Journal of Heuristics manuscript No.
(will be inserted by the editor)

Learning Linear Composite Dispatch Rules for Scheduling

Case study for the job- and flow-shop problem

Helga Ingimundardottir · **Thomas Philip Runarsson**

Received: January 19, 2015/ Accepted: date

Abstract Instead of creating new dispatching rules in an ad hoc manner, this study gives a framework on how to study simple heuristics for scheduling problems. Before starting to create new composite dispatching rules, meticulous research on optimal schedules can give an abundance of valuable information that can be utilised for learning new models. For instance, it's possible to seek out when the scheduling process is most susceptible to failure. Furthermore, the stepwise optimality of individual features imply their explanatory predictability. From which, a preference set is collected and a preference based learning model is created based on what feature states are preferable to others w.r.t. the end result, here minimising the final makespan. By doing so it's possible to learn new composite dispatching rules that outperform the models they are based on. Even though this study is based around the job-shop scheduling problem, it can be generalised to any kind of combinatorial problem.

Keywords Scheduling · Composite dispatching rules · Machine Learning · Feature Selection

H. Ingimundardottir
Dunhaga 5, IS-107 Reykjavik, Iceland
Tel.: +354-525-4704
Fax: +354-525-4632
E-mail: hei2@hi.is

T.P. Runarsson
Hjardarhagi 2-6, IS-107 Reykjavik, Iceland
Tel.: +354-525-4733
Fax: +354-525-4632
E-mail: tpr@hi.is

1 Introduction

Lure the reader in a with a good first sentence (which this is not!)

What is the problem?

A subclass of scheduling problems is the job-shop scheduling problem (JSP), which is widely studied in operations research. Job-shop deals with the allocation of tasks of competing resources where its goal is to optimise a single or multiple objectives. Its analogy is from manufacturing industry where a set of jobs are broken down into tasks that must be processed on several machines in a workshop.

Why is it interesting?

Furthermore, its formulation can be applied on a wide variety of practical problems in real-life applications which involve decision making, therefore its problem-solving capabilities has a high impact on many manufacturing organisations.

JSP is NP-hard [4], hence finding optimal solutions of high dimensionality is exceedingly difficult in a reasonable amount of time. As a result heuristics methods are adopted. Generally, this is done by applying a hand-crafted dispatching rule (DR) for a given problem space. Due to the exorbitant amounts of DRs to choose from, and any kind of alteration to the problem space, this can be come quite a time-consuming selection process for the heuristic designer, which any kind of automation would alleviate immensely.

What are your contributions?

For this reason, we propose a framework for learning the indicators of optimal solutions, such as done by [16]. The study shows that during the scheduling process it varies *when* it's most fruitful to make the 'right' decision, and depending on the problem space those pivotal moments can vary greatly.

What is the outline of what you will show?

Although, using optimal trajectory for creating training data gives vital information on how to learn good scheduling rules, it is a good starting point, but not sufficient. This is due to the fact our models are only based on optimal decisions, then once we make a suboptimal choice we are in uncharted territory and its effects are relatively unknown. For this reason, it is of paramount importance to inspect the actual end-performance when choosing a suitable model, not just staring blindly at the validation accuracy. Moreover, different measures on how to report training accuracy is discussed.

The outline of the paper is the following, Section 2 gives the mathematical formalities of the scheduling problem, and Section 3 goes into how their schedules are constructed, followed by Section 4 giving a background on what has been done previously in learning new dispatching rules in similar fields. Section 6 sets up the framework for learning from optimal schedules. In particular, the probability of choosing optimal decisions and the effects of making a suboptimal decision. Furthermore, the optimality of common dispatching rules is investigated, from which a blended dispatching rule is created. With those guidelines, Section 7 goes into detail how to create meaningful composite dispatching rules, with the importance of good feature selection and the polysemy of how to report accuracy. The paper finally concludes in Section 8 with discussion and conclusions.

2 Job and Permutation Flow-Shop Scheduling

The job-shop problem (JSP) involves the scheduling of jobs on a set of machines. Each job consists of a number of operations which are then processed on the machines in a predetermined order. An optimal solution to the problem will depend on the specific objective.

In this study we will consider the $n \times m$ JSP, where n jobs, $\mathscr{J} = \{J_j\}_{j=1}^{n}$, are scheduled on a finite set, $\mathscr{M} = \{M_a\}_{a=1}^{m}$, of m machines. The jobs are subject to the constraint that each job J_j must follow a predefined machine order, a chain or sequence of m operations $\boldsymbol{\sigma}_j = \{\sigma_{j1}, \sigma_{j2}, \ldots, \sigma_{jm}\}$. Furthermore, a machine can handle at most one job at a time. Additional constraints commonly considered are job release-dates and due-dates, however, those will not be considered here. The objective will be to schedule the jobs so as to minimize the maximum completion times for all tasks, also known as the makespan, $C_{\max}$. A common notion for this family of scheduling problems is $J||C_{\max}$ [18]. In the case when all jobs share the same permutation route $\boldsymbol{\sigma}_j$, the JSP is reduced to a permutation flow-shop scheduling problem (FSP) [6,24], denoted $F||C_{\max}$. Therefore, without the loss of generality, this study will be structured around the JSP.

Henceforth the index j refers to a job $J_j \in \mathscr{J}$ while the index a refers to a machine $M_a \in \mathscr{M}$. If a job requires a number of processing steps or operations, then the pair (j,a) refers to the operation, i.e., processing the task of job J_j on machine M_a. Note that once an operation is started, it must be completed uninterrupted, i.e., pre-emption is not allowed. Moreover, there are no sequence dependent set-up times.

3 Scheduling Heuristics

Heuristics algorithms for scheduling are typically either a construction or improvement heuristics. The improvement heuristic starts with a complete schedule and then tries to find similar, but better schedules. A construction heuristic starts with an empty schedule and adds one job at a time until the schedule is complete. The work presented here will focus on construction heuristics, although the techniques developed could be adapted to improvement heuristics also. In scheduling a construction heuristic is typically implemented as a priority dispatching rule. These are simple rules that basically determine which incomplete job should be dispatched next. However, knowing which job to dispatch is not sufficient, one must also know where to place it. In order to build tight schedules it is sensible to place a job, once it becomes available, such that the machine idle time is minimal. There may also be a number of different options for such a placement. Figure 1 illustrates the dispatching process with an example of a temporal partial schedule of six-jobs scheduled on five-machines. The numbers in the boxes represent the job identification j. The width of the box illustrates the processing times for a given job for a particular machine M_a (on the vertical axis). The dashed boxes represent the resulting partial schedule for when a particular job is scheduled next. Moreover, the current $C_{\max}$ is denoted by a dotted vertical line. In the figure we observe that J_2, to be scheduled on M_3, could be placed immediately in a slot between J_3 and J_4, or after J_4 on this machine. If J_6 had been placed earlier, a slot would have been created between it and J_4 thus creating a third alternative, namely scheduling J_2 after J_6. The construction heuristic must therefore decide where to place the job, and this may be independent of the dispatching rule applied. Different placement strategies could be considered, for example placing a job

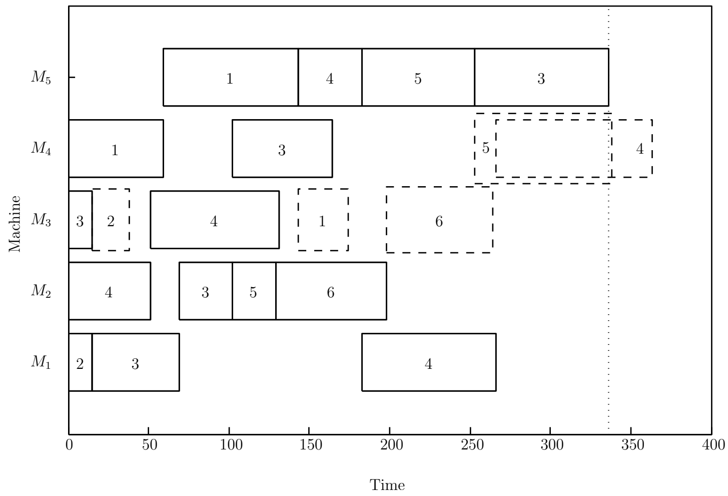

Fig. 1: Gantt chart of a partial JSP schedule after 15 dispatches: Solid and dashed boxes represent $\boldsymbol{\chi}$ and $\mathscr{L}^{(16)}$, respectively. Current $C_{\max}$ denoted as dotted line.

in smallest feasible slot. In our preliminary experiments we have discovered that such a placement can potentially rule out the possibility of constructing solutions with an optimal makespan. This problem, however, did not occur when jobs are simply placed as early as possible. For this reason it will be our placement strategy.

A *sequence* will refer to the sequential ordering of dispatches of tasks to machines, i.e., (j,a); the collective set of allocated tasks to machines is interpreted by its sequence, is referred to as a *schedule*; a *scheduling policy* will pertain to the manner in which the sequence is determined. As shown in our example given in Figure 1, there are 15 operations already scheduled. The sequence used to create the schedule was,

$$\boldsymbol{\chi} = (J_3,J_3,J_3,J_3,J_4,J_4,J_5,J_1,J_1,J_2,J_4,J_6,J_4,J_5,J_3) \tag{1}$$

and the available jobs to be scheduled $\mathscr{L}^{(k)} = \{J_1,J_2,J_4,J_5,J_6\}$ describes the five potential jobs to be dispatched at step $k = 16$ (note that J_3 is completed). An overview on dispatching rules, used to create such sequences, is given in the following section.

3.1 Priority Dispatching Rules

A priority dispatching rule inspects the job-list, $\mathscr{L}$, and dispatches the job with the highest priority. These rules typically use attributes for the corresponding operation, for example the processing time for the job. Consider again Figure 1, if the job with the shortest processing time (SPT) were to be scheduled next then J_2 would be dispatched. Similarly, for the longest processing time (LPT) heuristic J_5 would be dispatched. Dispatching can also be based on attributes related to the partial schedule. Examples of these are dispatching the job with the most work remaining (MWR) or

Table 1: Attribute space $\mathscr{A}$ for JSP where job J_j on machine M_a given the resulting temporal schedule after dispatching (j,a).

$\boldsymbol{\phi}$	Feature description	Mathematical formulation	Shorthand
	job related		
ϕ_1	job processing time	p_{ja}	proc
ϕ_2	job start-time	$x_s(j,a)$	startTime
ϕ_3	job end-time	$x_f(j,a)$	endTime
ϕ_4	job arrival time	$x_f(j,a-1)$	arrival
ϕ_5	total processing time	$\sum_{a\in\mathscr{M}} p_{ja}$	totalProc
ϕ_6	time job had to wait	$x_s(j,a)-x_f(j,a-1)$	wait
ϕ_7	total work remaining for job	$\sum_{a'\in\mathscr{M}\setminus\mathscr{M}_j} p_{ja'}$	wrmJob
ϕ_8	number of assigned operations for job	$\lvert\mathscr{M}_j\rvert$	jobOps
	machine related		
ϕ_9	machine ID	a	mac
ϕ_{10}	when machine is next free	$\max_{j'\in\mathscr{J}_a}\{x_f(j',a)\}$	macFree
ϕ_{11}	total work remaining for machine	$\sum_{j'\in\mathscr{J}\setminus\mathscr{J}_a} p_{j'a}$	wrmMac
ϕ_{12}	number of assigned operations for machine	$\lvert\mathscr{J}_a\rvert$	macOps
	flow related		
ϕ_{13}	change in idle time by assignment	$\Delta s(a,j)$	slotsReduced
ϕ_{14}	total idle time for machine	$\sum_{j'\in\mathscr{J}_a} s(a,j')$	slots
ϕ_{15}	total idle time for all machines	$\sum_{a'\in\mathscr{M}}\sum_{j'\in\mathscr{J}_{a'}} s(a',j')$	slotsTotal
	current makespan related		
ϕ_{16}	current makespan	$\max_{(j',a')\in\mathscr{J}\times\mathscr{M}_{j'}}\{x_f(j',a')\}$	makespan
ϕ_{17}	total work remaining for all jobs/mac	$\sum_{j'\in\mathscr{J}}\sum_{a'\in\mathscr{M}\setminus\mathscr{M}_{j'}} p_{j'a'}$	wrmTotal
ϕ_{18}	current step in the dispatching process	$\lvert\boldsymbol{\chi}\rvert$	step

alternatively the least work remaining (LWR). A survey of more than 100 of such rules are presented in [17], however the reader is referred to an in-depth survey for single-priority or *simple dispatching rules* (SDR) by [8]. SDRs assign an index to each job in the job-list and is generally only based on few attributes and simple mathematical operations.

Designing priority dispatching rules requires recognizing the important attributes of the partial schedules needed to create a good scheduling rule. These attributes attempt to grasp key features of the schedule being constructed. Which attributes are most important will necessarily depend on the objectives of the scheduling problem. Attributes used in this study applied for a job J_j to be dispatched on machine M_a are given in Table 1. The attributes of particular interest were obtained by inspecting the aforementioned SDRs. Attributes ϕ_1-ϕ_8 and ϕ_9-ϕ_{12} are job-related and machine-related, respectively. Then there are flow-related attributes, ϕ_{13}-ϕ_{15} which measure the influence of idle time on the schedule, and current makespan related, ϕ_{16}-ϕ_{18}. All of these attributes vary throughout the scheduling process, w.r.t. operation belonging to the same time step k, with the exception of ϕ_9, which is reported in order to distinguish which features are in conflict with each other;

@Helga: not sure how this works again... check code, can you recheck if this is all OK here? I mean do we need to talk about things we don't use in our experimental study!
@TPR: Features are fine in C# code. Only discrepancy is with flow-related variable w.r.t. MATLAB code. We use all features from Table 1 except for ϕ_{18} and ϕ_{17}, i.e., total number of features are $d = 16$

ϕ_{18} to keep track of features' evolution w.r.t. the scheduling process; and ϕ_5 and ϕ_{17} which are static for a given problem instance, but used for normalising other features, e.g., ϕ_{17} for work-remaining based ones (ϕ_7 and ϕ_{11}).

Dispatching rules are attractive since they are relatively easy to implement, fast and find good schedules. However, they can also fail unpredictably. Combining different SDRs can potentially enhance the scheduling performance.

3.2 Composite Priority Dispatching Rules

A careful combination of dispatching rules can perform significantly better [10]. These are referred to as *composite dispatching rules* (CDR), where the priority ranking is an expression of several single-based priority dispatching rules. CDRs can deal with greater number of features and more complicated form, in short, CDR are a combination of several SDRs. For instance let CDR be comprised of d dispatching rules (DR), then the index I for job J_j using CDR is,

$$I_j^{CDR} = \sum_{i=1}^{d} w_i \cdot \mathrm{DR}_i(\boldsymbol{\phi}_j) \tag{2}$$

where $w_i > 0$ and $\sum_{i=0}^{d} w_i = 1$ and w_i gives the weight of the influence of DR_i (which could be SDR or another CDR) to CDR. Note, each DR_i is function of the job J_j's attributes $\boldsymbol{\phi}_j$.

@Helga: can you please make it clear what the difference is between a blended and composite rule, for me its seems to be the same thing... are we confusing things here?!
@TPR: You're right, I've commented out the blended dispatching rules. It's hardly ever used in the literature, CDR are much more prevalent name.

At each time step k, an operation is dispatched which has the highest priority in the job-list, $\mathcal{L}^{(k)} \subset \mathcal{J}$. If there is a tie, some other priority measure is used. Generally the priority dispatching rules are static during the entire scheduling process.

@Helga: reword this paragraph so you don't start the sentence with a citation
@TPR: I copy/pasted from a document with the NATBIB package, where the `citet` prints out the author's name, making the sentence human readable.

Investigating 11 SDRs for JSP, [13] created a pool of 33 composite dispatching rules that strongly outperformed the ones they were based on. The CDRs were created with multi-contextual functions (MCFs) based on either on machine idle time or job waiting time, so one can say that the CDRs are a combination of those two key features of the schedule and then the SDRs. However, there are no combinations of the basic SDRs explored, only machine idle time and job waiting time. Similarly, using priority rules to combine 12 existing DRs from the literature, [27] had 48 priority rules combinations, yielding 48 different models to implement and test. It is intuitive to get a boost in performance by introducing new CDRs, since where one DR might be failing, another could be excelling so combining them together should yield a better CDR. However, these approaches introduce fairly ad hoc solutions and there is no guarantee the optimal combination of dispatching rules are found. Moreover, generally the weights **w** are chosen by the designer or the rule apriori. A more attractive approach would be to learn these weights from problem examples directly. We will now investigate how this may be accomplished.

4 Learning Dispatching Rules

A recent editorial of the state-of-the-art approaches in advanced dispatching rules for large-scale manufacturing systems by [1] points out that: "... most traditional dispatching rules are based on historical data. With the emergence of data mining and on-line analytic processing, dispatching rules can now take predictive information into account". The importance automated discovery of DR was also emphasised by [14]. Several of successful implementations in the field of semiconductor wafer fabrication facilities are discussed, however, this sort of investigation is still in its infancy.

@Helga: the remainder of this chapter should be about how learning has been used to find composite dispatching rules, from the literature, here you can cite you own work and the work of Olafson and those citing him. This section should conclude with a paragraph on instance generation and training data creation to connect to the next chapter. I leave this here below in case you would like to use something from it ...

With meta heuristics one can use existing DRs and use for example portfolio-based algorithm selection [19,5], either based on a single instance or class of instances [26] to determine which DR to choose from.

[11] point out that meta learning can be very fruitful in reinforcement learning, and in their experiments they discovered some key discriminants between competing algorithms for their particular problem instances, which provided them with a hybrid algorithm which combines the strengths of the algorithms.

[15] proposed a novel iterative dispatching rules (IDRs) for JSP which learns from completed schedules in order to iteratively improve new ones. At each dispatching step, the method can utilise the current feature space to *correctify* some possible *bad* dispatch made previously (sort of reverse lookahead). Their method is straightforward, and thus easy to implement and more importantly computationally inexpensive, although the authors do stress that there is still remains room for improvement.

[12] implement ant colony optimisation to select the best DR from a selection of nine DRs for JSP and their experiments showed that the choice of DR do affect the results and that for all performance measures considered it was better to have a all the DRs to choose from rather than just a single DR at a time.

5 Learning from Problem Instances

@Helga: what is this chapter is about?

This chapter need a rewrite, I will let you take the first iteration.

5.1 Problem Instances

@Helga: put here all material releated to Table 2.

For each problem class described in Table 2 there are N problem instances generated with a random problem generator using n jobs and m machines. The goal is to minimize the makespan, $C_{\max}$. The optimum makespan is denoted $C_{\max}^{\text{opt}}$, and the makespan obtained from the scheduling policy A under inspection by $C_{\max}^{A}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho = \frac{C_{\max}^{A} - C_{\max}^{\text{opt}}}{C_{\max}^{\text{opt}}} \cdot 100\% \tag{3}$$

which indicates the percentage relative deviation from optimality.

5.2 Schedule building

When building a complete 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 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[1] the schedule's current features (cf. Table 1) are updated based on the half-finished schedule.

It is easy to see that the sequence of task assignments is by no means unique. Inspecting a partial schedule further along in the dispatching process such as in Fig. 1, then let's say J_1 would be dispatched next, and in the next iteration J_2. Now this sequence would yield the same schedule as if J_2 would have been dispatched first and then J_1 in the next iteration, i.e., these are non-conflicting jobs. In this particular instance one can not infer that choosing J_1 is better and J_2 is worse (or vice versa) since they can both yield the same solution.

[1] Dispatch and time step are used interchangeably.

Note that in some cases there can be multiple optimal solutions to the same problem instance. Hence not only is the sequence representation ‘flawed’ in the sense that slight permutations on the sequence are in fact equivalent w.r.t. the end-result, but very varying permutations on the dispatching sequence (however given the same partial initial sequence) can result in very different complete schedules but can still achieve the same makespan, and thus same deviation from optimality, ρ, defined by (3), which is the measure under consideration. Care must be taken in this case that neither resulting features are labelled as undesirable. Only the resulting features from a dispatch resulting in a suboptimal solution should be labelled undesirable.

5.3 Labelling schedules w.r.t. optimal decisions

The optimum makespan is known for each problem instance. At each time step a number of feature pair are created, they consist of the features $\boldsymbol{\phi}_o$ resulting from optimal dispatches $o \in \mathscr{O}^{(k)}$, versus features $\boldsymbol{\phi}_s$ resulting from suboptimal dispatches $s \in \mathscr{S}^{(k)}$ at time k. Note, $\mathscr{O}^{(k)} \cup \mathscr{S}^{(k)} = \mathscr{L}^{(k)}$ and $\mathscr{O}^{(k)} \cap \mathscr{S}^{(k)} = \emptyset$. In particular, each job is compared against another job of the job-list, $\mathscr{L}^{(k)}$, and if the makespan differs, i.e., $C_{\max}^{(s)} \gneq C_{\max}^{(o)}$, an optimal/suboptimal pair is created, however if the makespan would be unaltered the pair is omitted since they give the same optimal makespan. This way, only features from a dispatch resulting in a suboptimal solution is labelled undesirable.

The approach taken here is to verify analytically, at each time step, by fixing the current temporal schedule as an initial state, whether it can indeed *somehow* yield an optimal schedule by manipulating the remainder of the sequence. This also takes care of the scenario that having dispatched a job resulting in a different temporal makespan would have resulted in the same final makespan if another optimal dispatching sequence would have been chosen. That is to say the data generation takes into consideration when there are multiple optimal solutions to the same problem instance.

5.4 Creating time-independent dispatching rules

Preliminary experiments for creating step-by-step model was done in [9] where an optimal trajectory was explored, i.e., at each dispatch some (random) optimal task is dispatched, resulting in local linear model for each dispatch; a total of ℓ linear models for solving $n \times m$ JSP. However, the experiments there showed that by fixing the weights to its mean value throughout the dispatching sequence, results remained satisfactory. A more sophisticated way, would be to create a *new* linear model, where the preference set, S, is the union of the preference pairs across the ℓ dispatches. This would amount to a substantial preference set, and for S to be computationally feasible to learn, S has to be reduced. For this several ranking strategies were explored in [**?**], the results there showed that it’s sufficient to use partial subsequent rankings, namely, combinations of r_i and r_{i+1} for $i \in \{1,\ldots,n'\}$, are added to the preference set, where $r_1 > r_2 > \ldots > r_{n'}$ ($n' \leq n$) are the rankings of the job-list, $\mathscr{L}^{(k)}$, at time step k, in such a manner that in the cases that there are more finruthan one operation with the same ranking, only one of that rank is needed to be compared to the subsequent rank. Moreover, in the case of this study, which deals with 10×10 problem instances, the partial subsequent ranking becomes necessary, as full ranking is computationally infeasible. This is due to the since the size of the preference set, $|S|$, becomes too large with full ranking, and would need sampling. For the following experimental set up, the preference set was limited to $|S| \leq 200,000$ by random sampling.

5.5 Linear Learning

@Helga: this is a condensed version of liblinear for our problem, please do not describe logistic regression just how the data is preprocessed and fed into liblinear.
@TPR: Duly noted, I've edited that out and simply cited the liblinear package. I also took out the scaling paragraph, as liblinear does that automatically.

Learning models considered in this study are based on ordinal regression in which the learning task is formulated as learning preferences. In the case of scheduling, learning which operations are preferred to others. Ordinal regression has been previously presented in [22] and in [9] for JSP, however given here for completeness.

Let $\boldsymbol{\phi}_o \in \mathbb{R}^d$ denote the post-decision state when dispatching J_o corresponds to an optimal schedule being built. All post-decisions states corresponding to suboptimal dispatches, J_s, are denoted by $\boldsymbol{\phi}_s \in \mathbb{R}^d$. One could label which feature sets were considered optimal, $\mathbf{z}_o = \boldsymbol{\phi}_o - \boldsymbol{\phi}_s$, and suboptimal, $\mathbf{z}_s = \boldsymbol{\phi}_s - \boldsymbol{\phi}_o$ by $y_o = +1$ and $y_s = -1$ respectively. Note, a negative example is only created as long as J_s actually results in a worse makespan, i.e., $C_{\max}^{(s)} \gneq C_{\max}^{(o)}$, since there can exist situations in which more than one operation can be considered optimal.

The preference learning problem is specified by a set of preference pairs,

$$S = \left\{ \{\mathbf{z}_o, +1\}_{k=1}^{\ell}, \{\mathbf{z}_s, -1\}_{k=1}^{\ell} \mid \forall o \in \mathscr{O}^{(k)}, s \in \mathscr{S}^{(k)} \right\} \subset \boldsymbol{\Phi} \times Y \tag{4}$$

where $\boldsymbol{\Phi} \subset \mathbb{R}^d$ is the training set of d features, $Y = \{-1, +1\}$ is the outcome space, $\ell = n \times m$ is the total number dispatches, from which $o \in \mathscr{O}^{(k)}$ and $s \in \mathscr{S}^{(k)}$ denote optimal and suboptimal dispatches, respectively, at step k. Note, $\mathscr{O}^{(k)} \cup \mathscr{S}^{(k)} = \mathscr{L}^{(k)}$, and $\mathscr{O}^{(k)} \cap \mathscr{S}^{(k)} = \emptyset$.

For JSP there are $d = 18$ features (cf. Table 1 and explained in more detail in Section 3.2), and the training set is created in the manner described in Section 5.

Now consider the model space $\mathscr{H} = \{h(\cdot) : X \mapsto Y\}$ of mappings from solutions to ranks. Each such function h induces an ordering $\succ$ on the solutions by the following rule,

$$\mathbf{x}_i \succ \mathbf{x}_j \quad \Leftrightarrow \quad h(\mathbf{x}_i) > h(\mathbf{x}_j) \tag{5}$$

where the symbol $\succ$ denotes "is preferred to." The function used to induce the preference is defined by a linear function in the feature space,

$$h(\mathbf{x}) = \sum_{i=1}^{d} w_i \phi_i(\mathbf{x}) = \langle \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}) \rangle. \tag{6}$$

Logistic regression learns the optimal parameters $\mathbf{w}^* \in \mathbb{R}^d$. For this study, L2-regularized logistic regression from the LIBLINEAR package [3] without bias is used to learn the preference set S, defined by (4). Hence, for each job on the job-list, $J_j \in \mathscr{L}$, let $\boldsymbol{\phi}_j$ denote its corresponding post-decision state. Then the job chosen to be dispatched, J_{j^*}, is the one corresponding to the highest preference estimate, i.e.,

$$J_{j^*} = \underset{J_j \in \mathscr{L}}{\operatorname{argmax}} \; h(\boldsymbol{\phi}_j) \tag{7}$$

where $h(\cdot)$ is the classification model obtained by the preference set.

5.6 Interpreting linear classification models

Looking at the features description in Table 1 it is possible for the ordinal regression to 'discover' the weights $\mathbf{w}$ in order for (6) corresponds to applying a single priority dispatching rules from Section 3.1. For instance,

$$\begin{aligned}
SPT: \quad w_i &= \begin{cases} -1 & \text{if } i = 1 \\ \;\;\,0 & \text{otherwise} \end{cases} \\
LPT: \quad w_i &= \begin{cases} +1 & \text{if } i = 1 \\ \;\;\,0 & \text{otherwise} \end{cases} \\
MWR: \quad w_i &= \begin{cases} +1 & \text{if } i = 7 \\ \;\;\,0 & \text{otherwise} \end{cases} \\
LWR: \quad w_i &= \begin{cases} -1 & \text{if } i = 7 \\ \;\;\,0 & \text{otherwise} \end{cases}
\end{aligned}$$

where $i \in \{1, \ldots, d\}$. When using a feature space based on SDRs, the linear classification models can very easily be interpreted as CDRs with predetermined weights.

For this study synthetic JSP and FSP problem instances will be considered with the problem size 10×10. For each problem space N_{train} and N_{test} instances were generated for training and testing, respectively. Moreover, of the training data 20% is reserved for validation. Summary of problem classes is given in Table 2. Note, that difficult problem instances are not filtered out beforehand, such as the approach in [25].

Problem instances for JSP are generated stochastically by fixing the number of jobs and machines and discrete processing time are i.i.d. and sampled from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e., $\mathbf{p} \sim \mathscr{U}(u_1, u_2)$. Two different processing times distributions were explored, namely $\mathscr{P}_{j.rnd}^{n \times m}$ where $I = [1, 99]$ and $\mathscr{P}_{j.rndn}^{n \times m}$ where $I = [45, 55]$. The machine order is a random permutation of all of the machines in the job-shop, hence they problem spaces $\mathscr{P}_{j.rnd}^{n \times m}$ and $\mathscr{P}_{j.rndn}^{n \times m}$ are referred to as random and random-narrow, respectively.

Although in the case of $\mathscr{P}_{j.rnd}^{n \times m}$ this may be an excessively large range for the uniform distribution, it is however chosen in accordance with the literature [2] for creating synthesised $J||C_{\max}$ problem instances. In addition, w.r.t. the machine ordering, one could look into a subset of JSP where the machines are partitioned into two

Table 2: Problem space distributions used in experimental studies. Note, problem instances are synthetic and each problem space is i.i.d.

type	name	size ($n \times m$)	N_{train}	N_{test}	note
JSP	$\mathscr{P}_{j.rnd}^{10\times10}$	10×10	300	200	random
	$\mathscr{P}_{j.rndn}^{10\times10}$	10×10	300	200	random-narrow
PFSP	$\mathscr{P}_{f.rnd}^{10\times10}$	10×10	300	200	random

(or more) sets, where all jobs must be processed on the machines from the first set (in some random order) before being processed on any machine in the second set, commonly denoted as $J|2\text{sets}|C_{\max}$ problems, but as discussed in [23] this family of JSP is considered "hard" (w.r.t. relative error from best known solution) in comparison with the "easy" or "unchallenging" family with the general $J||C_{\max}$ set-up. This is in stark contrast to [25] whose findings showed that structured $F||C_{\max}$ were quite easier to solve than completely random structures. Intuitively, an inherent structure in machine ordering should be exploitable for a better performance. However, for the sake of generality, a random structure is preferred as they correspond to difficult problem instances in the case of JSP.

Problem instances for FSP are such that processing times are i.i.d. and uniformly distributed, $\mathscr{P}_{f.rnd}^{n\times m}$ where $\mathbf{p} \sim \mathscr{U}(1,99)$, referred to as random. In the JSP context $\mathscr{P}_{f.rnd}^{n\times m}$ is analogous to $\mathscr{P}_{j.rnd}^{n\times m}$.

6 Performance of SDR and BDR

In order to create successful dispatching rules, a good starting point is to investigate the properties of optimal solutions and hopefully be able to learn how to mimic such "good" behaviour. For this, we follow an optimal solution, obtained by using a commercial software package [7], and inspect the evolution of its features, defined in Table 1. Moreover, it is noted, that there are several optimal solutions available for each problem instance. However, it is deemed sufficient to inspect only one optimal trajectory per problem instance as there are $N_{\text{train}} = 300$ independent instances which gives the training data variety.

6.1 Probability of choosing optimal decision

Firstly, we can observe that on a step-by-step basis there are several optimal dispatches to choose from. Figure 2 depicts how the number of optimal dispatches evolve at each dispatch iteration. Note, that only one optimal trajectory is pursued (chosen at random), hence this is only a lower bound of uniqueness of optimal solutions. As the number of possible dispatches decrease over time, Fig. 3 depicts the probability of choosing an optimal dispatch.

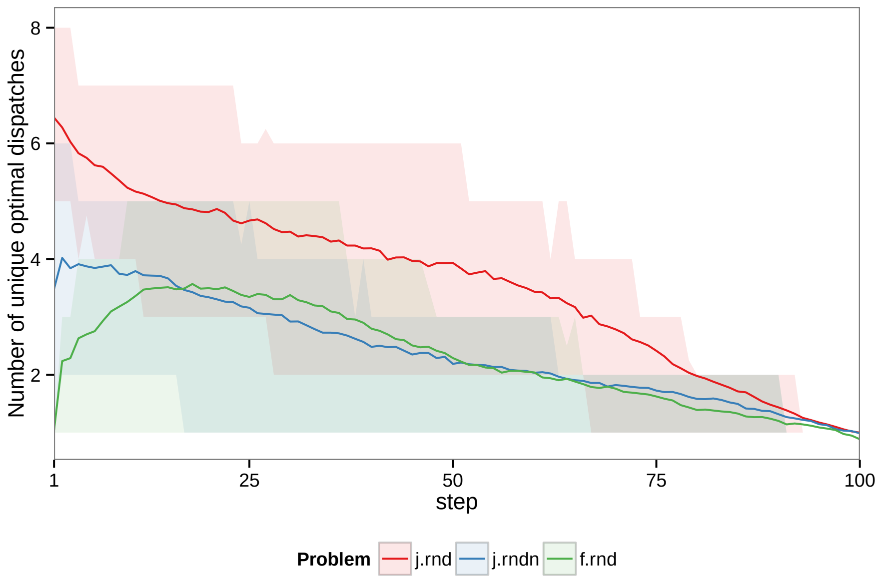

Fig. 2: Number of unique optimal dispatches (lower bound)

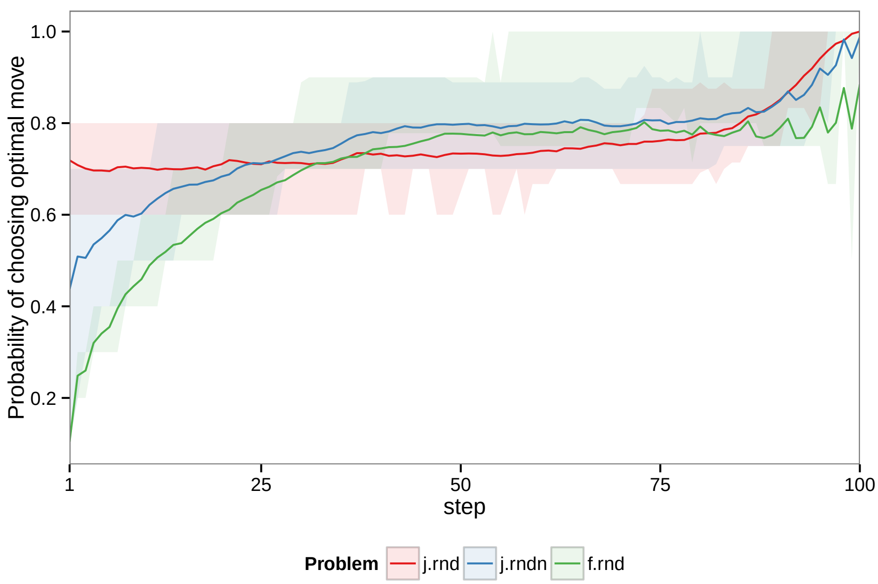

Fig. 3: Probability of choosing optimal move

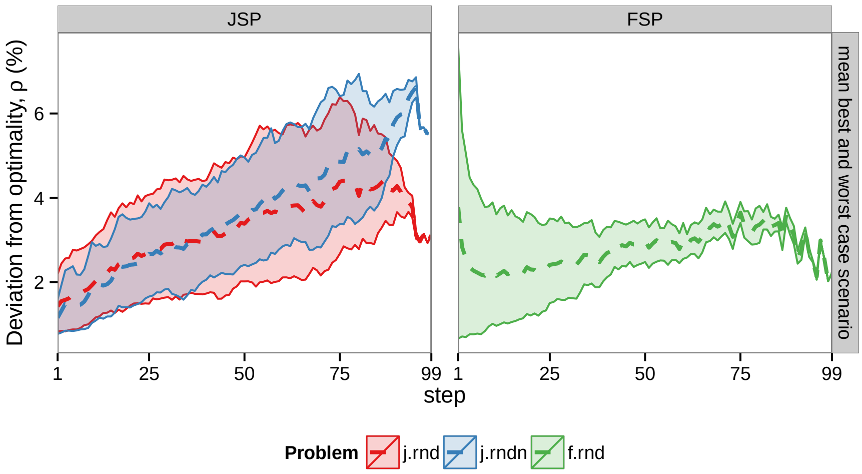

Fig. 4: Mean deviation from optimality, ρ, (%), for best (lower bound) and worst (upper bound) case scenario of choosing suboptimal dispatch for $\mathscr{P}_{j.rnd}^{10\times10}$, $\mathscr{P}_{j.rndn}^{10\times10}$ and $\mathscr{P}_{f.rnd}^{10\times10}$

6.2 Making suboptimal decisions

Looking at Fig. 3, $\mathscr{P}_{j.rnd}^{10\times10}$ has a relatively high probability (70% and above) of choosing an optimal job. However, it is imperative to keep making optimal decisions, because once off the optimal track the consequences can be dire. To demonstrate this Fig. 4 depicts mean worst and best case scenario of the resulting deviation from optimality, ρ, once you've fallen off the optimal track. Note, that this is given that you make *one* wrong turn. Generally, there will be more, and then the compound effects of making suboptimal decisions really start adding up.

It is interesting that for JSP, that over time making suboptimal decisions make more of an impact on the resulting makespan. This is most likely due to the fact that if suboptimal decision is made in the early stages, then there is space to rectify the situation with the subsequent dispatches. However, if done at a later point in time, little is to be done as the damage is already inflicted upon the schedule. However, for FSP, the case is the exact opposite. Then it's imperative to make good decisions right from the beginning. This is due to the major structural differences between JSP and FSP, namely the latter having a homogeneous machine ordering, constricting the solution immensely. Luckily, this does have the added benefit of making it less vulnerable for suboptimal decisions later in the decision process.

6.3 Optimality of simple priority dispatching rules

The probability of optimality of the aforementioned SDRs from Section 3.1, yet still maintaining our optimal trajectory, i.e., the probability of a job chosen by a SDR being able to yield an optimal makespan on a step-by-step basis, is depicted in Fig. 5. Moreover, the dashed line represents the benchmark of random guessing (cf. Fig. 3).

Now, let's bare in mind the deviation from optimality of applying SDRs throughout the dispatching process (box-plots of which are depicted in Fig. 6) then there is a some correspondence between high probability of stepwise optimality and low ρ. Alas, this isn't always the case, for $\mathscr{P}^{10\times 10}_{j.rnd}$, SPT always outperforms LPT w.r.t. stepwise optimality, however this does not transcend to SPT having a lower ρ value than LPT. Hence, it's not enough to just learn optimal behaviour, one needs to investigate what happens once we encounter suboptimal state spaces.

6.4 Simple blended dispatching rule

A naive approach to create a simple blended dispatching rule would be for instance be switching between two SDRs at a predetermined time point. Hence, going back to Fig. 5 a presumably good BDR for $\mathscr{P}^{10\times 10}_{j.rnd}$ would be starting with SPT and then switching over to MWR at around time step 40, where the SDRs change places in outperforming one another. A box-plot for ρ for all problem spaces is depicted in Fig. 7. Now, this little manipulation between SDRs does outperform SPT immensely, yet doesn't manage to gain the performance edge of MWR, save for $\mathscr{P}^{10\times 10}_{f.rnd}$. This gives us insight that for job-shop based problem spaces, the attribute based on MWR is quite fruitful for good dispatches, whereas the same cannot be said about SPT – a more sophisticated BDR is needed to improve upon MWR.

A reason for this lack of performance of our proposed BDR is perhaps that by starting out with SPT in the beginning, it sets up the schedules in such a way that it's quite greedy and only takes into consideration jobs with shortest immediate processing times. Now, even though it is possible to find optimal schedules from this scenario, as Fig. 5 show, the inherent structure that's already taking place, and might make it hard to come across by simple methods. Therefore it's by no means guaranteed that by simply swapping over to MWR will handle that situation which applying SPT has already created. Figure 7 does however show, that by applying MWR instead of SPT in the latter stages, does help the schedule to be more compact w.r.t. SPT. However, in the case of $\mathscr{P}^{10\times 10}_{j.rnd}$ and $\mathscr{P}^{10\times 10}_{j.rndn}$ the fact remains that the schedules have diverged too far from what MWR would have been able to achieve on its own. Preferably the blended dispatching rule should use best of both worlds, and outperform all of its inherited DRs, otherwise it goes without saying one would simply still use the original DR that achieved the best results.

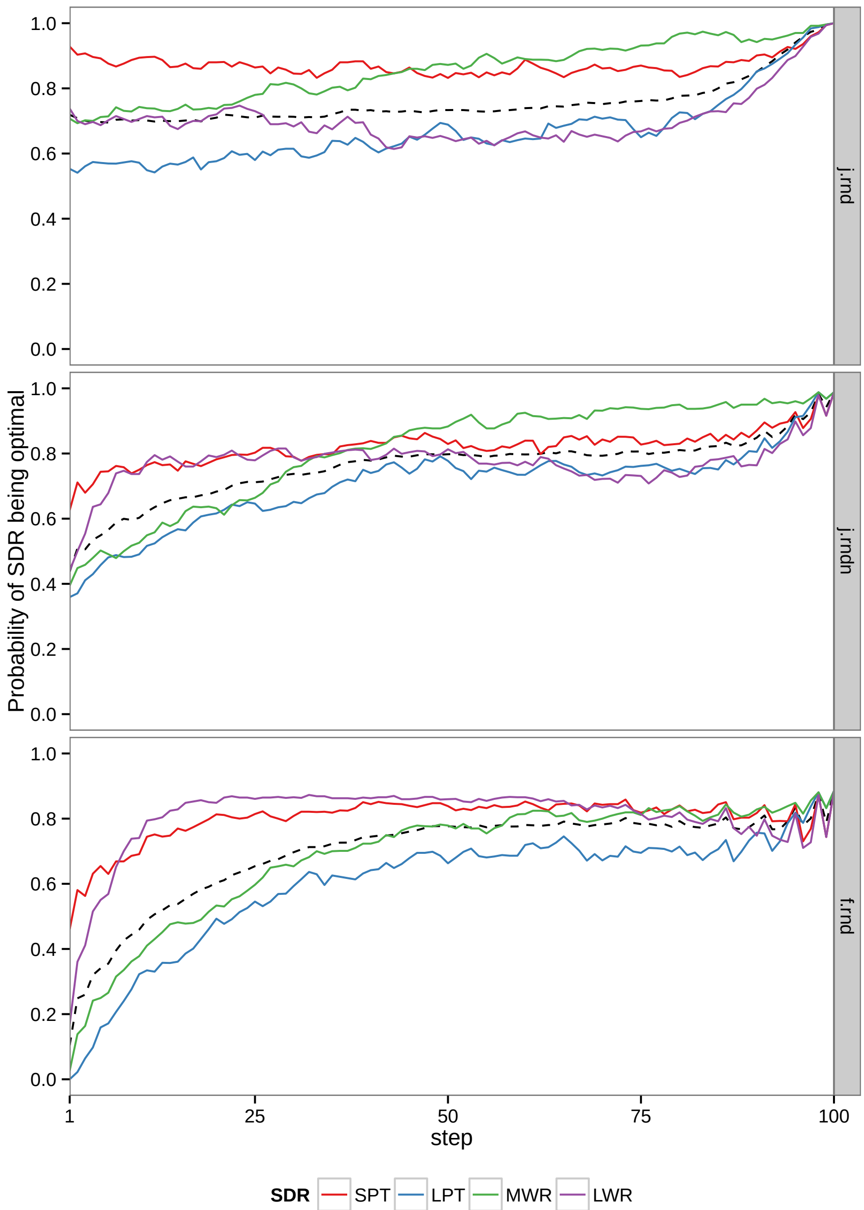

Fig. 5: Probability of SDR being optimal

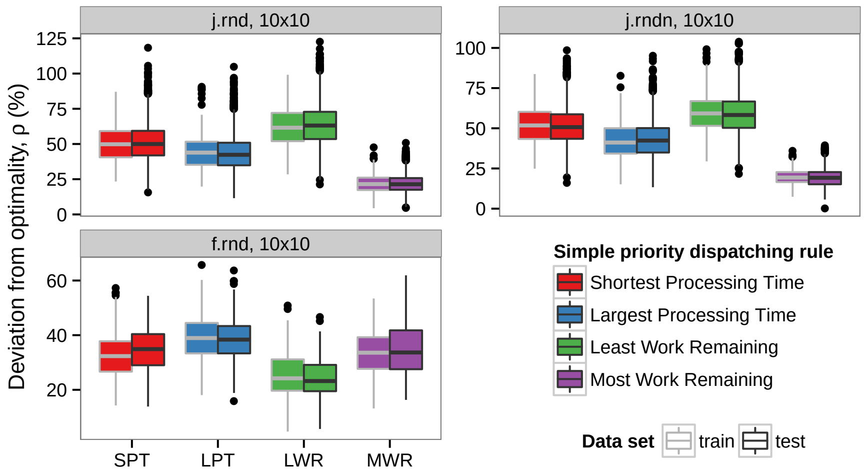

Fig. 6: Box plot for deviation from optimality, ρ, (%) for SDRs

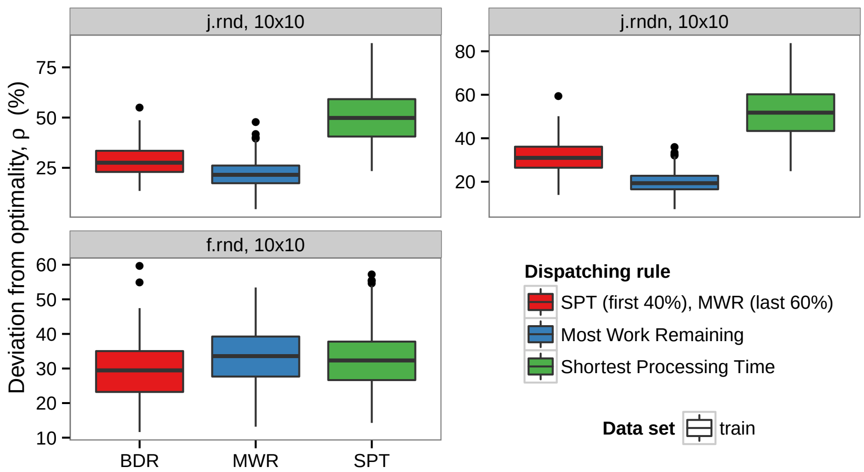

Fig. 7: Box plot for deviation from optimality, ρ, (%) for BDR where SPT is applied for the first 40% of the dispatches, followed by MWR

7 Learning CDR

Section 6.4 demonstrates there is definitely something to be gained by trying out different combinations, it's just non-trivial how to go about it, and motivates how it's best to go about learning such interaction, which will be addressed in this section.

7.1 Feature Selection

The SDRs we've inspected so-far are based on two features from Table 1, namely

- ϕ_1 for SPT and LPT
- ϕ_7 for LWR and MWR

by choosing the lowest value for the first SDR, and highest value for the latter SDR, i.e., the extremal values for those given features. There is nothing that limits us to using just those two features. From Table 1 we will limit our experiments to the first $d = 16$ features, as they are varying for each operation, save for ϕ_5 which is varying for each $J_j \in \mathscr{J}$.

For this study we will consider all combinations of features using either one, two, three or all of the features, for a total of $\binom{d}{1} + \binom{d}{2} + \binom{d}{3} + \binom{d}{d}$, i.e., total of 697 combinations. The reason for such a limiting number of active features, are due to the fact we want to keep the models simple enough for improved model interpretability

For each feature combination, a linear preference model is created in the manner described in Section 4, where Φ is limited to the predetermined feature combination. This was done with the software package from [3][2], by training on the full preference set S obtained from the $N_{\text{train}} = 300$ problem instances following the framework set up in Section 5. Note, in order to report the validation accuracy, 20% ($N = 60$) of the training set was set aside for validation of reporting the accuracy.

7.2 Validation accuracy

As the preference set S has both preference pairs belonging to optimal ranking, and subsequent rankings, it is not of primary importance to classify *all* rankings correctly, just the optimal ones. Therefore, instead of reporting the validation accuracy based on the classification problem of the correctly labelling the problem set S, it's opted the validation accuracy is obtained in the same manner as done in Section 6.3 for SDRs, i.e., the probability of choosing optimal decision given the resulting linear weights, however in this context, the mean throughout the dispatching process is reported. Figure 8 shows the difference between the two measures of reporting validation accuracy. Validation accuracy based on stepwise optimality only takes into consideration the likelihood of choosing the optimal move at each time step. However, the classification accuracy is also trying to correctly distinguish all subsequent rankings in addition of choosing the optimal move, as expected that measure is considerably lower.

7.3 Pareto front

When training the learning model one wants to keep the validation accuracy high, as that would imply a higher likelihood of making optimal decisions, which would in turn translate into a low final makespan. To test the validity of this assumptions, each of the 697 models is run on the preference set, and its mean ρ is reported

[2] Software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`

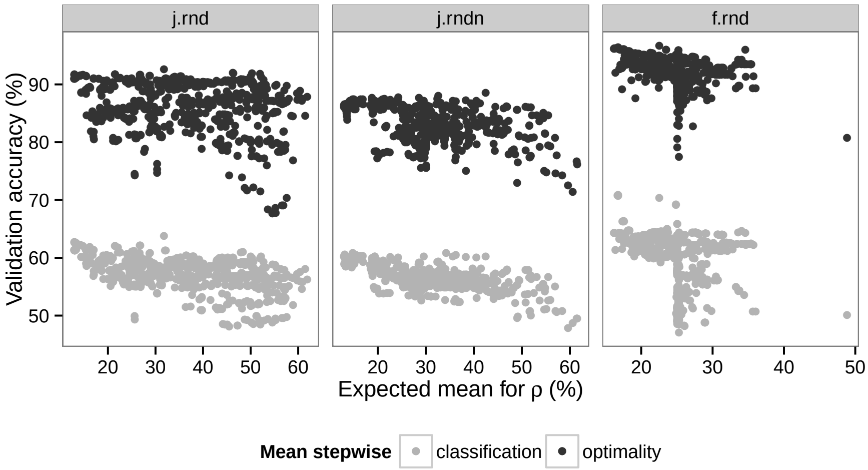

Fig. 8: Various methods of reporting validation accuracy for preference learning

against its corresponding validation accuracy in Fig. 9. The models are colour-coded w.r.t. the number of active features, and a line is drawn through its Pareto front. Moreover, those solutions are labelled with their corresponding model ID. Moreover, the Pareto front over all 697 models, irrespective of active feature count, is denoted with triangles. Moreover, their values are reported in Table 3, where the best objective is given in boldface.

For $\mathscr{P}_{j.rnd}^{10\times 10}$ there is no statistical difference between models 2.115, 3.503, 3.549 and 3.556 w.r.t. ρ, however only (2.115, 3.503) and (3.549, 3.556) w.r.t. validation accuracy. Other models were statistically significant to one another, using a Kolmogorov-Smirnov test with $\alpha = 0.05$. However, the solutions on the Pareto front for $\mathscr{P}_{j.rndn}^{10\times 10}$ are more or less with no (or minimal) statistical difference w.r.t. validation accuracy, and considerably fewer w.r.t. ρ. Most notably are the 2.107, 2.115, 3.486 and 3.549 (latter two have the lowest mean ρ) which are all statistically insignificant w.r.t. ρ and the latter three w.r.t. validation accuracy as well. For $\mathscr{P}_{f.rnd}^{10\times 10}$ 3.80, 3.120, 3.260 are equivalent to model corresponding to the lowest ρ, 3.244. Although, w.r.t. validation accuracy models 3.260 and 2.40 are statistically insignificant, where the latter yields a approx 1.5% worse mean ρ. So even looking at stepwise optimality by itself is very fickle, because slight variations can be quite dramatic to the end result.

Note, for both $\mathscr{P}_{j.rnd}^{10\times 10}$ and $\mathscr{P}_{j.rndn}^{10\times 10}$, model 1.16 is on the Pareto front. The model corresponds to feature ϕ_7, and in both cases has a weight strictly greater than zero (cf. Fig. 12). Revisiting Section 5.6, we observe that this implies the learning model was able to discover MWR as one of the Pareto solutions.

As one can see from Fig. 9, adding additional features to express the linear model boosts performance in both validation accuracy and expected mean for ρ, i.e., the Pareto fronts are cascading towards more desirable outcome with higher number of active features. However, there is a cut-off point for such improvement, as using

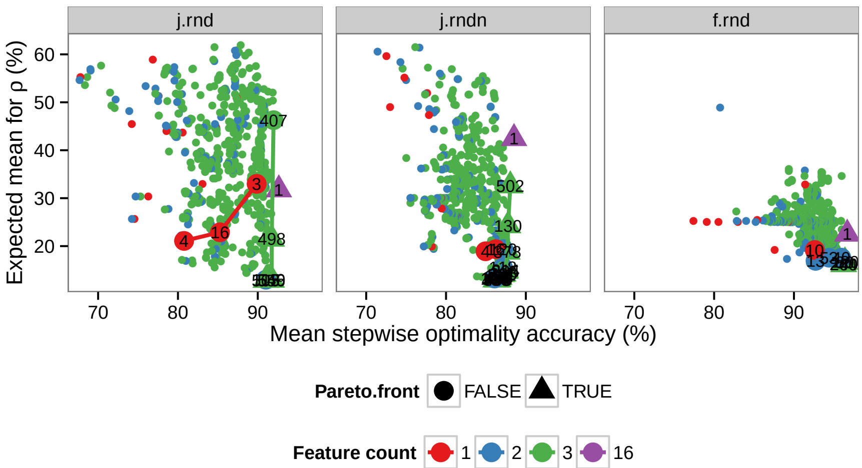

Fig. 9: Scatter plot for validation accuracy (%) against its corresponding mean expected ρ (%) for all 697 linear models, based on either one, two, three or all d combinations of features. Pareto fronts for each active feature count based on maximum validation accuracy and minimum mean expected ρ (%), and labelled with their model ID. Moreover, actual Pareto front over all models is marked with triangles.

all features is generally considerably worse off due to overfitting of classifying the preference set.

Now, let's inspect the models corresponding to the minimum mean ρ and highest mean validation accuracy, highlighted in Table 3 and inspect the stepwise optimality for those models in Fig. 10, again using probability of randomly guessing an optimal move from Section 6.1 as a benchmark. As one can see for both $\mathscr{P}_{j.rnd}^{10\times10}$ and $\mathscr{P}_{j.rndn}^{10\times10}$, despite having a higher mean validation accuracy overall, the probabilities vary significantly. A lower mean ρ is obtained when the validation accuracy is gradually increasing over time, and especially during the last phase of the scheduling.[3] Revisiting Fig. 4, this trend indicates that it's likelier for the resulting makespan to be considerably worse off if suboptimal moves are made at later stages, than at earlier stages. Therefore, it's imperative to make the 'best' decision at the 'right' moment, not just look at the overall mean performance. Hence, the measure of validation accuracy as discussed in Section 7.2 should take into consideration the impact a suboptimal move yields on a step-by-step basis, e.g., weighted w.r.t. a curve such as depicted in Fig. 4.

[3] It's almost too illegible to notice this shift directly from Fig. 10, as the difference between the two best models is oscillating up to only 3% at any given step. In fact $\mathscr{P}_{j.rndn}^{10\times10}$ has the most clear difference w.r.t. classification accuracy of indicating when a minimum ρ model excels at choosing the preferred move.

Table 3: Mean validation accuracy and mean expected deviation from optimality, ρ, for all CDR models on the Pareto front from Fig. 9.

Problem	PREF NrFeat.Model	Accuracy (%) Optimality	Classification	ρ (%)	Pareto
$\mathscr{P}_{j.rnd}^{10\times10}$	3.503	91.00	61.33	12.90	▲
	3.556	91.71	62.70	12.92	▲
	3.549	91.74	62.71	12.97	▲
	2.115	91.02	61.29	13.00	
	1.4	80.77	55.88	21.09	
	3.498	91.75	62.06	21.50	▲
	1.16	85.26	57.05	22.89	
	16.1	92.64	63.79	31.78	▲
	1.3	89.86	58.27	32.99	
	3.407	91.98	60.10	46.28	
$\mathscr{P}_{j.rndn}^{10\times10}$	3.549	86.42	60.16	12.99	▲
	3.486	86.22	60.30	12.99	▲
	3.493	86.48	58.92	13.03	▲
	3.456	86.52	58.90	13.09	▲
	2.107	86.08	59.27	13.25	
	2.115	86.17	58.93	13.38	
	3.492	86.57	58.80	13.43	▲
	3.458	86.67	58.81	13.53	▲
	2.116	86.59	59.26	13.86	
	3.521	86.97	59.21	14.09	▲
	2.40	86.65	58.90	14.12	
	3.205	87.16	58.90	14.22	▲
	3.335	87.43	59.20	14.78	▲
	3.214	87.46	59.25	15.03	▲
	2.118	87.12	60.42	15.56	
	3.378	87.69	58.70	18.79	▲
	1.4	84.95	57.46	18.93	
	1.16	86.22	58.04	19.37	
	2.120	87.16	60.22	19.39	
	3.130	87.77	59.01	24.30	▲
	3.502	88.05	59.40	32.62	▲
	16.1	88.52	60.22	42.48	▲
$\mathscr{P}_{f.rnd}^{10\times10}$	3.244	96.21	64.29	16.18	▲
	3.260	96.23	64.31	16.25	▲
	3.80	96.37	70.76	16.69	▲
	3.120	96.39	70.75	16.74	▲
	2.13	92.71	63.24	16.94	
	2.53	94.38	62.59	17.59	
	2.40	95.93	64.10	17.60	
	1.10	92.61	62.70	19.19	
	16.1	96.68	70.39	22.54	▲

Let's revert back to the original SDRs discussed in Section 6.3 and compare the best CDR models, a box-plot for ρ is depicted in Fig. 11. Firstly, there is a statistical difference between all models, and clearly the CDR model corresponding to minimum mean ρ value, is the clear winner, and outperforms the SDRs substantially. However, the best model w.r.t. maximum validation accuracy, then the CDR model shows a lacklustre performance. In some cases it's better off, e.g., compared to LWR, yet for job-shop it doesn't surpass the performance of MWR. This implies, the learning model is over-fitting the training data. Results hold for the test set.

7.4 Interpreting CDR

Section 5.6 showed how to interpret the linear preference models by their weights. Figure 12 depicts the linear weights, $\mathbf{w}$, from Eq. (5) for all of the CDR models reported in Table 3. The weights have been normalised for clarity purposes, such that it is scaled to $\|\mathbf{w}\| = 1$, thereby giving each feature their proportional contribution to the preference I_j^{CDR} defined by Eq. (2).

For $\mathcal{P}_{j.rndn}^{10\times10}$, there is no statistical difference between models 2.116 and 3.521 w.r.t. either ρ or validation accuracy. As Fig. 12 shows, ϕ_{16} and ϕ_7 are similar in value. However, looking at model 3.502 which has a slight statistical difference w.r.t. accuracy, the third feature yields the staggering difference in performance, about 20% increase in ρ. It's also interesting to inspect the full model for $\mathcal{P}_{f.rnd}^{10\times10}$, 16.1. Despite having similar contributions as all the active features of one of the its best model, 3.80, then the substantial interference from ϕ_8 along with other features present, hinders the full model from achieving a low ρ, thereby stressing the importance of feature selection, to steer clear of over-fitting.

Furthermore, in the case of models 3.80 and 3.120 for $\mathcal{P}_{f.rnd}^{10\times10}$ (equivalent both w.r.t. ρ and accuracy) the only difference is features ϕ_3 and ϕ_{10}. In addition, models 3.549 and 3.556 for $\mathcal{P}_{j.rnd}^{10\times10}$ show the same behaviour. As these features often coincide in job-shop, it is justifiable to use only either one, as the it contains the same information as its counterpart. Assuming this holds, then for models 3.498 and 3.503 where there is similar contributions between ϕ_{10} and ϕ_3, respectively, and ϕ_7 for both, the weights are similar, yet statistically significant from one another. There the third feature is the key to the success of the CDR, as opting for ϕ_{16} instead of ϕ_6 for 3.503 boosts the ρ performance by 8.6%. In addition, models 2.115 and 3.498, have similar contributions for ϕ_{10} and ϕ_7, however the additional ϕ_6, which causes the performance of ρ to diminish by 8.5%.

7.5 Resampling

This is still missing. Sampling strategies that have been applied (but not fully summarised)

- equal probability (current setting)
- w.r.t. best and worst case scenario
- inverted stepwise optimality
- or simply double emphasis on first half vs. second half (and vice versa)

Table 4: Mean validation accuracy and mean expected deviation from optimality, ρ, for all CDR models on the Pareto front using various re-sampling probabilities.

Problem	PREF NrFeat.Model	Sampling prob.	Accuracy (%) Optimality	Classification	ρ (%)
$\mathscr{P}_{j.rnd}^{10\times10}$	3.486	wcs	90.97	62.83	12.63
	3.486	bcs	91.17	62.91	12.71
	3.500	bcs	91.20	62.89	12.78
	3.556	equal	91.71	62.70	12.92
	3.549	equal	91.74	62.71	12.97
	3.556	opt	92.03	62.47	13.51
	3.498	wcs	92.11	62.27	19.48
	16.1	bcs	92.80	64.00	26.92
$\mathscr{P}_{j.rndn}^{10\times10}$	3.531	opt	85.87	59.91	12.74
	3.513	dbl2nd	86.67	58.41	12.80
	3.520	wcs	86.83	58.89	13.11
	3.513	wcs	86.89	58.87	13.47
	3.501	dbl1st	86.92	58.89	13.60
	3.458	bcs	86.99	58.83	13.72
	3.544	opt	87.08	59.03	13.75
	3.521	dbl1st	87.42	58.82	13.90
	3.544	dbl1st	87.44	58.83	13.97
	3.335	opt	87.58	58.86	14.49
	3.10	bcs	87.60	58.80	15.00
	3.130	bcs	87.70	59.10	16.37
	3.378	wcs	87.83	59.05	18.66
	3.368	wcs	87.85	59.94	21.41
	3.103	dbl1st	87.86	58.79	21.66
	3.26	dbl2nd	88.02	58.71	23.78
	3.26	dbl1st	88.05	58.75	23.84
	3.10	dbl2nd	88.09	58.67	23.90
	3.94	dbl2nd	88.12	58.64	24.43
	3.139	dbl1st	88.20	58.80	25.79
	3.130	dbl1st	88.70	58.92	36.79
	3.130	dbl2nd	88.74	58.92	36.84
	16.1	opt	89.07	60.04	42.07
	16.1	dbl1st	89.19	59.78	43.35
	16.1	dbl2nd	89.45	59.87	44.70
$\mathscr{P}_{f.rnd}^{10\times10}$	3.260	opt	96.74	63.86	15.32
	3.226	opt	96.72	63.85	15.32
	3.244	opt	96.76	63.80	15.61
	16.1	wcs	96.80	71.86	21.27
	16.1	dbl1st	96.83	70.78	22.19

8 Conclusions

Current literature still hold single priority dispatching rules in high regard, as they are simple to implement and quite efficient. However, they are generally taken for granted as there is clear lack of investigation of *how* these dispatching rules actually work, and what makes them so successful (or in some cases unsuccessful)? For instance, of the four SDRs this study focuses on, why does MWR outperform so significantly for job-shop, yet completely fail for flow-shop? MWR seems to be able to adapt to varying distributions of processing times, however manipulating the machine ordering causes MWR to break down. By inspecting optimal schedules, and meticulously researching what's going on, every step of the way of the dispatching sequence, some light is shed where these SDRs vary w.r.t. the problem space at hand. Once these simple rules are understood, then it's feasible to extrapolate the knowledge gained and create new composite rules that are likely to be successful.

"What general lessons might be learnt from this study?"

Creating new dispatching rules is by no means trivial. For job-shop there is the hidden interaction between processing times and machine ordering that's hard to measure. Due to this artefact, feature selection is of paramount importance, and then it becomes the case of not having too many features, as they are likely to hinder generalisation due to over-fitting in training. However, the features need to be explanatory enough to maintain predictive ability. For this reason Section 7 was limited to up to three active features, as the full feature set was clearly sub-optimal w.r.t. the SDRs used as a benchmark. By using features based on the SDRs, along with some additional local features describing the current schedule, it was possible to 'discover' the SDRs when given only one active feature. Furthermore, by adding on additional features, a boost in performance was gained, resulting in a composite dispatching rule that outperformed all of the SDR baseline.

When training the learning model, it's not sufficient to only optimize w.r.t. highest mean validation accuracy. As Section 7.3 showed, there is a trade-off between making the over-all best decisions versus making the right decision on crucial time points in the scheduling process, as Fig. 4 clearly illustrated. It is for this reason, traditional feature selection such as add1 and drop1 were unsuccessful in preliminary experiments, and thus resorting to having to exhaustively search all feature combinations. This also opens of the question of how should validation accuracy be measured? Since the model is based on learning preferences, both based on optimal versus suboptimal, and then varying degrees of sub-optimality. As we are only looking at the ranks in a black and white fashion, such that the makespans need to be strictly greater to belong to a higher rank, then it can be argued that some ranks should be grouped together if their makespans are sufficiently close. This would simplify the training set, making it (presumably) less of contradictions and more appropriate for linear learning. Or simply the validation accuracy could be weighted w.r.t. the difference in makespan.

Future work topic #1

During the dispatching process, there are some pivotal times which need to be especially taken care off. Figure 4 showed how making suboptimal decisions were more of a factor during the later stages, whereas for flow-shop the case was exact opposite.

Going to wait with this section until Section 7.5 has been completed

Despite the abundance of information gathered by following an optimal trajectory, the knowledge obtained is not enough by itself. Since the learning model isn't perfect, it is bound to make a mistake eventually. When it does, the model is in uncharted territory as there is not certainty the samples already collected are able to explain the current situation. For this we propose investigating features from suboptimal trajectories as well, since the future observations depend on previous predictions. A straight forward approach would be to inspect the trajectories of promising SDRs or CDRs. In fact, it would be worth while to try out imitation learning by [20, 21], such that the learned policy following an optimal trajectory is used to collect training data, and the learned model is updated. This can be done over several iterations, with the

Future work topic #2

benefit being, that the states that are likely to occur in practice are investigated, and as such used to dissuade the model from making poor choices. Alas, this comes at great computational cost due to the substantial amounts of states that need to be optimised for their correct labelling. Making it only practical for job-shop of a considerable lower dimension.

Future work topic #3

Although this study has been structured around the job-shop scheduling problem, it is easily extended to other types of deterministic optimisation problems that involve sequential decision making. The framework presented here collects snap-shots of the state space by following an optimal trajectory, and verifying the resulting optimal makespan from each possible state. From which the stepwise optimality of individual features can be inspected, which could for instance justify omittance in feature selection. Moreover, by looking at the best and worst case scenario of suboptimal dispatches, it is possible to pinpoint vulnerable times in the scheduling process.

Place work in wider context

Not done, but possible

References

1. Chen, T., Rajendran, C., Wu, C.W.: Advanced dispatching rules for large-scale manufacturing systems. The International Journal of Advanced Manufacturing Technology (2013)
2. Demirkol, E., Mehta, S., Uzsoy, R.: Benchmarks for shop scheduling problems. European Journal of Operational Research **109**(1), 137–141 (1998)
3. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research **9**, 1871–1874 (2008)
4. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
5. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2), 43–62 (2001)
6. Guinet, A., Legrand, M.: Reduction of job-shop problems to flow-shop problems with precedence constraints. European Journal of Operational Research **109**(1), 96–110 (1998)
7. Gurobi Optimization, Inc.: Gurobi optimization (version 5.6.2) [software] (2013). URL `http://www.gurobi.com/`
8. Haupt, R.: A survey of priority rule-based scheduling. OR Spectrum **11**, 3–16 (1989)
9. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: C. Coello (ed.) Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 6683, pp. 263–277. Springer, Berlin, Heidelberg (2011)
10. Jayamohan, M., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. European Journal of Operational Research **157**(2), 307–321 (2004)
11. Kalyanakrishnan, S., Stone, P.: Characterizing reinforcement learning methods through parameterized learning problems. Machine Learning **84**(1-2), 205–247 (2011)
12. Korytkowski, P., Rymaszewski, S., Wiśniewski, T.: Ant colony optimization for job shop scheduling using multi-attribute dispatching rules. The International Journal of Advanced Manufacturing Technology (2013)
13. Lu, M.S., Romanowski, R.: Multicontextual dispatching rules for job shops with dynamic job arrival. The International Journal of Advanced Manufacturing Technology (2013)
14. Mönch, L., Fowler, J.W., Mason, S.J.: Production Planning and Control for Semiconductor Wafer Fabrication Facilities, *Operations Research/Computer Science Interfaces Series*, vol. 52, chap. 4. Springer, New York (2013)
15. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Learning iterative dispatching rules for job shop scheduling with genetic programming. The International Journal of Advanced Manufacturing Technology (2013)
16. Olafsson, S., Li, X.: Learning effective new single machine dispatching rules from optimal scheduling data. International Journal of Production Economics **128**(1), 118–126 (2010)
17. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Operations Research **25**(1), 45–61 (1977)
18. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3 edn. Springer Publishing Company, Incorporated (2008)

19. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
20. Ross, S., Bagnell, D.: Efficient reductions for imitation learning. In: Y.W. Teh, D.M. Titterington (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10), vol. 9, pp. 661–668 (2010). URL `http://www.jmlr.org/proceedings/papers/v9/ross10a/ross10a.pdf`
21. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: G.J. Gordon, D.B. Dunson (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11), vol. 15, pp. 627–635. Journal of Machine Learning Research - Workshop and Conference Proceedings (2011). URL `http://www.jmlr.org/proceedings/papers/v15/ross11a/ross11a.pdf`
22. Runarsson, T.: Ordinal regression in evolutionary computation. In: T. Runarsson, H.G. Beyer, E. Burke, J. Merelo-Guervs, L. Whitley, X. Yao (eds.) Parallel Problem Solving from Nature - PPSN IX, *Lecture Notes in Computer Science*, vol. 4193, pp. 1048–1057. Springer, Berlin, Heidelberg (2006)
23. Storer, R.H., Wu, S.D., Vaccari, R.: New search spaces for sequencing problems with application to job shop scheduling. Management Science **38**(10), 1495–1509 (1992)
24. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers and Industrial Engineering **54**(3), 453–473 (2008)
25. 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 Journal on Computing **14**, 98–123 (2002)
26. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla-07: The design and analysis of an algorithm portfolio for SAT. Principles and Practice of . . . (2007)
27. Yu, J.M., Doh, H.H., Kim, J.S., Kwon, Y.J., Lee, D.H., Nam, S.H.: Input sequencing and scheduling for a reconfigurable manufacturing system with a limited number of fixtures. The International Journal of Advanced Manufacturing Technology (2013)

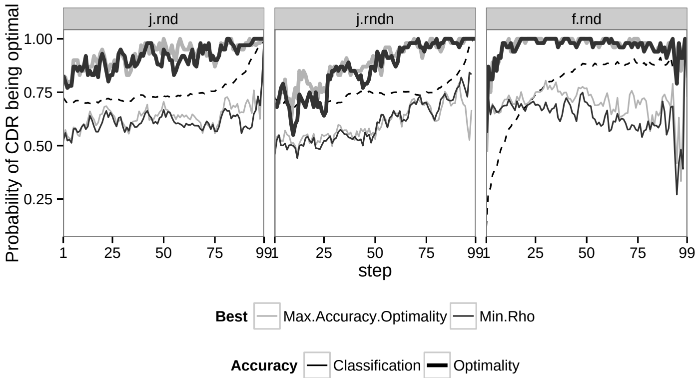

Fig. 10: Probability of choosing optimal move for models corresponding to highest mean validation accuracy (grey) and lowest mean deviation from optimality, ρ, (black) compared to the baseline of probability of choosing an optimal move at random (dashed).

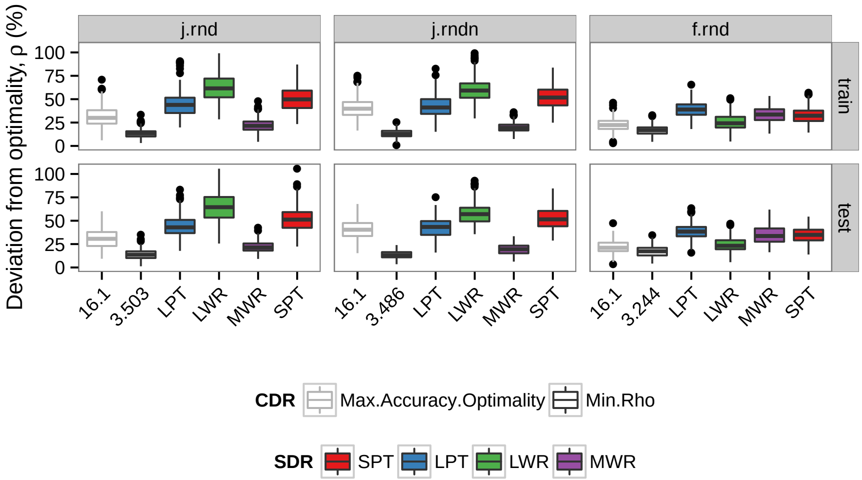

Fig. 11: Box plot for deviation from optimality, ρ, (%) for the best CDR models (cf. Table 3) and compared against SDRs from Section 6.3, both for training and test sets.

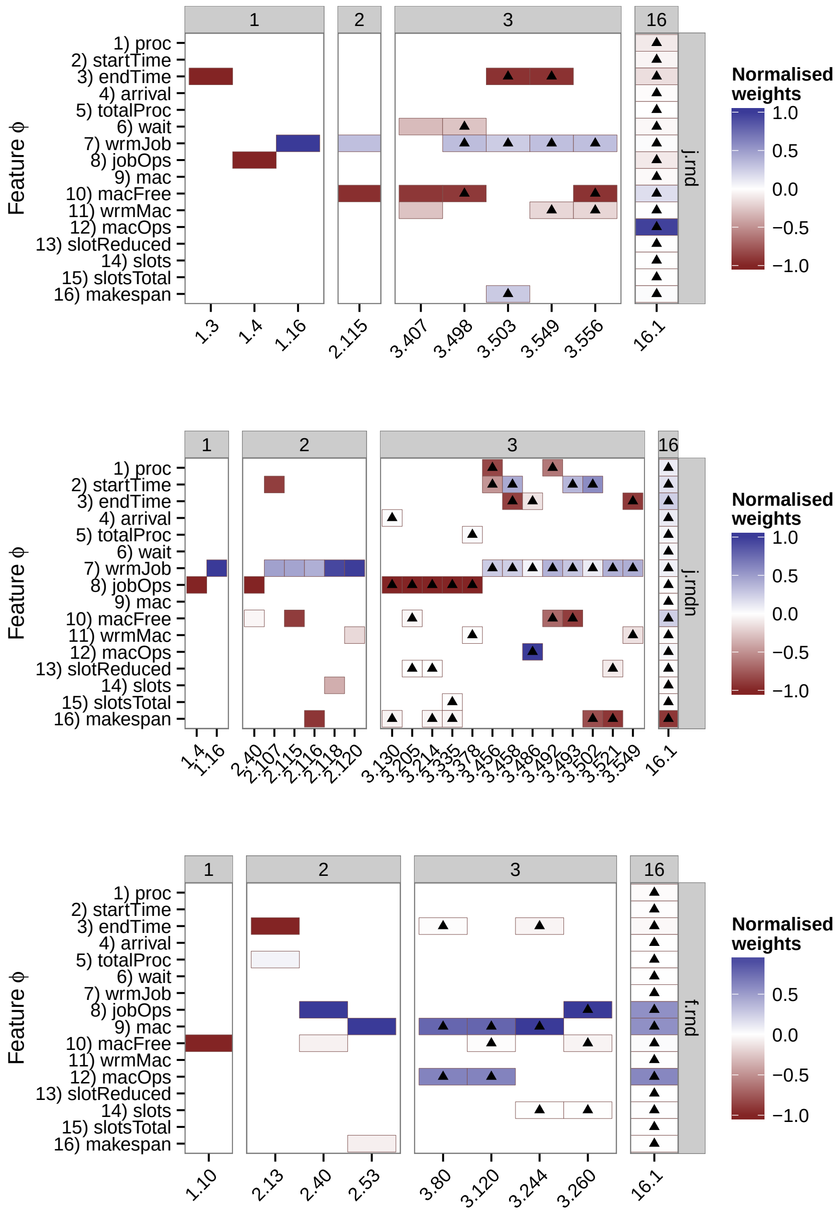

Fig. 12: Normalised weights for CDR models from Table 3, models are grouped w.r.t. its dimensionality, d. Note, a triangle indicates a solution on the Pareto front.