Evolutionary learning of weighted linear composite dispatching rules for scheduling

ABSTRACT

A prevalent approach to solving job shop scheduling problems is to combine several relatively simple dispatching rules such that they may benefit each other for a given problem space. Generally, this is done on an ad-hoc basis, requiring expert knowledge from heuristics designer, or extensive exploration of suitable combinations of heuristics. The approach here, is to automate that selection, by translating dispatching rules into measurable features and optimising what their contribution should be via evolutionary search. The framework is straight forward and easy to implement and shows promising results. Various data distributions are investigated, for both job shop and flow shop problems, as is scalability for higher dimensions.

Moreover, the study showed that the choice of objective function for evolutionary search is worth investigating. Since the optimisation is based on minimising the expected mean of the fitness function over a large set of problem instances, which can vary within. Then normalising the objective function can stabilise the optimisation process away from local minima.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Integer programming*

General Terms

Algorithms, Experimentation

Keywords

Job Shop Scheduling, Composite Dispatching Rules, Evolutionary Search

1. JOB SHOP SCHEDULING

The job-shop scheduling problem (JSP) deals with the allocation of tasks of competing resources where the goal is to optimise a single or multiple objectives, in particular minimising a schedule's maximum completion time, i.e. the makespan, denoted $C_{\max}$. Due to difficulty in solving this problem, heuristics are generally applied. Perhaps the simplest approach to generating good feasible solutions for JSP is by applying dispatching rules (DR), e.g., choosing a task corresponding to longest or shortest operation time; most or least successors; or ranked positional weight, i.e., sum of operation times of its predecessors. Ties are broken in an arbitrary fashion or by another heuristic rule. Combining dispatching rules for JSP is promising, however, there is a large number of rules to choose from thus its combinations relies on expert knowledge or extensive trial-and-error process to choose a suitable DR [20]. Hence given the diversity within the JSP paradigm, there is no "one-rule-fits-all" for all problem instances (or shop constraints), however single priority dispatching rules (SDR) based on job processing attributes have proven to be effective [7]. The classical dispatching rules are continually used in research; a summary of over 100 classical DR for JSP can be found in [14]. However, careful combinations of such simple rules, i.e. composite dispatching rules (CDR), can perform significantly better [11]. As a consequence a linear composite of dispatching rule for JSP was presented by in [9]. There the goal was to learn a set of weights, $\mathbf{w}$ via logistic regression such that

$$h(\mathbf{x}_j) = \left\langle \mathbf{w} \cdot \phi(\mathbf{x}_j) \right\rangle, \tag{1}$$

yields the preference estimate for dispatching job j that corresponds to post-decision state $\mathbf{x}_j$, where $\phi(\mathbf{x}_j)$ denotes the feature mapping. The job dispatched is the following,

$$j^* = \arg\max_j \left\{ h(\mathbf{x}_j) \right\}. \tag{2}$$

A more popular approach in recent JSP literature is applying genetic algorithms (GAs) [15]. However, in that case an extensive number of schedules need to be evaluated, and even for low dimensional JSP that can quickly become computationally infeasible. GAs can be used directly on schedules [3, 4, 21, 16, 1], however, then there are many concerns that need to be dealt with. To begin with there are nine encoding schemes for representing the schedules [3], in addition, special care must be taken when applying cross-over and mutation operators in order for schedules to still remain feasible. Moreover in case of JSP, GAs are not adapt for fine-tuning around optima, luckily a subsequent local search can mediate the optimisation [4].

Another approach is to apply GAs indirectly to JSP, via

GECCO'14, July 12-16, 2014, Vancouver, BC, Canada.

dispatching rules, i.e. dispatching rules based genetic algorithms (DRGA) [22, 5, 13] where a solution is no longer a *proper* schedule but a *representation* of a schedule via applying certain DRs consecutively. DRGA are a special case of genetic programming [12] which is the most predominant approach in hyper-heuristics is a framework of creating *new* heuristics from a set of predefined heuristics via GA optimisation [2].

There are two main viewpoints on how to approach scheduling problems, *a*) local level by building schedules for one problem instance at a time; and *b*) global level by building schedules for all problem instances at once. For local level construction a simple construction heuristic is applied, the schedule's features are collected at each dispatch iteration, from which a learning model will inspect the feature set to discriminate which operations are preferred to others via ordinal regression. The focus is essentially on creating a meaningful preference set composed of features and their ranks, as the learning algorithm is only run once to find suitable operators for the value function. This is the approach taken in [9]. Expanding on that work, this study will explore global level construction viewpoint, where there is no feature set collected beforehand since the learning model is optimised directly via evolutionary search. This involves numerous costly value function evaluations. In fact it involves an indirect method of evaluation whether one learning model is preferable to another, w.r.t. which one yields a better expected mean.

2. OUTLINE

In order to formulate the relationship between problem structure and heuristic efficiency one can utilise Rice's framework for algorithm selection [17]. The framework consists of four fundamental components, namely,

Problem space or instance space $\mathcal{P}$,
set of problem instances;

Feature space $\mathcal{F}$,
measurable properties of the instances in $\mathcal{P}$;

Algorithm space $\mathcal{A}$,
set of all algorithms under inspection;

Performance space $\mathcal{Y}$,
the outcome for $\mathcal{P}$ using an algorithm from $\mathcal{A}$.

For a given problem instance $\mathbf{x} \in \mathcal{P}$ with k features $\phi(\mathbf{x}) = \{\phi_1(\mathbf{x}), ..., \phi_k(\mathbf{x})\} \in \mathcal{F}$ and using algorithm $a \in \mathcal{A}$ the performance is $y = Y(a, \phi(\mathbf{x})) \in \mathcal{Y}$, where $Y : \mathcal{A} \times \mathcal{F} \mapsto \mathcal{Y}$ is the mapping for algorithm and feature space onto the performance space. [18, 19, 10] formulate JSP in the following manner: *a*) problem space $\mathcal{P}$ is defined as the union of N problem instances consisting of processing time and ordering matrices given in section 3; *b*) feature space $\mathcal{F}$, which is outlined in section 4. Note, these are not the only possible set of features, however, they are built on the work by [9, 18] and deemed successful in capturing the essence of a JSP data structure; *c*) algorithm space $\mathcal{A}$ is simply the scheduling policies under consideration and discussed in section 5; *d*) performance space is based on the resulting $C_{\max}$. Different fitness measures are investigated in section 5.1; and *e*) mapping Y is the step-by-step scheduling process.

In the context of Rice's framework, and returning to the aforementioned approaches to scheduling problems, then the objective is to maximise its expected performance, i.e.

a) Local level

$$\max_{\mathcal{P}' \subset \mathcal{P}} \mathbb{E}\left[Y\left(a, \phi(\mathbf{x})\right)\right] \tag{3}$$

where $\mathbf{x} \in \mathcal{P}'$ and algorithm a is obtained via ordinal regression based on the feature space $\mathcal{F}$, i.e. $\mathcal{F}|_{\mathcal{P}'} \mapsto \mathcal{A}$, such as the approach taken in [9], and will be used as a benchmark for the following,

b) Global level

$$\max_{a \in \mathcal{A}} \mathbb{E}\left[Y\left(a, \phi(\mathbf{x})\right)\right] \tag{4}$$

where training data $\mathbf{x} \in \mathcal{P}$ is guided by its algorithm a, i.e. $\mathcal{A} \mapsto \mathcal{P}$. This will be the focus of this study.

Note that the mappings $\phi : \mathcal{P} \mapsto \mathcal{F}$ and $Y : \mathcal{A} \mapsto \mathcal{Y}$ are the same for both paradigms.

The paper concludes in section 6 with discussion and conclusions.

3. PROBLEM SPACE

For this study synthetic JSP and its subclass, permutation flow shop problem (PFSP), data instances will be considered with the problem size $n \times m$, where n and m denotes number of jobs and machines, respectively.

There are two fundamental types of problem classes: non-structured versus structured. Firstly there are the "conventional" structured problem classes, where problem instances for are generated stochastically by fixing the number of jobs and machines and processing time are i.i.d. and sampled from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e. $p \sim \mathcal{U}(u_1, u_2)$. Two different processing times distributions are explored, namely $\mathcal{P}_{jrnd}$ where $I = [1, 99]$ and $\mathcal{P}_{jrndn}$ where $I = [45, 55]$, referred to as random and random-narrow, respectively. The machine order is a random permutation of all of the machines in the job-shop.

Analogous to $\mathcal{P}_{jrnd}$ and $\mathcal{P}_{jrndn}$ the problem classes $\mathcal{P}_{frnd}$ and $\mathcal{P}_{frndn}$, respectively, correspond to the structured PFSP problem classes, however with a homogeneous machine order permutation. Secondly, there is a structured problem classes of PFSP which is modelled after real-world flow-shop manufacturing, namely, job-correlated $\mathcal{P}_{fjc}$, where job processing times are dependent on job index, however independent of machine index. Problem instances for PFSP are generated using [23] problem generator[1].

For each JSP and PFSP class N_{train} and N_{test} instances were generated for training and testing, respectively. Values for N are given in table 1. Note, that difficult problem instances are not filtered out beforehand, such as the approach in [23].

[1]Both code, written in `C++`, and problem instances used in their experiments can be found at: `http://www.cs.colostate.edu/sched/generator/`

Table 1: Problem space distributions used in section 5. Note, problem instances are synthetic and each problem space is i.i.d. and '–' denotes not available.

name	size ($n \times m$)	N_{train}	N_{test}	note
Permutation flow shop problem (PFSP)				
$\mathcal{P}^{6\times5}_{frnd}$	6×5	500	–	random
$\mathcal{P}^{6\times5}_{frndn}$	6×5	500	–	random-narrow
$\mathcal{P}^{6\times5}_{fjc}$	6×5	500	–	job-correlated
$\mathcal{P}^{10\times10}_{frnd}$	10×10	–	500	random
$\mathcal{P}^{10\times10}_{frndn}$	10×10	–	500	random-narrow
$\mathcal{P}^{10\times10}_{fjc}$	10×10	–	500	job-correlated
Job shop problem (JSP)				
$\mathcal{P}^{6\times5}_{jrnd}$	6×5	500	–	random
$\mathcal{P}^{6\times5}_{jrndn}$	6×5	500	–	random-narrow
$\mathcal{P}^{10\times10}_{jrnd}$	10×10	–	500	random
$\mathcal{P}^{10\times10}_{jrndn}$	10×10	–	500	random-narrow

4. FEATURE SPACE

When building a complete JSP schedule $\ell = n \cdot m$ dispatches must be made sequentially. A job is placed at the earliest available time slot for its next machine, whilst still fulfilling constraints that each machine can handle at most one job at each time, and jobs need to have finished their previous machines according to its machine order. Unfinished jobs are dispatched one at a time according to some heuristic. After each dispatch the schedule's current features are updated. Features are used to grasp the essence of the current state of the schedule. Temporal scheduling features applied in this study, for each possible post-decision state, are given in table 2. An example of a schedule being built is given in fig. 1, where there are a total of five possible jobs that could be chosen to be dispatched next by some dispatching rule. These features would serve as the input for eq. (1).

It's noted that some of the features directly correspond to a SDR commonly used in practice, for example if the weights were zero, save for $w_6 = 1$, then (2) yields the job with the highest ϕ_6 value, i.e. equivalent to dispatching rule most work remaining (MWR).

5. EXPERIMENTAL STUDY

The optimum makespan is denoted $C^{\text{opt}}_{\max}$, and the makespan obtained from the linear learning model by $C^{\text{model}}_{\max}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho := \frac{C^{\text{model}}_{\max} - C^{opt}_{\max}}{C^{\text{opt}}_{\max}} \cdot 100\% \tag{5}$$

which indicates the percentage relative deviation from optimality.

Inspired by DRGA, the approach taken in this study is to optimise the weights $\mathbf{w}$ in eq. (1) directly, via evolutionary search such as covariance matrix adaptation evolution strategy (CMA-ES) [6], which has been proven to be a very efficient numerical optimisation technique.

Using standard set-up of parameters of the CMA-ES optimisation, the runtime was limited to 288 hours on a cluster for each training set given in section 3, and in every case the optimisation reached its maximum walltime.

Table 2: Feature space $\mathcal{F}$ for $\mathcal{P}$ given the resulting temporal schedule after dispatching an operation.

ϕ	Feature description
ϕ_1	job processing time
ϕ_2	job start-time
ϕ_3	job end-time
ϕ_4	when machine is next free
ϕ_5	current makespan
ϕ_6	total work remaining for job
ϕ_7	most work remaining for all jobs
ϕ_8	total idle time for machine
ϕ_9	total idle time for all machines
ϕ_{10}	ϕ_9 weighted w.r.t. number of assigned tasks
ϕ_{11}	time job had to wait
ϕ_{12}	idle time created
ϕ_{13}	total processing time for job

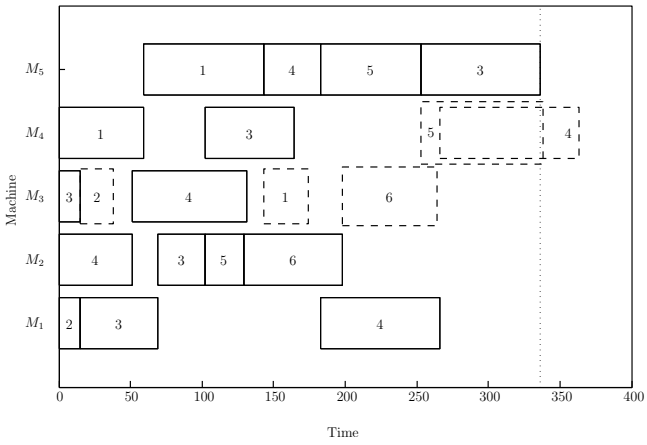


Figure 1: Gantt chart of a partial JSP schedule after 15 dispatches: Solid boxes represent previously dispatched jobs, and dashed boxes represent the jobs that could be scheduled next. Current $C_{\max}$ denoted as dotted line.

5.1 Performance measures

Generally, evolutionary search only needs to minimise the expected fitness value, however the approach in [9] was to use the known optimum to correctly label which operations' features were indeed optimal compared to other possible operations, then it would be of interest to inspect if there is any performance edge gained in incorporating optimal labelling in evolutionary search. Therefore, two objective functions will be considered, namely,

$$ES_{C_{\max}} := \min \mathbb{E}[C_{\max}] \tag{6}$$

$$ES_{\rho} := \min \mathbb{E}[\rho] \tag{7}$$

Main statistics of the experimental run are given in table 3 and depicted in fig. 3 for both approaches. In addition,

Table 3: Final results for CMA-ES optimisation.

$\mathcal{P}$	#gen	#eval	$\mathrm{ES}_{C_{\max}}$	#gen	#eval	ES_{ρ}
jc	5984	65835	567.688	1625	17886	0.361
frnd	5088	55979	571.394	4546	50006	7.479
frndn	5557	61138	544.764	2701	29722	0.938
jrnd	4707	51788	448.612	1944	21395	8.258
jrndn	4802	52833	449.942	1974	21725	8.691

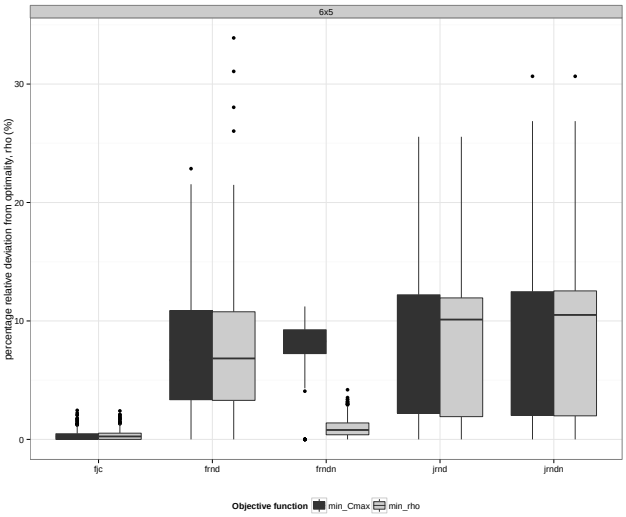


Figure 2: Box-plot of training data for percentage relative deviation from optimality, defined by eq. (5), when implementing the final weights obtained from CMA-ES optimisation, using both objective functions from eqs. (6) and (7), left and right, respectively.

evolving decision variables, here weights $\mathbf{w}$ for eq. (1), are depicted in fig. 4.

In order to compare the two objective functions, the best weights reported were used for eq. (1) on the corresponding training data. Its box-plot of percentage relative deviation from optimality, defined by eq. (5), is depicted in fig. 2 and main statistics detailed in table 4.

In the case of $\mathcal{P}_{frndn}$, eq. (7) gave a considerably worse results, since the optimisation got trapped in a local minimum, as the erratic evolution of the weighs in fig. 4a suggest. For other problem spaces, eq. (6) gave slightly better results than eq. (7), however, there was no statical difference between adopting either objective function. Therefore, minimisation of expectation of ρ, is preferred over simply using the unscaled resulting makespan.

5.2 Problem difficulty

The evolution of fitness per generation from the CMA-ES optimisation of eq. (7) is depicted in fig. 3, and since all problem spaces reached their allotted computational time, without converging. In fact $\mathcal{P}_{frnd}$ and $\mathcal{P}_{jrndn}$ needed restarting during the optimisation process. Furthermore, the evolution of the decision variables, $\mathbf{w}$, are depicted in fig. 4. As one can see, the relative contribution for each weight clearly differs between problem spaces. Note that in the case of $\mathcal{P}_{jrndn}$

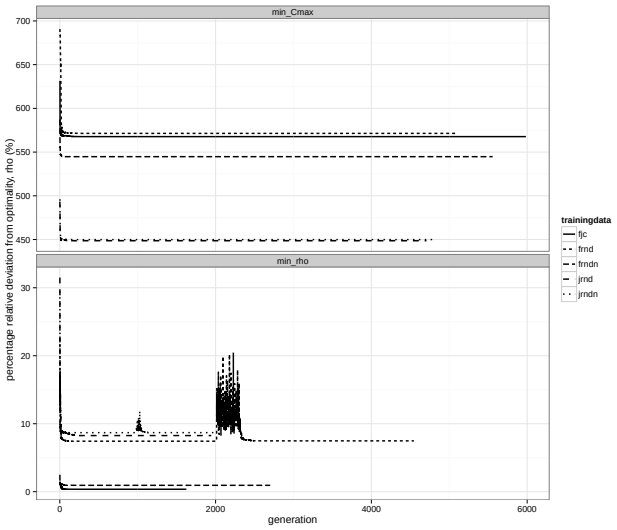


Figure 3: Fitness for optimising (w.r.t. eqs. (6) and (7) above and below, receptively), per generation of the CMA-ES optimisation.

(cf. fig. 4b), CMA-ES restarts around generation 1,000 and quickly converges back to its previous fitness, however lateral relation of weights has completely changed. Implying that there are many optimal combinations of weights to be used, which can be expected due to the fact some features in table 2 are a linear combination of one others, e.g. $\phi_3 = \phi_1 + \phi_2$.

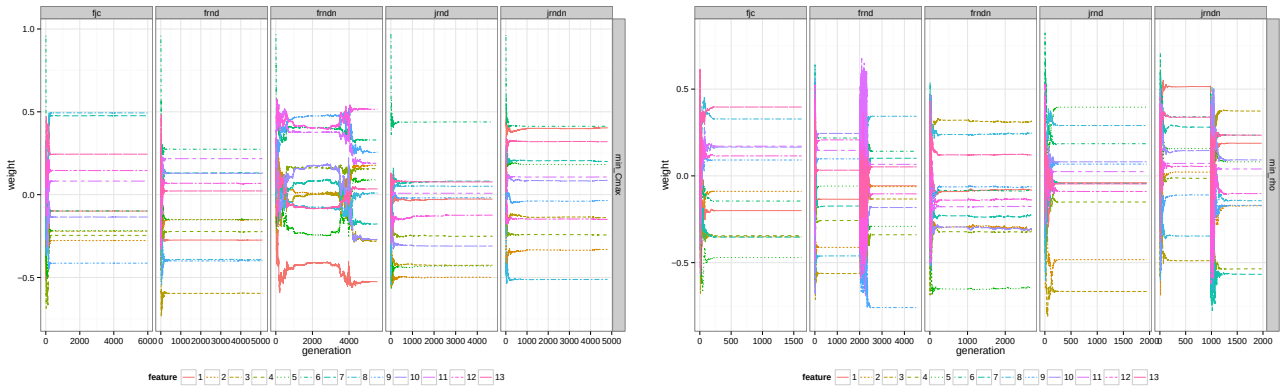
5.3 Robustness and scalability

As a benchmark, the linear ordinal regression model (PREF) from [9] was created. Using the weights obtained from optimising eq. (7) and applying them on their 6×5 training data, their main statistics of eq. (5) are reported in table 4, for all training sets described in table 1. Moreover, the best SDR, from which the features in table 2 were inspired by, are also reported for comparison, i.e. most work remaining (MWR) for all JSP problem spaces, and least work remaining (LWR) for all PFSP problem spaces.

To explore the scalability of the learning methods, a similar comparison to section 5.3 is made for the applying the learning models on their corresponding 10×10 testing data, results are reported in table 5. Note that only resulting $C_{\max}$ is reported, as the optimum makespan is not known.

6. DISCUSSION AND CONCLUSIONS

Data distributions considered in this study either varied w.r.t. the processing times distributions, continuing the preliminary experiments in [9] , or w.r.t. the job ordering permutations, i.e. homogeneous σ matrices in PFSP versus heterogeneous σ matrices in JSP. From the results based on 6×5 training data, given in table 4, it's obvious that CMA-ES optimisation substantially outperforms the previous PREF methods from [9], for all problem spaces considered. Furthermore, the results hold when testing on 10×10, (cf. table 5), suggesting the method is indeed scalable for higher dimensions.



(a) minimise w.r.t. eq. (6)

(b) minimise w.r.t. eq. (7)

Figure 4: Evolution of weights of features (given in table 2) at each generation of the CMA-ES optimisation. Note, weights are normalised such that $\|\mathbf{w}\| = 1$.

Moreover, the study showed that the choice of objective function for evolutionary search is worth investigating. There was no statistical difference from minimising the fitness function directly and its normalisation w.r.t. true optimum (cf. eqs. (6) and (7)), save for $\mathcal{P}_{frndn}$. Implying, even though ES doesn't rely on optimal solutions, there are some problem spaces where it can be of great benefit. This is due to the fact that the problem instances can vary greatly within the same problem space [10], thus normalising the objective function would help the evolutionary search to deviate the from giving too much weight for problematic problem instances for the greater good.

The main drawback of using evolutionary search for learning optimal weights for eq. (1) is how computationally expensive it is to evaluate the mean expected fitness. Even for a low problem dimension, 6-job 5-machine JSP, each optimisation run reached their walltime of 288hrs, without converging. Now, 6×5 JSP requires 30 sequential dispatches, where at each time step there are up to 6 jobs to choose from, i.e. its complexity is $\mathcal{O}(n^{n \cdot m})$, making it computationally infeasible to apply this framework for higher dimensions as is. However, evolutionary search only requires the rank of the candidates, and therefore it is appropriate to retain a sufficiently accurate surrogate for the value function during evolution in order to reduce the number of costly true value function evaluations, such as the approach in [8]. This could reduce the computational cost of the evolutionary search considerably, making it feasible to conduct the experiments from section 5 for problems of higher dimensions, e.g. with these adjustments it is possible to train on 10×10 and test on for example 14×14 to verify whether scalability holds for even higher dimensions.

7. REFERENCES

[1] B. Ak and E. Koc. A Guide for Genetic Algorithm Based on Parallel Machine Scheduling and Flexible Job-Shop Scheduling. *Procedia - Social and Behavioral Sciences*, 62:817–823, Oct. 2012.

[2] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[3] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithmsI. Representation. *Computers & Industrial Engineering*, 30(4):983–997, 1996.

[4] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, 36(2):343–364, Apr. 1999.

[5] A. Dhingra and P. Chandna. A bi-criteria M-machine SDST flow shop scheduling using modified heuristic genetic algorithm. *International Journal of Engineering, Science and Technology*, 2(5):216–225, 2010.

[6] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, June 2001.

[7] R. Haupt. A survey of priority rule-based scheduling. *OR Spectrum*, 11:3–16, 1989.

[8] H. Ingimundardottir and T. P. Runarsson. Sampling strategies in ordinal regression for surrogate assisted evolutionary optimization. In *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on*, pages 1158–1163, nov. 2011.

[9] H. Ingimundardottir and T. P. Runarsson. Supervised learning linear priority dispatch rules for job-shop scheduling. In C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 263–277. Springer Berlin, Heidelberg, 2011.

[10] H. Ingimundardottir and T. P. Runarsson. Determining the characteristic of difficult job shop scheduling instances for a heuristic solution method. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 408–412. Springer Berlin Heidelberg, 2012.

[11] M. Jayamohan and C. Rajendran. Development and analysis of cost-based dispatching rules for job shop

scheduling. *European Journal of Operational Research*, 157(2):307–321, 2004.

[12] J. R. Koza and R. Poli. Genetic programming. In E. Burke and G. Kendal, editors, *Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 5. Springer, 2005.

[13] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology*, Feb. 2013.

[14] S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.

[15] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3 edition, 2008.

[16] R. Qing-dao-er ji and Y. Wang. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 39(10):2291–2299, Oct. 2012.

[17] J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

[18] K. Smith-Miles, R. James, J. Giffin, and Y. Tu. A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. In T. Stützle, editor, *Learning and Intelligent Optimization*, volume 5851 of *Lecture Notes in Computer Science*, pages 89–103. Springer Berlin, Heidelberg, 2009.

[19] K. Smith-Miles and L. Lopes. Generalising algorithm performance in instance space: A timetabling case study. In C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 524–538. Springer Berlin, Heidelberg, 2011.

[20] J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective

Table 4: Main statistics of percentage relative deviation from optimality, ρ, defined by eq. (5) for various models, using corresponding 6×5 training data.

(a) $\mathcal{P}^{6\times5}_{jrnd}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	8.54	10	6	0	26
ES_{ρ}	8.26	10	6	0	26
PREF	10.18	11	7	0	30
MWR	16.48	16	9	0	45

(b) $\mathcal{P}^{6\times5}_{jrndn}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	8.68	11	6	0	31
ES_{ρ}	8.69	11	6	0	31
PREF	10.00	11	6	0	31
MWR	14.02	13	8	0	37

(c) $\mathcal{P}^{6\times5}_{frnd}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	7.44	7	5	0	23
ES_{ρ}	7.48	7	5	0	34
PREF	9.87	9	7	0	38
LWR	20.05	19	10	0	71

(d) $\mathcal{P}^{6\times5}_{frndn}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	8.09	8	2	0	11
ES_{ρ}	0.94	1	1	0	4
PREF	2.38	2	1	0	7
LWR	2.25	2	1	0	7

(e) $\mathcal{P}^{6\times5}_{fjc}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	0.33	0	0	0	2
ES_{ρ}	0.36	0	0	0	2
PREF	1.08	1	1	0	5
LWR	1.13	1	1	0	6

Table 5: Main statistics of $C_{\max}$ for various models, using corresponding 10×10 test data.

(a) $\mathcal{P}^{10\times10}_{jrnd}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	922.51	914	73	741	1173
ES_{ρ}	931.37	931	71	735	1167
PREF	1011.38	1004	82	809	1281
MWR	997.01	992	81	800	1273

(b) $\mathcal{P}^{10\times10}_{jrndn}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	855.85	857	50	719	1010
ES_{ρ}	855.91	856	51	719	1020
PREF	899.94	898	56	769	1130
MWR	897.39	898	56	765	1088

(c) $\mathcal{P}^{10\times10}_{frnd}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	1178.73	1176	80	976	1416
ES_{ρ}	1181.91	1179	80	984	1404
PREF	1215.20	1212	80	1006	1450
LWR	1284.41	1286	85	1042	1495

(d) $\mathcal{P}^{10\times10}_{frndn}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	1065.48	1059	32	992	1222
ES_{ρ}	980.11	980	8	957	1006
PREF	987.49	988	9	958	1011
LWR	986.94	987	9	959	1010

(e) $\mathcal{P}^{10\times10}_{fjc}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	1135.44	1134	286	582	1681
ES_{ρ}	1135.47	1134	286	582	1681
PREF	1136.02	1135	286	582	1685
LWR	1136.49	1141	287	581	1690

flexible job-shop problems. *Computers and Industrial Engineering*, 54(3):453–473, 2008.

[21] J.-T. Tsai, T.-K. Liu, W.-H. Ho, and J.-H. Chou. An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. *The International Journal of Advanced Manufacturing Technology*, 38(9-10):987–994, Aug. 2007.

[22] J. A. Vázquez-Rodríguez and S. Petrovic. A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793, Dec. 2009.

[23] J.-P. Watson, L. Barbulescu, L. D. Whitley, and A. E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14:98–123, 2002.