Many-Objective Genetic Programming for Job-Shop Scheduling

Atiya Masood [*], Yi Mei[*], Gang Chen[*], Mengjie Zhang[*]
[*] School of Engineering and Computer Science, Victoria University of Wellington
Wellington, 6012, New Zealand
{masoodatiy, yi.mei, aaron.chen, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—In Job Shop Scheduling (JSS) problems, there are usually many conflicting objectives to consider, such as the makespan, mean flowtime, maximal tardiness, number of tardy jobs, etc. Most studies considered these objectives separately or aggregated them into a single objective (fitness function) and treat the problem as a single-objective optimization. Very few studies attempted to solve the multi-objective JSS with two or three objectives, not to mention the many-objective JSS with more than three objectives. In this paper, we investigate the many-objective JSS, which takes all the objectives into account. On the other hand, dispatching rules have been widely used in JSS due to its flexibility, scalability and quick response in dynamic environment. In this paper, we focus on evolving a set of trade-off dispatching rules for many-objective JSS, which can generate non-dominated schedules given any unseen instance. To this end, a new hybridized algorithm that combines Genetic Programming (GP) and NSGA-III is proposed. The experimental results demonstrates the efficacy of the newly proposed algorithm on the tested job-shop benchmark instances.**

I. Introduction

Job Shop Scheduling (JSS) [1] is a significant scheduling problem with a wide range of applications in many industries such as manufacturing and cloud computing. Given a set of jobs and machines to process the jobs, JSS aims to design a schedule to process the jobs in an optimal way by the machines, so that the predefined objectives are optimized and the constraints (e.g. order of the operations of the jobs and the eligible machines for the jobs) are satisfied.

As a well-known problem, JSS has been intensively investigated in the literature [2]. It has been proven to be *NP-hard* [3], since it can be reduced to the *Traveling Salesman Problem* (TSP). Thus, no polynomial method exists to find the global optimal solution, and exact methods are not applicable in practice when the problem size is large. In this situation, heuristics and meta-heuristics become promising methods to obtain near-optimal solutions within the limited time budget. There have been a variety of heuristics [4] and meta-heuristics [5], [6] proposed for JSS.

As a branch of heuristics, *dispatching rules* have been successfully used in JSS due to its flexibility, scalability and quick response to the dynamic environment. So far, there have been a variety of dispatching rules designed for different JSS variants [4], [7].

A dispatching rule is a rule to decide which job in the queue is to be processed by the idle machine at each decision point. A dispatching rule can be seen as a *priority function*, which is used to assign priority to each waiting job. Then, the next job will be selected based on the priority value. For example, in the well-known *Shortest Processing Time* (SPT) rule, jobs with shorter processing times will enjoy higher priority. There can be two types of dispatching rules: (1) *non-delay* and (2) *active* rules. As the name indicates, non-delay rules do not allow any delay on the idle machines as long as the waiting queue is not empty. On the other hand, active rules allow some reasonable delay (which is no more than the minimal processing time of the waiting jobs) to handle the potential new job arrivals with urgent due date. Obviously, non-delay rules cannot guarantee optimality. On the other hand, active rules can guarantee optimality, while it is not trivial to find a proper delay factor. To simply the study, we focus on non-delay rules in this paper.

In designing a dispatching rule for JSS, there are mainly two challenges. First, it is hard to identify all the important factors and understand precisely how they affect the quality of the rule. In other words, it is hard to find the best *priority function* of the rule. Second, the behavior of a dispatching rule can vary a lot from one scenario to another. For example, the rule which is good at minimizing the mean flow time may perform poorly in minimizing the maximal tardiness. Therefore, different rules will be needed to suit different job shop scenarios.

To address these issues, hyper-heuristics have been adopted to automatically design the dispatching rules, i.e. optimize the priority function. A comprehensive survey of automatic design of dispatching rules is given in [8]. The existing hyper-heuristics for evolving dispatching rules in JSS can be divided into two categories, (1) linear regression, and (2) symbolic regression. The linear regression methods formulate the priority function as the linear function of all the job shop attributes and optimize the weights/coefficients [9]. In contrast, the symbolic regression methods optimize both the structure and coefficients of the priority function. Genetic Programming (GP) [10] is a commonly used symbolic regression method for evolving dispatching rules.

Although it is technically simple to consider only a single optimization objective for JSS, it is now widely evidenced in the literature that JSS by nature presents several potentially conflicting objectives, including the makespan, mean flow time, maximum tardiness, maximum lateness, total work load, proportion of tardy jobs. Due to this reason, since early 2000s, the research on *multi-objective* JSS starts to gain increasing popularity. A survey [11] shows that many real-world scheduling problems have inherently multiple objectives. In line with this view, a lot of independent studies have been carried out in the literature for multi-objective JSS [12].

Despite of the rapid development of EC techniques for multi-objective JSS, existing studies of JSS problems having more than three objectives (so-called *many-objective*) are still very limited. There are only a few studies on four objectives [13]. However, the mutual conflicting nature among different objectives in a job shop was still not clearly revealed. In this paper, we aim to explicitly study JSS problems with many conflicting objectives. Specifically, we proposed a new algorithm that considers the challenges of both JSS and many-objective optimization. It combines the evolution process of GP and the selection scheme of NSGA-III [14], which is one of the state-of-the-art many-objective evolutionary algorithm. We evaluated the proposed algorithm by comparing with other multi-objective evolutionary algorithms (i.e. NSGA-II [15] and SPEA2 [16]) to verify the efficacy of the proposed algorithm in solving many-objective JSS.

The rest of the paper is organized as follows: Section II gives the background, including the problem description and related work. Section III describes the proposed algorithm for many-objective JSS. Section IV shows the experimental studies and discussions. Finally, Section V gives the conclusions and future work.

II. Background

In this section, the problem description will be introduced first and then we will discuss some related works.

A. Problem Description

In JSS problem, we are given a set of N jobs and a set of M machines. Each job j_i, $1 \leq i \leq N$ has a sequence of m operations to be performed, i.e. $\{o_i^1, \ldots, o_i^m\}$. Stringent restrictions apply to the processing order over all operations of a job. Specifically, for any job j_i, its operation o_i^{k+1} cannot start until its previous operation o_i^k has been finished. Every operation o_i^k, $1 \leq k \leq m$, is further associated with a fixed processing time $p_i^k > 0$ and has to be processed on a specific machine $m_i, 1 \leq i \leq M$. Any solution to such a JSS problem has to comply with several important rules, as described below.

- Each machine can only process at most one operation of any job at any given time.
- All operations are *non-preemptive*. This means that once an operation starts its processing on a machine, the machine will continue to process it until completion. During this period, the machine cannot be interrupted by other operations.
- All the M machines in the job shop are *persistently available* and can be used to process new operations whenever they become idle.
- The number of jobs is known *a priori*. Moreover, for every job, the operations to be performed and the processing time of each operation are also fixed in advance.
- The due date of each job is pre-determined and fixed during job execution.
- All jobs are immediately ready for processing in the job shop. No precedence constraints apply to them.

Based on the JSS problem described above, our goal is to evolve useful dispatching rules that can build the complete schedules incrementally. The quality of the schedules created by using these rules will then be evaluated with respect to a number of objectives $\boldsymbol{f} = (f_1, \ldots, f_D)$, where D stands for the number of objectives. Without losing generality, we assume that all the dimensions of $\boldsymbol{f}$ are minimized. Here, we considered $D \geq 4$, i.e. there are *four or more* objectives. In this case, the problem is called *many-objective* JSS.

Given two schedules Δ_1 and Δ_2, it is said that Δ_1 *dominates* Δ_2 if and only if

$$\forall i, 1 \leq i \leq D, f_i(\Delta_1) \leq f_i(\Delta_2)$$

and

$$\exists i, f_i(\Delta_1) < f_i(\Delta_2).$$

Consequently, if P_1 is the dispatching rule that produces schedule Δ_1 and P_2 is the dispatching rule that produces schedule Δ_2, then we can also say that, for the given problem instance I, P_1 dominates P_2. Based on the above explanations, it should now be clear that our goal is to evolve a collection of *Pareto optimal* dispatching rules, jointly known as the *Pareto front*. Each dispatching rule in the Pareto front is not dominated by any other rules evolved by using our algorithms.

B. Related Works

Huge efforts have been made in developing effective evolutionary computation (EC) algorithm for JSS [17]. In the literature, many research works have been conducted with the focus on optimizing a single objective [18], [19]. However, it is becoming more and more clear that JSS essentially has multiple (or many) different objectives. In general, there are two alternative approaches for handling multiple objectives in a job shop, i.e. the *aggregation method* and the *Pareto dominance method* [20]. In a typical aggregation method, multiple optimization objectives have to be aggregated together to form a scalar-valued fitness function through weighted sum [21]. Obviously, the usefulness of this method is restricted to the situation when the preferences over different objectives can be quantified before applying any EC techniques. On the other hand, without using any aggregation functions, the *Pareto dominance* concept can be exploited to define the optimization criteria for guided search of *Pareto optimal schedules* [21]. Based on this idea, many EC algorithms have been proposed with the aim of evolving the *Pareto front* [14], [15], [16].

In the literature, the Pareto dominance method has attracted substantial research attention. Prominent examples include the Non-dominated Sorting Genetic Algorithm (NSGA-II) [15] and the Strength Pareto Evolutionary Algorithm (SPEA2) [16]. Specific techniques have also been successfully developed for multi-objective JSS. For instance, Murata et al. proposed a multi-objective GA for flow shop scheduling problems. Their research specifically considered problem instances with concave Pareto fronts and at most three optimization objectives (i.e. the makespan, total tardiness, total flow time) [22]. While solving bi-objective JSS problems, a tradeoff between the makespan and the availability of machines has also been identified in [23]. In particular, whenever the makespan is minimized, the availability of machines will be negatively affected, and vice versa.

In addition to the research works mentioned above, some researchers have started to consider JSS problems with more than three objectives. For example, in [24], Nguyen et al. have considered the problem of designing dispatching rules for general JSS problems with up to five different objectives. In [13], four different objectives have been considered while evolving optimal schedules by using multi-objective GA. While showing promise, it is unclear whether standard multi-objective optimization algorithms such as NSGA-II and SPEA2 are competent at tackling many-objective JSS problems. It is also unclear whether the objectives under consideration are mutually conflicting. If not, we may easily reduce the number of objectives to three or less.

In fact, as emphasized by Deb in [14], a large proportion of real-world problems can be described naturally as many-objective problems. In this paper, we will also demonstrate that JSS inherently has many (i.e. more than three) conflicting objectives. [21] gives a nice review of some interesting many-objective problems investigated in the literature. There is also a growing interest in industries to tackle problems with many objectives [25], [26]. In line with this research trend, we believe that many-objective JSS will attract increasing research attention in the near future. This paper serves as the first initiative towards addressing this challenging problem.

It is worthwhile to note that it is not our intention in this paper to develop a new general-purpose algorithm for many-objective optimization. In fact, quite a few interesting many-objective algorithms have already been developed in recent years [21]. A short review on evolutionary many-objective optimization techniques can be found in [21]. Among all the different technologies, we have particular interests in NSGA-III [14].

As we discussed in the Introduction, rather than directly evolving Pareto optimal schedules, dynamic and responsive scheduling requires us to evolve dispatching rules. We hope that the evolved dispatching rules could achieve a good balance over many conflicting objectives. This is essentially a machine learning approach where rules can be evolved based on a training set of problem instances and then further evaluated on a different set of testing instances. In the literature, the most widely explored technique is known as the Genetic Programming based Hyper-Herestic (GP-HH) [27], which will also be adopted in this paper. The effectiveness of GP-HH has been extensively demonstrated in the field of JSS [28], [29]. To our knowledge, we are among the first to use GP-HH for many-objective JSS.

III. A New Algorithm for Many-Objective JSS

In this section, we describe a new algorithm to solve many-objective JSS problems by combining GP and NSGA-III. GP is a commonly used hyper-heuristic for evolving dispatching rules for JSS and has achieved great success [28], [29]. NSGA-III is one of the state-of-the-art many-objective evolutionary algorithms. Thus, it is expected that the combination of GP and NSGA-III can lead to a competitive algorithm for evolving a set of trade-off rules in many-objective JSS. The framework of the newly proposed algorithm, which is named as GP-NSGA-III, is given in Algorithm 1. It can be seen that GP-NSGA-III combines the initialization, evaluation and evolutionary operators of GP, and the selection scheme of NSGA-III.

Algorithm 1: The framework of GP-NSGA-III.

```
Input  : A training set $\mathcal{I}_{\text{train}}$
Output: A set of non-dominated rules $\boldsymbol{P}^*$
1  Initialize and evaluate the population $\boldsymbol{P}_0$ of rules by the
   ramped-half-and-half method;
2  Calculate the reference points $\boldsymbol{Z}$;
3  Set $g \leftarrow 0$;
4  while $g < g_{\max}$ do
5      Generate the offspring population $\boldsymbol{Q}_g$ using the
       crossover, mutation and reproduction of GP;
6      foreach $Q \in Q_g$ do Evaluate rule $Q$;
7      $\boldsymbol{R}_g \leftarrow \boldsymbol{P}_g \cup \boldsymbol{Q}_g$;
8      Form the new population $\boldsymbol{P}_{g+1}$ from $\boldsymbol{R}_g$ by the
       NSGA-III selection;
9      $g \leftarrow g + 1$;
10 end
11 return The non-dominated individuals $P^* \subseteq P_{g_{\max}}$;
```

A. Representation of Rules

A dispatching rule is usually described as a priority function (when the delay factor is not considered). Therefore, optimizing the dispatching rule is equivalent to optimizing the priority function. In GP, a priority function can be represented as a GP tree. Fig. 1 shows the GP tree representation of the 2PT+WINQ+NPT rule [4].

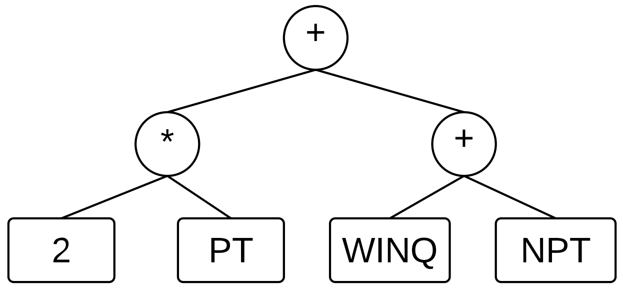

Fig. 1. The GP tree representation of the 2PT+WINQ+NPT rule.

In the GP tree representation, there are two different sets to be determined: (1) the *terminal* set and (2) the *non-terminal* (function) set. In the example in Fig. 1, the terminals in the tree are $\{2, \text{PT}, \text{WINQ}, \text{NPT}\}$, and the non-terminals are $\{+, *\}$. Proper terminal and non-terminal sets are crucial for constructing promising and concise search space for GP. Here, the commonly considered job shop attributes are included in the terminals, which is given in Table I. The non-terminal set is set to $\{+, -, *, /, \min, \max, \text{ifte}\}$, where "+", "−" and

"$*$" are basic arithmetic operators. The function "/" is the protected division operator, which returns 1 if the denominator is 0. The functions "$\min$" and "$\max$" take two arguments and return either the smaller or the larger value, respectively. The function "ifte" takes three arguments a, b and c. It returns b if $a > 0$, and c otherwise.

TABLE I
TERMINAL SET FOR GP FOR JSS.

Attribute	Symbol
Processing time of the operation	PT
Inverse processing time of the operation	IPT
Processing time of the next operation	NPOT
Ready time of the operation	ORT
Ready time of the next machine	NMRT
Work Remaining	WKR
Number of operation remaining	NOR
Work in the next queue	WINQ
Number of operations in the next queue	NOINQ
Flow due date	FDD
Due Date	DD
Weight	W
Number of operations in the queue	NOIQ
Work in the queue	WIQ
Ready time of the machine	MRT

B. Fitness Evaluation

The fitness evaluation is required in lines 1 and 6 of Algorithm 1. It calculates the objective values by applying the rule to the training instances. Detailed process for fitness evaluation is further presented in Algorithm 2. $\boldsymbol{f}(\cdot)$ is the objective vector, which includes the makespan, maximal flowtime, mean weighted tardiness, maximal weighted tardiness and proportion of tardy jobs.

Algorithm 2: The evaluation of a dispatching rule.

```
  Input  : A training set $\mathcal{I}_{\text{train}}$ and a rule $P$
  Output: The fitness $\boldsymbol{f}(P)$ of the rule $P$
1 foreach $\mathcal{I} \in \mathcal{I}_{train}$ do
2     Construct a schedule $\Delta(P, \mathcal{I})$ by applying the rule $P$
      to the JSS instance $\mathcal{I}$;
3     Calculate the objective values $\boldsymbol{f}(\Delta(P, \mathcal{I}))$;
4 end
5 $\boldsymbol{f}(P) \leftarrow \frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{\mathcal{I} \in \mathcal{I}_{\text{train}}} \boldsymbol{f}(\Delta(P, \mathcal{I}))$;
6 return $\boldsymbol{f}(P)$;
```

C. Evolutionary Operators

Standard GP crossover and mutation operators are adopted for evolving GP trees. Specifically, for crossover, two parents are randomly selected from the population through the *tournament selection* method. Then, for each parent, a node is randomly selected from each parent GP tree. Afterwards, the sub-trees rooted from the selected nodes are swapped. For mutation, a randomly node is selected from the child GP tree, and the sub-tree rooted from the selected node is replaced by another randomly generated sub-tree.

D. Population Update

Unlike traditional GP, which generates the new population from scratch, GP-NSGA-III first combines the offspring population $\boldsymbol{Q}$ and the parent population $\boldsymbol{P}$ together. Then, all the individuals in the combined population $\boldsymbol{R}$ is divided into different ranks using *non-dominated sorting* [14]. After that, the new population is obtained by filling in the individuals from the lowest rank, until the current rank cannot be fully inserted into the population. Then, each individual of the current rank is assigned to a reference point $Z \in \boldsymbol{Z}$, and selected one by one using the niching method. Details of the population update mechanism can be found in [14].

IV. EXPERIMENTAL STUDIES

In this section, experimental studies will be conducted to compare the proposed GP-NSGA-III to NSGA-II [15] and SPEA2 [16], which are two well-known multi-objective evolutionary algorithms. Both compared algorithms were combined with GP as well, i.e. they adopted the solution representation and evolutionary operators of GP. For the sake of convenience, they are referred to as GP-NSGA-II and GP-SPEA2 hereafter. In the following, we will describe the dataset used in the experiments, parameter settings of the algorithms, and results and discussions.

A. Dataset

In this paper, the widely used Taillard static job shop benchmark set [30] is selected as the dataset for the experiments. Although the effectiveness of dispatching rules is more prominent for dynamic JSS, as a starting point, using static problem instances can make it easier to analyze and understand the behavior of our algorithm. For the sake of convenience, the Taillard job shop benchmark set is denoted as the TA set hereafter.

There are 80 indexed problem instances in the TA set (i.e. each problem instance has a different ID from 1 to 80), which can be further divided into eight groups (denoted as the TA-1, ..., TA-8 groups). The problems instances in the same group have the same number of jobs and machines, but the processing time matrices were generated by using different random seeds. Across different groups, the number of jobs varies from 15 to 100, and the number of machines varies from 15 to 20. In the experiments, the total 80 instances were divided into the *training set* and the *test set*, each consisting of 40 instances. Specifically, the training set consists of all the instances with the odd ID, and the test set contains all the instances with the even ID.

The release times of all jobs in all problem instances are safely set to zero. Since there is no due date information included in the original dataset, we set the due date using the standard *total workload strategy*. That is,

$$dd(j_i) = \lambda \times \sum_{k=1}^{m} p_i^k,$$

where $dd(j_i)$ stands for the due date of the job j_i, and λ is the due date factor, which is set to 1.3 in our experiments. The job weights are set according to the 2:6:2 rule [31]. That is, the weight is set to 4, 2 and 1 with the probabilities of 20%, 60% and 20%, respectively.

In terms of objectives, we designed two sets of experiments. The first set of experiments considers minimizing four objectives, i.e. the mean flowtime, maximal flowtime, mean weighted tardiness and maximal weighted tardiness. The second set of experiments has an additional objective on the number of tardy jobs. For the sake of convenience, the first set is referred to as the "4-obj" experiment, and the second one the "5-obj" experiment.

For each run of each tested algorithm, the experiment consists of two steps as follows.

1) Apply the algorithm (GP-NSGA-III, GP-NSGA-II or GP-SPEA2) to the training set to obtain a set of tradeoff dispatching rules;
2) For each problem instance in the test set, apply every obtained rule to produce a schedule and calculate all the considered objective values with respect to the schedule.

The above steps are repeated for the compared GP-NSGA-III, GP-NSGA-II and GP-SPEA2 algorithms.

B. Parameter Settings

Table II summarized the parameter settings of GP-NSGA-III in detail. To make a fair comparison, the same settings are also adopted for the compared GP-NSGA-II and GP-SPEA2. All experimented algorithms were implemented with ECJ [32], which is a commonly used research system in Java for GP.

TABLE II
THE PARAMETER SETTING OF THE PROPSED GP-NSGA-III.

Parameter	**Value**
Population Size	1024
Generation	50
Crossover rate	85%
Mutation rate	10%
Reproduction Rate	5%
Max-depth	8
Tournament size	7

C. Performance Measures

There are a variety of performance measures proposed for evaluating multi-objective optimization algorithms from different perspectives. In this paper, following a common practice, we choose the *Hyper-Volume* (HV) [33] and *Inverted Generational Distance* (IGD) [34] as the two main performance measures. Theoretically, a set of tradeoff dispatching rules with better performance should have a *larger* HV value and a *smaller* IGD value.

All the objectives have been normalized into the range between 0 and 1 before calculating the above performance measures. Then, for HV, the nadir point is set to (1,1) since all the objective are to be minimized. For IGD, since the true Pareto front is unknown, an approximate Pareto front was obtained by selecting the non-dominated solutions among the final solutions acquired from all the runs of all the three tested algorithms.

D. Results and Discussions

TABLE III
THE MEAN AND STANDARD DEVIATION OVER THE HV VALUES OF THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS IN THE 4-OBJ EXPERIMENT. THE SIGNIFICANTLY BETTER RESULTS ARE IN BOLD.

ID	#J_#M	GP-NSGA-III	GP-NSGA-II	GP-SPEA2
1	15_15	**.0292(.0068)**	.0035(.0007)	.0170(.0028)
2	15_15	.0541(.0049)	**.3430(.0493)**	.0345(.0037)
3	15_15	**.0668(.0062)**	.0197(.0021)	.0179(.0017)
4	15_15	.0329(.0049)	**.1136(.0250)**	.0126 (.0012)
5	15_15	**.0344(.0034)**	.0122(.0012)	.0116(.0017)
6	20_15	.1027(.0087)	**.1796(.0094)**	.0410(.0032)
7	20_15	.0668(.0085)	**.2354(.0122)**	.0425(.0017)
8	20_15	.1193(.0118)	**.2506(.0138)**	.0457(.0029)
9	20_15	**.1287(.0109)**	.0593(.0035)	.0602(.0047)
10	20_15	**.1106(.0118)**	.0876(.0038)	.0533(.0032)
11	20_20	.0945(.0056)	**.2706(.0139)**	.0307(.0031)
12	20_20	.0873(.0064)	**.5793(.0209)**	.0361(.0021)
13	20_20	.0348(.0057)	.0332(.0015)	.0346(.0024)
14	20_20	**.0826(.0047)**	.0241(.0017)	.0330(.0025)
15	20_20	.0030(.0032)	**.0161(.0012)**	.0158(.0017)
16	30_15	**.2235(.0188)**	.1729(.0048)	.1263(.0062)
17	30_15	**.2092(.0132)**	.1514(.0046)	.1139(.0047)
18	30_15	**.1932(.0132)**	.1402(.0041)	.1194(.0057)
19	30_15	**.1976(.0161)**	.1525(.0057)	.1058(.0040)
20	30_15	**.1863(.0157)**	.1533(.0044)	.1163(.0046)
21	30_20	**.1863(.0184)**	.1188(.0038)	.0839(.0047)
22	30_20	**.0777(.0067)**	.0230(.0019)	.0249(.0031)
23	30_20	**.1739(.0127)**	.1119(.0034)	.0880(.0058)
24	30_20	**.1687(.0147)**	.1401(.0028)	.0900(.0034)
25	30_20	**.1692(.0127)**	.1252(.0035)	.0920(.0044)
26	50_15	**.2609(.0139)**	.1999(.0062)	.1713(.0070)
27	50_15	**.3092(.0229)**	.2035(.0047)	.1750(.0061)
28	50_15	**.2453(.0159)**	.1820(.0070)	.1577(.0058)
29	50_15	**.2811(.0216)**	.1789(.0062)	.1578(.0067)
30	50_10	**.2708(.0177)**	.1845(.0053)	.1599(.0098)
31	50_20	**.2143(.0135)**	.1738(.0062)	.1366(.0054)
32	50_20	**.3182(.0193)**	.2001(.0042)	.1582(.0045)
33	50_20	**.2568(.0162)**	.1829(.0066)	.1383(.0051)
34	50_20	**.2955(.0189)**	.1876(.0046)	.1522(.0055)
35	50_20	**.2737(.0130)**	.1851(.0043)	.1500(.0064)
36	50_20	**.3333(.0163)**	.2169(.0062)	.2131(.0081)
37	100_20	**.2687(.0164)**	.0896(.0037)	.0796(.0048)
38	100_20	**.2659(.0145)**	.2141(.0057)	.2132(.0061)
39	100_20	**.2345(.0134)**	.1141(.0668)	.1111(.0055)
40	100_20	**.3564(.0190)**	.2217(.0046)	.2211(.0059)

Tables III and IV show the mean and standard deviation over the HV values of the 30 independent runs of the compared algorithms with respect to the 40 test instances in the 4-obj experiment. For each instance, Wilcoxon's rank sum test [35] was conducted to compare the best performing algorithm against the other two algorithms. The corrected significance level under Bonferroni correction is set to $0.05/2 = 0.025$. That is, if both p-values are smaller than 0.025, then the best algorithm is considered significantly better than the other two algorithm. The significantly better results are marked in bold.

In the tables, "ID" is the instance ID, and #J and #M indicate the number of jobs and machines, respectively. One can see that as the instance ID increases, the problem size increases as well. Tables III and IV together suggest that GP-NSGA-III

TABLE IV
THE MEAN AND STANDARD DEVIATION OVER THE IGD VALUES OF THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS IN THE 4-OBJ EXPERIMENT. THE SIGNIFICANTLY BETTER RESULTS ARE IN BOLD.

ID	#J_#M	GP-NSGA-III	GP-NSGA-II	GP-SPEA2
1	15_15	**.0071(4.0E-05)**	.0077(4.9E-05)	.0087(1.4E-05)
2	15_15	.0071(6.9E-05)	**.0037(2.1E-05)**	.0105(9.3E-05)
3	15_15	**.0009(1.9E-05)**	.0018(6.1E-06)	.0017(1.8E-05)
4	15_15	.0057(3.9E-05)	**.0036(1.2E-05)**	.0119(4.7E-05)
5	15_15	**.0013(1.6E-05)**	.0023(1.4E-05)	.0037(4.9E-05)
6	20_15	**.0024(2.2E-05)**	.0028(3.3E-05)	.0051(6.2E-05)
7	20_15	.0110(7.5E-05)	**.0088(5.9E-05)**	.0167(1.4E-05)
8	20_15	**.0027(4.0E-05)**	.0038(3.3E-05)	.0060(9.2E-05)
9	20_15	**.0023(2.3E-05)**	.0033(3.9E-05)	.0054(7.5E-05)
10	20_15	**.0029(3.0E-05)**	.0044(5.6E-05)	.0072(9.0E-05)
11	20_20	**.0017(1.4E-05)**	.0025(1.5E-05)	.0046(6.3E-05)
12	20_20	**.0023(1.8E-05)**	.0031(2.4E-05)	.0048(5.9E-05)
13	20_20	**.0016(9.7E-06)**	.0027(1.0E-05)	.0042(8.1E-05)
14	20_20	**.0021(2.3E-05)**	.0027(2.2E-05)	.0045(5.7E-05)
15	20_20	**.0032(3.6E-05)**	.0050(4.0E-05)	.0083(9.3E-05)
16	30_15	**.0009(1.7E-05)**	.0013(1.9E-05)	.0020(2.5E-05)
17	30_15	**.0024(3.3E-05)**	.0032(4.9E-05)	.0054(5.5E-05)
18	30_15	**.0017(2.0E-05)**	.0023(3.0E-05)	.0036(3.4E-05)
19	30_15	**.0025(3.4E-05)**	.0033(3.8E-05)	.0052(6.3E-05)
20	30_15	**.0015(2.3E-05)**	.0018(3.1E-05)	.0030(2.8E-05)
21	30_20	**.0026(4.1E-05)**	.0033(4.7E-05)	.0050(1.0E-05)
22	30_20	**.0040(2.5E-05)**	.0048(2.7E-05)	.0080(7.5E-05)
23	30_20	**.0027(2.0E-05)**	.0031(3.1E-05)	.0048(4.4E-05)
24	30_20	**.0033(5.1E-05)**	.0037(5.0E-05)	.0059(6.3E-05)
25	30_20	**.0030(4.0E-05)**	.0037(3.6E-05)	.0059(6.1E-05)
26	50_15	**.0013(1.3E-05)**	.0018(2.9E-05)	.0031(3.5E-05)
27	50_15	**.0007(1.2E-05)**	.0013(2.2E-05)	.0022(2.7E-05)
28	50_15	**.0041(4.2E-05)**	.0047(8.3E-05)	.0081(9.8E-05)
29	50_15	**.0017(3.0E-05)**	.0023(4.2E-05)	.0041(4.9E-05)
30	50_15	**.0020(3.6E-05)**	.0024(4.0E-05)	.0041(4.6E-05)
31	50_20	**.0010(1.0E-05)**	.0015(4.2E-05)	.0024(4.3E-05)
32	50_20	**.0008(9.6E-06)**	.0013(2.1E-05)	.0021(3.2E-05)
33	50_20	**.0033(5.0E-05)**	.0035(4.5E-05)	.0058(6.6E-05)
34	50_20	**.0012(2.4E-05)**	.0016(2.8E-05)	.0026(2.7E-05)
35	50_20	**.0013(2.2E-05)**	.0020(3.3E-05)	.0034(3.4E-05)
36	100_20	**.0011(1.6E-05)**	.0016(1.9E-05)	.0030(3.8E-05)
37	100_20	**.0005(8.4E-06)**	.0012(1.9E-05)	.0010(1.4E-05)
38	100_20	**.0015(1.7E-05)**	.0018(3.3E-05)	.0035(8.8E-05)
39	100_20	**.0012(1.3E-05)**	.0027(3.7E-05)	.0047(5.8E-05)
40	100_20	**.0009(1.4E-05)**	.0015(2.3E-05)	.0028(4.6E-05)

TABLE V
THE MEAN AND STANDARD DEVIATION OVER THE HV VALUES OF THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS IN THE 5-OBJ EXPERIMENT. THE SIGNIFICANTLY BETTER RESULTS ARE IN BOLD.

ID	#J_#M	GP-NSGA-III	GP-NSGA-II	GP-SPEA2
1	15_15	.0113(.0033)	**.0259(.0030)**	.0007(.0002)
2	15_15	.0216(.0031)	**.0225(.0049)**	.0012(.0003)
3	15_15	.0223(.0037)	**.0250(.0035)**	.0012(.0003)
4	15_15	.0187(.0035)	.0179(.0024)	.0010(.0003)
5	15_15	.0193(.0031)	**.0244(.0036)**	.0011(.0002)
6	20_15	**.0444(.0055)**	.0345(.0038)	.0025(.0005)
7	20_15	**.0657(.0046)**	.0312(.0035)	.0040(.0005)
8	20_15	**.0457(.0038)**	.0281(.0045)	.0027(.0004)
9	20_15	**.0455(.0063)**	.0300(.0036)	.0028(.0005)
10	20_15	**.0362(.0041)**	.0312(.0056)	.0022(.0004)
11	20_20	.0231(.0031)	**.0306(.0040)**	.0012(.0002)
12	20_20	**.0248(.0035)**	.0143(.0017)	.0014(.0003)
13	20_20	**.0294(.0048)**	.0265(.0034)	.0017(.0003)
14	20_20	.0321(.0051)	**.0352(.0029)**	.0018(.0004)
15	20_20	.0148(.0036)	.0148(.0024)	.0009(.0002)
16	30_15	**.0667(.0070)**	.0212(.0046)	.0040(.0008)
17	30_15	**.0791(.0189)**	.0195(.0080)	.0041(.0015)
18	30_15	**.0758(.0052)**	.0272(.0036)	.0046(.0005)
19	30_15	**.0713(.0058)**	.0273(.0052)	.0044(.0008)
20	30_15	**.1186(.0072)**	.0311(.0057)	.0070(.0008)
21	30_20	**.0601(.0063)**	.0338(.0057)	.0035(.0006)
22	30_20	**.0404(.0048)**	.0273(.0045)	.0024(.0005)
23	30_20	**.0732(.0059)**	.0293(.0042)	.0044(.0006)
24	30_20	**.0554(.0048)**	.0296(.0042)	.0033(.0004)
25	30_20	.0363(0.005)	.0361(.0041)	.0021(.0005)
26	50_15	**.2507(.0092)**	.0321(.0046)	.0155(.0013)
27	50_15	**.2286(.0121)**	.0280(.0050)	.0140(.0016)
28	50_15	**.2283(.0091)**	.0284(.0052)	.0139(.0014)
29	50_15	**.2243(.0130)**	.0222(.0039)	.0139(.0016)
30	50_15	**.1822(.0134)**	.0241(.0045)	.0112(.0017)
31	50_20	**.1389(.0070)**	.0276(.0045)	.0082(.0010)
32	50_20	**.1821(.0082)**	.0301(.0053)	.0111(.0014)
33	50_20	**.1414(.0054)**	.0286(.0061)	.0084(.0010)
34	50_20	**.1614(.0086)**	.0295(.0045)	.0100(.0010)
35	50_20	**.2040(.0063)**	.0303(.0049)	.0124(.0013)
36	100_20	**.1285(.0077)**	.0220(.004)	.0159(.0017)
37	100_20	**.1338(.0082)**	.0226(.005)	.0165(.0015)
38	100_20	**.1646(.0060)**	.0285(.0042)	.0145(.0017)
39	100_20	**.1176(.0049)**	.0189(.0046)	.0138(.0016)
40	100_20	**.1503(.0077)**	.0253(.0048)	.0178(.0016)

performed significantly better than both GP-NSGA-II and GP-SPEA2 in most of the tested instances with 4 objectives. In terms of HV, GP-NSGA-III performed the best with statistical significance on 31 out of the total 40 instances. GP-NSGA-II outperformed other algorithms in the remaining 8 instances. SPEA2 failed to achieve the best performance in any instances.

In terms of IGD, Table IV showed a similar pattern. GP-NSGA-III performed significantly better on 37 out of the 40 instances, and GP-NSGA-II achieved the best performance on the remaining 3 instances. Again, there is no instance where GP-SPEA2 performed the best.

When taking a closer look at the tables, it can be found that GP-NSGA-II performed better on smaller instances (no more than 20 jobs and 20 machines). On the other hand, GP-NSGA-III appeared to be more effective on larger instances. This demonstrates the advantage of GP-NSGA-III especially on challenging problems since a large size usually indicates a more difficult problem. In addition, GP-NSGA-III showed better performance in terms of IGD than in terms of HV. This might be due to the generation of the approximate Pareto front based on the results of all the compared algorithms. That is, GP-NSGA-III seems to contribute most of the elements in the approximate Pareto front.

Tables V and VI jointly present the corresponding results for the 5-obj experiment. The tables exhibited the same patterns to that of the 4-obj experiment. Overall, GP-NSGA-III performed the best, followed by GP-NSGA-II. GP-SPEA2 performed the worst. In addition, in terms of HV, it can be seen that the performance of all the algorithms are worse for 5 objectives than for 4 objectives. This is consistent with our intuition that

TABLE VI
THE MEAN AND STANDARD DEVIATION OVER THE IGD VALUES OF THE 30 INDEPENDENT RUNS OF THE COMPARED ALGORITHMS IN THE 5-OBJ EXPERIMENT. THE SIGNIFICANTLY BETTER RESULTS ARE IN BOLD.

ID	#J_#M	GP-NSGA-III	GP-NSGA-II	GP-SPEA2
1	15_15	**.0040(4.7E-05)**	.0049(5.9E-05)	.0053(1.0E-04)
2	15_15	**.0040(3.0E-05)**	.0060(6.1E-05)	.0055(8.1E-05)
3	15_15	**.0011(6.5E-06)**	.0021(1.5E-05)	.0015(2.6E-05)
4	15_15	.0071(4.3E-05)	.0071(5.5E-05)	.0094(7.5E-05)
5	15_15	**.0021(1.4E-05)**	.0035(2.7E-05)	.0027(2.9E-05)
6	20_15	**.0033(2.0E-05)**	.0039(3.1E-05)	.0043(4.5E-05)
7	20_15	**.0062(3.3E-05)**	.0078(8.2E-05)	.0082(8.5E-05)
8	20_15	**.0022(1.6E-05)**	.0030(4.7E-05)	.0029(3.3E-05)
9	20_15	**.0060(3.9E-05)**	.0065(6.4E-05)	.0079(6.9E-05)
10	20_15	**.0024(1.5E-05)**	.0035(3.4E-05)	.0032(4.5E-05)
11	20_20	**.0038(2.3E-05)**	.0039(3.1E-05)	.0051(3.2E-05)
12	20_20	**.0036(2.0E-05)**	.0050(3.9E-05)	.0047(3.0E-05)
13	20_20	**.0050(3.3E-05)**	.0052(2.9E-05)	.0067(6.8E-05)
14	20_20	**.0030(2.0E-05)**	.0036(2.8E-05)	.0040(4.2E-05)
15	20_20	**.0071(7.4E-05)**	.0112(7.6E-05)	.0094(1.0E-05)
16	30_15	**.0021(1.8E-05)**	.0030(4.4E-05)	.0028(5.3E-05)
17	30_15	**.0034(1.5E-05)**	.0035(2.6E-05)	.0048(3.2E-05)
18	30_15	**.0032(1.7E-05)**	.0036(3.0E-05)	.0045(3.4E-05)
19	30_15	**.0018(1.1E-05)**	.0029(3.0E-05)	.0024(5.2E-05)
20	30_15	**.0024(1.6E-05)**	.0030(3.2E-05)	.0031(3.0E-05)
21	30_20	**.0032(2.2E-05)**	.0042(3.9E-05)	.0042(6.1E-05)
22	30_20	**.0030(2.5E-05)**	.0038(3.9E-05)	.0040(4.9E-05)
23	30_20	**.0025(1.7E-05)**	.0032(2.2E-05)	.0033(3.5E-05)
24	30_20	**.0028(1.8E-05)**	.0031(2.5E-05)	.0037(3.2E-05)
25	30_20	**.0031(1.9E-05)**	.0036(2.7E-05)	.0041(5.0E-05)
26	50_15	**.0021(7.7E-06)**	.0026(2.8E-05)	.0028(1.9E-05)
27	50_15	**.0021(1.3E-05)**	.0034(3.1E-05)	.0028(4.1E-05)
28	50_15	**.0022(8.7E-06)**	.0030(3.2E-05)	.0029(2.6E-05)
29	50_15	**.0022(1.1E-05)**	.0031(2.7E-05)	.0028(3.1E-05)
30	50_15	**.0024(1.6E-05)**	.0035(3.7E-05)	.0032(6.1E-05)
31	50_20	**.0020(1.2E-05)**	.0027(2.9E-05)	.0026(2.7E-05)
32	50_20	**.0021(1.0E-05)**	.0029(3.4E-05)	.0028(4.3E-05)
33	50_20	**.0024(1.1E-05)**	.0032(4.2E-05)	.0032(4.9E-05)
34	50_20	**.0012(8.9E-06)**	.0020(2.3E-05)	.0016(2.0E-05)
35	50_20	**.0024(8.1E-06)**	.0031(3.5E-05)	.0031(4.3E-05)
36	100_20	.0021(6.6E-06)	.0021(2.0E-05)	.0033(3.6E-05)
37	100_20	.0021(8.7E-06)	.0021(2.5E-05)	.0034(2.4E-05)
38	100_20	.0024(7.8E-06)	.0024(2.1E-05)	.0040(4.2E-05)
39	100_20	**.0020(7.4E-06)**	.0029(2.9E-05)	.0047(4.7E-05)
40	100_20	.0027(1.1E-05)	.0027(3.4E-05)	.0042(3.2E-05)

increasing the number of objective can dramatically increase the difficulty of the problem. Note that when 5 objectives are being considered, there are some problems instances in which no significance can be found (e.g. instance 4). For these instances, GP-NSGA-III was still one of the best performing algorithms. It is also found that when the number of objectives increases, the advantage of GP-NSGA-III becomes more obvious. Meanwhile, Table V shows that, with 5 objectives, GP-NSGA-III performed statistically the same as GP-NSGA-II on instances 4, 15 and 25. Notice that, for the same two instances, GP-NSGA-II actually performed better in the 4-obj experiment. In terms of IGD, GP-NSGA-III performed at least statistically the same as the other algorithms, and achieved significantly better performance in most instances.

Figs. 2 and 3 depict the non-dominated set obtained by the 30 independent runs of GP-NSGA-II and GP-NSGA-III on instance 6 with 4 objectives. It can be seen that GP-NSGA-II managed to achieve better values in objectives 2 and 4, and covered a wider range of objective 3. This explains why GP-NSGA-II outperformed GP-NSGA-III in this instance. It also shows that the four objectives are conflicting to each other. Although the two figures demonstrate better performance for GP-NSGA-II, for a majority of the problem instances, GP-NSGA-III in fact achieved consistently better results.

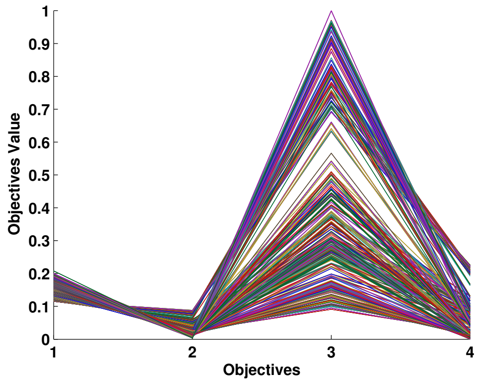

Fig. 2. The non-dominated set obtained by the 30 independent runs of GP-NSGA-II on instance 6 with 4 objectives.

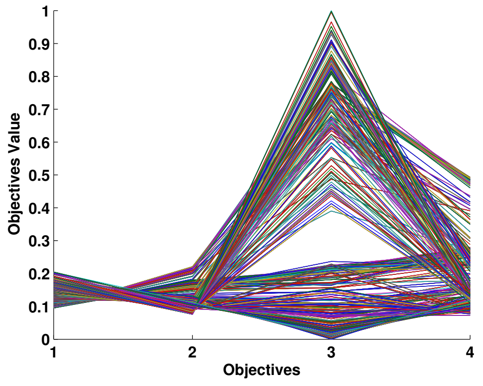

Fig. 3. The non-dominated set obtained by the 30 independent runs of GP-NSGA-III on instance 6 with 4 objectives.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have discussed the importance of considering many (i.e. four or more) potentially conflicting objectives for effective JSS. Aimed at solving JSS problems with many objectives, we have also successfully developed a new algorithm that seamlessly combines GP-HH for evolving dispatching rules with the selection technique introduced in NSGA-III. To examine the effectiveness of our algorithm, experimental studies have been carried out by using the Taillard static job-shop benchmark set. Through experimental comparison with two competing algorithms for multi-objective optimization, i.e. NSGA-II and SPEA2, we obtained clear evidence that our algorithm performed consistently better on a majority of the problem instances, especially when five objectives were considered. This finding leads us to believe that our algorithm is more effective for many-objective JSS than commonly used multi-objective optimization algorithms. Our experiments also showed that many-objective JSS is more challenging than multi-objective JSS and therefore demands new and effective algorithms. Our study serves as the first initiative towards achieving this goal.

In the future, we have the plan to further examine the effectiveness of our algorithm for dynamic JSS where GP-HH is considered more useful. We would also like to investigate the potential enhancements of our algorithm by using archiving methods, niching mechanisms and local search techniques so as to achieve better results on a wide range of JSS problems.

References

[1] M. L. Pinedo, *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

[2] E. Hart, P. Ross, and D. Corne, "Evolutionary Scheduling: A Review." *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 191–220, 2005.

[3] J. Błażewicz, W. Domschke, and E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques," *European journal of operational research*, vol. 93, no. 1, pp. 1–33, 1996.

[4] O. Holthaus and C. Rajendran, "Efficient jobshop dispatching rules: further developments," *Production Planning & Control*, vol. 11, no. 2, pp. 171–178, 2000.

[5] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[6] B. Peng, Z. Lü, and T. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Computers & Operations Research*, vol. 53, pp. 154–164, 2015.

[7] 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, 2012.

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

[9] A. M. Kuczapski, M. V. Micea, L. A. Maniu, and V. I. Cretu, "Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling," *Information Technology and Control*, vol. 39, no. 1, pp. 32–37, 2010.

[10] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[11] M. M. Yenisey and B. Yagmahan, "Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends," *Omega*, vol. 45, pp. 119–135, 2014.

[12] R. Tavakkoli-Moghaddam, A. Rahimi-Vahed, and A. H. Mirzaei, "A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness," *Information Sciences*, vol. 177, no. 22, pp. 5072–5090, 2007.

[13] D. C. Mattfeld and C. Bierwirth, "An efficient genetic algorithm for job shop scheduling with tardiness objectives." *European Journal of Operational Research*, vol. 155, no. 3, pp. 616–630, 2004.

[14] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 4, pp. 577–601, 2014.

[15] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, pp. 182–197, 2002.

[16] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," in *EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems*, 2002, pp. 95–100.

[17] J. E. C. Arroyo and V. A. Armentano, "Genetic local search for multi-objective flowshop scheduling problems," *European Journal of Operational Research*, vol. 167, no. 3, pp. 717–738, 2005.

[18] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *Proceedings of Genetic and Evolutionary Computation Conference*. ACM, 2010, pp. 257–264.

[19] J. Park, S. Nguyen, M. Zhang, and M. Johnston, "Evolving ensembles of dispatching rules using genetic programming for job shop scheduling," in *Genetic Programming*. Springer, 2015, pp. 92–104.

[20] I. Kacem, S. Hammadi, and P. Borne, "Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic," *Mathematics and computers in simulation*, vol. 60, no. 3, pp. 245–276, 2002.

[21] B. Li, J. Li, K. Tang, and X. Yao, "Many-objective evolutionary algorithms: A survey," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, p. 13, 2015.

[22] H. Ishibuchi and T. Murata, "A multi-objective genetic local search algorithm and its application to flowshop scheduling," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 28, no. 3, pp. 392–403, 1998.

[23] Z. Yaqin, L. Beizhi, and W. Lv, "Study on job-shop scheduling with multi-objectives based on genetic algorithms," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, vol. 10. IEEE, 2010, pp. V10–294.

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

[25] K. Narukawa and T. Rodemann, "Examining the Performance of Evolutionary Many-Objective Optimization Algorithms on a Real-World Application." in *ICGEC*. IEEE, 2012, pp. 316–319.

[26] J. R. Kasprzyk, P. M. Reed, G. W. Characklis, and B. R. Kirsch, "Many-objective de Novo water supply portfolio planning under deep uncertainty." *Environmental Modelling and Software*, vol. 34, pp. 87–104, 2012.

[27] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational intelligence*. Springer, 2009, pp. 177–201.

[28] N. Ho and J. Tay, "Evolving dispatching rules for solving the flexible job-shop problem," in *IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 2005, pp. 2848–2855.

[29] D. Jakobović and K. Marasović, "Evolving priority scheduling heuristics with genetic programming," *Applied Soft Computing*, vol. 12, no. 9, pp. 2781–2789, 2012.

[30] E. Taillard, "Benchmarks for basic scheduling problems," *european journal of operational research*, vol. 64, no. 2, pp. 278–285, 1993.

[31] 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, 1999.

[32] S. Luke *et al.*, "A java-based evolutionary computation research system," https://cs.gmu.edu/ eclab/projects/ecj/.

[33] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[34] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, "Multiobjective optimization test instances for the CEC 2009 special session and competition," *University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report*, pp. 1–30, 2008.

[35] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.