Generating Training Data for Learning Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir(✉) and Thomas Philip Rúnarsson

School of Engineering and Natural Sciences,
University of Iceland, Reykjavik, Iceland
{hei2,tpr}@hi.is

Abstract. A supervised learning approach to generating composite linear priority dispatching rules for scheduling is studied. In particular we investigate a number of strategies for how to generate training data for learning a linear dispatching rule using preference learning. The results show, that when generating a training data set from only optimal solutions, it is not as effective as when suboptimal solutions are added to the set. Furthermore, different strategies for creating preference pairs is investigated as well as suboptimal solution trajectories. The different strategies are investigated on 2000 randomly generated problem instances using two different problem generator settings.

When applying learning algorithms, the training set is of paramount importance. A training set should have sufficient knowledge of the problem at hand. This is done by the use of features which are supposed to capture the essential measures of a problem's state. For this purpose, the job-shop scheduling problem (JSP) is used as a case study to illustrate a methodology for generating meaningful training data which can be successfully learned.

JSP deals with the allocation of tasks of competing resources where the goal is to minimise a schedule's maximum completion time, i.e., the makespan denoted $C_{\max}$. In order to find good solutions, heuristics are commonly applied in research, such as the simple priority based dispatching rules (SDR) from [11]. Composites of such simple rules can perform significantly better [6]. As a consequence, a linear composite of dispatching rules (LCDR) was presented in [3]. The goal there 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_j that corresponds to post-decision state $\mathbf{x}_j$, where $\phi(\mathbf{x}_j)$ denotes its feature mapping. The job dispatched is the following,

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

The approach was to use supervised learning to determine which feature states are preferable to others. The training data was created from optimal solutions of randomly generated problem instances.

C. Dhaenens et al. (Eds.): LION 9 2015, LNCS 8994, pp. 236–248, 2015.
DOI: 10.1007/978-3-319-19084-6_22

An alternative would be minimising the expected $C_{\max}$ by directly usin brute force search such as CMA-ES [2]. Preliminary experiments were conduc in [5], which showed that optimising the weights in Eq. (1) via evolution search actually resulted in a better LCDR than the previous approach. T nature of the CMA-ES is to explore suboptimal routes until it converges to optimal route. This implies that the previous approach, of restricting the train data only to *one* optimal route, may not produce a sufficiently rich training That is, the training set should incorporate a more complete knowledge of possible preferences, i.e., it should make the distinction between suboptimal a sub-suboptimal features, etc. This approach would require a Pareto ranking preferences which can be used to make the distinction of which feature sets equivalent, better or worse – and to what degree, e.g. by giving a weight to preference. This would result in a very large training set, which of course co be re-sampled in order to make it computationally feasible to learn. In t study we will investigate a number of different ranking strategies for creat preference pairs.

Alternatively, training data could be generated using suboptimal solut trajectories. For instance [7] used decision trees to 'rediscover' largest process time (LPT, a single priority based dispatching rule) by using LPT to cre its training data. The limitations of using heuristics to label the training d is that the learning algorithm will mimic the original heuristic (both when works poorly and well on the problem instances) and does not consider the r optimum. In order to learn heuristics that can outperform existing heurist then the training data needs to be correctly labelled. This drawback is confron in [8,10,15] by using an optimal scheduler, computed off-line. In this stu we will both follow optimal and suboptimal solution trajectories, but for e partial solution the preference pair will be labelled correctly by solving the par solution to optimality using a commercial software package [1]. For this stu most work remaining (MWR), a promising SDR for the given data distributi [4], and the CMA-ES optimised LCDRs from [5] will be deemed worthwhile generating suboptimal trajectories.

To summarise, the study considers two main aspects of the generation training data: (*a*) how preference pairs are added at each decision stage, and which solution trajectorie(s) should be sampled. That is, optimal, random, suboptimal trajectories, based on a good heuristic, etc.

The outline of the paper is as follows, first we illustrate how JSP can seen as a decision tree where the depth of the tree corresponds to the to number of job-dispatches needed to form a complete schedule. The feature sp is also introduced and how optimal dispatches and suboptimal dispatches labelled at each node in the tree. This is followed by detailing the strateg investigated in this study by selecting preference pairs ranking and sampl solution trajectories. The authors then perform an extensive study compar these strategies. Finally, this paper concludes with discussions and a summ of main results.

Table 1. Problem space distributions, $\mathcal{P}$.

Name	Size ($n \times m$)	N_{train}	N_{test}	Note
$\mathcal{P}_{j.rnd}$	6×5	500	500	Random
$\mathcal{P}_{j.rndn}$	6×5	500	500	Random-narrow

Table 2. Feature space, $\mathcal{F}$.

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

1 Problem Space

In this study synthetic JSP data instances are considered with the problem size $n \times m$, where n and m denotes number of jobs and machines, respectively. Problem instances are generated stochastically. By fixing the number of jobs and machines while processing time are i.i.d. samples 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 time distributions are explored, namely $\mathcal{P}_{j.rnd}$ where $I = [1, 99]$ and $\mathcal{P}_{j.rndn}$ where $I = [45, 55]$ are referred to as random and random-narrow, respectively. The machine order is a random permutation of all of the machines in the job-shop.

For each data distribution N_{train} and N_{test} problem 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 [16].

2 JSP Tree Representation

When building a complete JSP schedule $\ell = n \cdot m$ dispatches must be made consecutively. A job is placed at the earliest available time slot for its next

machine, whilst still fulfilling constraints that each machine can handle, wh is at most one job at each time, and jobs need to have finished their previ machines according to its machine order. Unfinished jobs, referred to as job-list denoted $\mathcal{L}$, are dispatched one at a time according to a heuristic. A each dispatch, the schedule's current features are updated based on its result partial schedule. For each possible post-decision state the temporal features, applied in this study are given in Table 2. These features are based on SD which are widespread in practice. For example if $\mathbf{w}$ is zero, save for $w_6 = 1$, t Eq. (1) gives $h(\mathbf{x}_j) > h(\mathbf{x}_i)$, $\forall i$ which are jobs with less work remaining th job J_j, namely Eq. (2) yields the job with the highest ϕ_6 value, i.e., equival to dispatching rule most work remaining (MWR).

Figure 1 illustrates how the first two dispatches could be executed for a 6 JSP with the machines $a \in \{M_1, ..., M_5\}$ on the vertical axis and the horizon axis yields the current makespan, $C_{\max}$. The next possible dispatches are deno as dashed boxes with the job index j within and its length corresponding processing time p_{ja}. In the top layer one can see an empty schedule. In the mid layer one of the possible dispatches from the layer above is fixed (depicted so and one can see the resulting schedule (i.e., what are the next possible dispatc given this new scenario?). Finally, the bottom layer depicts all outcomes if job on machine M_3 would be dispatched. This sort of tree representation is sim to *game trees* [9] where the root node denotes the initial (i.e., empty) sched and the leaf nodes denote the complete schedule. Therefore, the distance k fr an internal node to the root yields the number of operations already dispatch Traversing from root to leaf node, one can obtain a sequence of dispatches t yielded the resulting schedule, i.e., the sequence indicates in which order tasks should be dispatched for that particular schedule.

However, one can easily see that this sequence of task assignments is by means unique. Inspecting a partial schedule further along in the dispatch process such as in Fig. 1 (top layer), then let's say J_1 would be dispatched ne and in the next iteration J_2. This sequence would yield the same schedule a J_2 would have been dispatched first and then J_1 in the next iteration (since th are non-conflicting jobs). This indicates that some of the nodes in the tree merge despite states of the partial schedules being different in previous lay In this particular instance one can not infer that choosing J_1 is better and J worse (or vice versa) since they can both yield the same solution.

Furthermore, in some cases there can be multiple optimal solutions to same problem instance. Hence not only is the sequence representation 'flawed the sense that slight permutations on the sequence are in fact equivalent w. the end-result, but varying permutations on the dispatching sequence (given same partial initial sequence) can result in very different complete schedu with the same makespan, and thus same deviation from optimality, ρ defined Eq. (4), which is the measure under consideration. Care must be taken in t case that neither resulting features are labelled as undesirable or suboptim Only the resulting features from a dispatch resulting in a suboptimal solut should be labelled undesirable.

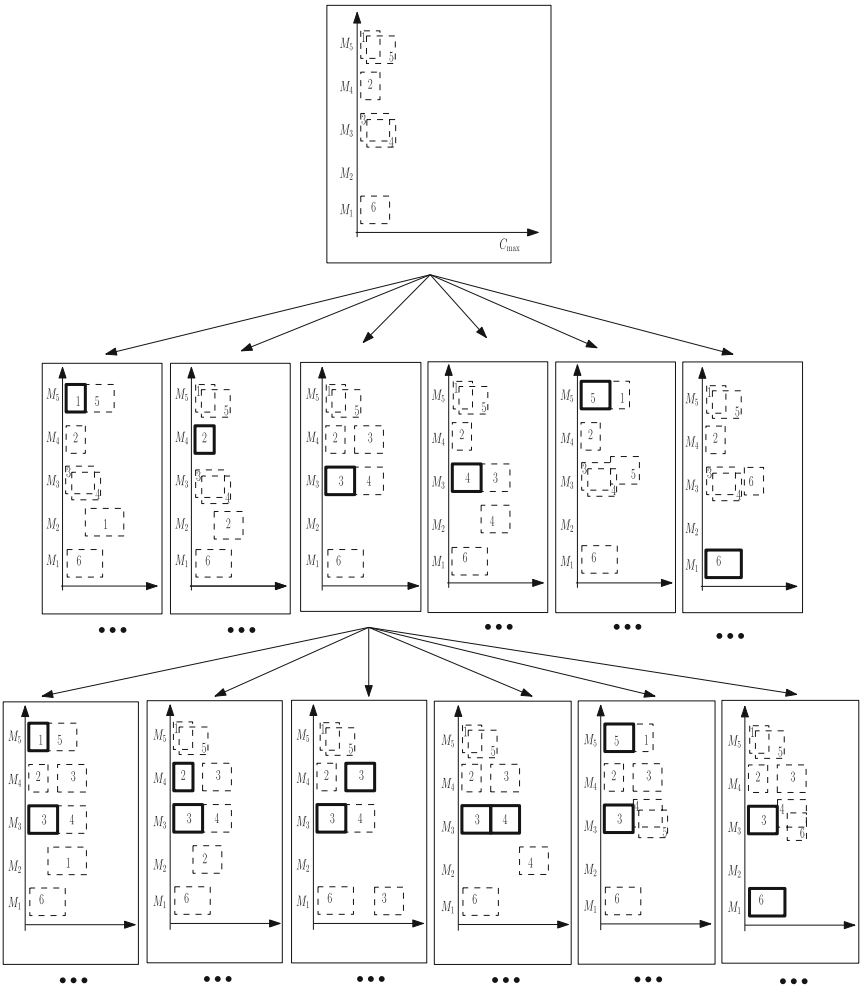


Fig. 1. Partial Tree for JSP for the first two dispatches. Executed dispatches are depicted solid, and all possible dispatches are dashed.

The creation of the tree for job-shop scheduling can be done recursively for all possible permutation of dispatches in the manner described above, resulting in a full n-ary tree of height $\ell = n \cdot m$. Such an exhaustive search would yield at the most n^{ℓ} leaf nodes (worst case scenario being that no sub-trees merge). Now, since the internal vertices (i.e., partial schedules) are only of interest to learn,[1] the number of those can be at the most $^{n^{\ell}-1}/_{n-1}$ [12]. Even for small dimensions of n and m the number of internal vertices are quite substantial and thus computationally expensive to investigate them all.

[1] The root is the empty initial schedule and for the last dispatch there is only one option left to dispatch, so there is no preferred 'choice' to learn.

The optimum makespan is known for each problem instance. At each ti step (i.e., layer of the tree) a number of feature pairs are created. The feat pairs consist of the features ϕ_o resulting from optimal dispatches $o \in \mathcal{O}^{(k)}$, ver features ϕ_s resulting from suboptimal dispatches $s \in \mathcal{S}^{(k)}$ at time k. Note, $\mathcal{O}^{(k}$ $\mathcal{S}^{(k)} = \mathcal{L}^{(k)}$ and $\mathcal{O}^{(k)} \cap \mathcal{S}^{(k)} = \emptyset$. In particular, each job is compared agai another job from the job-list, $\mathcal{L}^{(k)}$, and if the makespan differs, i.e., $C_{\max}^{(s)}$ $C_{\max}^{(o)}$, an optimal/suboptimal pair is created. However, if the makespan wo be unaltered the pair is omitted since they give the same optimal makesp This way, only features from a dispatch resulting in a suboptimal solutio labelled undesirable.

The approach taken in this study is to verify analytically, at each time st whether it can indeed *somehow* yield an optimal schedule by manipulating remainder of the sequence, while maintaining the current temporal schedule fi as its initial state. This also takes care of the scenario that having dispatche job resulting in a different temporal makespan would have resulted in the sa final makespan even if another optimal dispatching sequence would have b chosen. That is to say the data generation takes into consideration when th are multiple optimal solutions to the same problem instance.

3 Selecting Preference Pairs

At each dispatch iteration k, a number of preference pairs are created, whic then iterated over all N_{train} instances available. A separate data set is delib ately created for each dispatch iteration, as the initial feeling is that DRs use the beginning of the schedule building process may not necessarily be the sa as in the middle or end of the schedule. As a result there are ℓ linear sched ing rules for solving a $n \times m$ job-shop specified by a set of preference pairs each step,

$$S = \left\{ \{\phi_o - \phi_s, +1\}, \{\phi_s - \phi_o, -1\} \right\} \subset \Phi \times Y$$

for all $o \in \mathcal{O}^{(k)}, s \in \mathcal{S}^{(k)}, k \in \{1, \ldots, \ell\}$ where $Y = \{-1, 1\}$ denotes, suboptir or optimal preferences, respectively, and $\phi_o, \phi_s \in \Phi \subset \mathcal{F}$ are features from collected training set Φ. The reader is referred to [3] for a detailed descriptio how the linear ordinal regression model is trained on preference set S. Defin the size of the preference set as $l = |S|$, then if l is too large re-sampling n be needed to be done in order for the ordinal regression to be computationa feasible.

3.1 Trajectory Sampling Strategies

The following trajectory sampling strategies were explored for adding featu to the training set Φ,

Φ^{opt} at each dispatch some (random) optimal task is dispatched.

Φ^{cma} at each dispatch the task corresponding to highest priority, computed with fixed weights $\mathbf{w}$, which were obtained by directly optimising the mean of the performance measure defined in Eq. (4) with CMA-ES.
Φ^{mwr} at each dispatch the task corresponding to most work remaining is dispatched, i.e., following the simple dispatching rule MWR.
Φ^{rnd} at each dispatch some random task is dispatched.
Φ^{all} all aforementioned trajectories are explored, i.e.,

$$\Phi^{all} = \Phi^{opt} \cup \Phi^{cma} \cup \Phi^{mwr} \cup \Phi^{rnd}.$$

In the case of Φ^{mwr} and Φ^{cma} it is sufficient to explore each trajectory exactly once for each problem instance, since they are static DRs. Whereas, for Φ^{opt} and Φ^{rnd} there can be several trajectories worth exploring. However, only one is chosen at random, this is deemed sufficient as the number of problem instances N_{train} is relatively large.

3.2 Ranking Strategies

The following ranking strategies were implemented for adding preference pairs to S,

S_b all optimum rankings r_1 versus all possible suboptimum rankings r_i, $i \in \{2, \ldots, n'\}$, preference pairs are added, i.e., same basic set-up as in [3].
S_f full subsequent rankings, i.e., all possible combinations of r_i and r_{i+1} for $i \in \{1, \ldots, n'\}$, preference pairs are added.
S_p partial subsequent rankings, i.e., sufficient set of combinations of r_i and r_{i+1} for $i \in \{1, \ldots, n'\}$, are added to the preference set – e.g. in the cases that there are more than one operation with the same ranking, only one of that rank is needed to compared to the subsequent rank. Note that $S_p \subset S_f$.
S_a all rankings, i.e., all possible combinations of r_i and r_j for $i, j \in \{1, \ldots, n'\}$, $i \neq j$, preference pairs are added.

where $r_1 > r_2 > \ldots > r_{n'}$ ($n' \leq n$) are the rankings of the job-list, $\mathcal{L}^{(k)}$, at time step k.

4 Experimental Study

To test the validity of different rankings and strategies, the problem spaces outlined in Table 1 were used. The optimum makespan is denoted $C_{\max}^{\text{opt}}$, and the makespan obtained from the heuristic model is $C_{\max}^{\text{model}}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho = \frac{C_{\max}^{\text{model}} - C_{\max}^{opt}}{C_{\max}^{\text{opt}}} \cdot 100\,\% \tag{4}$$

which indicates the percentage relative deviation from optimality.

The preference set, S, across varying trajectories and ranking strategies depicted in Fig. 2, where the figure is divided vertically by problem space a horizontally by trajectory scheme.

A linear ordinal regression model (PREF) was created for each preference set, S, for problem spaces $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$. A box-plot with the results of p centage relative deviation from optimality, ρ, defined by Eq. (4), is presented Fig. 3. The box-plots are grouped w.r.t. trajectory strategies and colour-co w.r.t. ranking schemes. Moreover, the simple priority dispatching rule MW and the weights obtained by the CMA-ES optimisation used to obtain the tra ing sets Φ^{mwr} and Φ^{cma} respectively are shown in black in the far left of group for comparison. From Fig. 3 it is apparent there can be a performa edge gained by implementing a particular ranking or trajectory strategy. Mo over, the behaviour is analogous across different disciplines. Main statistics reported in Table 3a and b for $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$, respectively. Models are sor w.r.t. mean relative error.

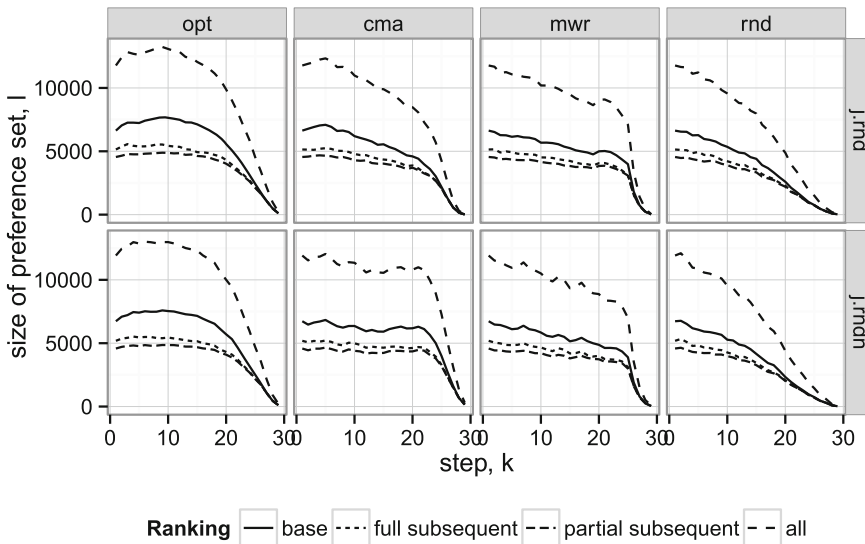


Fig. 2. Size of preference set, $l = |S|$, for different trajectories and ranking strate obtained from the training set for problem spaces $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$.

4.1 Ranking Strategies

There is no statistical difference between PREF_f and PREF_p ranking-mod across all trajectory disciplines (cf. Fig. 3), which is expected since S_p is desig to contain the same preference information as S_f. The results hold for b problem spaces.

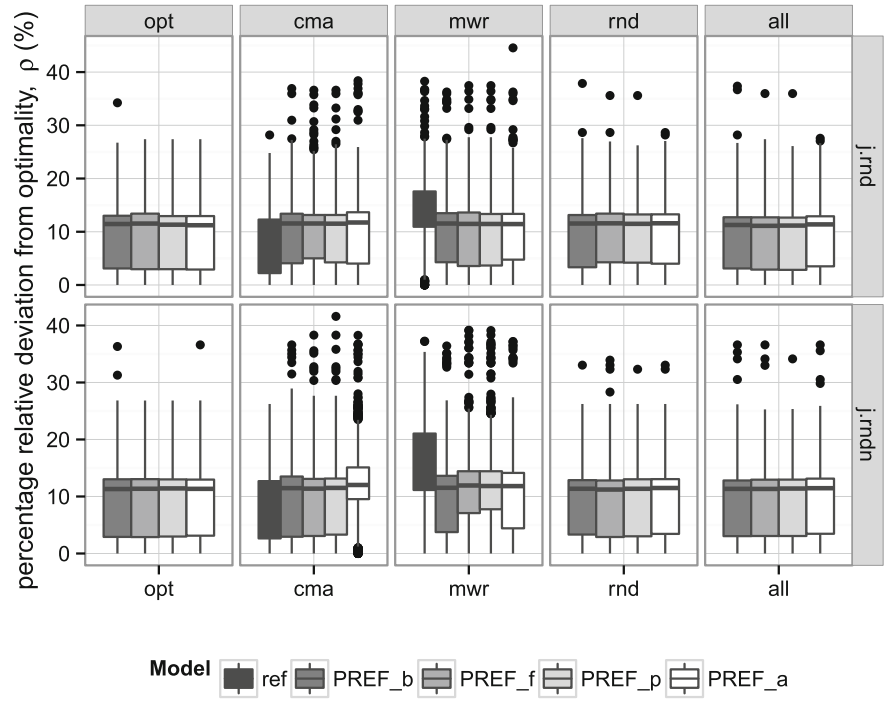


Fig. 3. Box-plot of results for linear ordinal regression model trained on various preference sets using test sets for problem spaces $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$.

Combining the ranking schemes, S_a, does not improve the individual ranking-schemes as there is no statistical difference between PREF_a and PREF_b, PREF_f nor PREF_p across all disciplines, save PREF_a^{cma} for $\mathcal{P}_{j.rndn}$ which yielded a considerably worse mean relative error.

Moreover, there is no statistical difference between either of the subsequent ranking-schemes outperforming the original S_b set-up from [3]. However overall, the subsequent ranking schemes results in lower mean relative error, and since a smaller preference set is preferred, it is opted to use the S_p ranking scheme.

Furthermore, it is noted that PREF^{mwr} is able to significantly outperform the original heuristic (MWR) used to create its training data Φ^{mwr}, irrespective of the ranking schemes. Whereas the fixed weights found via CMA-ES outperform the PREF^{cma} models for all ranking schemes. This implies that ranking scheme is relatively irrelevant. The results hold for both problem spaces.

4.2 Trajectory Sampling Strategies

Learning preference pairs from good scheduling policies, as done in PREF^{cma} and PREF^{mwr}, can give favourable results. However, tracking optimal paths yield generally a lower mean relative error.

Table 3. Main statistics of percentage relative deviation from optimality, ρ, defi by Eq. (4) for various models.

(a) $\mathcal{P}_{j.rnd}$ test set

model	track	rank	mean	med	sd	max
CMA			8.84	10.59	6.14	28.18
PREF	all	p	9.63	11.16	6.32	35.97
PREF	all	f	9.68	11.11	6.38	35.97
PREF	opt	a	9.92	11.22	6.49	27.39
PREF	all	b	9.98	11.27	6.61	37.36
PREF	opt	b	10.05	11.45	6.53	34.23
PREF	opt	p	10.13	11.33	6.74	27.39
PREF	all	a	10.15	11.38	6.30	27.57
PREF	opt	f	10.31	11.54	6.87	27.39
PREF	rnd	b	10.51	11.55	6.86	37.87
PREF	rnd	p	10.75	11.49	6.70	35.60
PREF	cma	p	10.78	11.52	6.89	36.60
PREF	rnd	a	10.82	11.59	6.73	28.65
PREF	cma	f	10.90	11.55	6.89	36.60
PREF	cma	b	10.90	11.55	7.10	36.91
PREF	mwr	p	10.95	11.46	7.26	37.47
PREF	mwr	f	11.07	11.48	7.35	37.47
PREF	rnd	f	11.09	11.58	6.92	35.60
PREF	mwr	a	11.09	11.44	7.21	44.55
PREF	mwr	b	11.30	11.54	7.63	36.26
PREF	cma	a	11.39	11.74	7.59	38.38
MWR			13.76	12.72	7.41	38.27

(b) $\mathcal{P}_{j.rndn}$ test set

model	track	rank	mean	med	sd	max
CMA			9.13	10.91	6.16	26.23
PREF	rnd	b	9.82	11.36	6.07	33.05
PREF	rnd	f	9.87	11.22	6.57	33.92
PREF	opt	b	9.94	11.31	6.52	36.32
PREF	opt	f	9.98	11.36	6.58	26.84
PREF	rnd	p	9.99	11.35	6.42	32.33
PREF	opt	a	10.01	11.34	6.31	36.60
PREF	all	f	10.05	11.33	6.53	36.60
PREF	opt	p	10.06	11.42	6.52	26.84
PREF	all	p	10.08	11.39	6.49	34.15
PREF	all	b	10.12	11.34	6.73	36.60
PREF	rnd	a	10.14	11.49	6.25	33.05
PREF	all	a	10.39	11.45	6.69	36.60
PREF	cma	f	10.56	11.38	7.28	38.31
PREF	cma	b	10.73	11.47	7.62	36.60
PREF	cma	p	10.74	11.51	7.43	41.60
PREF	mwr	b	11.33	11.52	7.72	36.41
PREF	mwr	a	11.70	11.82	7.88	37.20
PREF	mwr	f	12.07	11.93	8.07	39.17
PREF	mwr	p	12.14	11.84	8.32	39.12
PREF	cma	a	12.59	12.02	7.94	38.27
MWR			14.16	12.74	7.59	37.25

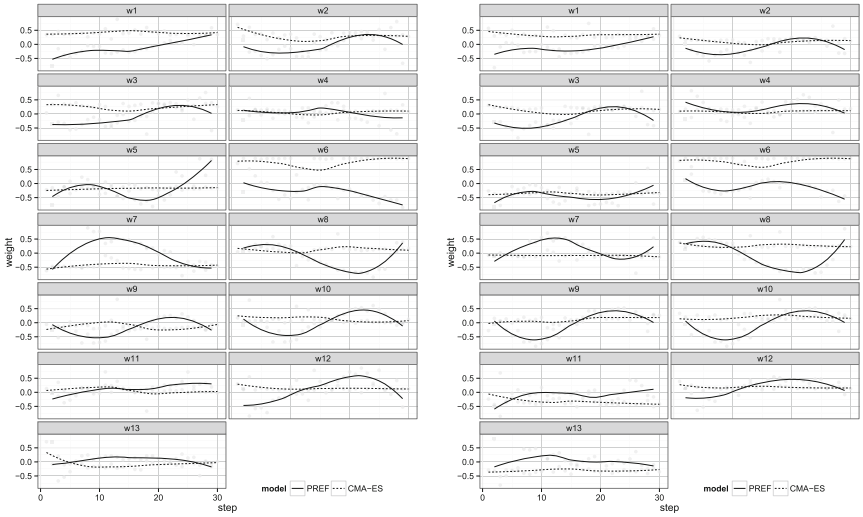
It is particularly interesting there is no statistical difference between PREF and PREF^{rnd} for both $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$ ranking-models. That is to say, track optimal dispatches gives the same performance as completely random dispatch This indicates that exploring only optimal trajectories can result in a train set where the learning algorithm is inept to determine good dispatches in circumstances when newly encountered features have diverged from the lear feature set labelled to optimum solutions.

Finally, PREF^{all} and PREF^{opt} gave the best combination for $\mathcal{P}_{j.rnd}$ $\mathcal{P}_{j.rndn}$. However, in the latter case PREF^{rnd} had the best mean relative e although not statistically different from PREF^{all} and PREF^{opt}.

For $\mathcal{P}_{j.rnd}$ the best mean relative error was for PREF^{all}. In that case add random suboptimal trajectories with the optimal trajectories gave the learn algorithm a greater variety of preference pairs for getting out of local mini Therefore, a general trajectory scheme would explore both optimal with sub timal paths.

4.3 Following CMA-ES Guided Trajectory

The rational for using the Φ^{cma} strategy was mostly due to the fact tha linear classifier created the training data (using the weights found via CM ES optimisation). Hence the training data created should be linearly separa which in turn should boost the training accuracy for a linear classification lea ing model. However, this is not the case since PREF^{cma} does not improve



(a) $\mathcal{P}_{j.rnd}$ (b) $\mathcal{P}_{j.rndn}$

Fig. 4. Linear weights (w_1 to w_{13} from left to right, top to bottom) found via CMA-ES optimisation (dashed), and weights found via learning classification PREF_p^{cma} model (solid).

original CMA-ES heuristic which was used to guide its training set Φ^{cma}. However, the PREF^{cma} approach is preferred to that of PREF^{mwr}, so there is some information gained by following the CMA-ES obtained weights instead of simple priority dispatching rules, such as MWR. Inspecting the CMA-ES guided training data more closely, in particular the linear weights for Eq. (1). The weights are depicted in Fig. 4 for problem spaces $\mathcal{P}_{j.rnd}$ (left) and $\mathcal{P}_{j.rndn}$ (right). The original weights found via CMA-ES optimisation that are used to guide the collection of training data are depicted dashed whereas weights obtained by the linear classification PREF_p^{cma} model are depicted solid.

From the CMA-ES experiments it is clear that a lot of weight is applied to decision variable w_6 which corresponds to implementing MWR, yet the existing weights for other features directs the evolutionary search to a "better" training data to learn than the PREF models. Arguably, the training data could be even better, however implementing CMA-ES is rather costly. In [5] the optimisation had not fully converged given its allocated 288 hrs of computation time.

It might also be an artefact because the sampling of the feature space during CMA-ES search is completely different to the data generation described in this study. Hence the different scaling parameters for the features might influence the results. Moreover, the CMA-ES is minimising the makespan directly, whereas the PREF models are learning to discriminate optimal versus suboptimal features sets that are believed to imply a better deviation from optimality later on. However, in that case, the process is very vulnerable when it comes to any divergence

from the optimal path. Ideally, it would be best to combine both methodolog
Collect training data from the CMA-ES optimisation which optimises w.r.t.
ultimate performance measure used, and in order to improve upon those weig
even further, use a preference based learning approach to deter from any lo
minima.

5 Summary and Conclusion

The study presents strategies for how to generate training data to be used
supervised learning of linear composite dispatching rules for job-shop scheduli
The experimental results provide evidence of the benefit of adding subopti
solutions to the training set apart from optimal ones. The subsequent ranki
are not of much value, since they are disregarded anyway, but the classificat
of optimal[2] and suboptimal features are of paramount importance. However,
trajectories to create training instances have to be varied to boost performan
This is due to the fact that sampling only states that correspond to optimal
close-to optimal schedules isn't of much use when the model has diverged
far. Since we are dealing with sequential decision making, all future observati
are dependent on previous operations. Therefore, to account for this drawba
an imitation learning approach by [13,14] could fruitful. In that case, we co
continue with our PREF^{opt} model and collect a new training set by following
learned policy and use that to create a new model similar to the Φ^{all} scheme
short, using the model to update itself. This can be done several times until
weights converge. The benefit of this approach is that the states that are lik
to occur in practice are investigated and as such used to dissuade the mo
from making poor choices. Alas, due to the computational cost[3] of collecting
training set Φ, this sort of methodology isn't suitable for high dimensionality
job-shops.

Unlike [8,10,15] learning only optimal training data was not fruitful. He
ever, inspired by the original work of [7], having heuristics guide the generat
of training data (while using optimal labelling based on a solver) gave me
ingful preference pairs which the learning algorithm could learn. In conclusi
henceforth, the training data will be generated with PREF$_p^{all}$ scheme for
authors' future work. Based on these preliminary experiments, we continue
test on a greater variety of problem data distributions for scheduling, namely j
shop and permutation flow-shop problems. Once training data has been carefu
created, global dispatching rules can finally be learned with the hope of imp
menting them for a greater number of jobs and machines. This is the focus
our current work.

[2] Here the tasks labelled 'optimal' do not necessarily yield the optimum makes
(except in the case of following optimal trajectories), instead these are the opti
dispatches for the given partial schedule.

[3] Note, each partial schedule corresponding to a feature in Φ is optimised to obt
its correct labelling.

References

1. Gurobi Optimization Inc: Gurobi optimization (version 5.6.2) [software] (2013). http://www.gurobi.com/
2. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
3. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 263–277. Springer, Heidelberg (2011)
4. 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.) LION 2012. LNCS, vol. 7219, pp. 408–412. Springer, Heidelberg (2012)
5. Ingimundardottir, H., Runarsson, T.P.: Evolutionary learning of weighted linear composite dispatching rules for scheduling. In: International Conference on Evolutionary Computation Theory and Applications (ECTA) (2014)
6. 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)
7. Li, X., Olafsson, S.: Discovering dispatching rules using data mining. J. Sched. **8**, 515–527 (2005)
8. Malik, A.M., Russell, T., Chase, M., Beek, P.: Learning heuristics for basic block instruction scheduling. J. Heuristics **14**(6), 549–569 (2008)
9. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior (Commemorative Edition). Princeton University Press, Princeton Classic Editions, Princeton (2007)
10. Olafsson, S., Li, X.: Learning effective new single machine dispatching rules from optimal scheduling data. Int. J. Prod. Econ. **128**(1), 118–126 (2010)
11. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Oper. Res. **25**(1), 45–61 (1977)
12. Rosen, K.H.: Discrete Mathematics and its Applications, Chap. 9, 5th edn. McGraw-Hill Inc, New York (2003)
13. Ross, S., Bagnell, D.: Efficient reductions for imitation learning. In: Teh, Y.W., Titterington, D.M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010), vol. 9, pp. 661–668 (2010). http://www.jmlr.org/proceedings/papers/v9/ross10a/ross10a.pdf
14. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Gordon, G.J., Dunson, D.B. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTAT 2011), vol. 15, pp. 627–635, Journal of Machine Learning Research - Workshop and Conference Proceedings (2011). http://www.jmlr.org/proceedings/papers/v15/ross11a/ross11a.pdf
15. Russell, T., Malik, A.M., Chase, M., van Beek, P.: Learning heuristics for the superblock instruction scheduling problem. IEEE Trans. Knowl. Data Eng. **21**(10), 1489–1502 (2009)
16. 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)