Evolutionary Learning of Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardottir and Thomas Philip Runarsson

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 in an ad-hoc fashion, requiring expert knowledge from heuristics designers, 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 shows 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 the set, then normalising the objective function can stabilise the optimisation process away from local minima.

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

H. Ingimundardottir · T.P. Runarsson (✉)
Industrial Engineering, Mechanical Engineering and Computer Science,
University of Iceland, Reykjavik, Iceland
e-mail: tpr@hi.is

H. Ingimundardottir
e-mail: hei2@hi.is

J.J. Merelo et al. (eds.), *Computational Intelligence*,
Studies in Computational Intelligence 620, DOI 10.1007/978-3-319-26393-9_4

applying dispatching rules (DR), e.g., choosing a task corresponding to longest or shortest processing time, most or least successors, or ranked positional weight, i.e., sum of processing 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 rely on expert knowledge or extensive trial-and-error process to choose a suitable DR [21]. 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 [8]. The classical dispatching rules are continually used in research; a summary of over 100 classical DRs for JSP can be found in [16]. However, careful combinations of such simple rules, i.e., composite dispatching rules (CDRs) can perform significantly better [12]. As a consequence, a linear composite of dispatching rules for JSP was presented in [10]. There the goal was to learn a set of weights, $\mathbf{w}$ via ordinal regression such that

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

yields the preference estimate for dispatching job j that corresponds to post-decision state $\mathbf{x}_j$, where $\boldsymbol{\phi}(\mathbf{x}_j)$ denotes the feature mapping (cf. Sect. 4). In short, Eq. 1 is a simple linear combination of features found using a classifier which is trained by giving more weight to instances that are preferred w.r.t. optimality in a supervised learning fashion. As a result, 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) [17]. However, in that case an extensive number of schedules need to be evaluated, and even for low dimensional JSP, it can quickly become computationally infeasible. GAs can be used directly on schedules [1, 3, 4, 13, 22], 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].

The most predominant approach in hyper-heuristics, a framework of creating *new* heuristics from a set of predefined heuristics, is genetic programming [2]. Dispatching rules based genetic algorithms (DRGA) [5, 15, 23] are a special case of genetic programming [14], where GAs are applied indirectly to JSP via dispatching rules, i.e., where a solution is no longer a *proper* schedule but a *representation* of a schedule via applying certain DRs consecutively.

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 [10]. Expanding on that work, this study will explore a 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 [18]. The framework consists of four fundamental components, namely,

Problem Space or Instance Space $\mathscr{P}$,
set of problem instances;
Feature Space $\mathscr{F}$,
measurable properties of the instances in $\mathscr{P}$;
Algorithm Space $\mathscr{A}$,
set of all algorithms under inspection;
Performance Space $\mathscr{Y}$,
the outcome for $\mathscr{P}$ using an algorithm from $\mathscr{A}$.

For a given problem instance $\mathbf{x} \in \mathscr{P}$ with k features $\boldsymbol{\phi}(\mathbf{x}) = \{\phi_1(\mathbf{x}), \ldots, \phi_k(\mathbf{x})\} \in \mathscr{F}$ and using algorithm $a \in \mathscr{A}$ the performance is $y = Y(a, \boldsymbol{\phi}(\mathbf{x})) \in \mathscr{Y}$, where $Y : \mathscr{A} \times \mathscr{F} \mapsto \mathscr{Y}$ is the mapping for algorithm and feature space onto the performance space. [11, 19, 20] formulate JSP in the following manner: (*a*) problem space $\mathscr{P}$ is defined as the union of N problem instances consisting of processing time and ordering matrices given in Sect. 3; (*b*) feature space $\mathscr{F}$, which is outlined in Sect. 4. Note, these are not the only possible set of features, however, they are built on the work by [10, 19] and deemed successful in capturing the essence of a JSP data structure; (*c*) algorithm space $\mathscr{A}$ is simply the scheduling policies under consideration and discussed in Sect. 5; (*d*) performance space is based on the resulting $C_{\max}$. Different fitness measures are investigated in Sect. 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 heuristic performance, i.e.,

(*a*) Local level

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

where $\mathbf{x} \in \mathscr{P}'$ and algorithm a is obtained via ordinal regression based on the feature space $\mathscr{F}$, i.e., $\mathscr{F}|_{\mathscr{P}'} \mapsto \mathscr{A}$, such as the approach taken in [10], and will be used as a benchmark for the following,

(*b*) Global level

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

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

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

The paper concludes in Sect. 6 with discussion and conclusions.

3 Problem Space

For this study synthetic JSP and its subclass, permutation flow shop problem (PFSP), the scheduling task considered here is where n jobs are scheduled on a set of m machines, i.e., problem size $n \times m$, subject to the constraint that each job must follow a predefined machine order and that a machine can handle at most one job at a time. The pair (j, a) refers to the operation of dispatching job j on machine a. As a result, a total of $\ell = n \cdot m$ sequential operations need to be made for a complete schedule.

The objective is to schedule the jobs so as to minimize the maximum completion times, $C_{\max}$, also known as the makespan. For a mathematical formulation of JSP the reader is recommended [10].

There are two fundamental types of problem classes: non-structured versus structured. Firstly there are the "conventional" structured problem classes, where problem instances are generated stochastically by fixing the number of jobs and machines, as well as processing times are i.i.d. and sampled from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e., $p \sim \mathscr{U}(u_1, u_2)$. Two different processing time distributions are explored, namely $\mathscr{P}_{j.rnd}$ where $I = [1, 99]$ and $\mathscr{P}_{j.rndn}$ 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 $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$ the problem classes $\mathscr{P}_{f.rnd}$ and $\mathscr{P}_{f.rndn}$, respectively, correspond to the structured PFSP problem classes, however with a homogeneous machine order permutation. Secondly, there are structured problem classes of PFSP which are modelled after real-world flow-shop manufacturing namely job-correlated $\mathscr{P}_{f.jc}$ where job processing times are dependent on job index and

Table 1 Problem space distributions used in Sect. 5

Name	Size	N_{train}	N_{test}	Note
Permutation flow shop problem (PFSP)				
$\mathscr{P}^{6\times5}_{f.rnd}$	6×5	500	–	Random
$\mathscr{P}^{6\times5}_{f.rndn}$	6×5	500	–	Random-narrow
$\mathscr{P}^{6\times5}_{f.jc}$	6×5	500	–	Job-correlated
$\mathscr{P}^{10\times10}_{f.rnd}$	10×10	–	500	Random
$\mathscr{P}^{10\times10}_{f.rndn}$	10×10	–	500	Random-narrow
$\mathscr{P}^{10\times10}_{f.jc}$	10×10	–	500	Job-correlated
Job shop problem (JSP)				
$\mathscr{P}^{6\times5}_{j.rnd}$	6×5	500	–	Random
$\mathscr{P}^{6\times5}_{j.rndn}$	6×5	500	–	Random-narrow
$\mathscr{P}^{10\times10}_{j.rnd}$	10×10	–	500	Random
$\mathscr{P}^{10\times10}_{j.rndn}$	10×10	–	500	Random-narrow

Note Problem instances are synthetic and each problem space is i.i.d. and '–' denotes not available

independent of machine index. Problem instances for PFSP are generated using [24] 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, difficult problem instances are not filtered out beforehand, such as the approach in [24].

4 Feature Space

When building a complete JSP schedule, a job is placed at the earliest available time slot for its next machine while still fulfilling constraints that each machine can handle at most one job at a 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. As seen in Table 2, temporal scheduling features applied in this study are given for each possible post-decision state. 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 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 $\mathbf{w}$ in Eq. 1 were all zero, save for $w_6 = 1$, then Eq. 2 yields the job with the highest ϕ_6 value, i.e., equivalent to dispatching rule most work remaining (MWR).

[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 2 Feature space $\mathscr{F}$ for $\mathscr{P}$ given the resulting temporal schedule after dispatching an operation (j, a)

$\boldsymbol{\phi}$	Feature description
ϕ_1	Job j processing time
ϕ_2	Job j start-time
ϕ_3	Job j end-time
ϕ_4	When machine a is next free
ϕ_5	Current makespan
ϕ_6	Total work remaining for job j
ϕ_7	Most work remaining for all jobs
ϕ_8	Total idle time for machine a
ϕ_9	Total idle time for all machines
ϕ_{10}	ϕ_9 weighted w.r.t. number of assigned tasks
ϕ_{11}	Time job j had to wait
ϕ_{12}	Idle time created
ϕ_{13}	Total processing time for job j

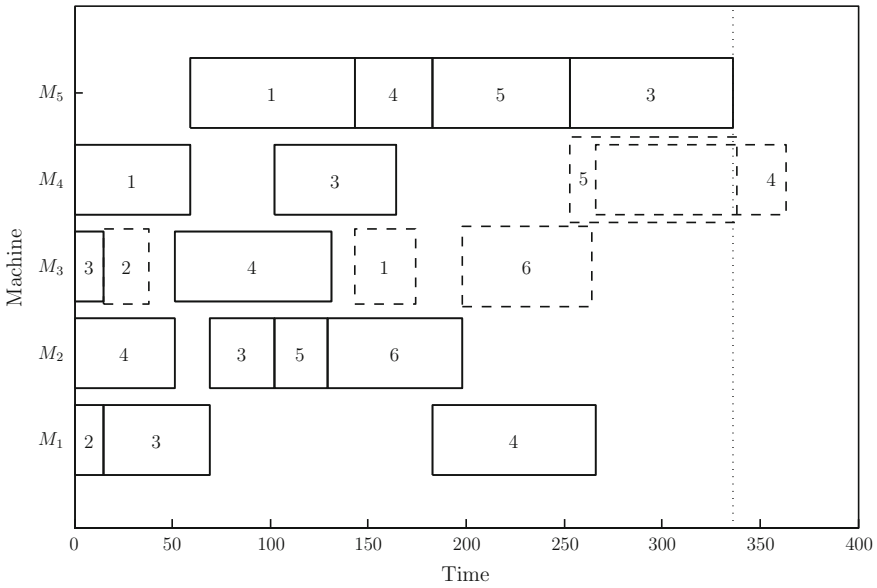

Fig. 1 Gantt chart of a partial JSP schedule after 15 operations: *Solid boxes* represent previously dispatched jobs, and *dashed boxes* represent the jobs that could be scheduled next. Current $C_{\max}$ denoted as *dotted line*

5 Experimental Study

The optimum makespan[2] is denoted $C_{\max}^{\text{opt}}$, and the makespan obtained from the heuristic model by $C_{\max}^{\text{model}}$. Since the optimal makespan varies between problem instances the performance measure is the following,

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

which indicates the percentage relative deviation from optimality. Throughout a Kolmogorov-Smirnov test with $\alpha = 0.05$ is applied to determine statistical significance between methodologies.

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) [7]. This 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 h on a cluster for each training set given in Sect. 3 and in every case the optimisation reached its maximum walltime.

5.1 Performance Measures

Generally, evolutionary search only needs to minimise the expected fitness value. However, the approach in [10] was to use the known optimum to correctly label which operations' features were optimal when compared to other possible operations (Fig. 2). Therefore, it would be of interest to inspect if there is any performance edge gained by 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, 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 Table 4 present its main statistics; mean, median, standard deviation, minimum and maximum values.

[2]Optimum values are obtained by using a commercial software package [6].

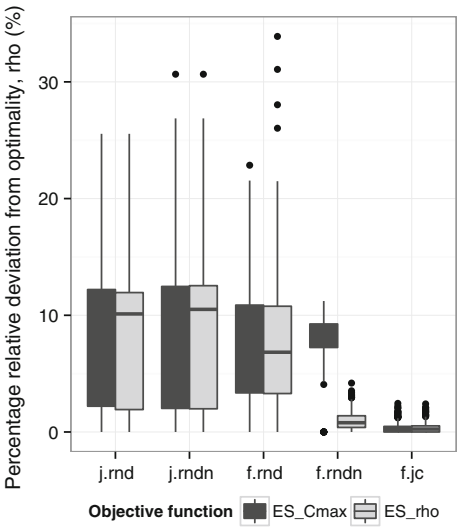

Fig. 2 Box-plot of training data for percentage relative deviation from optimality, defined by Eq. (5), when implementing the final weights obtained from CMA-ES optimisation, using both objective functions from Eqs. (6) and (7), *left* and *right*, respectively

Table 3 Final results for CMA-ES optimisation; total number of generations and function evaluations and its resulting fitness value for both performance measures considered

(a) w.r.t. Eq. 6			
$\mathscr{P}$	#gen	#eval	$ES_{C_{max}}$
j.rnd	4707	51788	448.612
j.rndn	4802	52833	449.942
f.rnd	5088	55979	571.394
f.rndn	5557	61138	544.764
f.jc	5984	65835	567.688
(b) w.r.t. Eq. 7			
$\mathscr{P}$	#gen	#eval	ES_{ρ}
j.rnd	1944	21395	8.258
j.rndn	1974	21725	8.691
f.rnd	4546	50006	7.479
f.rndn	2701	29722	0.938
f.jc	1625	17886	0.361

In the case of $\mathscr{P}_{f.rndn}$, Eq. 6 gave a considerably worse results, since the optimisation got trapped in a local minima, as the erratic evolution of the weights in Fig. 4a suggest. For other problem spaces, Eq. 6 gave slightly better results than Eq. 7. However, there was no statistical difference between adopting either objective function. Therefore, minimisation of expectation of ρ, is preferred over simply using the unscaled resulting makespan.

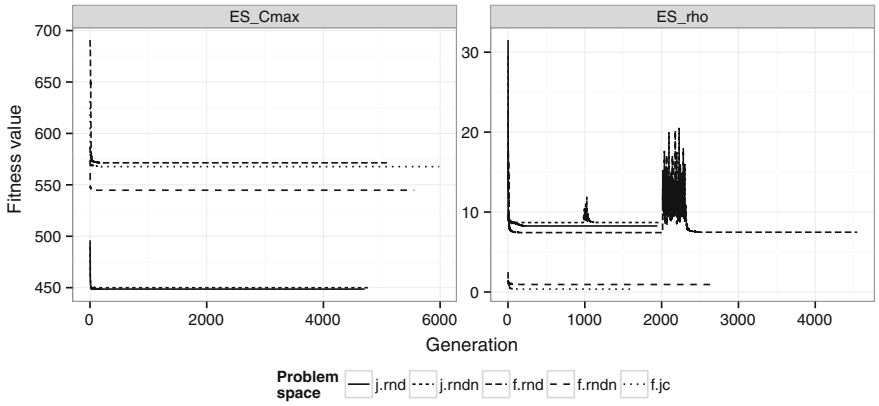

Fig. 3 Fitness for optimising (w.r.t. Eqs. (6) and (7) *above* and *below*, receptively), per generation of the CMA-ES optimisation

5.2 Problem Difficulty

The evolution of fitness per generation from the CMA-ES optimisation of Eq. 7 is depicted in Fig. 3. Note, all problem spaces reached their allotted computational time without converging. In fact $\mathscr{P}_{f.rnd}$ and $\mathscr{P}_{j.rndn}$ 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 $\mathscr{P}_{j.rndn}$ (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. This can be expected due to the fact some features in Table 2 are a linear combination of others, e.g. $\phi_3 = \phi_1 + \phi_2$.

5.3 Scalability

As a benchmark, the linear ordinal regression model (PREF) from [10] 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 models, a similar comparison to Sect. 5.2 is made for 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 and Eq. 5 is not applicable.

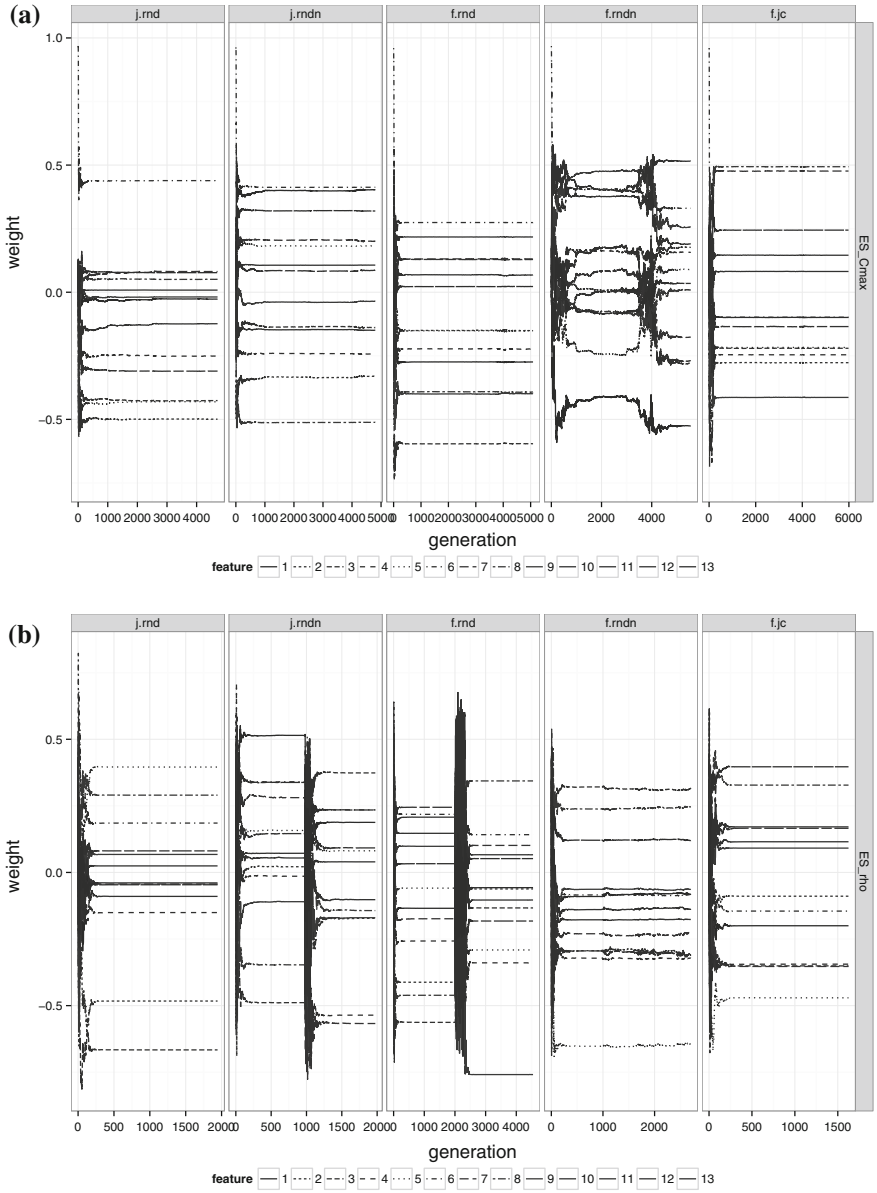

Fig. 4 Evolution of weights of features (given in Table 2) at each generation of the CMA-ES optimisation. Note, weights are normalised such that $||\mathbf{w}|| = 1$. **a** Minimise w.r.t. Eq. 6. **b** Minimise w.r.t. Eq. 7

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) $\mathscr{P}^{6\times5}_{j.rnd}$					
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) $\mathscr{P}^{6\times5}_{j.rndn}$					
$ES_{C_{max}}$	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) $\mathscr{P}^{6\times5}_{f.rnd}$					
$ES_{C_{max}}$	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) $\mathscr{P}^{6\times5}_{f.rndn}$					
$ES_{C_{max}}$	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) $\mathscr{P}^{6\times5}_{f.jc}$					
$ES_{C_{max}}$	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

6 Discussion and Conclusions

Data distributions considered in this study either varied w.r.t. the processing time distributions, continuing the preliminary experiments in [10] , or w.r.t. the job ordering permutations—i.e., homogeneous machine order for PFSP versus heterogeneous machine order for 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 [10] for all problem spaces considered. Furthermore, the results hold when testing on 10×10 (cf. Table 5), suggesting the method is indeed scalable to higher dimensions.

Moreover, the study showed that the choice of objective function for evolutionary search is worth investigating. There was no statistical difference from minimising the

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

(a) $\mathscr{P}^{10\times10}_{j.rnd}$					
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) $\mathscr{P}^{10\times10}_{j.rndn}$					
$ES_{C_{\max}}$	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) $\mathscr{P}^{10\times10}_{f.rnd}$					
$ES_{C_{\max}}$	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) $\mathscr{P}^{10\times10}_{f.rndn}$					
$ES_{C_{\max}}$	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) $\mathscr{P}^{10\times10}_{f.jc}$					
$ES_{C_{\max}}$	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

fitness function directly and its normalisation w.r.t. true optimum (cf. Eqs. (6) and (7)), save for $\mathscr{P}_{f.rndn}$. 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 [11]. Thus normalising the objective function would help the evolutionary search to deviate from giving too much weight for problematic problem instances.

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 288 h without converging. Now, 6×5 JSP requires 30 sequential operations where at each time step there are up to 6 jobs to choose from—i.e., its complexity is $\mathscr{O}(n^{n \cdot m})$ 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 [9]. This could reduce the computational cost of the evolutionary search considerably, making it feasible to conduct the experiments from Sect. 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.

References

1. Ak, B., Koc, E.: A guide for genetic algorithm based on parallel machine scheduling and flexible job-shop scheduling. Procedia—Soc. Behav. Sci. **62**, 817–823 (2012)
2. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
3. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms-I. Representation. Comput. Ind. Eng. **30**(4), 983–997 (1996)
4. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Comput. Ind. Eng. **36**(2), 343–364 (1999)
5. Dhingra, A., Chandna, P.: A bi-criteria M-machine SDST flow shop scheduling using modified heuristic genetic algorithm. Int. J. Eng., Sci. Technol. **2**(5), 216–225 (2010)
6. Gurobi Optimization, Inc.: Gurobi optimization (version 5.6.2) [software]. http://www.gurobi.com/ (2013)
7. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
8. Haupt, R.: A survey of priority rule-based scheduling. OR Spectr. **11**, 3–16 (1989)
9. Ingimundardottir, H., Runarsson, T.P.: Sampling strategies in ordinal regression for surrogate assisted evolutionary optimization. In: 2011 11th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 1158–1163 (2011)
10. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: Coello, C. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6683, pp. 263–277. Springer, Heidelberg (2011)
11. Ingimundardottir, H., Runarsson, T.P.: Determining the characteristic of difficult job shop scheduling instances for a heuristic solution method. In: Hamadi, Y., Schoenauer, M. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, pp. 408–412. Springer, Heidelberg (2012)
12. Jayamohan, M., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. Eur. J. Oper. Res. **157**(2), 307–321 (2004)
13. Qing-dao-er ji, R., Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. Comput. Oper. Res. **39**(10), 2291–2299 (2012)
14. Koza, J.R., Poli, R.: Genetic programming. In: Burke, E., Kendal, G. (eds.) Introductory Tutorials in Optimization and Decision Support Techniques, chap. 5. Springer (2005)
15. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Learning iterative dispatching rules for job shop scheduling with genetic programming. Int. J. Adv. Manuf. Technol. (2013)
16. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Oper. Res. **25**(1), 45–61 (1977)
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer Publishing Company, Incorporated (2008)
18. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
19. Smith-Miles, K., James, R., Giffin, J., Tu, Y.: A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. In: Sttzle,

T. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 5851, pp. 89–103. Springer, Heidelberg (2009)
20. Smith-Miles, K., Lopes, L.: Generalising algorithm performance in instance space: a timetabling case study. In: Coello, C. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6683, pp. 524–538. Springer, Heidelberg (2011)
21. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput. Ind. Eng. **54**(3), 453–473 (2008)
22. Tsai, J.T., Liu, T.K., Ho, W.H., Chou, J.H.: An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. Int. J. Adv. Manuf. Technol. **38**(9–10), 987–994 (2007)
23. Vázquez-Rodríguez, J.A., Petrovic, S.: A new dispatching rule based genetic algorithm for the multi-objective job shop problem. J. Heuristics **16**(6), 771–793 (2009)
24. Watson, J.P., Barbulescu, L., Whitley, L.D., Howe, A.E.: Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. INFORMS J. Comput. **14**, 98–123 (2002)