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

Supervised Learning Linear Composite Dispatch Rules for Scheduling

Case study for JSP and PFSP

Helga Ingimundardottir · Thomas Philip Runarsson

Received: August 29, 2014/ 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 training 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 · JSP · PFSP · Generating training data · Scalability · Feature Evolution

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

What is the problem?

Why is it interesting?

What are your contributions?

What is the outline of what you will show?

We show how it matters *when* during the scheduling process it's most fruitful to make the 'right' decision. Moreover, we give a framework on how to measure it.

We show how using optimal trajectory for creating training data, such as done by [17], is a good starting point, but not sufficient.

We show that it is important to look at the end-performance when chooing a suitable model, not just staring blindly at the training accuracy. Moreover, different measured on how to report training accuracy is discussed.

2 Background

2.1 Job Shop and Flow Shop Scheduling

In the job-shop problem (JSP), a set of jobs must be scheduled on a set of machines. Each job consists of a number of operations which are processed on the machines in a predetermined order. The optimal schedule is the one where the time to complete all jobs is minimal (minimum makespan).

For an $n \times m$ JSP considered here is 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, subject to the constraint that each job J_j must follow a predefined machine order (a chain or sequence of m operations $\boldsymbol{\sigma}_j = \{\sigma_{j1}, \sigma_{j2}, \ldots, \sigma_{jm}\}$) and that a machine can handle at most one job at a time. An additional constraint commonly considered are job release-dates and due-dates, however, those will not be considered here. The objective is 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, i.e. JSP w.r.t. minimising makespan, is $J||C_{\max}$ [19].

However, in the case that all jobs share the same permutation route $\boldsymbol{\sigma}_j$, JSP is reduced to a permutation flow-shop scheduling problem (PFSP) [5,23], denoted $F||C_{\max}$. Therefore, without the loss of generality, this study is structured around 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 setup times.

For this study synthetic JSP and PFSP problem instances will be considered with the problem sizes 8×8, 10×10 and 12×12. Summary of problem classes is given in Table 1. 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 $\mathscr{P}_{j.rnd}$ where $I = [1, 99]$ and $\mathscr{P}_{j.rndn}$ where $I = [45, 55]$. The machine order is a random permutation of all of the machines in the job-shop, hence they problem spaces $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$ are referred to as random and random-narrow, respectively.

For each JSP class N_{train} and N_{test} instances were generated for training and testing, respectively. Values for N are given in Table 1.

Although in the case of $\mathscr{P}_{j.rnd}$ 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 (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 [22] 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 $J||C_{\max}$ setup. 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 PFSP are such that processing times are i.i.d. and uniformly distributed, $\mathscr{P}_{f.rnd}$ where $\mathbf{p} \sim \mathscr{U}(1, 99)$, referred to as random. In the JSP context $\mathscr{P}_{f.rnd}$ is analogous to $\mathscr{P}_{j.rnd}$.

There are N_{train} and N_{test} instances were generated for training and testing, respectively. Values for N are given in Table 1.

2.2 Construction heuristics

Construction heuristics are designed in such a manner that it limits the search space in a logical manner, as to not to exclude the optimum. The construction heuristic here is to schedule the operations as closely together as possible and as soon as possible. Figure 1 illustrates the dispatching process with an example of a temporal partial schedule for a six-job and five-machine JSP where 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 with a dotted line.

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

type	name	size ($n \times m$)	N_{train}	N_{test}	note
JSP	$\mathscr{P}_{j.rnd}^{8\times8}$	8×8	–	500	random
	$\mathscr{P}_{j.rnd}^{10\times10}$	10×10	300	200	random
	$\mathscr{P}_{j.rnd}^{12\times12}$	12×12	–	500	random
	$\mathscr{P}_{j.rndn}^{8\times8}$	8×8	–	500	random-narrow
	$\mathscr{P}_{j.rndn}^{10\times10}$	10×10	300	200	random-narrow
	$\mathscr{P}_{j.rndn}^{12\times12}$	12×12	–	500	random-narrow
PFSP	$\mathscr{P}_{f.rnd}^{8\times8}$	8×8	–	500	random
	$\mathscr{P}_{f.rnd}^{10\times10}$	10×10	300	200	random
	$\mathscr{P}_{f.rnd}^{12\times12}$	12×12	–	500	random

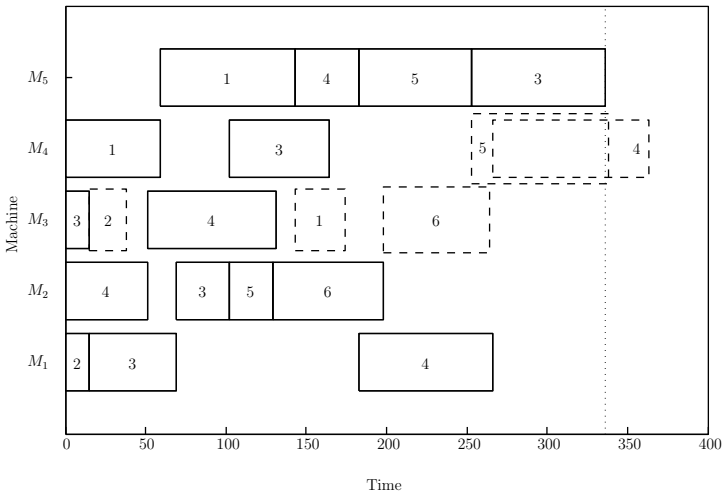

Fig. 1: Gantt chart of a partial JSP schedule after 15 dispatches: Solid and dashed boxes represent $\boldsymbol{\chi}$ and $\mathscr{R}^{(16)}$, respectively. Current $C_{\max}$ denoted as dotted line.

As one can see, there are 15 operations already scheduled. The sequence used to create the schedule was,

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

hence the ready-list is $\mathscr{R} = \{J_1, J_2, J_4, J_5, J_6\}$ (note that J_3 has traversed through all of its machines) indicating the 5 potential jobs to be dispatched at step $k = 16$.

If the job with the shortest processing time were to be scheduled next, i.e. implementing the SPT heuristic, then J_2 would be dispatched. Similarly, for the LPT heuristic then J_5 would be dispatched. Other dispatching rules, such as MWR and

LWR, use features not directly observable from looking at the current partial schedule (but easy to keep record of).

Henceforth, a *sequence* will refer to the sequential ordering of the dispatches of tasks to machines, i.e. (j,a); the collective set of allocated tasks to machines, which is interpreted by its sequence, is referred to as a *schedule*; a *scheduling policy* will pertain to the manner in which the sequence is manufactured for an (near) optimal schedule, e.g. MWR.

2.3 Simple priority dispatching rules

Dispatching rules are of a construction heuristics, where one starts with an empty schedule and adds sequentially on one operation (or tasks) at a time. When a machine is free the dispatching rule inspects waiting jobs, i.e. ready-list $\mathscr{R}$, and selects a job with the highest priority. A single-based priority dispatching rule is a function of features of the jobs and/or machines of the schedule. The features can be constant or vary throughout the scheduling process. For instance, the priority may depend on job processing attributes, such as which job has,

- shortest immediate processing time (SPT),
- longest immediate processing time (LPT),
- most work remaining (MWR)
- least work remaining (LWR)

These rules are the ones most commonly applied in the literature due to their simplicity and effectiveness However there are many more available, e.g. randomly selecting an operation with equal possibility (RND); minimum slack time (MST); smallest slack per operation (S/OP); and using the aforementioned dispatching rules with predetermined weights. A survey of more than 100 of such rules are presented in [18], however the reader is referred to an in-depth survey for single-priority or *simple* dispatching rules (SDR) by [7]. SDRs assign an index to each job of the ready-list waiting to be scheduled, and are generally only based on few features and simple mathematical operations.

2.4 Features

In order to apply a dispatching rule a number of features of the schedule being built must be computed, the features are used to grasp the essence of the current state of the schedule. The features of particular interest were obtained from inspecting the aforementioned single priority-based dispatching rules from Section 2.3. Some features are directly observed from the partial schedule. The temporal scheduling features in this study applied for a job J_j to be dispatched on machine M_a are given in Table 2. Note, from a job-oriented viewpoint, for a job already dispatched $J_j \in \mathscr{J}$ the corresponding set of machines now processed is $\mathscr{M}_j \subset \mathscr{M}$. Similarly from the machine-oriented viewpoint, $M_a \in \mathscr{M}$ with corresponding $\mathscr{J}_a \subset \mathscr{J}$.

The features of particular interest were obtained from inspecting the aforementioned SDRs from Section 2.3: ϕ_1-ϕ_8 and ϕ_9-ϕ_{12} are job-related and machine-related attributes of the current schedule, respectively.

Some features are directly observed from the partial schedule, such as the job- and machine-related features. In addition there are flow-related, ϕ_{13}-ϕ_{15} which measure the influence of idle time on the schedule, and current makespan-related, ϕ_{16}-ϕ_{18}.

All of the features vary throughout the scheduling process, w.r.t. operation belonging to the same time step k, save for ϕ_9, which is reported in order to distinguish which features are in conflict with each other; 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}).

2.5 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 DRs, 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 feature state $\boldsymbol{\phi}_j$.

Since each DR yield a priority index I^{DR} then it is easy to translate its index as a performance measure a. Then it is possible to combine several performance measures into a single DR, these are referred to as blended dispatching rules (BDR), where an overall blended priority index P is defined as

$$P_j = \sum_{a=l} w_a \cdot a \tag{3}$$

where $w_a > 0$ and $\sum_{a=0}^{C} w_a = 1$ and w_a gives the weight of the proportional influence of performance measure a (based on some SDR or CDR) to the overall priority.

Generally the weights $\mathbf{w}$ chosen by the algorithm designer apriori. A more sophisticated approach would to learn have the algorithm discover these weights autonomously, for instance via GAs or reinforcement learning.

[15] stress the importance automated discovery of DR and named several of successful implementations in the field of semiconductor wafer fabrication facilities however this sort of investigation is still in its infancy and subject for future research.

A recent editorial of the state-of-the-art approaches in advanced dispatching rules for large-scale manufacturing systems by [1] points out that:

Table 2: Feature space $\mathscr{F}$ 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

> [..] most traditional dispatching rules are based on historical data. With the emergence of data mining and online analytic processing, dispatching rules can now take predictive information into account.

implying that there has not been much automation in the process of discovering new dispatching rules, which is the ultimate goal of this dissertation, i.e. automate creating optimisation heuristics for scheduling.

With meta heuristics one can use existing DRs and use for example portfolio-based algorithm selection [20,4], either based on a single instance or class of instances [26] to determine which DR to choose from. Instead of optimising which algorithm to use under what data distributions, such as the case of portfolio algorithms, the approach taken in this dissertation is more similar to that of 'meta

learning' [24] which is the study of how learning algorithms can be improved, i.e. exploiting their strengths and remedy their failings, in order for a better algorithm design. Thus creating an adaptable learning algorithm that dynamically finds the appropriate dispatching rule to the data distribution at hand.

[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.

[16] 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.

[14] investigate 11 simple dispatching rules for JSP to create a pool of 33 composite dispatching rules that strongly outperformed the ones they were based on, which is intuitive since where one SDR might be failing, another could be excelling so combining them together should yield a better CDR. [14] create their composite dispatching rules with multi-contextual functions (MCFs) based on either on machine idle time or job waiting time, so one can say that the composite dispatching rules are a combination of those two key features of the schedule and then the basic dispatching rules. However, there are no combinations of the basic DR explored, only machine idle time and job waiting time. [27] used priority rules to combine 12 existing dispatching rules from the literature, in their approach they had 48 priority rules combinations, yielding 48 different models to implement and test. This is a fairly ad-hoc solution and there is no guarantee the optimal combination of dispatching rules is found.

At each time step k, an operation is dispatched which has the highest priority of the ready-list, $\mathscr{R}^{(k)} \subset \mathscr{J}$, i.e. the jobs who still have operations unassigned. If there is a tie, some other priority measure is used. Generally these priority dispatching rules are static during the scheduling process.

2.6 Supervised learning models

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 [21] and in [8] 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,

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

where $\Phi \subset \mathbb{R}^d$ 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, $\mathscr{O}^{(k)} \cup \mathscr{S}^{(k)} = \mathscr{R}^{(k)}$, and $\mathscr{O}^{(k)} \cap \mathscr{S}^{(k)} = \emptyset$.

For JSP there are $d = 18$ features (cf. Table 2 and explained in more detail in Section 2.5), and the training set is created in the manner described in Section 3.

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,

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

Let $\mathbf{z}$ denote either $\boldsymbol{\phi}_o - \boldsymbol{\phi}_s$ with $y = +1$ or $\boldsymbol{\phi}_s - \boldsymbol{\phi}_0$ with $y = -1$ (positive and negative example respectively). Logistic regression learns the optimal parameters $\mathbf{w} \in \mathbb{R}^d$ determined by solving the following task,

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\left\langle \mathbf{w} \cdot \mathbf{w} \right\rangle + C \sum_{i=1}^{|S|} \log\left(1 + e^{-y_i \left\langle \mathbf{w} \cdot \mathbf{z}_i \right\rangle}\right) \tag{7}$$

where $C > 0$ is a penalty parameter, and the negative log-likelihood is due to the fact the given data point $\mathbf{z}_i$ and weights $\mathbf{w}$ are assumed to follow the probability model,

$$\mathscr{P}(y = \pm 1 | \mathbf{z}, \mathbf{w}) = \frac{1}{1 + e^{-y \left\langle \mathbf{w} \cdot \mathbf{z}_i \right\rangle}}. \tag{8}$$

The logistic regression defined in (7) is solved iteratively, in particular using Trust Region Newton method [13], which generates a sequence $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ converging to the optimal solution $\mathbf{w}^*$ of (7).

The regulation parameter C in (7), controls the balance between model complexity and training errors, and must be chosen appropriately. It is also important to scale the features $\boldsymbol{\phi}$ first. A standard method of doing so is by scaling the training set such that all points are in some range, typically $[-1, 1]$. That is, scaled $\tilde{\boldsymbol{\phi}}$ is,

$$\tilde{\phi}_i = 2(\phi_i - \underline{\phi}_i)/(\overline{\phi}_i - \underline{\phi}_i) - 1 \qquad \forall i \in \{1, \ldots, d\} \tag{9}$$

where $\underline{\phi}_i$, $\overline{\phi}_i$ are the maximum and minimum i-th component of all the feature variables in set Φ, namely,

$$\underline{\phi}_i = \min\{\phi_i \mid \forall \boldsymbol{\phi} \in \Phi\} \quad \text{and} \quad \overline{\phi}_i = \max\{\phi_i \mid \forall \boldsymbol{\phi} \in \Phi\}. \tag{10}$$

where $i \in \{1 \dots d\}$. Moreover, scaling makes the features less sensitive to processing times.

Logistic regression makes optimal decisions regarding optimal dispatches and at the same time efficiently estimates a posteriori probabilities. The optimal $\mathbf{w}^*$ obtained by the training set, can be used on any new data point, $\boldsymbol{\phi}$, and their inner product is proportional to probability estimate (8). Hence, for each job on the ready-list, $J_j \in \mathscr{R}$, 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.,

$$J_{j^*} = \underset{J_j \in \mathscr{R}}{\operatorname{argmax}}\ h(\boldsymbol{\phi}_j) \tag{11}$$

where $h(\cdot)$ is the classification model obtained by the preference set, S, defined by (4).

2.7 Interpreting linear classification models

Looking at the features description in Table 2 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 2.3. For instance,

$$\text{SPT:} \quad w_i = \begin{cases} -1 \text{ if } i = 1 \\ 0 \text{ otherwise} \end{cases}$$

$$\text{LPT:} \quad w_i = \begin{cases} +1 \text{ if } i = 1 \\ 0 \text{ otherwise} \end{cases}$$

$$\text{MWR:} \quad w_i = \begin{cases} +1 \text{ if } i = 7 \\ 0 \text{ otherwise} \end{cases}$$

$$\text{LWR:} \quad w_i = \begin{cases} -1 \text{ if } i = 7 \\ 0 \text{ otherwise} \end{cases}$$

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

3 Generating training data

For each problem class described in Table 1 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 $C^{\text{opt}}_{\max}$, and the makespan obtained from the scheduling policy A under inspection by $C^{A}_{\max}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho = \frac{C^{A}_{\max} - C^{\text{opt}}_{\max}}{C^{\text{opt}}_{\max}} \cdot 100\% \tag{12}$$

which indicates the percentage relative deviation from optimality.

3.1 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 2) 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.

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 (12), 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.

3.2 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, $\mathscr{O}^{(k)} \cup \mathscr{S}^{(k)} = \mathscr{R}^{(k)}$ and $\mathscr{O}^{(k)} \cap \mathscr{S}^{(k)} = \emptyset$. In particular, each job is compared against another job of the ready-list, $\mathscr{R}^{(k)}$, 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

[1] Dispatch and time step are used interchangeably.

into consideration when there are multiple optimal solutions to the same problem instance.

3.3 Creating time-independent dispatching rules

Preliminary experiments for creating step-by-step model was done in [8] 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 training set, and for S to be computationally feasible to learn, S has to be reduced. For this several ranking strategies were explored in [9], 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 training set, where $r_1 > r_2 > \ldots > r_{n'}$ ($n' \leq n$) are the rankings of the ready-list, $\mathscr{R}^{(k)}$, at time step k, in such a manner that in the cases that there are more than 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 training set, $|S|$, becomes too large with full ranking, and would need sampling.

4 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 [6], and inspect the evolution of its features, defined in Table 2. 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.

Note, for this section, only 10×10 problem instances will be considered from the problem spaces described in Table 1. Leaving dimensionality 8×8 and 12×12 solely for testing scalability in Section 5.7. Figures within this section depict the mean over all the training data.

4.1 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

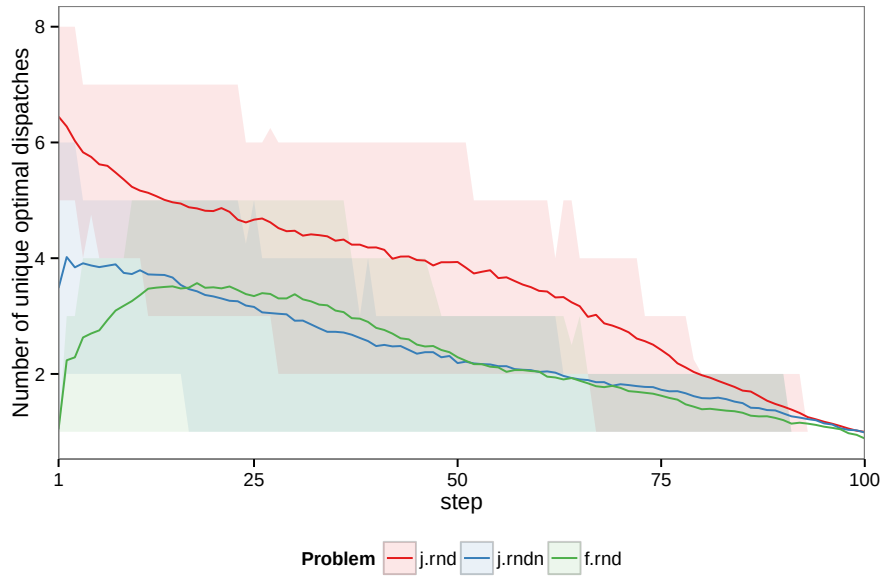

Fig. 2: Number of unique optimal dispatches (lower bound)

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.

4.2 Making suboptimal decisions

Looking at Fig. 3, $\mathscr{P}_{j.rnd}$ 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 the 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 $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$, 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 done. However, for flow shop, 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 PFSP, namely the latter having a homogeneous machine ordering, constricting the solution

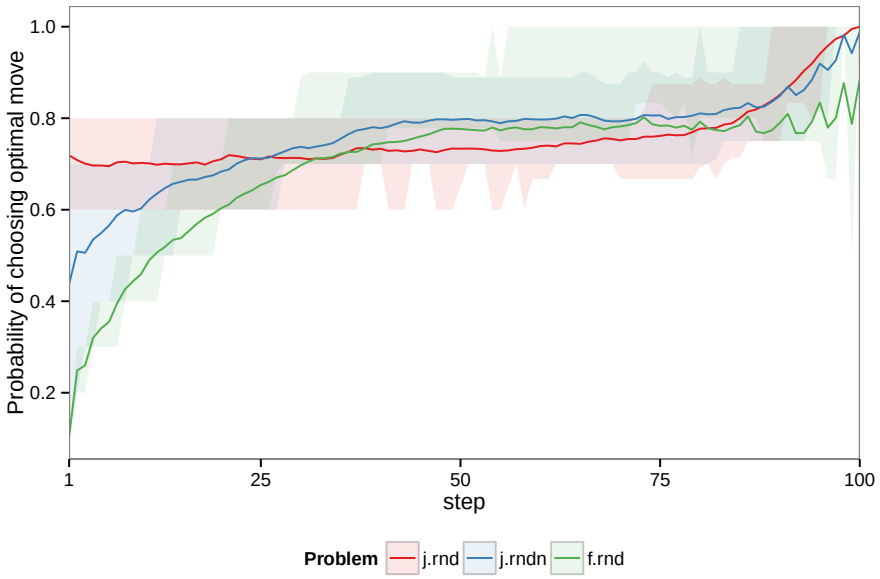

Fig. 3: Probability of choosing optimal move

immensely. Luckily, this does have the added benefit of making it less vulnerable for suboptimal decisions later in the decision process.

4.3 Optimality of simple priority dispatching rules

The probability of optimality of the aforementioned SDRs from Section 2.3, 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 $\mathscr{P}_{j.rnd}$, 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.

4.4 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

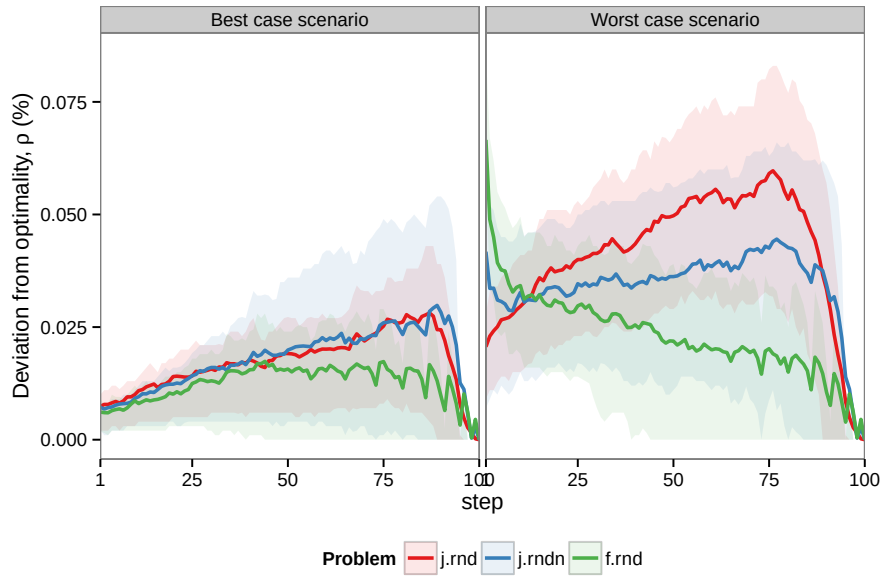

Fig. 4: Deviation from optimality, ρ, (%), for best and worst case scenario of choosing suboptimal dispatch for $\mathscr{P}_{j.rnd}$, $\mathscr{P}_{j.rndn}$ and $\mathscr{P}_{f.rnd}$

to Fig. 5 a presumably good BDR for $\mathscr{P}_{j.rnd}$ 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 $\mathscr{P}_{f.rnd}$. 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 $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$ 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,

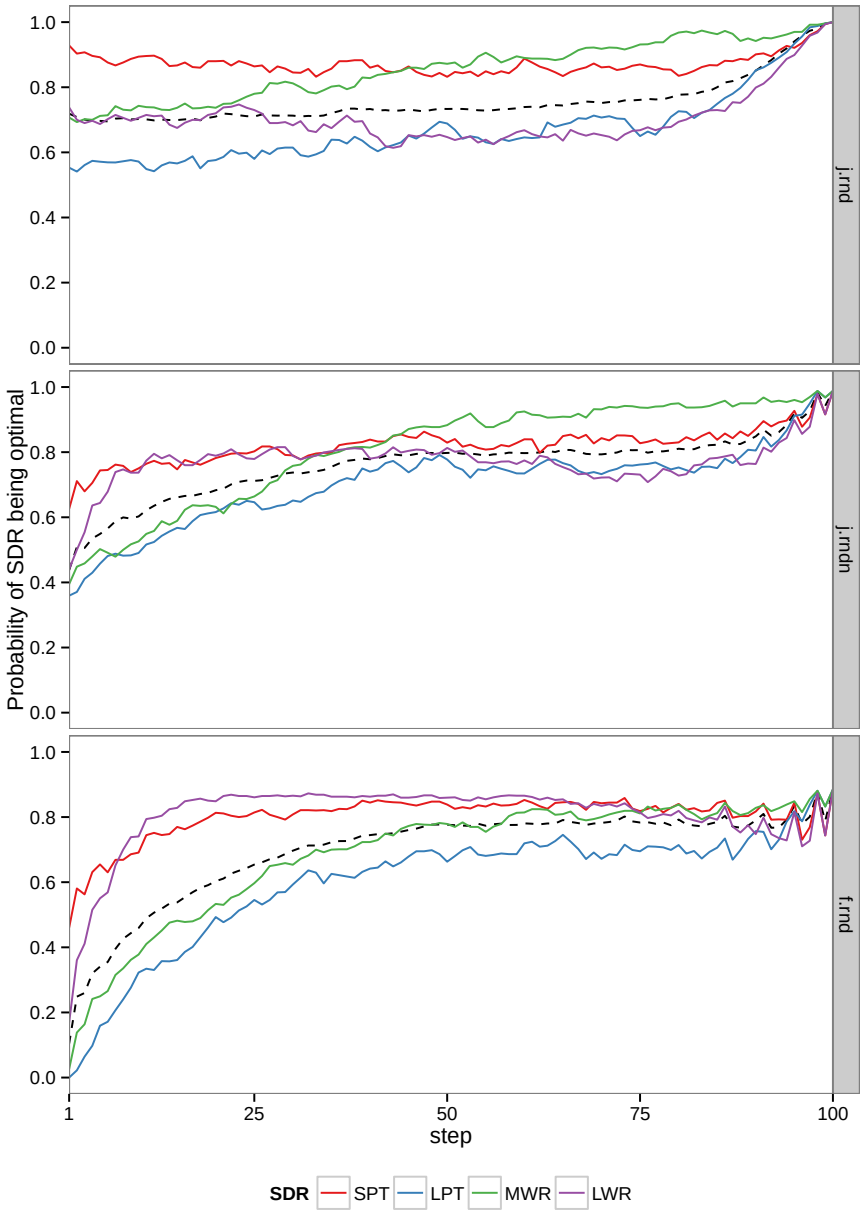

Fig. 5: Probability of SDR being optimal

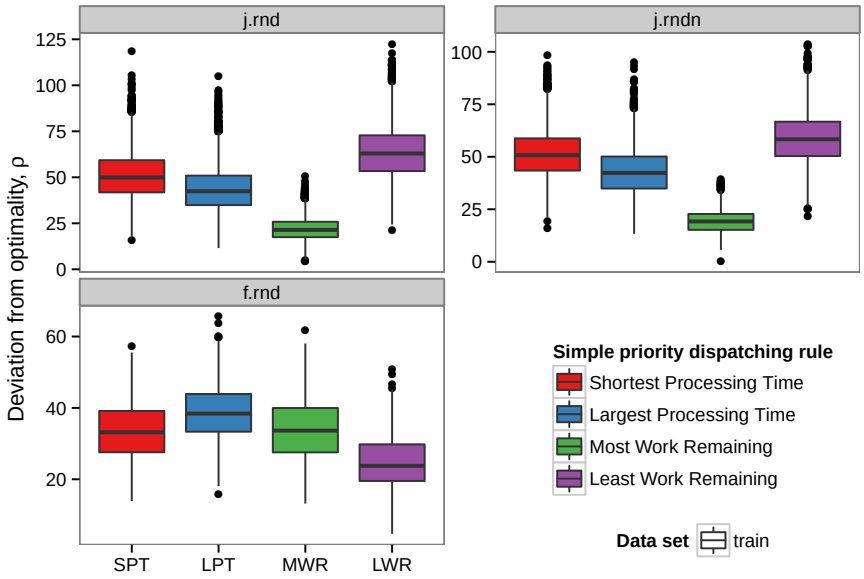

Fig. 6: Box plot for deviation from optimality, ρ, (%) for SDRs

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.

5 Learning CDR

Section 4.4 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.

5.1 Feature Selection

The SDRs we've inspected so-far are based on two features from Table 2, 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 2 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 $J_j \in \mathscr{J}$.

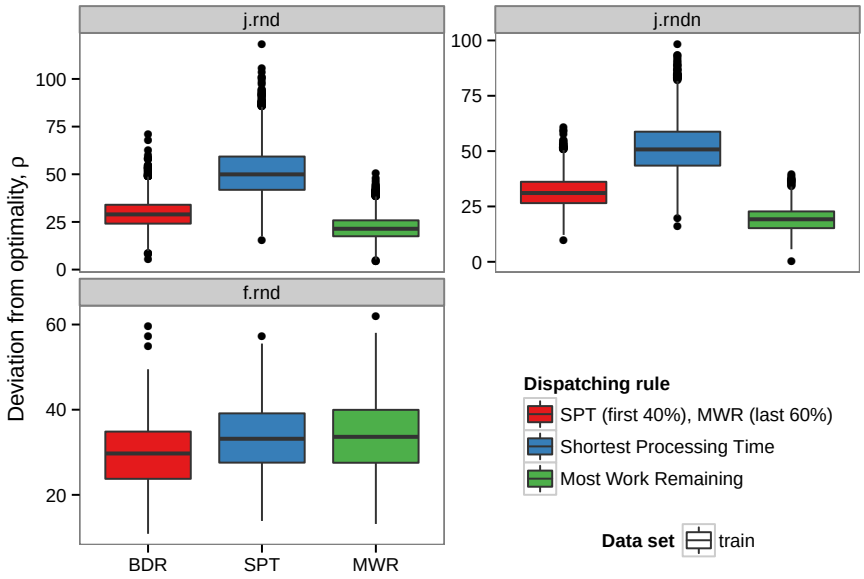

Fig. 7: Box plot for deviation from optimality, ρ, (%) for BDR where SPT is applied for the first 40% of the dispatches, followed by MWR

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 to be reasonably easy to visualize for Section 5.6.

For each feature combination, a linear preference model is created in the manner described in Section 2.6, 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 3.

5.2 Training 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 training accuracy based on the classification problem of the correctly labelling the problem set S, it's opted the training accuracy is obtained in the same manner as done in Section 4.3 for SDRs, i.e. the probability of choosing optimal decision given the resulting linear weights,

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

however in this context, the mean throughout the dispatching process is reported. Figure 8 shows the difference between the two measures of reporting training accuracy. Training 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.

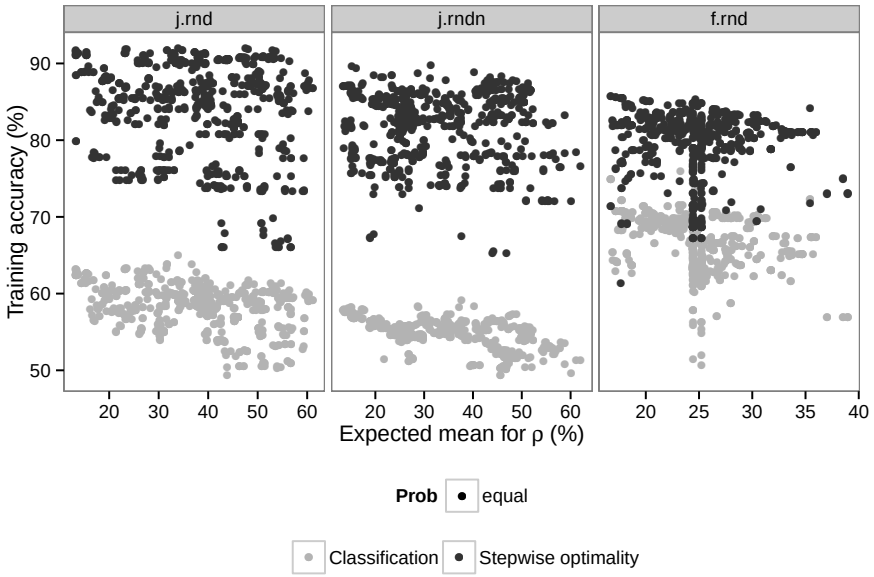

Fig. 8: Various methods of reporting training accuracy for preference learning

5.3 Pareto front

When training the learning model one wants to keep the training 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 training set, and its mean ρ is reported against its corresponding training 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. Note for $\mathscr{P}_{j.rndn}$ there is no statistical difference between models 3.501, 3.508 and 3.510 w.r.t. training accuracy, however only 3.501 and 3.508 w.r.t. ρ. Other

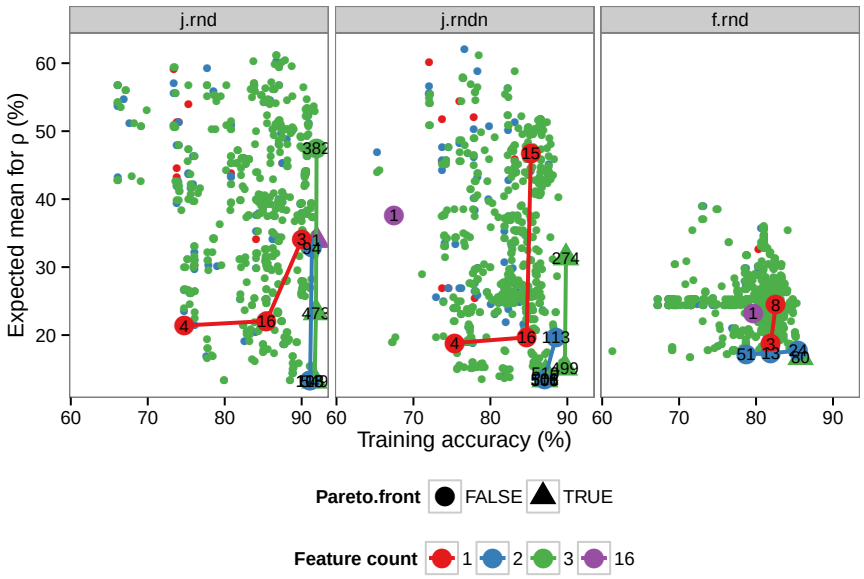

Fig. 9: Scatter plot for training accuracy (%) against its corresponding mean expected ρ (%) for all 697 linear models, based on either one, two, three or all d combinations of features. Pareto fronts for each active feature count based on maximum training accuracy and minimum mean expected ρ (%), and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

models were statistically significant to one another, using a Kolmogorov-Smirnov test with $\alpha = 0.05$.

Note, for both $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$, 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 2.7, 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 training 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 all features is generally considerably worse off.

Now, let's inspect the models corresponding to the minimum mean ρ and highest training 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 4.1 as a benchmark. Note, only one CDR model is plotted for $\mathscr{P}_{f.rnd}$ as its Pareto front constitutes of only a single model. As one can see for both $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$, despite having a higher mean training accuracy overall, the probabilities vary significantly. A lower mean ρ is obtained when the training

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

Problem	NrFeat.Model	Acc	ρ	Pareto
$\mathscr{P}_{j.rnd}$	1.3	90.03	34.08	
	1.4	74.74	21.41	
	1.16	85.43	22.06	
	2.94	91.34	32.84	
	2.108	91.10	13.32	
	2.115	91.08	13.31	
	3.382	91.94	47.53	
	3.473	91.86	23.31	▲
	3.549	91.68	**13.26**	▲
	16.1	**91.95**	33.96	▲
$\mathscr{P}_{j.rndn}$	1.4	75.38	18.84	
	1.15	85.26	46.77	
	1.16	84.72	19.66	
	2.113	88.53	19.66	
	2.116	87.04	13.52	
	3.274	**89.82**	31.39	▲
	3.499	89.68	15.19	▲
	3.501	87.09	13.50	▲
	3.508	87.07	**13.44**	▲
	3.510	87.17	14.42	▲
	16.1	67.48	37.66	
$\mathscr{P}_{f.rnd}$	1.3	81.91	18.70	
	1.8	82.55	24.45	
	2.13	81.91	17.30	
	2.24	85.46	17.74	
	2.51	78.72	17.17	
	3.80	**85.79**	**16.72**	▲
	16.1	79.63	23.25	

accuracy is gradually increasing over time, because revisiting Fig. 4, 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 training accuracy as discussed in Section 5.2 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.

Let's revert back to the original SDRs discussed in Section 4.3 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, for $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$, where the best model w.r.t. minimum ρ doesn't coincide with the model corresponding to the maximum training accuracy, such as the case with $\mathscr{P}_{f.rnd}$, then the CDR model shows a lacklustre performance. In some cases it's better off, e.g. compared to LWR, yet doesn't surpass the performance of

MWR. This implies, the learning model is overfitting the training data. Results hold for the test set.

5.4 Interpreting CDR

Section 2.7 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).

As discussed in Section 5.3 for $\mathscr{P}_{j.rndn}$, there is no statistical difference between models 3.501, 3.508 and 3.510 w.r.t. training accuracy. As Fig. 12 shows, ϕ_{16} and ϕ_7 are similar in value, however it's the third feature that yields the difference in performance. In fact, the contribution from ϕ_1 in 3.501 is on par with ϕ_9 in 3.508, as those models are not statistically different w.r.t. ρ performance. However, the decreased contribution of ϕ_{16} in favour for ϕ_{10} in 3.510 results in approximately 1% increase in ρ. Furthermore, it's sufficient to use only ϕ_{16} and ϕ_7 as active features, as model 2.116 has no statistical difference from either 3.508 or 3.510, for both ρ and training accuracy.

Similarly for $\mathscr{P}_{j.rnd}$, ϕ_7 and ϕ_3 are similar for models 3.473 and 3.549, yet statistically significant from one another. There the third feature is the key to the success of the CDR, as opting for ϕ_{11} instead of ϕ_6 for 3.549 boosts the ρ performance by about 10%.

It's also interesting to inspect the full model for $\mathscr{P}_{f.rnd}$, 1.16. Despite having similar contributions as all the active features of its best model, 3.80, then the substantial interference from ϕ_8 along with other features present, hinders the full model from both objectives, i.e. high training accuracy and low ρ, thereby stressing the importance of feature selection.

5.5 Resampling

5.6 Adaboost

Run Adaboost experiment

5.7 Scalability

Up till now, only 10×10 problem instances have been investigated. However, it is interesting to know whether the hypotheses still hold for both lower and higher dimensions, i.e. 8×8 and 12×12 respectively.

Check 8×8 scalability

Check 12×12 scalability

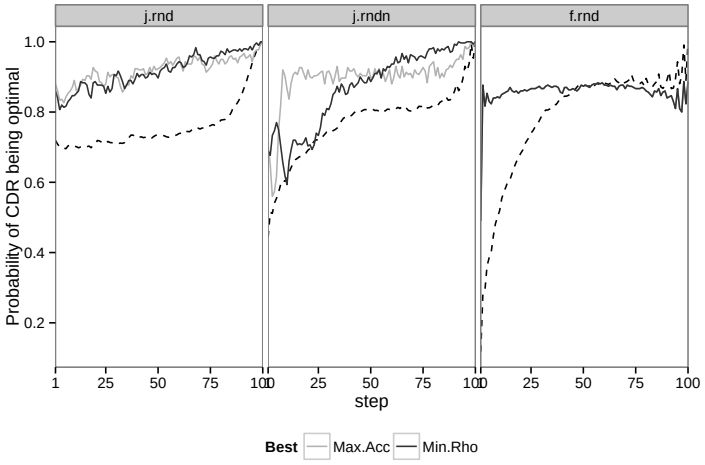

Fig. 10: Probability of choosing optimal move for models corresponding to highest training accuracy (grey) and lowest mean deviation from optimality, ρ, (black) compared to the baseline of probability of choosing an optimal move at random (dashed).

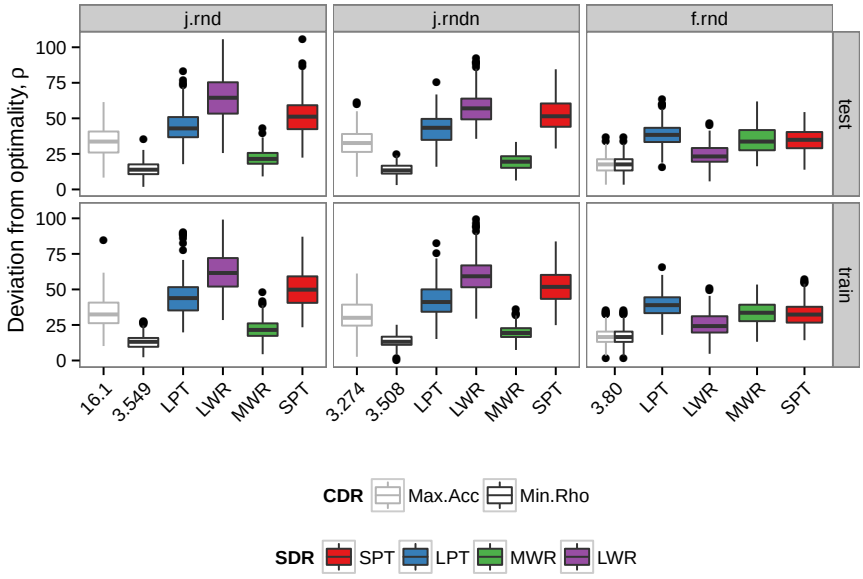

Fig. 11: Box plot for deviation from optimality, ρ, (%) for the best CDR models (cf. Table 3) and compared against SDRs from Section 4.3, both for training and test sets.

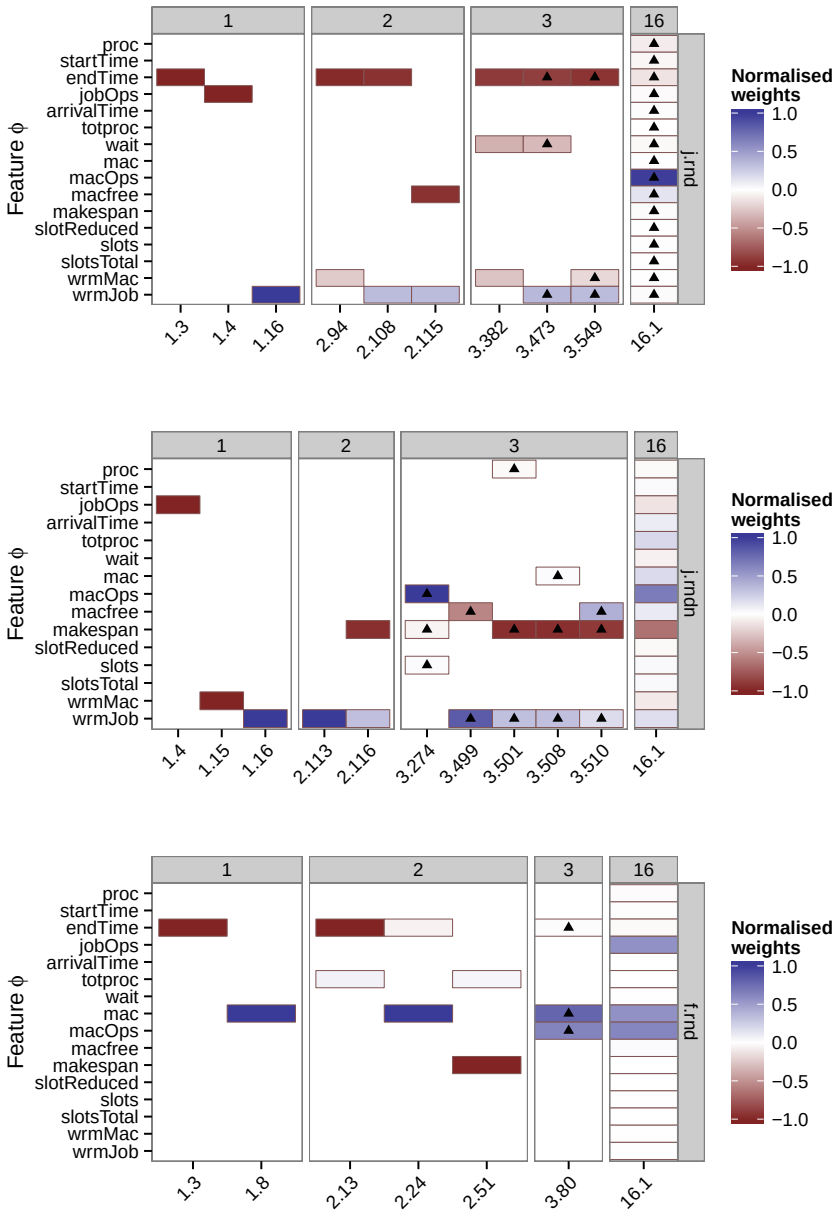

Fig. 12: Normalised weights for CDR models from Table 3, models are grouped w.r.t. its dimensionality, d. Note, a triangle indicates a solution on the Pareto front.

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

Problem	NrFeat.Model	Prob	Acc	ρ	Pareto
$\mathscr{P}_{j.rnd}$	2.94	wcs	91.50	32.31	
	2.101	bcs	91.52	32.36	
	2.108	equal	91.10	13.32	
	2.108	opt	91.22	13.42	
	2.108	bcs	90.94	13.18	
	2.115	equal	91.08	13.31	
	2.115	opt	91.19	13.41	
	3.498	opt	91.98	24.79	▲
	3.500	bcs	91.08	**13.09**	▲
	3.510	wcs	91.12	13.24	▲
	3.549	equal	91.68	13.26	▲
	3.549	opt	91.87	13.40	▲
	16.1	equal	91.95	33.96	
	16.1	opt	**92.00**	34.68	▲
$\mathscr{P}_{j.rndn}$	2.113	equal	88.53	19.66	
	2.116	opt	87.24	**13.43**	▲
	3.508	opt	**90.97**	13.45	▲
	3.532	opt	87.43	13.44	▲
	16.1	equal	67.48	37.66	
	16.1	opt	89.64	46.20	
$\mathscr{P}_{f.rnd}$	2.13	equal	81.91	17.30	
	2.42	opt	81.90	17.13	
	2.44	wcs	85.70	17.58	
	3.80	wcs	**85.87**	16.75	▲
	3.116	opt	71.61	**16.55**	▲
	3.499	opt	85.80	16.67	▲
	16.1	equal	79.63	23.25	

6 Conclusions

Remind reader of what you have done

Place work in wider context

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

Flag all the exciting open research directions

Current literature still hold simple priority dispatching rules in high regard, as they are simple to implement and quite effective. 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, e.g. 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, in order to shed some information where these SDRs vary w.r.t. the problem distribution

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.

Creating new dispatching rules is by no means trivial. For job shop scheduling there is the hidden interaction between processing times and machine ordering that's hard to measure. Due to this artefact, feature selection becomes of paramount importance, and then it becomes 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 5 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 training accuracy. As Section 5.3 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.

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. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2), 43–62 (2001)
5. 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)
6. Gurobi Optimization, Inc.: Gurobi optimization (version 5.6.2) [software] (2013). URL `http://www.gurobi.com/`
7. Haupt, R.: A survey of priority rule-based scheduling. OR Spectrum **11**, 3–16 (1989)
8. 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)
9. Ingimundardottir, H., Runarsson, T.P.: Generating training data for supervised learning linear composite dispatch rules for scheduling (2014). Submitted
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. Lin, C.J., Weng, R.C., Keerthi, S.S.: Trust region newton method for logistic regression. J. Mach. Learn. Res. **9**, 627–650 (2008)
14. Lu, M.S., Romanowski, R.: Multicontextual dispatching rules for job shops with dynamic job arrival. The International Journal of Advanced Manufacturing Technology (2013)

15. 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)
16. 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)
17. 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)
18. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Operations Research **25**(1), 45–61 (1977)
19. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3 edn. Springer Publishing Company, Incorporated (2008)
20. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
21. 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)
22. 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)
23. 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)
24. Vilalta, R., Drissi, Y.: A perspective view and survey of meta-learning. Artificial Intelligence Review (2002)
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)