_r(m, j)$ for each machine $m \in \mathcal{M}_{ji}$ based on the information of the job j and current workload of m. The job j will visit the machine m^* with the highest $\mathcal{P}_r(m, j)$. When a machine m become idle and its queue $\mathcal{Q}(m)$ is not empty, the dispatching rule will be applied to calculate the priority $\mathcal{P}_d(j, \mathcal{M})$ for each job $j \in \mathcal{Q}(m)$ based on the information of the job j and the current status of the shop. The objective of the dynamic FJSS problem here is to minimise the total weighted tardiness:

$$TWT = \sum_{j \in \mathbb{C}} w_j T_j \tag{2}$$

where $T_j = (d_j - C_j)$ is the tardiness, and C_j is the completion time of job j. In this equation, $\mathbb{C}$ is the set of completed jobs used to evaluate scheduling heuristics. For the purpose of measuring the effectiveness of evolved scheduling heuristics, the interval from the beginning of the simulation (replication or instance) until the arrival of the 500^{th} job is considered as the warm-up time and the statistics from the next completed 5000 jobs will be recorded for calculate the objective value (i.e. $|\mathbb{C}| = 5000$).

III. People-centric Evolutionary System

Fig. 1 shows how the proposed PES works. The algorithm starts with a random population $\mathbb{H} = \{\mathcal{H}_1, \mathcal{H}_2, \cdots, \mathcal{H}_N\}$ of scheduling heuristics which contains two priority functions

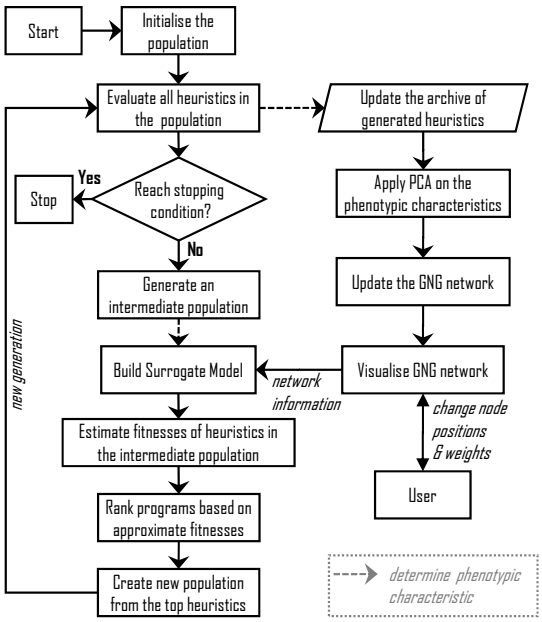


Fig. 1. People-centric evolutionary system.

for dispatching (or sequencing) decisions $\mathcal{P}_d$ and routing decisions $\mathcal{P}_r$. If the stopping condition is not reached, all heuristics in the population will be evaluated by using discrete event simulation (DES) [29] as described in Section II. The phenotypic characteristics $phenotype(\mathcal{H})$, the corresponding fitnesses *fitness*$(\mathcal{H})$, and the sizes of evaluated heuristics are recorded into an archive $\mathcal{A}$. Then principal component analysis (PCA) is applied to transform the dataset X of phenotypic characteristics (with the dimension of D stored in $\mathcal{A}$) to X' with a lower dimensionality (by using the first K principal components where $K << D$). Then, a modified version of growing neural gas (GNG) [33] is used to learn topological relations of generated heuristics from the transformed dataset X'. The network $\mathcal{N} = (V, E)$ obtained with the modified GNG (mGNG) contains a set of nodes V and a set of undirected edges E that represent the distributions of generated heuristics on the search space. After the network is learned, the user can optionally modify the network either by changing node positions or selection pressures. PES will build a surrogate model based on the historical search data and the information from the obtained mGNG network. An intermediate population is then generated by applying genetic operations. The fitnesses of these newly generated heuristics are estimated by using the surrogate model with inputs from the phenotypic characteristics and sizes of those heuristics, archive $\mathcal{A}$, and the network $\mathcal{N}$. The heuristics in the intermediate population are ranked based on the estimated fitnesses and only the top heuristics are used to build the population for the next generations. In the rest of this section, we will provide the details of each key component in this algorithms including the GP representation of scheduling heuristics, genetic operations, the mGNG algorithm, and the surrogate model.

A. Representation

The proposed PES utilises both genotype characteristics (heuristic sizes) and phenotype characteristics (rankings of

TABLE I
ATTRIBUTES/FUNCTIONS OF ROUTING RULES $\mathcal{P}_r(j, m)$

Notation	Description
WLN	workload of machine $m = workload(Q(m))$
NON	number of operations waiting at $m = \|Q(m)\|$
rPT	ratio between workload of operations in $Q(m)$ with processing times larger than $p_(o_{ji}, m)$.
$rSLACK$	ratio between workload of operations in $Q(m)$ with slack larger than the slack of job j
Function set	$+$,$-$,$\times$, protected division %, min, max

TABLE II
ATTRIBUTES/FUNCTIONS OF DISPATCHING RULES $\mathcal{P}_d(j, \mathcal{M})$

Notation	Description
rrJ	time-in-system of job $(t - r_j)$
rRJ	job queuing time $(t - mr_j)$
RO	number of remaining operations $\|O_j^r\|$.
RT	work remaining of the job $p_(o_{jk}, m) + \sum_{o_{ji} \in O_j^r \setminus k} p_{avg}(o_{ji})$
PT	operation processing time $p_(o_{jk}, m)$
rDD	time to due date = $d_j - t$
SL	slack of the job $= d_j - (t + RT)$
W	weight of job w_j
#	Random number from 0 to 1
NPT	average processing time of the next operation $p_{avg}(o_{ik+1})$
$WINQ$	average workload in the next queue $\frac{1}{\|\mathcal{M}_{ik+1}\|} \sum_{m' \in \mathcal{M}_{ik+1}} workload(Q(m'))$
Function set	$+$,$-$,$\times$, protected division %, min, max

jobs) of evolved heuristics to improve its efficiency.

1) Genotype: Both routing rules $\mathcal{P}_r$ and dispatching rules $\mathcal{P}_d$ are represented as expression trees in GP. Table I and Table II show the attributes used to construct routing rules and dispatching rules respectively. In these tables, t is the current simulation time, mr_j is the time job j visit machine m, and o_{jk} is its current operation. The function $workload(Q(m))$ will calculate the total processing time of all jobs waiting in the queue of machine m. These attributes have been extracted from the previous studies [34], [16], [20] and they have been used in many existing rules in the literature. For the routing rules, the last two attributes are added to provide additional comparative information between the considered job and other jobs waiting in the queue of the corresponding machines.

2) Phenotype: This study used the phenotypic characteristics proposed by [20] to capture the behaviours of dispatching rules. The phenotype of an evolved heuristic is characterised by a decision vector with the dimension of D, where D is the number of decision situations (each decision situation includes a number of jobs to be prioritised). First, the ranks of jobs (smaller ranks for jobs with higher priorities) in each situation determined by a reference rule are recorded. For each dispatching rule, the corresponding ranks are also determined and the decision value for each decision situation is the rank determined by the reference rule of the job whose rank obtained by the evolved priority function is 1. The distance between two rules is computed using the Euclidean distance between their decision vectors. As the routing rules considered in this study also make routing decisions based on the assigned priorities, the same technique can be applied to determine the phenotypic characteristics of routing rules. The phenotype of a scheduling heuristic is simply a concatenation of the phenotypes of its dispatching rule and routing rule. An example of phenotypes for the considered scheduling heuristic can be found in [20] and [35].

B. Genetic operations

In the proposed algorithm, subtree crossover, subtree mutation [36], and subtree extraction [37] are employed to generate new rules for the scheduling heuristic $\mathcal{H} = \{\mathcal{P}_r, \mathcal{P}_d\}$. For a selected scheduling heuristic, each rule will be randomly picked and a random genetic operation is applied. The subtree crossover creates new rules for the next generation by randomly recombining subtrees from two selected parent rules. The subtree mutation is performed by selecting a node of a chosen rule and replacing the subtree rooted by that node with a newly randomly-generated subtree. Meanwhile, the subtree extraction simply replaces a rule with its random subtree so the size of new rule is always smaller than the parent rule. To improve the diversity of the population, all the same duplicated heuristics (based on phenotypic similarity) will be eliminated before applying the proposed surrogate model [20].

C. Mapping evolutionary process

To efficiently explore competitive scheduling heuristics, it is crucial for PES to capture the topological relations and distributions of previous generated heuristics. Since the heuristics generated by genetic operations are complicated and usually contain redundant components, phenotypic characteristics are more useful than the genotypic characteristics to determine the similarity between generated heuristics. Also, phenotypic characteristics is a good predictor of program fitnesses [20], [21] and the phenotypic diversity has good correlations with the fitness improvements [38]. Therefore, we employ the phenotypic characteristics to map the evolutionary process of PES. There are two main steps in mapping evolutionary process with PES: (1) dimensionality reduction with PCA, and (2) topogical learning with mGNG.

1) Dimensionality reduction: When the number of decision situations or the dimension D increases, we will have a high-dimensional dataset X of decision vectors. High-dimensional data can slow down the learning process, strongly influence the scalability of the proposed algorithm, and may reduce the accuracy of the machine learning techniques. To avoid this problem, a dimensionality reduction technique is needed to preprocess the input data. There have been many dimensionality reduction methods developed in the literature to cope with different types of data. In this paper, we use PCA for dimensionality reduction because it is an efficient and well-established approach. Another reason for using PCA here is the multicollinearity issue in the decision vectors. Since decision vectors are obtained by applying scheduling rules to different random decisions situations, it is not uncommon to encounter similar decision situations which result in similar rankings (jobs with high priorities in one situation is likely to have high priorities in similar situations).

2) Modified growing neural gas: Algorithm 1 shows how mGNG can adapt the network to the input data. All data points in $X = \{phenotype(\mathcal{H}) | \mathcal{H} \in \mathcal{A}\}$ are transformed into X' by PCA. In each epoch, an input x is then sampled from the transformed data X' and mGNG determines, in terms of Euclidean distance, the nearest unit (s_1) and the second nearest unit (s_2) which are adapted towards the input data. An edge connecting these two units is also created (if it has not yet existed) and its age is set to zero. After each epoch, a new unit will be inserted near to the unit with the maximum error in order to reduce the total error of the network. If an unit has a very low utility as compared to the maximum error, it will be removed from the network. This mechanism is proposed in [33] to deal with non-stationary distributions. In PES, removing nodes with low utilities has two main advantages: (1) reducing the computation costs of mGNG, and (2) discarding nodes representing rare or outdated heuristics (i.e. ones that are rarely generated in recent generations). The second advantage is that it helps mGNG naturally track the dynamics of the population in the most recent generations. The parameter θ can be considered as a convenient way to control the resolution of networks. If θ is high, nodes will be unlikely to be removed and mGNG will try to fit the network to the whole dataset. If θ is low, mGNG will focus on more representative programs (frequently reproduced) and main evolutionary traits. The maximum age of edges are set to the size of the dataset to preserve most relations of programs. We also adopt a scheme like *k-means* to adapt the learning rate over time [39], i.e. $e_b = 1/n_{win}$ and $e_n = 100/n_{win}$ (see steps 9–10 in Algorithm 1) where n_{win} is the number of input signals for which the considered node has been a winner.

It is noted that mGNG is applied at the end of each generation. The obtained network $\mathcal{N}$ can show what programs are generated during the evolutionary process and monitor the distributions of the GP population. As compared to existing mapping-based method in the literature such as MAP-Elites [40], mGNG has a number of advantages. First, mGNG does not require any prior knowledge about the phenotype search space, which is important for MAP-Elites to predefine the map size. Second, mGNG produces a network which is more powerful than MAP-Elites in terms of representing the topological relations of generated programs. Finally, mGNG can efficiently control and updating the map (i.e. network) during the evolutionary process. These properties make mGNG an great approach to mapping the explored search space. In the next section, we show how the obtained network can help GP adaptively explore the search space.

Algorithm 1 Modified growing neural gas (mGNG)

Input: dataset X, current network $\mathcal{N}$ (if exist)
Output: (updated) mGNG network $\mathcal{N}$, PCA model

1: $X' \leftarrow$ transform dataset X with PCA
2: $epoch \leftarrow 0$
3: randomly initialise $\mathcal{N}$ with two nodes if $\mathcal{N}$ is empty
4: **repeat**
5: randomly shuffle X'
6: **for** each data point $x \in X'$ **do**
7: $w_{s_1} \leftarrow$ the nearest (winning) node s_1 for input x
8: update the error of s_1: $E_{s_1} = E_{s_1} + ||x - w_{s_1}||^2$
9: updating unit s_1: $w_{s_1} = w_{s_1} + \epsilon_b(x - w_{s_1})$
10: updating neighbour n of s_1: $w_n = w_n + \epsilon_n(x - w_n)$
11: create an edge between s_1 and s_2 (the second-nearest unit to x) and set its age to zero if the edge does not exist; otherwise set the edge's age to zero
12: increment the age of edges connecting s_1 and each neighbor $n \in \mathcal{N}$
13: removing edges with age larger than $a_{max} = |X|$ and nodes with no emanating edges
14: updating the utility of s_1: $U_{s_1} = U_{s_1} + ||x - w_{s_2}||^2 - ||x - w_{s_1}||^2$
15: let q is the unit with maximum error and l is the unit with minimum utility
16: remove the unit l if $E_q/U_l > \theta$
17: **for** each unit $c \in \mathcal{N}$ **do**
18: $E_c = (1 - \beta)E_c$
19: $U_c = (1 - \beta)U_c$
20: insert a new unit to $\mathcal{N}$ between q and its neighbor if the number of nodes in $\mathcal{N} < max\_node$
21: **until** $epoch = maximum\_epoch$ or mGNG *converges*

D. Adaptive surrogate model

When a new heuristic $\mathcal{H} = (\mathcal{P}_r, \mathcal{P}_d)$ is generated by genetic operations, its corresponding phenotypic characteristic $phenotype(\mathcal{H}) = (phenotype(\mathcal{P}_r) \| phenotype(\mathcal{P}_d))$ is determined by using the technique discussed in section III-A2. In the previous study [20], the fitness was estimated by:

$$\hat{f}(\mathcal{H}) = \textit{fitness}\left(\arg\min_{\mathcal{H}' \in \mathcal{A}} \|phenotype(\mathcal{H}) - phenotype(\mathcal{H}')\|\right) \quad (3)$$

In the proposed algorithm, we generalise this surrogate model by adding a *control factor* based on the information from the network $\mathcal{N}$. The new model will estimate the fitness as:

$$\hat{f}_a(\mathcal{H}) = \hat{f}(\mathcal{H}) \times control\_factor(\mathcal{H}, \mathcal{N}) \quad (4)$$

in which:

$$control\_factor(\mathcal{H}, \mathcal{N}) = df(\mathcal{H}, \mathcal{N}) \times \frac{bf(\mathcal{H}, \mathcal{N})}{pf(\mathcal{H}, \mathcal{N})} \quad (5)$$

where $df(\mathcal{H}, \mathcal{N})$, $bf(\mathcal{H}, \mathcal{N})$, and $pf(\mathcal{H}, \mathcal{N})$ are three key factors to control the behaviours of PES in terms of diversity, sizes of evolved heuristics, and user preferences. If $control\_factor(\mathcal{H}, \mathcal{N}) = 1$, the proposed model will be the same as the surrogate model in [20] because $\hat{f}_a(\mathcal{H}) = \hat{f}(\mathcal{H})$. Heuristics with a higher $control\_factor(\mathcal{H}, \mathcal{N})$ will be less likely to be selected.

1) Diversity control: PES uses a simple strategy to control the diversity of evolved heuristics, i.e. reduce the chance to select heuristics located in a well-explored area of the search space. Since the distribution of evolved heuristics has been captured by mGNG, we can easily determine the density of a certain area in the search space and use it to adjust the estimate

fitness. Given a newly generated heuristic $\mathcal{H}$, a diversity factor is determined:

$$df(\mathcal{H},\mathcal{N}) = \begin{cases} \frac{nd(\mathcal{H},\mathcal{N})}{T} & nd(\mathcal{H},\mathcal{N}) > T \\ 1 & otherwise \end{cases} \quad (6)$$

where $nd(\mathcal{H},\mathcal{N})$ is the *neighbourhood density* of the heuristic $\mathcal{H}$ given the network $\mathcal{N}$ and T is a predefined diversity threshold ($0 < T \leq 1$). To calculate $nd(\mathcal{H},\mathcal{N})$, $phenotype(\mathcal{H})$ is first transformed by the PCA model obtained from the Algorithm 1 and the node $n_{\mathcal{H}}^{\mathcal{N}*}$ in $\mathcal{N}$ with the minimum distance to the transformed $phenotype(\mathcal{H})$ is determined. Then, the neighbourhood density is calculated as:

$$nd(\mathcal{H},\mathcal{N}) = \frac{1}{|\mathcal{A}|}\left(freq(n_{\mathcal{H}}^{\mathcal{N}*}) + \sum_{n \in NB(n_{\mathcal{H}}^{\mathcal{N}*})} freq(n)\right) \quad (7)$$

where $freq(n)$ is the matching frequency of node $n \in \mathcal{N}$ determined by the number of entries in $\mathcal{A}$ that best match node n (noted that $|\mathcal{A}| = \sum_{n' \in \mathcal{N}} freq(n')$). Meanwhile, $NB(n_{\mathcal{H}}^{\mathcal{N}*})$ is the set of neighbours of $n_{\mathcal{H}}^{\mathcal{N}*}$, i.e. basically any nodes in $\mathcal{N}$ that are directly connected to $n_{\mathcal{H}}^{\mathcal{N}*}$ by an edge. A higher neighbourhood density indicates that there have been more heuristics generated within the neighbourhood of the considered heuristic $\mathcal{H}$.

From equations (4)–(6), it is easy to see that the new surrogate model gives a heuristic $\mathcal{H}$ a worse fitness (higher value) if it is located in the area with the neighbourhood density greater than T. This mechanism is created to discourage GP from guiding its search towards too crowded areas. If T is high, PES will focus on areas where good heuristics have been identified. If T is extremely small, PES will guide its search to explore all areas in the search space without strong emphases on the fitnesses. In other words, PES will explore the search space more aggressively with a smaller T and the exploitation level of GP will be enhanced if a high T is used. However, it should be noted that T provides a convenient way for GP to adapt its search during the evolutionary process rather than predefine a fix behaviour (e.g. with tournament size for tournament selection). At the beginning of the evolution, T will have less impact on the approximate fitnesses because heuristics are randomly generated, i.e. the density of each node in the network $\mathcal{N}$ will be similarly low. In this case, $\hat{f}(\cdot)$ will play an important role to accelerate the search towards heuristics with higher approximate fitnesses as long as better and new heuristics are found, i.e. more exploitation. However, when PES stops progressing in a few generations, the neighbourhood density the areas where top evolved heuristics are located will increase due to the selection mechanism that favours high performing heuristics (see Fig. 1). If the neighbourhood density rises above the threshold level T, PES will guide the search towards heuristics with slightly worse fitnesses but in the less explored areas. Depending on the evolutionary progress, the proposed PES algorithm will adaptively adjust its exploration and exploitation intensity.

2) Bloat control: The most common approach to control bloat issue in GP is to predefine the maximum length, the maximum depth, or the penalty parameter to control sizes of evolved programs. However, without prior knowledge about the problem, it is too hard to determine these parameters with good values. To control bloat in PES, we further take advantage of the obtained network $\mathcal{N}$. Given a newly generated heuristic $\mathcal{H}$, a bloat factor is determined:

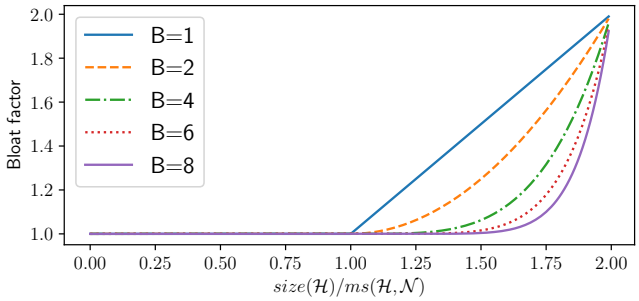


Fig. 2. Bloat factor curves.

$$bf(\mathcal{H},\mathcal{N}) = \begin{cases} 1 + \left(\frac{size(\mathcal{H})}{ms(\mathcal{H},\mathcal{N})} - 1\right)^B & \frac{size(\mathcal{H})}{ms(\mathcal{H},\mathcal{N})} > 1 \\ 1 & otherwise \end{cases} \quad (8)$$

where $size(\mathcal{H})$ is the size of heuristic $\mathcal{H}$ (total number of terminals and functions) and $ms(\mathcal{H},\mathcal{N})$ is the maximum size of heuristics $\mathcal{H}' \in \mathcal{A}$ that best matched node $n_{\mathcal{H}}^{\mathcal{N}*}$ (defined in the previous section). If $size(\mathcal{H}) \leq ms(\mathcal{H},\mathcal{N})$, the bloat factor will have no impact on the estimated fitness of $\mathcal{H}$. Otherwise, the parameter $B \geq 1$ governs the tradeoffs between the sizes of the evolved heuristics and their quality. The bloat factor curves with different B values are shown in Fig. 2. If B is higher, PES will put a higher priority on the fitnesses (a lower bloat factor) and a lower priority on the heuristic sizes. If B is low, PES will focus on generating compact rules. Similar to diversity control, this bloat control is also adaptive to the evolutionary progress, i.e. larger heuristics are only preferred when they have better fitnesses and there is no similar heuristic previously generated. It should be noted that B will have a reverse impact on the bloat factor is $size(\mathcal{H})/ms(\mathcal{H},\mathcal{N}) > 2$, i.e. a higher B will control the sizes of newly generated heuristics more aggressively if the their sizes $size(\mathcal{H})$ are double the maximum size of heuristics $ms(\mathcal{H},\mathcal{N})$. This property helps PES avoid generating heuristics with too many redundant components in one generation which are likely to be included in the parent heuristics if a higher B value is used.

3) User preferences: PES allows the users and decision makers to interact with evolutionary process by selecting the areas in the search space that they want PES to explore. This property is useful if there has been already an effective heuristic applied in the production system and the decision makers only want to make some improvements without dramatically changing the current scheduling practice. Another useful case is when the decision makers want to find a good tradeoff between program sizes and complexity on the fly rather than using a predefined objective function. With the obtained network $\mathcal{N}$, the decision makers can easily guide the search of PES simply by selecting a node or a set of nodes, corresponding to specific areas in the search space, to further explore. When the network $\mathcal{N}$ is generated or updated, the default preference for each node n is $pref(n) = 1$.

The decision makers can assign a particular preference value to each node within the range $[1, maxp]$. Given a newly generated heuristic $\mathcal{H}$, a preference factor is determined:

$$pf(\mathcal{H}, \mathcal{N}) = 1 + \frac{pref(n_{\mathcal{H}}^{\mathcal{N}*}) - 1}{maxp - 1} \quad (9)$$

From equation (5) and equation (9), it is clear that PES will select heuristics located near to nodes with high preference factors because of high preference values $pref(n_{\mathcal{H}}^{\mathcal{N}*})$. The users can select and increase the preference values of one or multiple nodes in $\mathcal{N}$ and higher preference values should be assigned to more desirable nodes (e.g. similar to certain rules or explainable rules). The users may choose to interact with PES at the end of each generation or set up a periodic review depending on the speed of the evolutionary process.

In general, PES is developed to overcome a number of key challenges in existing GP algorithms. First, PES can efficiently capture the distributions of evolved heuristics across generations with the mGNG network. This network can provide useful aggregate information about the evolutionary progress. Second, PES provides a convenient way to adaptively control the diversity based on the evolutionary progress rather than simply applying random genetic operations, which is inefficient and sensitive to predefined parameters such as tournament size, crossover probability, and mutation probability. Third, PES can adaptively control the sizes of newly generated heuristics based on the sizes of their similar heuristics evolved in the previous generations instead of explicitly fixing the maximum size of evolved heuristics. Finally, PES has a visualisation and interaction component that allows users to monitor and guide the evolutionary process. These properties help PES discover scheduling heuristics more efficiently, especially for highly complex environments with multiple decisions such as DFJSS.

IV. Experiment Settings

This section describes the simulation configurations, parameter settings, and performance metrics used to evaluate the performance of the proposed PES.

A. Simulation configurations

In our experiments, six different simulation scenarios based on two utilisation levels ($\rho = 85\%$ and 95%) and three levels of flexibility ($f\% = 30\%$, 50%, and 70%) are used to examine the performance of PES. PES will be compared to a simple genetic programming (referred to as GP) algorithm [41] and the surrogate-assisted GP (simply referred to as SGP) [20]. For the three algorithms, an independent simulation replication is used to calculate the fitness (i.e. total weighted tardiness in equation(2)) in each generation as this strategy shows significant improvements in previous studies [41], [42]. For testing, the quality of the best heuristics evolved by the three algorithms are based on the performance obtained with 50 simulation replications and $|\mathbb{C}| = 5000$. Common random number [29] is used as the variance reduction technique in our experiments for both DFJSS simulation and population initialisation. To reduce the computational cost, the simulation will terminate early if the number of jobs in the shop exceeds 500, i.e. the shop is overloaded [41].

TABLE III
Parameter Settings

Parameter	GP	SGP	PES
Population size	200		
Initialisation	ramp-half-and-half [36]		
Tournament size	5		
Maximum generation	50		
Maximum depth	17	17	–
Crossover rate	80%	80%	–
Mutation rate	15%	20%	–
Reproduction rate	5%	–	–
# of decision situations D	–	200	200
Size of intermediate population	–	2000	2000
# of principal components K	–	–	5
Diversity threshold T	–	–	0.05
Bloat control parameter B	–	–	1, 8

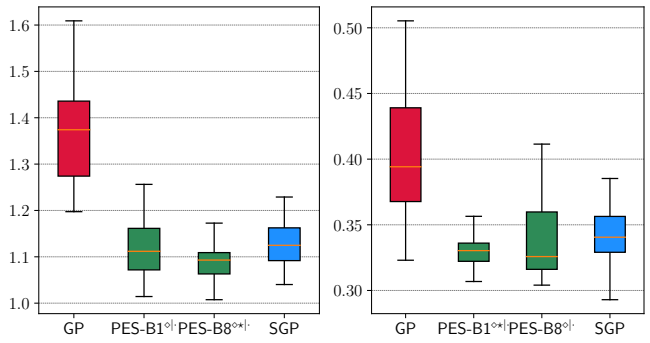
B. Parameter settings

The default parameters for the three algorithms are shown in Table III. All algorithms use the same terminal set and function set in Table I and Table II. The maximum number of nodes in mGNG is 500. The utility factor θ is 50 (to ensure that representative programs generated during the evolution are captured) and the maximum number of epoches to learn the mGNG networks is 1000 or when the algorithm is converged, i.e. the total error is not improved in 5 epoches. These parameters are selected via preliminary experiments. It is noted that the mGNG algorithm is very efficient and does not show significant impacts on the running times of PES. As compared to the simulation time used to evaluate heuristics, the mapping time is negligible. The preference parameter $maxp$ for interactions is 4. For SGP and PES, the phenotype of each rule in an evolved heuristic is determined by 100 decision situations ($D = 2 \times 100$) and with a reference heuristic using *least waiting time* (LWT) [43] as the routing rule and 2PT + WINQ + NPT [34] as the dispatching rule.

C. Performance metrics

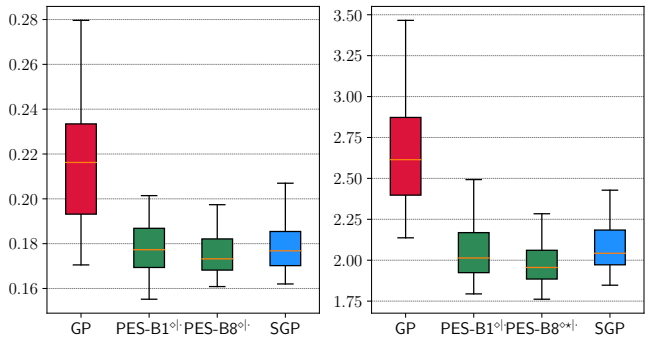
For comparisons, 30 independent runs of GP, SGP, and PES are performed and their best evolved heuristics are recorded. The Wilcoxon test with $\alpha = 0.05$ is used for statistical significance tests of the *test performance* and *sizes* of evolved heuristics. The test performance of an evolved heuristic is the average TWT from 50 independent simulation replications. For these two performance metrics, the lower the better.

We also examine the behaviours of the three algorithms in terms of the *training fitnesses*, the *number of components* in the network $\mathcal{N}$, and the *number of nodes* in $\mathcal{N}$. It should be noted that the mGNG networks $\mathcal{N}$ are only used to capture the evolutionary patterns of GP and SGP and do not influence their search mechanisms. The number of connected components shows the connectivity of evolved heuristics across generations. The connectivity can help explain the behaviours of each algorithm. Meanwhile, the number of nodes represents the diversity of evolved heuristics. If an algorithm converges to a certain area in the search space, the number of nodes will be low. If an algorithm continues to explore different areas in the search space, the number of nodes will be high.



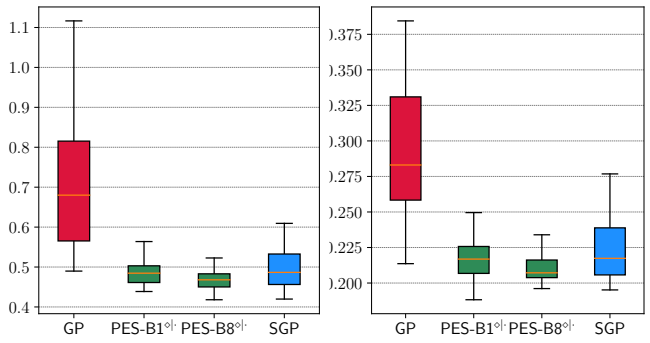
(a) $\langle 85\%, 30\% \rangle$

(b) $\langle 85\%, 50\% \rangle$



(c) $\langle 85\%, 70\% \rangle$

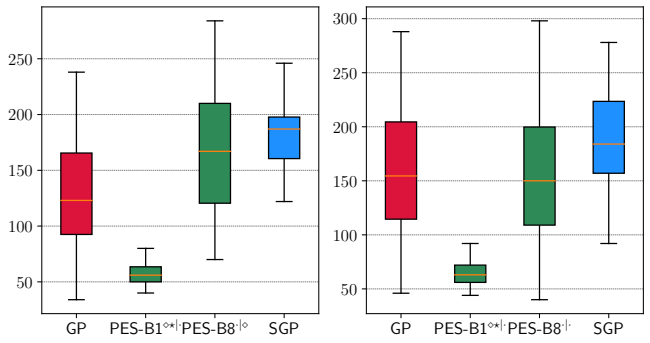
(d) $\langle 95\%, 30\% \rangle$



(e) $\langle 95\%, 50\% \rangle$

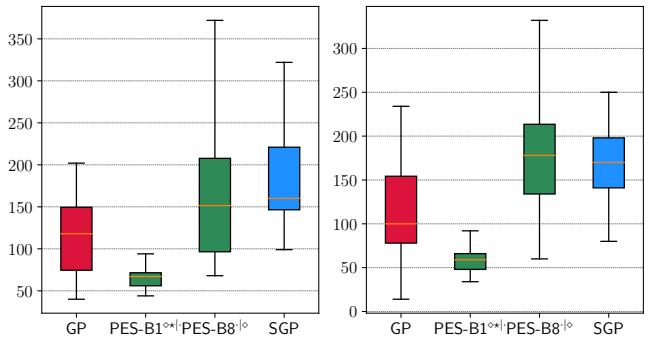
(f) $\langle 95\%, 70\% \rangle$

Fig. 3. Test performance (average TWT of evolved heuristics.



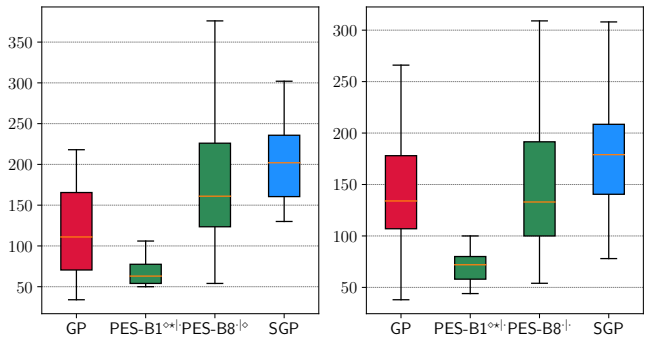
(a) $\langle 85\%, 30\% \rangle$

(b) $\langle 85\%, 50\% \rangle$



(c) $\langle 85\%, 70\% \rangle$

(d) $\langle 95\%, 30\% \rangle$



(e) $\langle 95\%, 50\% \rangle$

(f) $\langle 95\%, 70\% \rangle$

Fig. 4. Sizes of evolved heuristics.

V. Computational Results

This section presents the experimental results for PES, GP, and SGP. First, these three algorithms are compared in terms of the test performance and heuristic sizes. Then, the test results on unseen scenarios are provided to investigate their generalisation. Finally, we investigate the behaviours of the algorithms using visualisation and illustrate how users can intervene the evolutionary process.

A. Test performance

The test performance of the three algorithms are shown in Fig. 3. PES-B1 and PES-B8 represent the PES algorithm using the default parameters in Table III with $B = 1$ and $B = 8$ respectively. For each PES variant, a notation $l|r$ is used show the outputs of statistical tests. l and r are the algorithms, i.e. $\diamond$ for GP and $\star$ for SGP, which are significantly worse (or better) than a PES variant. In all six simulation scenarios, PES variants are significantly better than GP. Meanwhile, PES variants are either significantly better than or not significantly different from SGP in all scenarios. PES-B8 has a slightly lower variances as compared to PES-B1 and SGP. In general, PES shows consistent and superior performance in a wide range of scenarios with different levels of flexibility and utilisation.

B. Program size

Fig. 4 shows the sizes of the best heuristics evolved by PES, GP, and SGP. PES-B1 obviously produces the most compact heuristics in all scenarios. Heuristics evolved by PES-B1 are

TABLE IV
TEST PERFORMANCE OF EVOLVED SCHEDULING HEURISTICS (RESULTS IN THE FORM OF MEDIAN [MINIMUM, MAXIMUM])

Test Scenario	Trained with $\langle 85\%, 30\% \rangle$				Trained with $\langle 95\%, 30\% \rangle$			
	GP	PES-B1	PES-B8	SGP	GP	PES-B1	PES-B8	SGP
$\langle 0.85\%, 30\% \rangle$	1.37 [1.20,1.78]	**1.11 [1.01,1.26]**	***1.09 [1.01,1.21]***	1.12 [1.04,1.32]	1.51 [1.24,2.33]	**1.16 [1.03,1.43]**	***1.13 [1.03,1.28]***	1.19 [1.06,1.38]
$\langle 0.95\%, 30\% \rangle$	2.45 [2.17,3.32]	**2.02 [1.79,2.35]**	***2.01 [1.75,2.19]***	2.04 [1.86,2.53]	2.61 [2.14,3.47]	**2.01 [1.79,2.49]**	***1.96 [1.76,2.33]***	2.04 [1.85,2.43]
$\langle 0.85\%, 50\% \rangle$	0.46 [0.37,0.67]	**0.34 [0.30,0.40]**	***0.33 [0.30,0.42]***	0.35 [0.31,0.46]	0.58 [0.40,1.22]	**0.39 [0.32,0.49]**	***0.37 [0.32,0.47]***	0.41 [0.32,0.53]
$\langle 0.95\%, 50\% \rangle$	0.75 [0.54,1.18]	**0.50 [0.42,0.62]**	***0.48 [0.42,0.63]***	0.51 [0.45,0.73]	0.90 [0.60,1.55]	**0.57 [0.46,0.77]**	***0.52 [0.44,0.70]***	0.61 [0.46,0.81]
$\langle 0.85\%, 70\% \rangle$	0.27 [0.20,0.41]	**0.20 [0.16,0.25]**	***0.19 [0.17,0.26]***	0.20 [0.17,0.27]	0.36 [0.23,0.89]	**0.22 [0.18,0.28]**	***0.22 [0.17,0.29]***	0.24 [0.18,0.32]
$\langle 0.95\%, 70\% \rangle$	0.38 [0.27,0.65]	**0.25 [0.19,0.31]**	***0.23 [0.19,0.34]***	0.25 [0.21,0.38]	0.51 [0.30,1.04]	**0.29 [0.22,0.40]**	***0.28 [0.21,0.40]***	0.32 [0.22,0.46]
Test Scenario	Trained with $\langle 85\%, 50\% \rangle$				Trained with $\langle 95\%, 50\% \rangle$			
	GP	PES-B1	PES-B8	SGP	GP	PES-B1	PES-B8	SGP
$\langle 0.85\%, 30\% \rangle$	1.43 [1.16,2.66]	**1.26 [1.11,1.47]**	**1.28 [1.05,1.57]**	1.27 [1.11,1.62]	1.44 [1.23,1.81]	**1.19 [1.09,1.59]**	**1.17 [1.04,1.30]**	1.22 [1.06,1.37]
$\langle 0.95\%, 30\% \rangle$	2.98 [2.25,8.29]	**2.58 [2.16,3.49]**	**2.60 [1.98,3.54]**	2.67 [2.14,3.46]	2.81 [2.43,3.81]	**2.34 [2.06,3.20]**	**2.26 [1.84,2.70]**	2.33 [2.05,3.06]
$\langle 0.85\%, 50\% \rangle$	0.39 [0.32,1.00]	***0.33 [0.30,0.38]***	**0.33 [0.30,0.41]**	0.34 [0.29,0.43]	0.43 [0.34,0.65]	**0.33 [0.31,0.42]**	**0.32 [0.30,0.37]**	0.33 [0.28,0.44]
$\langle 0.95\%, 50\% \rangle$	0.63 [0.49,2.78]	**0.50 [0.45,0.60]**	**0.51 [0.45,0.66]**	0.53 [0.43,0.71]	0.68 [0.49,1.12]	**0.48 [0.44,0.64]**	**0.47 [0.42,0.57]**	0.49 [0.42,0.65]
$\langle 0.85\%, 70\% \rangle$	0.21 [0.18,0.58]	**0.18 [0.15,0.21]**	**0.18 [0.16,0.22]**	0.18 [0.16,0.23]	0.24 [0.18,0.38]	**0.18 [0.17,0.22]**	**0.18 [0.16,0.20]**	0.18 [0.16,0.25]
$\langle 0.95\%, 70\% \rangle$	0.28 [0.22,1.41]	***0.22 [0.19,0.26]***	**0.22 [0.20,0.29]**	0.23 [0.20,0.32]	0.32 [0.22,0.60]	**0.22 [0.20,0.29]**	**0.22 [0.19,0.26]**	0.22 [0.18,0.33]
Test Scenario	Trained with $\langle 85\%, 70\% \rangle$				Trained with $\langle 95\%, 70\% \rangle$			
	GP	PES-B1	PES-B8	SGP	GP	PES-B1	PES-B8	SGP
$\langle 0.85\%, 30\% \rangle$	1.64 [1.30,2.63]	**1.43 [1.10,1.77]**	**1.38 [1.19,1.79]**	1.42 [1.17,1.87]	1.52 [1.26,2.22]	**1.35 [1.17,1.65]**	**1.30 [1.19,1.80]**	1.30 [1.14,1.59]
$\langle 0.95\%, 30\% \rangle$	3.59 [2.65,6.28]	**3.16 [2.12,4.23]**	**3.06 [2.30,4.33]**	3.18 [2.19,5.06]	3.07 [2.50,5.64]	**2.79 [2.19,5.82]**	2.81 [2.40,4.09]	2.65 [2.16,3.55]
$\langle 0.85\%, 50\% \rangle$	0.43 [0.33,0.69]	**0.35 [0.30,0.44]**	**0.33 [0.31,0.41]**	0.35 [0.31,0.43]	0.40 [0.32,0.60]	**0.33 [0.30,0.41]**	**0.32 [0.30,0.44]**	0.33 [0.29,0.39]
$\langle 0.95\%, 50\% \rangle$	0.73 [0.52,1.32]	**0.57 [0.44,0.74]**	**0.54 [0.46,0.72]**	0.57 [0.47,0.78]	0.67 [0.46,1.04]	**0.52 [0.44,0.66]**	**0.50 [0.45,0.73]**	0.50 [0.45,0.65]
$\langle 0.85\%, 70\% \rangle$	0.22 [0.17,0.33]	**0.18 [0.16,0.20]**	**0.17 [0.16,0.24]**	0.18 [0.16,0.21]	0.21 [0.17,0.34]	**0.17 [0.15,0.22]**	***0.17 [0.16,0.21]***	0.18 [0.16,0.20]
$\langle 0.95\%, 70\% \rangle$	0.30 [0.21,0.54]	**0.23 [0.19,0.29]**	**0.22 [0.19,0.32]**	0.23 [0.20,0.30]	0.28 [0.21,0.52]	**0.22 [0.19,0.29]**	**0.21 [0.20,0.27]**	0.22 [0.20,0.28]

roughly twice smaller than GP and three times smaller than SGP. In addition, the variances of heuristic sizes for PES-B1 are also much smaller than those for other algorithms. Given than PES-B1 is better than SGP in terms of the test performance, it is safe to conclude that PES-B1 is a superior algorithm to evolve scheduling heuristics for DFJSS. Although PES-B8 is slightly better than PES-B1 in terms of the test performance, PES-B8 evolves much larger heuristics than PES-B1.

C. Generalisation

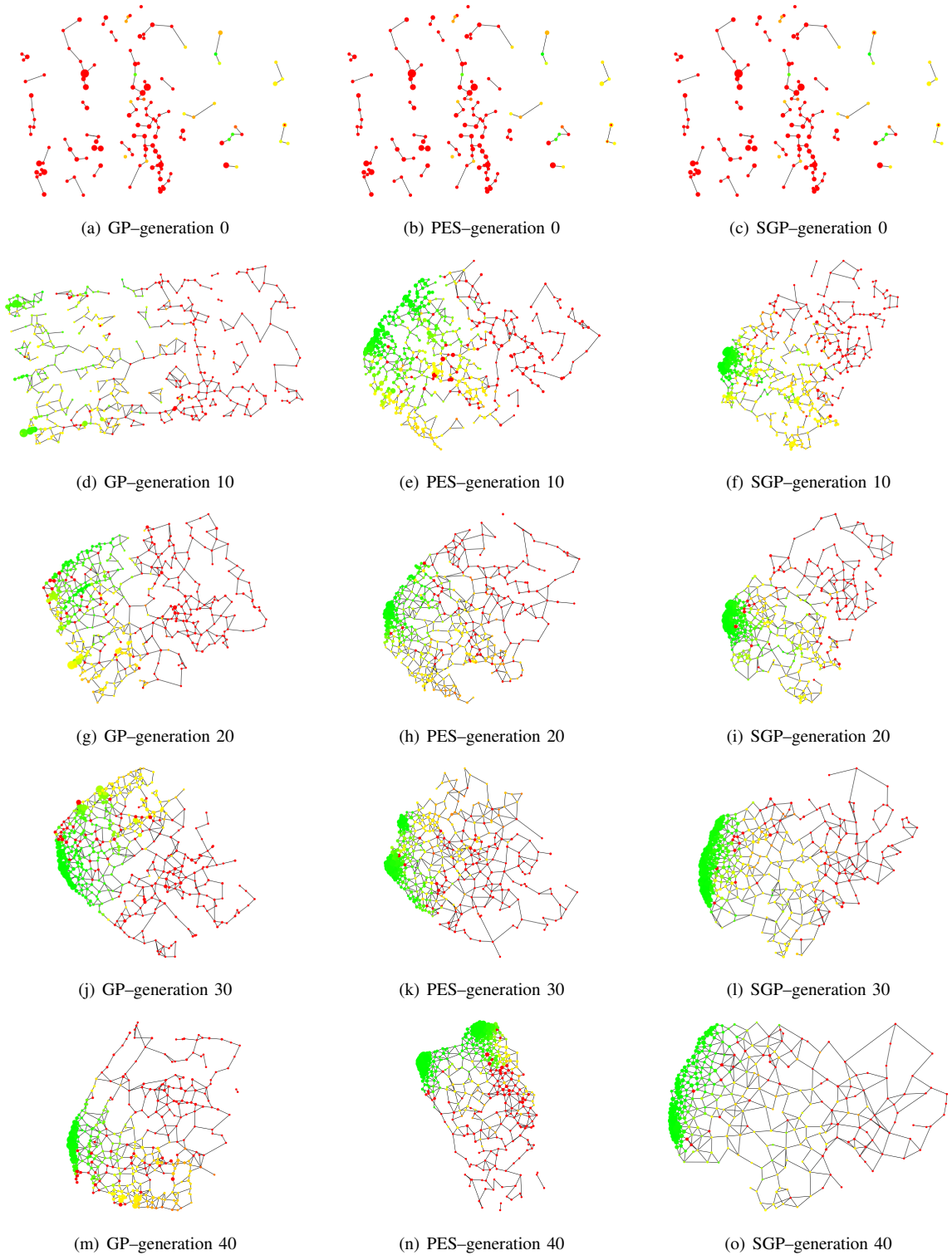
To examine the generalisation of the three algorithms, we show the detailed results for unseen scenarios in Table IV. The results of PES are **bold** if they are significantly better than GP and *italic* if they are better than SGP (***bold-italic*** if PES is significantly better than both GP and SGP). In general, PES and SGP are better than GP in the scenarios used for training and are able to maintain their dominance in unseen scenarios. These results showed that PES, especially the PES-B8 variant, and SGP are not overfitted to the training scenarios. It means that loosely controlling bloat with a high B does not have a negative impact on the generalisation of PES.

In terms of training scenarios, heuristics evolved with $\langle 85\%, 30\% \rangle$ are most effective across scenarios. The main reason is that a low flexibility and a low utilisation force the algorithms to evolve dispatching rules and routing routes that assigned accurate priorities to effectively utilise machines. For a high flexibility, an operation will have many options and it is easy to find a reasonably good routing rule. Similarly, for a high utilisation, simple time-based dispatching rules such as shortest processing time (SPT) and 2PT+WINQ+NPT perform quite well and their variances can also easily be discovered. Therefore, heuristics evolved with a high utilisation or a high flexibility may not generalise well to unseen scenarios.

D. Visualisation

With the mGNG, PES has a built-in ability to visualise the evolutionary process. Fig. 5 show the visualisation of PES with $B = 8$ and $K = 2$, GP, and SGP through generations of a particular run for the scenario $\langle 95\%, 30\% \rangle$. It is noted that a higher K value can be used but it will lead to a complex mGNG network which is hard to understand. Each node in a network visualised in Fig. 5 is a node in the obtained mGNG network at the end of each generation. The position of a node is determined using the weights of that node. The node colour (from red to green) and node size represent the average fitness and the number of all heuristics $\mathcal{H} \in \mathcal{A}$ that are nearest to that node. Red (green) nodes represent the area with bad (good) heuristics.

After initialisation, all algorithms have the same population (the same network), in which heuristics are scatter randomly. After a few generations, we start to see the difference in their evolutionary processes. At generation 10, while GP still randomly explores the search space (small nodes scattered across the network), PES and SGP have a focus on a certain area and the diversity in PES is slightly higher than that of SGP. At generation 20, GP starts to narrow down its search but it evolves a mix of both good and bad heuristics. PES also narrows its search but in a more efficient way, i.e. focusing on good heuristics (a green area on the left of the network). SGP does not change significantly. At generation 30, GP shows a tendency to converge to the left area. SGP still focus on one



(a) GP–generation 0 (b) PES–generation 0 (c) SGP–generation 0

(d) GP–generation 10 (e) PES–generation 10 (f) SGP–generation 10

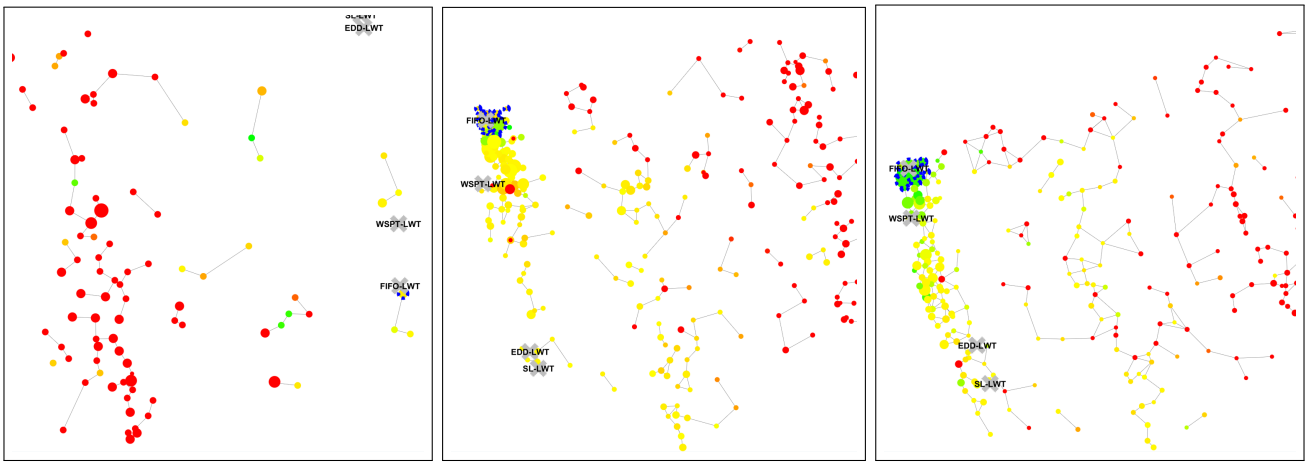
(g) GP–generation 20 (h) PES–generation 20 (i) SGP–generation 20

(j) GP–generation 30 (k) PES–generation 30 (l) SGP–generation 30

(m) GP–generation 40 (n) PES–generation 40 (o) SGP–generation 40

Fig. 5. Visualisation of the evolutionary process. The node colour and node size shows the average fitness and the density for each corresponding node. The network rotation through generation indicates a genetic drift, i.e. the changes in the distributions of generated heuristics.

area but slightly expand the search. PES, on the other hand, starts to form two clusters in the network. At generation 40, GP and SGP converges to a particular in their networks but PES converges to two distinct areas represented by the two large clusters. This visualisation suggests that SGP and PES are much more efficient than GP as they can identify and evolve good heuristics rather than random heuristics. PES, with the ability to maintain its diversity and efficiently explore the search space, shows a better adaptive ability than SGP which highly restricts its exploration and loses its diversity quickly in latter generations (the number of nodes are reduced).



(a) Generation 0:
$\mathcal{P}_d = \max(rRJ, W)$
$\mathcal{P}_r = ((rPT - NON) - (NON - NON))$

(b) Generation 3:
$\mathcal{P}_d = W$
$\mathcal{P}_r = (rPT - WLN)$

(c) Generation 6:
$\mathcal{P}_d = (W * div(W, PR))$
$\mathcal{P}_r = (rPT - WLN)$

Fig. 6. Visualisation of the evolutionary process. For better presentation, only the areas with high fitnesses in the mGNG network are shown. The gray crosses represent benchmark rules. The blue boundaries indicates that the nodes are preferred. $div(\cdot, \cdot)$ is the protected division.

E. Human-guided evolutionary process

To illustrate how a decision maker can intervene the evolutionary process of PES. We conduct a small experiment in which the decision maker can interact with the mGNG network and decide which nodes or areas that PES should explore further. To make the visualisation easier to intepret, we use four benchmark scheduling heuristics: (1) FIFO-LWT (first-in-first-out for the dispatching rule and least-waiting time for the routing rule), (2) WSPT-LWT (weighted shortest processing time for the dispatching rule and LWT), (3) EDD-LWT (earliest due date for the dispatching rule and LWT), and (4) SL-LWT (minimum slack for the dispatching rule and LWT). Their phenotypic characteristics are obtained and used to identify their position in the network (gray crosses). In this case, we assume that the production system is currently employing FIFO-LWT because of its simplicity and the decision maker wants to make some improvements without making significant changes in the scheduling behaviours. To achieve this goal, the decision maker will interact with the mGNG network at the end of each generation and select the nodes nearest to FIFO-LWT and adjust their preference values (see Section III-D3) for PES to explore.

Fig. 6 shows the outcomes of the interaction. With the inputs of the decision makers, PES narrows its search to areas close to FIFO-LWT. In the caption of each sub figure, we also show the best-so-far scheduling heuristics. In the initial generation, the best heuristic is quite random. At generation 3, after two rounds of interaction, the best heuristic has changed to $\mathcal{P}_d = W$ and $\mathcal{P}_r = (rPT - WLN)$. This heuristic is actually a variant of FIFO-LWT. With $\mathcal{P}_d = W$, jobs are basically prioritised as FIFO if they have the same weight. Similarly, $\mathcal{P}_r = (rPT - WLN)$ will behave like LWT if the term rPT is removed. Having rPT will help the routing rule break the tie when the considered machine have the same workload. At generation 6, a better heuristic is identified. The new heuristic has the same routing rule as one found at generation 3 but its dispatching rule is a variant of WSPT. It is reasonable as PES has explored a new good area between FIFO-LWT and WSPT-LWT. This experiment demonstrates that it is possible to include decision makers' inputs on the fly by using the proposed PES. This property is very useful in the case that the user or decision maker wants to navigate the evolutionary process to find the solutions of interest without hardcoding the search algorithm or the fitness function.

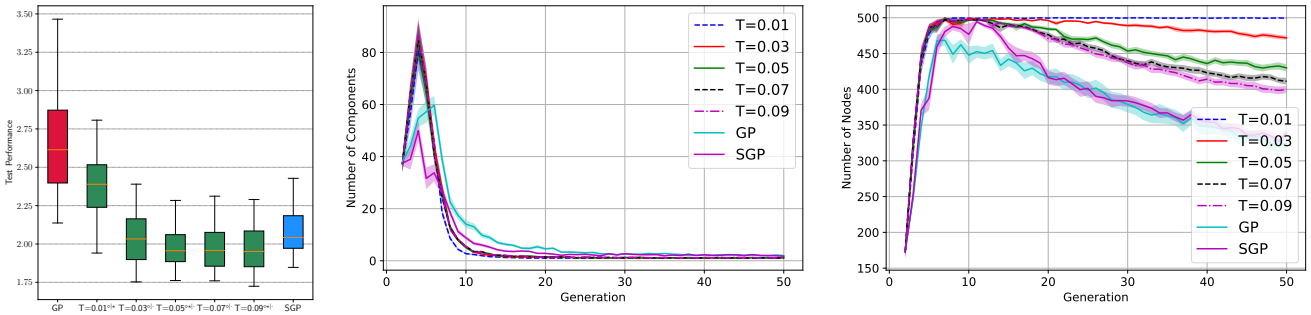
VI. Further Analyses

The previous section has shown effectiveness and capability of PES. In this section, we further analyses the key parameters of PES to understand how they influence the evolutionary process of PES. Because of the space limitation, we only present the results for the scenario $\langle 95\%, 30\% \rangle$. Similar patterns are also observed for other scenarios.

A. Influence of diversity control

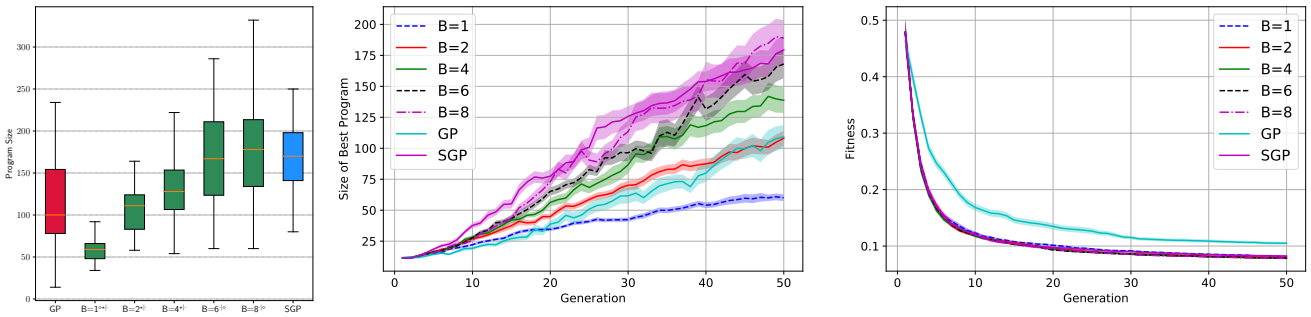
The threshold T is key parameter of PES to control the diversity of PES across generations. Fig. 7(a) shows the test performance of evolved heuristics when T increases from 0.01 to 0.09. It is easy to see that setting T too low has a negative effect on the test performance as PES tries too hard to maintain the diversity rather than boosting the quality of evolved heuristics. However, with the lowest $T = 0.01$, PEs is still better than GP. The T values from 0.05 to 0.09 seems to be consistent. As SGP is a special case of PES with $T = 1$ (without bloat control and user preferences), it is reasonable to say that using a high T value is also undesirable.

Fig. 7(b) and Fig. 7(c) show the number of connected components and the number of nodes in the obtained mGNG network. it is easy to see that PES has a different behaviour as compared to GP and SGP. A common observation is that the numbers of connected component increase quickly in the first few generations and then converge to 1 (i.e. each node has at least one connection). However, GP and SGP are slower than



(a) Test performance (b) Number of network components (c) Number of network nodes

Fig. 7. Influence of diversity control.



(a) Sizes of final heuristics (b) Sizes of best-so-far heuristics (c) Best fitness

Fig. 8. Influence of bloat control.

PES in terms of the increasing speed and the convergence speed. There is no significant differences in terms of the number of connected components when different T values are applied. The differences in terms the number of nodes are much more obvious. All algorithms search unexplored areas very quickly and reach the maximum number of nodes (500) in the first few generations. Then most algorithms except for PES with $T = 0.01$ starts to lose their diversity. As expected, a low T helps PES maintain a high diversity across generations. GP and SGP lose their diversity very quickly as compared to PES. These results confirm that T is a very useful and predictable parameter to control the population diversity.

B. Influence of bloat control

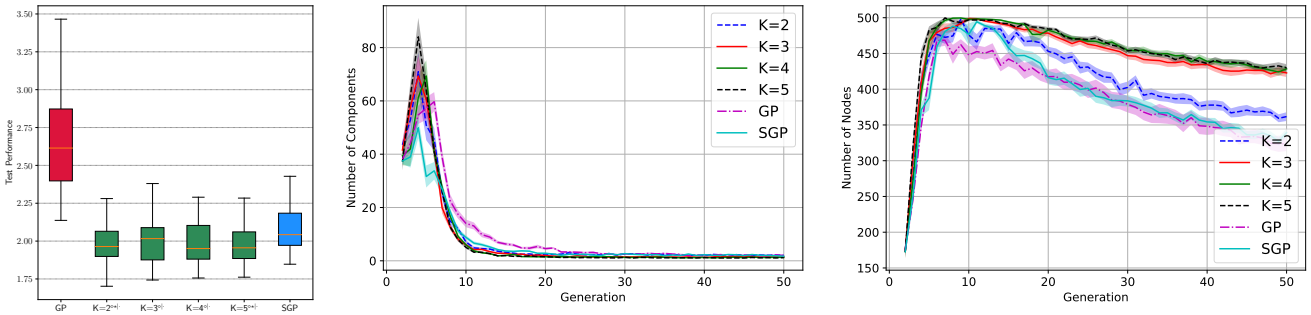
The ability to produce compact heuristic is an attractive property of PES. Fig. 8(a) shows that the sizes of evolved heuristics increase very quickly when B increases from 1 to 8. When $B \geq 4$, PES behaves like SGP. This can be further observed in Fig. 8(b). Clearly, PES with $B = 1$ is best at control the heuristic sizes when the average sizes of its best heuristics only increase from 10 to more than 50. Meanwhile, the sizes of final heuristics evovled by PES with $B = 8$ and SGP are about 20 times larger than their initial heuristics. PES with $B = 2$ is similar to GP in terms of heuristic sizes. It should be noted that the best fitness of all algorithms except for GP are not significantly as shown in Fig. 8(c) although the heuristic sizes are very different.

C. Influence of dimensionality reduction

As dimensionality reduction plays an important role to improve the efficiency of mGNG, it is important to see if they have a negative impact on the efficiency of the overall PES algorithm. Fig. 9(a) show the test performance of PES when different numbers of principal components K are applied. The results show that there is no significant difference in terms of the test performance when different K values are used. K also does not influence the number of connected components as shown in Fig. 9(b). However, the diversity seems to be reduced when a smaller K is used as presented in Fig. 9(c). One explanation for this phenomenon is that a lower K will reduce the discrimination ability when mGNG tries to learn the topological relations of evolved heuristics. Therefore, PES cannot effectively see the difference in terms of phenotypic characteristics of evolved heuristics to accurately determine their estimate fitnesses.

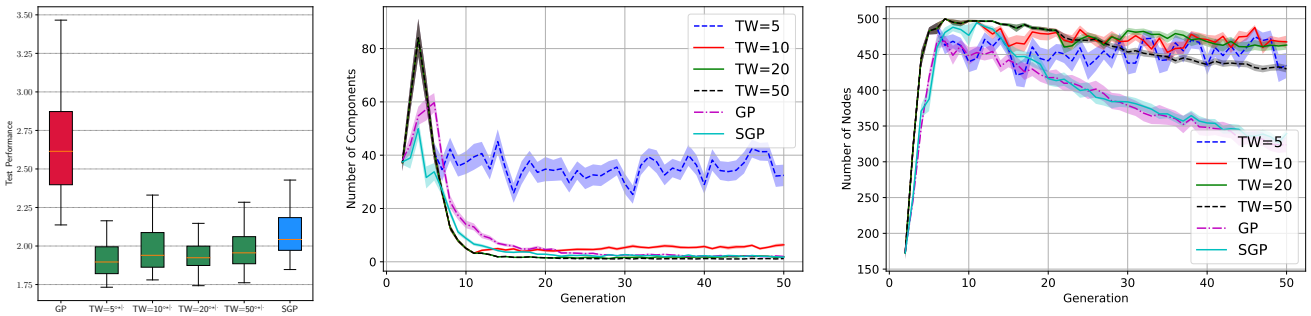
D. Influence of time window

The archive $\mathcal{A}$ is needed for mGNG to map the explore areas in the search space. However, keeping all evolved heuristics in $\mathcal{A}$ may reduce the efficiency of mGNG and PES. In this section, we will examine the recency of the archive by the restricting the number of recorded heuristics only to the most recent generation, i.e. time window TW. Fig. 10(a) shows the test performance when different TW are used (e.g. $TW = 5$ means that only heuristics evolved in the last 5 generations



(a) Sizes of final heuristics (b) Number of network components (c) Number of network nodes

Fig. 9. Influence of dimensionality reduction.



(a) Test performance (b) Number of network components (c) Number of network nodes

Fig. 10. Influence of time window.

are kept in $\mathcal{A}$). The results show that PES is not sensitive to TW. In Fig. 10(b), it is interesting to see that the number of components are very high for $TW = 5$ and this number is significantly reduced when $TW = 10$. The reason is that the number of evolved heuristics in $\mathcal{A}$ is not large enough for mGNG to form a well connected network. Fig. 10(c) shows that the number of nodes with $TW = 5$ is still higher than those of GP and SGP although their variances are much higher (because there are significant changes in $\mathcal{A}$ in each generation). These results suggest that TW can be significantly reduced without influencing the performance of PES.

VII. Conclusions

This paper presents a new evolutionary system for learning powerful and compact scheduling heuristics for a complex production environment. The proposed system is a systematic combination of conventional genetic operations, surrogate-assisted modelling, topological data analysis, and visualisation. The core principle of PES is to monitor the evolutionary progress by incrementally learning topological relations of evolved heuristics and using that knowledge to guide the evolution. The experiments have shown that PES can outperform the state-of-the-art SGP in terms of heuristic performance. Moreover, PES has overcome a major limitation of SGP by successfully controlling the sizes of evolved heuristics without deteriorating its performance. On average, heuristics evolved by PES are twice smaller than the simple GP and three time smaller than SGP. Smaller evolved heuristics will make it much easier to analyse and interpret how scheduling decisions are made. Mapping the evolutionary progress also provide PES with the visualisation and interaction abilities that allow the users and decision makers integrate their inputs on the fly. The extensive analyses presented in this paper also demonstrate that the key parameters are intuitive and can be used to control many aspects of the evolution.

Future studies will focus on improving the efficiency of PES by investigating other machine learning methods for topological data analysis such as growing self-organizing map and self-organizing incremental neural network. As finding the right parameters for PES for an arbitrary problem is not straightforward, an important task is to reduce the number of parameters and make the algorithm more adaptive. It would be interesting to also examine how PES can be used for multi-objective optimisation and other machine learning tasks such as classification, symbolic regression, and computer vision.

References

[1] J. Moyne and J. Iskandar, "Big data analytics for smart manufacturing: Case studies in semiconductor manufacturing," *Processes*, vol. 5, no. 3, 2017.

[2] T. Kaihara and Y. Yao, "A new approach on CPS-based scheduling and WIP control in process industries," in *Proceedings of the 2012 Winter Simulation Conference (WSC)*, Dec. 2012, pp. 1–11.

[3] D. Ivanov, A. Dolgui, B. Sokolov, F. Werner, and M. Ivanova, "A dynamic model and an algorithm for short-term supply chain scheduling in the smart factory industry 4.0," *International Journal of Production Research*, vol. 54, no. 2, pp. 386–402, Jan. 2016.

[4] Y.-C. Wang and J. M. Usher, "Application of reinforcement learning for agent-based production scheduling," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 73–82, Feb. 2005.

[5] K. Arviv, H. Stern, and Y. Edan, "Collaborative reinforcement learning for a two-robot job transfer flow-shop scheduling problem," *International Journal of Production Research*, vol. 54, no. 4, pp. 1196–1209, Feb. 2016.

[6] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: ACM, 2016, pp. 50–56.

[7] X. N. Shen and X. Yao, "Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems," *Information Sciences*, vol. 298, pp. 198–224, Mar. 2015.

[8] J. Li, Q. Pan, and P. Duan, "An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping," *IEEE Transactions on Cybernetics*, vol. 46, no. 6, pp. 1311–1324, June 2016.

[9] Y. Han, D. Gong, Y. Jin, and Q. Pan, "Evolutionary multiobjective blocking lot-streaming flow shop scheduling with machine breakdowns," *IEEE Transactions on Cybernetics*, vol. 49, no. 1, pp. 184–197, Jan 2019.

[10] S. Nguyen, D. Thiruvady, A. Ernst, and D. Alahakoon, "Genetic programming approach to learning multi-pass heuristics for resource constrained job scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 1167–1174.

[11] C. D. Geiger, R. Uzsoy, and H. Aytu, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Scheduling*, vol. 9, no. 1, pp. 7–34, Feb. 2006.

[12] S. Chand, Q. Huynh, H. Singh, T. Ray, and M. Wagner, "On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems," *Information Sciences*, vol. 432, pp. 146–163, 2018.

[13] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, Feb. 2016.

[14] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter, "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems," *International Journal of Production Economics*, vol. 145, no. 1, pp. 67–77, 2013.

[15] 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*, ser. Studies in Computational Intelligence, A. S. Uyar, E. Ozcan, and N. Urquhart, Eds. Springer Berlin Heidelberg, 2013, no. 505, pp. 251–282.

[16] L. Nie, L. Gao, P. Li, and X. Li, "A GEP-based policies constructing approach for dynamic flexible job shop scheduling problem with job release dates," *Journal of Intelligent Manufacturing*, vol. 24, no. 4, pp. 763–774, Aug. 2013.

[17] 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, Jun. 2015.

[18] 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, Apr. 2014.

[19] M. Durasevic, D. Jakobovic, and K. Knezevic, "Adaptive scheduling on unrelated machines with genetic programming," *Applied Soft Computing*, vol. 48, pp. 419–430, Nov. 2016.

[20] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary Computation*, vol. 23, no. 3, pp. 343–367, Sep. 2015.

[21] S. Nguyen, M. Zhang, and K. C. Tan, "Adaptive charting genetic programming for dynamic flexible job shop scheduling," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '18. New York, NY, USA: ACM, 2018, pp. 1159–1166.

[22] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, "An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 5, pp. 339–353, Oct 2017.

[23] E. Hart and K. Sim, "A hyper-heuristic ensemble method for static job-shop scheduling," *Evolutionary Computation*, vol. 24, no. 4, pp. 609–635, Dec. 2016.

[24] M. Durasević and D. Jakobović, "Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1, pp. 53–92, Jun 2018.

[25] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling," *Applied Soft Computing*, vol. 63, pp. 72–86, 2018.

[26] S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2951–2965, Sep. 2017.

[27] F. Doshi-Velez and B. Kim, "A roadmap for a rigorous science of interpretability," *CoRR*, vol. abs/1702.08608, 2017. [Online]. Available: http://arxiv.org/abs/1702.08608

[28] Google AI, "People + AI research," in *People + AI Research Initiative Symposium*, Sep. 2017. [Online]. Available: https://ai.google/research/teams/brain/pair

[29] A. M. Law and D. M. Kelton, *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 1999.

[30] S. Kreipl, "A large step random walk for minimizing total weighted tardiness in a job shop," *Journal of Scheduling*, vol. 3, pp. 125–138, May 2000.

[31] 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," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, Oct. 2013.

[32] M. Pinedo and M. Singer, "A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop," *Naval Research Logistics*, vol. 46, no. 1, pp. 1–17, Jan. 1999.

[33] B. Fritzke, "A self-organizing network that can follow non-stationary distributions," in *Proceeedings of International Conference on Artificial Neural Networks: ICANN'97*, W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 613–618.

[34] V. Sels, N. Gheysen, and M. Vanhoucke, "A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions," *International Journal of Production Research*, vol. 50, no. 15, pp. 4255–4270, Sep. 2011.

[35] S. Nguyen, M. Zhang, D. Alahakoon, and K. C. Tan, "Visualizing the evolution of computer programs for genetic programming [research frontier]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 4, pp. 77–94, Nov 2018.

[36] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[37] S. Nguyen, M. Zhang, M. Johnston, and K. Tan, "Automatic Programming via Iterated Local Search for Dynamic Job Shop Scheduling," *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 1–14, 2015.

[38] E. K. Burke, S. Gustafson, and G. Kendall, "Diversity in genetic programming: An analysis of measures and correlation with fitness," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 1, pp. 47–62, Feb. 2004.

[39] S. Furao and O. Hasegawa, "An incremental network for on-line unsupervised classification and topology learning," *Neural Networks*, vol. 19, no. 1, pp. 90–106, 2006.

[40] J. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *CoRR*, vol. abs/1504.04909, 2015.

[41] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios – A genetic programming approach," in *Proceedings of 2010 Genetic and Evolutionary Computation Conference*, ser. GECCO'10, M. Pelikan and J. Branke, Eds. Portland, Oregon, USA: ACM Press, 2010, pp. 257–264.

[42] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Selection Schemes in Surrogate-Assisted Genetic Programming for Job Shop Scheduling," in *SEAL'14: Simulated Evolution and Learning*. Springer International Publishing, 2014, pp. 656–667.

[43] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, Apr. 2008.