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

Learning Linear Composite Dispatch Rules for Scheduling

Case study for the job-shop problem

Helga Ingimundardottir · Thomas Philip Runarsson

Received: September 10, 2015/ Accepted: date

Abstract Instead of creating new dispatching rules in an ad-hoc manner, this study gives a framework on how to analyse heuristics for scheduling problems. Before starting to create new composite priority 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 priority 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 sequential combinatorial problem.

Needs a rewrite

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

The problem is to learn new problem specific priority dispatching rules for scheduling. A subclass of scheduling problems is the job-shop scheduling problem (JSP), which is widely studied in operations research. JSP 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. 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.

Generally, job-shop is solved 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 become quite a time-consuming selection process for the heuristic designer, which any kind of automation would alleviate immensely. With meta heuristics one can use existing DRs, and use for example portfolio-based algorithm selection [33, 11], either based on a single instance or class of instances [43] to determine which DR to choose from. Implementing ant colony optimisation to select the best DR from a selection of nine DRs for JSP, experiments from [21] showed that the choice of DR do affect the results and that for all performance measures considered. They showed that it was better to have a all the DRs to choose from rather than just a single DR at a time.

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, dispatching rules, although the techniques developed could be adapted to improvement heuristics also.

Genetic algorithms (GA) are one of the most widely used approaches in JSP literature [32]. However, in that case an extensive number of schedules need to be evaluated, and even for low dimensional JSP that can quickly become computationally infeasible. GAs can be used directly on schedules [6, 7, 41, 19, 1, 26], however, in that case there are many concerns that need to be dealt with. To begin with there are nine encoding schemes for representing the schedules [6]. In addition, there has to be special care taken when applying cross-over and mutation operators in order for the schedules, now in the role of 'chromosomes,' to still remain feasible.

Another approach is to apply GAs indirectly to JSP, via dispatching rules, i.e., Dispatching Rules Based Genetic Algorithms (DRGA) [42, 8, 28] where a solution is no longer a *proper* schedule but a *representation* of a schedule via applying certain dispatching rules consecutively. DRGA are a special case of *genetic programming* [22] which is the most predominant approach in hyper-heuristics is a framework of creating *new* heuristics from a set of predefined heuristics via GA optimisation [4].

A prevalent approach to solving JSP is to combine several relatively single priority dispatching rules such that they may benefit each other for a given problem space. The approach in [16], was to automate that selection, by translating dispatching rules into measurable features and optimising what their contribution should be via evolutionary search. The framework is straight forward and easy to implement and showed promising and robust results, as models were trained on a lower dimension, and validated on higher dimension. Moreover, [16] showed that the choice of objective function for evolutionary search is worth investigating. Since the optimisation is based on minimising the expected mean of the fitness function over a large set of problem instances, which can vary within. Then normalising the objective function can stabilise the optimisation process away from local minima.

By applying genetic programming (GP) on a terminal set of job-attributes for flexible job-shop,[1] [40] optimised a multi-objective job-shop (transformed into a single objective by linearly combining their objectives) with promising results compared to the benchmarks DRs from the literature. The main drawback, is that the rules from their GP framework is quite complex, and difficult to contrive a meaningful description to a layman. In fact, [14] revisited the experiments from [40] for dynamic job-shop,[2] and tested it against some single priority dispatching rules, and found that it only slightly outperformed ERD-rule, and was beat by SPT-rule. The reason behind this staggering change in performance, is most likely due to the choice of objective function, and the underlying problem spaces that were used in training. It's argued that the randomly generated problem instances aren't a proper representative for real-world long-term job-shop applications, e.g., by the narrow choice of release times, yielding schedules that are overloading in the beginning phases.

A novel iterative dispatching rules that were evolved with GP for JSP, [28] learned 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.

Adopting a two-stage hyper-heuristic approach to generate a set of machine-specific DRs for dynamic job-shop, [31] used GP to evolve composite priority dispatching rules (CDR) from basic attributes, along with evolutionary algorithm to assign a CDR to a specific machine. The problem space consists of job-shops in semiconductor manufacturing, with additional shop constraints, as machines are grouped to similar work centres, which can have different set-up time, workload, etc. In fact, the GP emphasised on efficiently dispatching on the work centres with set-up requirements and batching capabilities, which are rules that are non-trivial to determine manually.

In the field of Artificial Intelligence, [26] point out that despite their 'intelligent' solutions, the effectiveness of finding the optimum has been rather limited. This is the general case for GAs, as they are not well suited for fine-tuning around the optimum [7]. However, combined with local-search methodologies, they can be improved upon significantly, as [26] showed with the use of a hybrid method using Genetic Algorithms (GA) and Tabu Search (TS). Therefore, getting the best of both worlds, namely, the diverse global search obtained from GA and being complemented with

[1] Flexible job-shop allows jobs to be processed on different machines, i.e., the problem has the additional complexity of making a routing decision of operations to machines.

[2] Job-shop is considered dynamic when the processing times are not known prior to their arrival.

the intensified local search capabilities of TS. Now, hybridisation of global and local methodologies is non-trivial. In general combination of the two improves performance, however, they often come at a great computational cost.

Using improvement heuristics, [45] studied space shuttle payload processing by using reinforcement learning (RL), in particular, temporal difference learning. Starting with a relaxed problem, each job was scheduled as early as its temporal partial order would permit, there by initially ignoring any resource constraints on the machines, yielding the schedule's critical path. Then the schedule would be repaired so the resource constraints were satisfied in the minimum amount of iterations.

Meta learning can be very fruitful in RL, as experiments from [20] 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.

Using case based reasoning for timetable scheduling, training data in [3] is guided by the two best heuristics in the literature. They point out that in order for their framework to be successful, problem features need to be sufficiently explanatory and training data need to be selected carefully so they can suggest the appropriate solution for a specific range of new cases. Stressing the importance of meaningful feature selection.

A recent editorial of the state-of-the-art approaches in advanced dispatching rules for large-scale manufacturing systems by [5] 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 of automated discovery of DR was also emphasised by [27]. Several of successful implementations in the field of semiconductor wafer fabrication facilities are discussed, however, this sort of investigation is still in its infancy.

The alternative to hand-crafting heuristics, is to implement an automatic way of learning heuristics using a data driven approach. Data can be generated using a known heuristic, such an approach is taken in [23] for single-machine job-shop where a LPT-rule is applied. Afterwards, a decision tree is used to create a dispatching rule with similar logic. However, this method cannot outperform the original LPT-rule used to guide the search. This drawback is confronted in [25,38,29] by using an optimal scheduler, computed off-line. The optimal solutions are used as training data and a decision tree learning algorithm applied as before. Preferring simple to complex models, the resulting dispatching rules gave significantly better schedules than using popular heuristics in that field, and a lower worst-case factor from optimality.

The focus on this study is better understanding of *how* and *when* dispatching rules work in general, in order to mediate the set-up for the learning problem. For this reason, we propose a framework for learning the indicators of optimal solutions, such as done by [29], as an in-depth analysis of a expert policy gives a benchmark of what is theoretically possible to learn.

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.

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 describes the main attributes for job-shop, and goes into how to create schedules with dispatching rules. Section 4 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 single priority dispatching rules is investigated. With those guidelines, Section 5 goes into detail how to create meaningful composite priority dispatching rules using preference learning, focusing on how to compare operations and collect training data with the importance of good feature selection and the polysemy of how to report accuracy. The paper finally concludes in Section 6 with discussion and conclusions.

2 Job-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 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.

Each job J_j has an indivisible operation time (or cost) on machine M_a, p_{ja}, which is assumed to be integral and finite. Starting time of job J_j on machine M_a is denoted $x_s(j,a)$ and its end time is denoted $x_e(j,a)$ where,

$$x_e(j,a) := x_s(j,a) + p_{ja} \tag{1}$$

Each job J_j has a specified processing order through the machines, it is a permutation vector, $\boldsymbol{\sigma}_j$, of $\{1,..,m\}$, representing a job J_j can be processed on $M_{\boldsymbol{\sigma}_j(a)}$ only after it has been completely processed on $M_{\boldsymbol{\sigma}_j(a-1)}$, i.e.,

$$x_s(j, \boldsymbol{\sigma}_j(a)) \geq x_e(j, \boldsymbol{\sigma}_j(a-1)) \tag{2}$$

for all $J_j \in \mathscr{J}$ and $a \in \{2,..,m\}$. Note, that each job can have its own distinctive flow pattern through the machines, which is independent of the other jobs. However, in the case that all jobs share the same *fixed* permutation route, referred to as flow-shop (FSP). A commonly used subclass of FSP in the literature is permutation flow-shop, which has the added constraint that the processing order of the jobs on the machines must be identical as well, i.e., no passing of jobs allowed [39].

The disjunctive condition that each machine can handle at most one job at a time is the following,

$$x_s(j,a) \geq x_e(j',a) \quad \text{or} \quad x_s(j',a) \geq x_e(j,a) \tag{3}$$

for all $J_j, J_{j'} \in \mathscr{J}$, $J_j \neq J_{j'}$ and $M_a \in \mathscr{M}$.

The objective function is to minimise its maximum completion times for all tasks, commonly referred to as the makespan, $C_{\max}$, which is defined as follows,

$$C_{\max} := \max\left\{x_e(j, \boldsymbol{\sigma}_j(m)) \;\middle|\; J_j \in \mathscr{J}\right\}. \tag{4}$$

This family of scheduling problems is denoted by $J||C_{\max}$ [32]. Additional constraints commonly considered are job release-dates and due-dates or sequence dependent set-up times, however, these will not be considered here.

In order to find an optimal (or near optimal) solution for scheduling problems one could either use exact methods or heuristics methods. Exact methods guarantee an optimal solution. However, job-shop scheduling is strongly NP-hard [10]. Any exact algorithm generally suffers from the curse of dimensionality, which impedes the application in finding the global optimum in a reasonable amount of time. Using a state-of-the-art software for solving scheduling problems, such as LiSA (A Library of Scheduling Algorithms) [2], which includes a specialised version of branch and bound that manages to find optimums for job-shop problems of up to 14×14 [37]. However, problems that are of greater size, become intractable. Heuristics are generally more time efficient but do not necessarily attain the global optimum. Therefore, job-shop has the reputation of being notoriously difficult to solve. As a result, it's been widely studied in deterministic scheduling theory and its class of problems has been tested on a plethora of different solution methodologies from various research fields [26], all from simple and straight forward dispatching rules to highly sophisticated frameworks.

3 Priority Dispatching Rules

Priority dispatching rules determine, from a list of incomplete jobs, $\mathscr{L}$, which job should be dispatched next. This process is illustrated in Figure 1, where an example of a temporal partial schedule of six-jobs scheduled on five-machines is illustrated. 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. The object is to keep this value as small as possible once all operations are complete. As shown in the example there are 15 operations already scheduled. The *sequence* of dispatches used to create this partial schedule is,

$$\boldsymbol{\chi} = (J_3, J_3, J_3, J_3, J_4, J_4, J_5, J_1, J_1, J_2, J_4, J_6, J_4, J_5, J_3) \tag{5}$$

and refers to the sequential ordering of job dispatches to machines, i.e., (j,a); the collective set of allocated jobs 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 from the available jobs to be scheduled. In our example the available jobs is given by the job-list $\mathscr{L}^{(k)} = \{J_1, J_2, J_4, J_5, J_6\}$ with the five potential jobs to be dispatched at step $k = 16$ (note that J_3 is completed).

Deciding which job to dispatch is, however, not sufficient, one must also know where to place it. In order to build tight schedules it is sensible to place a job as soon as it becomes available and such that the machine idle time is minimal, i.e., schedules are non-delay. There may also be a number of different options for such a placement. In Fig. 1 one observes 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 time in which machine M_a is idle between consecutive jobs J_j and $J_{j'}$ is called idle time, or slack,

$$s(a,j) := x_s(j,a) - x_e(j',a) \tag{6}$$

where J_j is the immediate successor of $J_{j'}$ on M_a.

Construction heuristics are designed in such a way that it limits the search space in a logical manner, respecting not to exclude the optimum. Here, the construction heuristic is to schedule the dispatches as closely together as possible, i.e., minimise the schedule's idle time. More specifically, once an operation (j,a) has been chosen from the job-list, $\mathscr{L}$, by some dispatching rule, it can placed immediately after (but not prior) $x_e(j,\boldsymbol{\sigma}_j(a-1))$ on machine M_a due to constraint Eq. (2). However, to guarantee that constraint Eq. (3) is not violated, idle times M_a are inspected, as they create flow time which J_j can occupy. Bearing in mind that J_j release time is $x_e(j,\boldsymbol{\sigma}_j(a-1))$ one cannot implement Eq. (6) directly, instead it has to be updated as follows,

$$\tilde{s}(a,j') := x_s(j'',a) - \max\{x_e(j',a), x_e(j,\boldsymbol{\sigma}_j(a-1))\} \tag{7}$$

for all already dispatched jobs $J_{j'}, J_{j''} \in \mathscr{J}_a$ where $J_{j''}$ is $J_{j'}$ successor on M_a. Since pre-emption is not allowed, the only applicable slots are whose idle time can process the entire operation, i.e.,

$$\tilde{S}_{ja} := \left\{ J_{j'} \in \mathscr{J}_a \;\middle|\; \tilde{s}(a,j') \geq p_{ja} \right\}. \tag{8}$$

The placement rule applied will decide where to place the job and is an intrinsic heuristic of the construction heuristic, chosen independently of the priority dispatching rule applied. Different placement rules could be considered for selecting a slot from Eq. (8), e.g., if the main concern were to utilise the slot space, then choosing the slot with the smallest idle time would yield a closer-fitted schedule and leaving greater idle times undiminished for subsequent dispatches on M_a. In our experiments cases were discovered where such a placement can rule out the possibility of constructing optimal solutions. This problem, however, did not occur when jobs are simply placed as early as possible, which is beneficial for subsequent dispatches for J_j. For this reason it will be the placement rule applied here.

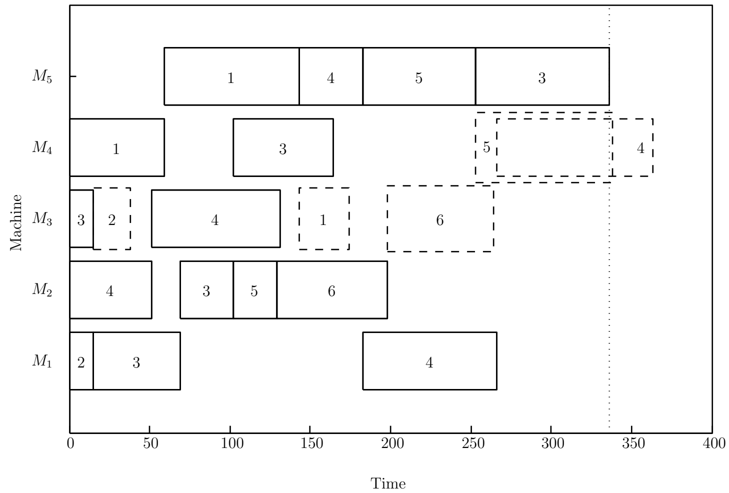

Fig. 1 Gantt chart of a partial JSP schedule after 15 dispatches: Solid and dashed boxes represent $\boldsymbol{\chi}$ and $\mathscr{L}^{(16)}$, respectively. Current $C_{\max}$ denoted as dotted line.

Priority dispatching rules will use attributes of operations, such as processing time, in order to determine the job with the highest priority. 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 have the highest priority. 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 alternatively the least work remaining (LWR). A survey of more than 100 of such rules are presented in [30]. However, the reader is referred to an in-depth survey for simple or *single priority dispatching rule* (SDR) by [13]. The SDRs assign an index to each job in the job-list and is generally only based on a few attributes and simple mathematical operations.

Designing priority dispatching rules requires recognizing the important attributes of the partial schedules needed to create a reasonable 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 each possible operation are given in Table 1, where the set of machines already dispatched for J_j is $\mathscr{M}_j \subset \mathscr{M}$, and similarly, M_a has already had the jobs $\mathscr{J}_a \subset \mathscr{J}$ previously dispatched. The attributes of particular interest were obtained by inspecting the aforementioned SDRs. Attributes ϕ_1-ϕ_8 and ϕ_9-ϕ_{16} are job-related and machine-related, respectively. In fact, [31] note that in the current literature, there is a lack of global perspective in the attribute space, as omitting them won't address the possible negative impact an operation (j,a) might have on other machines at a later time, it is for that reason we consider attributes such as ϕ_{13}-ϕ_{15}, that are slack related and are a means of indicating the current quality of the schedule. All of the attributes, $\boldsymbol{\phi}$, vary throughout the scheduling process, w.r.t. operation belonging to the same time step k, with the exception of ϕ_6 and ϕ_{10} which are static for a given problem instance but varying for each J_j and M_a, respectively.

Table 1 Attribute space $\mathscr{A}$ for JSP where job J_j on machine M_a given the resulting temporal schedule after operation (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	$x_e(j,a)$	endTime
ϕ_4	job arrival time	$x_e(j,a-1)$	arrival
ϕ_5	time job had to wait	$x_s(j,a)-x_e(j,a-1)$	wait
ϕ_6	total processing time for job	$\sum_{a\in\mathscr{M}} p_{ja}$	jobTotProcTime
ϕ_7	total work remaining for job	$\sum_{a'\in\mathscr{M}\setminus\mathscr{M}_j} p_{ja'}$	jobWrm
ϕ_8	number of assigned operations for job	$\lvert\mathscr{M}_j\rvert$	jobOps
	machine related		
ϕ_9	when machine is next free	$\max_{j'\in\mathscr{J}_a}\{x_e(j',a)\}$	macFree
ϕ_{10}	total processing time for machine	$\sum_{j\in\mathscr{J}} p_{ja}$	macTotProcTime
ϕ_{11}	total work remaining for machine	$\sum_{j'\in\mathscr{J}\setminus\mathscr{J}_a} p_{j'a}$	macWrm
ϕ_{12}	number of assigned operations for machine	$\lvert\mathscr{J}_a\rvert$	macOps
ϕ_{13}	change in idle time by assignment	$\Delta s(a,j)$	reducedSlack
ϕ_{14}	total idle time for machine	$\sum_{j'\in\mathscr{J}_a} s(a,j')$	macSlack
ϕ_{15}	total idle time for all machines	$\sum_{a'\in\mathscr{M}}\sum_{j'\in\mathscr{J}_{a'}} s(a',j')$	allSlack
ϕ_{16}	current makespan	$\max_{(j',a')\in\mathscr{J}\times\mathscr{M}_{j'}}\{x_f(j',a')\}$	makespan

Priority dispatching rules are attractive since they are relatively easy to implement, fast and find reasonable schedules. In addition, they are relatively easy to interpret, which makes them desirable for the end-user. However, they can also fail unpredictably. A careful combination of dispatching rules have been shown to perform significantly better [18]. These are referred to as *composite priority dispatching rules* (CDR), where the priority ranking is an expression of several dispatching rules. CDRs deal with a greater number of more complicated functions (or features) constructed from the schedules attributes. In short, a CDR is a combination of several DRs. For instance let π be a CDR comprised of d DRs, then the index I for $J_j \in \mathscr{L}^{(k)}$ using π is,

$$I_j^{\pi} = \sum_{i=1}^{d} w_i \pi_i(\boldsymbol{\chi}^j) \tag{9}$$

where $w_i > 0$ and $\sum_{i=0}^{d} w_i = 1$ with w_i giving the weight of the influence of π_i (which could be a SDR or another CDR) to π. Note, each π_i is function of J_j's attributes from the current sequence $\boldsymbol{\chi}$, where $\boldsymbol{\chi}^j$ implies that J_j was the latest dispatch, i.e., the partial schedule given $\chi_k = J_j$.

At each time step k, an operation is dispatched which has the highest priority. If there is a tie, some other priority measure is used. Generally the dispatching rules are static during the entire scheduling process, however, ties could also be broken randomly (RND).

Investigating 11 SDRs for JSP, [24] created a pool of 33 CDRs that strongly outperformed the ones they were based on, by using multi-contextual functions based on either on job waiting time or machine idle time (similar to ϕ_5 and ϕ_{14} in Table 1), i.e., the CDRs are a combination of those two key attributes and then the SDRs. However, there are no combinations of the basic SDRs explored, only those two attributes. Similarly, using priority rules to combine 12 existing DRs from the literature, [44] had 48 CDR 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.

The composite priority dispatching rule presented in Eq. (9) can be considered as a special case of a the following general linear value function,

$$\pi(\boldsymbol{\chi}^j) = \sum_{i=1}^{d} w_i \phi_i(\boldsymbol{\chi}^j). \tag{10}$$

when $\pi_i(\cdot) = \phi_i(\cdot)$, i.e., a composite function of the features from Table 1. Finally, the job to be dispatched, J_{j^*}, corresponds to the one with the highest value, i.e.,

$$J_{j^*} = \underset{J_j \in \mathscr{L}}{\operatorname{argmax}} \; \pi(\boldsymbol{\chi}^j) \tag{11}$$

Similarly, single priority dispatching rules may be described by this linear model. For instance, let all $w_i = 0$, but with following exceptions: $w_1 = -1$ for SPT, $w_1 = +1$ for LPT, $w_7 = -1$ for LWR and $w_7 = +1$ for MWR. Generally the weights $\mathbf{w}$ 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 Performance Analysis of Priority Dispatching Rules

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 the construction of such solutions. For this, we follow optimal solutions, obtained by using a commercial software package [12], and inspect the probability of SDRs being optimal, which serves as an indicator of how hard it is to put our objective up as a machine learning problem. However, we must also take into consideration the end-goal, which is minimising deviation from optimality, ρ, because it's its relationship to stepwise optimality is not fully understood.

In this section we will describe concerns that must be addressed when learning new priority dispatching rules. At the same time we will describe the experimental set-up used in our study.

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

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

4.1 Problem Instances

The class of problem instances used in our studies is the job-shop scheduling problem described in Section 2. Each instance will have different processing times, machine ordering and dimensions. Each instance will therefore create different challenges for a priority dispatching rule. Dispatching rules learned will be customized for the problems used for their training. For real world application using historical data would be most appropriate. The aim would be to learn a dispatching rule that works well on average for a given distribution of problem instances. To illustrate the performance difference of priority dispatching rules on different problem distributions, within the same class of problems, consider the following three cases. Problem instances for JSP are generated stochastically by fixing the number of jobs and machines to ten. A discrete processing time is sampled independently from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e., $\mathbf{p} \sim \mathscr{U}(u_1, u_2)$. The machine order is a random permutation of all of the machines in the job-shop. 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 $I = [45, 55]$. These instances are referred to as random and random-narrow, respectively. In addition we consider the case where the machine order is fixed and the same for all jobs, i.e. $\sigma = \{1, \ldots, m\}$ where $\mathbf{p} \sim \mathscr{U}(1, 99)$. These jobs are denoted by $\mathscr{P}_{f.rnd}^{n\times m}$ and is analogous to $\mathscr{P}_{j.rnd}^{n\times m}$.

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 π under inspection by $C_{\max}^{\pi}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho = \frac{C_{\max}^{\pi} - C_{\max}^{\text{opt}}}{C_{\max}^{\text{opt}}} \cdot 100\% \tag{12}$$

which indicates the percentage relative deviation from optimality. Figure 2 depicts the box-plot for Eq. (12) when using the SDRs from Section 3 for all of the problem spaces from Table 2. These box-plots show the difference in performance of the various SDRs. The MWR performs on average the best on the $\mathscr{P}_{j.rnd}^{n\times m}$ and $\mathscr{P}_{j.rndn}^{n\times m}$ problems instances, whereas for $\mathscr{P}_{f.rnd}^{n\times m}$ is is LWR that performs best. It is also interesting to observe that all but the MWR perform statistically worse than random job dispatching on the $\mathscr{P}_{j.rnd}^{n\times m}$ and $\mathscr{P}_{j.rndn}^{n\times m}$ problems instances.

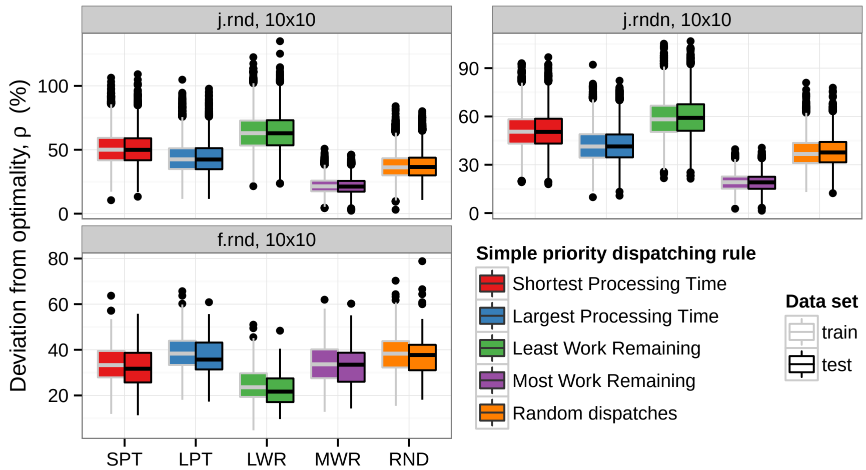

Fig. 2 Box-plot for deviation from optimality, ρ, (%) for SDRs

4.2 Reconstructing optimal solutions

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 their machine order. Unfinished jobs are dispatched one at a time according to some heuristic. After each dispatch[3] 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 cannot infer that choosing J_1 is better and J_2 is worse (or vice versa) since they can both yield the same solution. Furthermore, there may 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 (although given the same partial initial sequence) can result in very different complete schedules but can still achieve the same makespan.

The redundancy in building optimal solutions using dispatching rules means that many different dispatches may yield an optimal solution to the problem instance. The probability that a job chosen by a SDR yields an optimal makespan, on a step-by-step basis, is depicted in Fig. 3. These probabilities vary quite a bit between the different

[3] Dispatch and time step are used interchangeably.

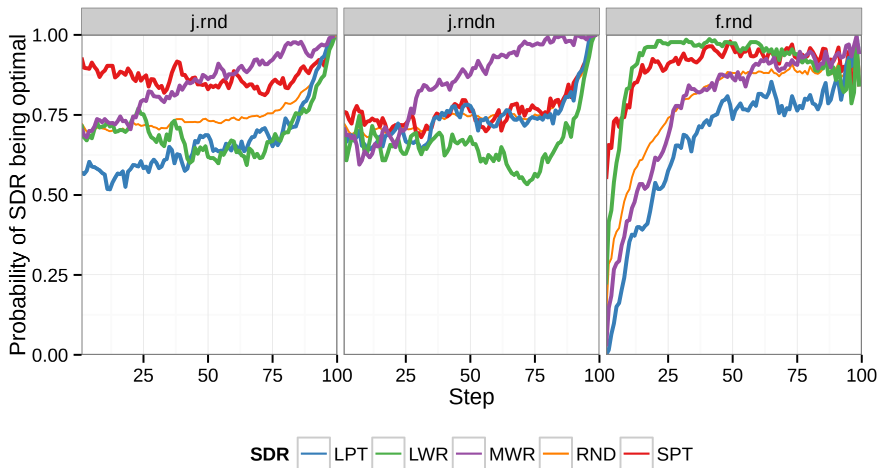

Fig. 3 Probability of SDR being optimal

problem instances distributions studied. From Fig. 3 one observed that MWR has a higher probability than random guessing, in choosing a dispatch which may result in an optimal schedule. This is especially true towards the end of the schedule building process. Similarly, the LWR chooses dispatches resulting in optimal schedules with a higher probability. This would appear to be support the idea that the higher the probability of dispatching jobs that may lead to an optimal schedule, the better the SDRs performance, as illustrated by Fig. 2. However, there is a counter example. The SPT has a higher probability than random dispatching of selecting a jobs that may lead to an optimal solution. Nevertheless, the random dispatching performs better than SPT on problem instances $\mathscr{P}_{j.rnd}^{10\times10}$ and $\mathscr{P}_{j.rndn}^{10\times10}$.

Looking at Fig. 3, then $\mathscr{P}_{j.rnd}^{10\times10}$ has a relatively high probability (70% and above) of choosing an optimal job at random. However, it is imperative to keep making optimal decisions, because once off the optimal track the consequences are unknown. To demonstrate this Fig. 4 depicts mean worst and best case scenario of the resulting deviation from optimality, ρ, once off the optimal track. Note, that this is given that one makes *one* non-optimal dispatch. Generally, there will be more, and then the compound effects of making suboptimal decisions cumulate.

It is interesting to observe that for $\mathscr{P}_{j.rnd}^{10\times10}$ and $\mathscr{P}_{j.rndn}^{10\times10}$ making suboptimal decisions later impacts on the resulting makespan more than doing a mistake early. The opposite seems to be the case for $\mathscr{P}_{f.rnd}^{10\times10}$. In this case it is imperative to make good decisions right from the start. This is due to the major structural differences between JSP and FSP, namely the latter having a homogeneous machine ordering, constricting the solution immensely.

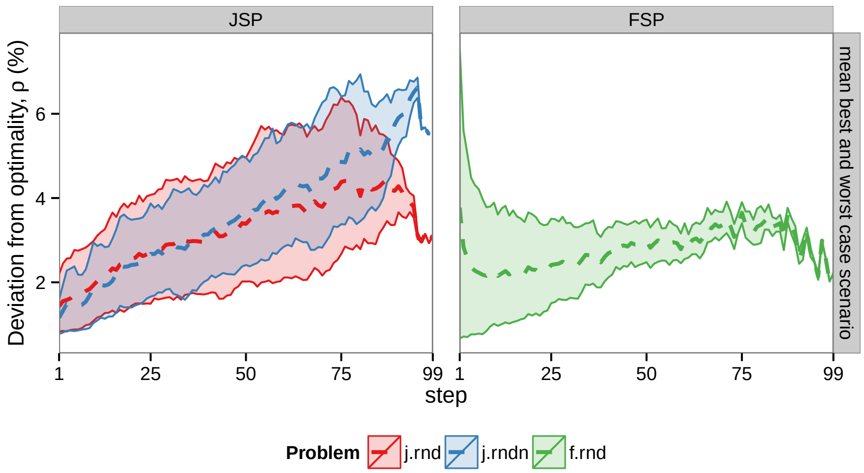

Fig. 4 Mean deviation from optimality, ρ, (%), for best (lower bound) and worst (upper bound) case scenario of choosing suboptimal dispatch for $\mathscr{P}_{j.rnd}^{10\times10}$, $\mathscr{P}_{j.rndn}^{10\times10}$ and $\mathscr{P}_{f.rnd}^{10\times10}$

4.3 Blended dispatching rules

A naive approach to create a simple blended dispatching rule (BDR) would be to switch between SDRs at a predetermined time. Observing again Fig. 3, a presumably good BDR for $\mathscr{P}_{j.rnd}^{10\times10}$ would be to start with SPT and then switch over to MWR at around time step 40, where the SDRs change places in outperforming one another. A box-plot for ρ for the BDR compared with MWR and SPT is depicted in Fig. 5. This simple swap between SDRs does outperform the SPT, yet doesn't manage to gain the performance edge of MWR. Using SPT downgrades the performance of MWR.

add random dispatching on this figure, does it perform like random? No beter!

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. 3 show, the inherent structure that's already taking place, 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 5 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 $\mathscr{P}_{j.rnd}^{10\times10}$ and $\mathscr{P}_{j.rndn}^{10\times10}$ the fact remains that the schedules have diverged too far from what MWR would have been able to achieve on its own.

this last paragraph ends with the hope the CDRs will perform better than BDRs, and so motivate the next section, we also observe here that there will no be one SDR that fits all problem distributions and that custom build ones will be required for each instance distribution.

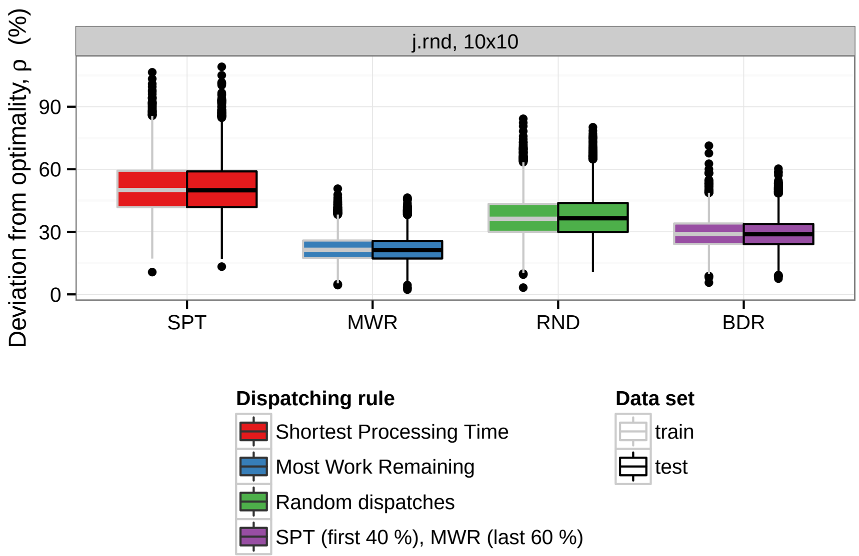

Fig. 5 Box-plot for deviation from optimality, ρ, (%) for BDR where SPT is applied for the first 40% of the dispatches, followed by MWR

5 Learning CDR

Section 4.3 demonstrated there is definitely something to be gained by trying out different combinations of DRs, 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 Preference Learning

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 [36] and in [15] for JSP, and given here for completeness.

The optimum makespan is known for each problem instance. At each time step k, a number of feature pair are created. Let $\boldsymbol{\phi}^o \in \mathbb{R}^d$ denote the post-decision state when dispatching $J_o \in \mathscr{O}^{(k)}$ corresponds to an optimal schedule being built. All post-decisions states corresponding to suboptimal dispatches, $J_s \in \mathscr{S}^{(k)}$, are denoted by $\boldsymbol{\phi}^s \in \mathbb{R}^d$. Note, $\mathscr{O}^{(k)} \cup \mathscr{S}^{(k)} = \mathscr{L}^{(k)}$, and $\mathscr{O}^{(k)} \cap \mathscr{S}^{(k)} = \emptyset$.

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 training data generation takes into consideration when there are multiple optimal solutions[4] to the same problem instance.

One could label which feature sets were considered optimal, $\boldsymbol{\psi}^o = \boldsymbol{\phi}^o - \boldsymbol{\phi}^s$, and suboptimal, $\boldsymbol{\psi}^s = \boldsymbol{\phi}^s - \boldsymbol{\phi}^o$ by $y_o = +1$ and $y_s = -1$ respectively. Then, the preference learning problem is specified by a set of preference pairs,

$$\Psi = \left\{ (\boldsymbol{\psi}^o, +1), (\boldsymbol{\psi}^s, -1) \;\middle|\; \forall (J_o, J_s) \in \mathscr{O}^{(k)} \times \mathscr{S}^{(k)} \right\}_{k=1}^{\ell} \subset \Phi \times Y \tag{13}$$

where $\Phi \subset \mathbb{R}^d$ is the training set of $d = 16$ features (cf. Table 1), $Y = \{+1, -1\}$ is the outcome space from job pairs, $J_o \in \mathscr{O}^{(k)}$ and $J_s \in \mathscr{S}^{(k)}$, for all dispatch steps k.

To summarise, each job is compared against another job of the job-list, $\mathscr{L}^{(k)}$, and if the makespan differs, i.e., $C_{\max}^{(s)} \gneqq C_{\max}^{(o)}$, an optimal/suboptimal pair is created. However, if the makespans are identical 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.

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

$$\boldsymbol{\chi}^i \succ \boldsymbol{\chi}^j \quad \Leftrightarrow \quad \pi(\boldsymbol{\chi}^i) > \pi(\boldsymbol{\chi}^j) \tag{14}$$

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,

$$\pi(\boldsymbol{\chi}^j) = \sum_{i=1}^{d} w_i \phi_i(\boldsymbol{\chi}^j) = \left\langle \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{\chi}^j) \right\rangle. \tag{15}$$

Logistic regression learns the optimal parameters $\mathbf{w}^* \in \mathbb{R}^d$. For this study, L2-regularized logistic regression from the LIBLINEAR package [9] without bias is used to learn the preference set Ψ, defined by Eq. (13). Hence, for each job on the job-list, $J_j \in \mathscr{L}$, let $\boldsymbol{\phi}^j := \boldsymbol{\phi}(\boldsymbol{\chi}^j)$ 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.,

$$J_{j^*} = \underset{J_j \in \mathscr{L}}{\operatorname{argmax}}\; \pi(\boldsymbol{\phi}^j) \tag{16}$$

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

Preliminary experiments for creating step-by-step model was done in [15] 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

[4] There can be 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.

the preference set, Ψ, is the union of the preference pairs across the ℓ dispatches, such as described in Eq. (13). This would amount to a substantial preference set, and for Ψ to be computationally feasible to learn, Ψ has to be reduced. For this several ranking strategies were explored in [17], 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, 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, for this study, which deals with 10×10 problem instances, the partial subsequent ranking becomes necessary, as full ranking is computationally infeasible due to its size. Furthermore, for the following experimental set up, the preference set was limited to $|\Psi| \leq 500,000$ by random sampling.

5.2 Feature Selection

The SDRs we've inspected so-far are based on two job-attributes from Table 1, namely: *i)* ϕ_1 for SPT and LPT, and *ii)* ϕ_7 for LWR and MWR, by choosing the lowest value for SPT and LWR, and highest value for LPT and MWR, i.e., the extremal values for those attributes. There is nothing that limits us to using just only these two attributes.

For this study we will consider all combinations of attributes using either one, two, three or all d of them, 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 attributes, 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, where Φ is limited to the predetermined feature combination. This was done with the software package from [9],[5] by training on the full preference set Ψ obtained from $N_{\text{train}} = 300$ problem instances following the framework set up in Section 5.1. Note, in order to report the validation accuracy, 20% ($N_{\text{val}} = 60$) of the training set was set aside for validation of reporting the accuracy.

5.3 Validation accuracy

As the preference set Ψ 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 Ψ, it's opted the validation accuracy is obtained in the same manner as done in Section 4.2 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 6 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

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

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.

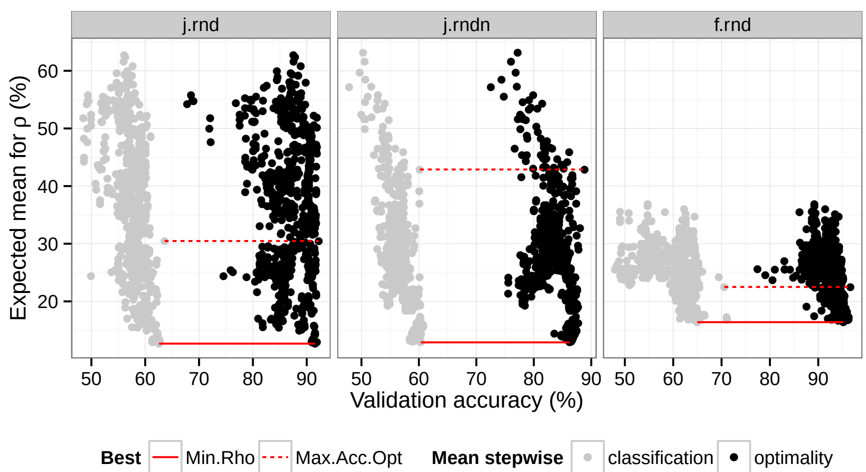

Fig. 6 Various methods of reporting validation accuracy for preference learning

5.4 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 against its corresponding validation accuracy in Fig. 7. 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.

Equation (10) showed how to interpret the linear preference models by their weights $\mathbf{w}$. Figure 8 depicts $\mathbf{w}$ 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^π defined by Eq. (9). These weights will now be explored further, along with testing whether models are statistically significant to one another, using a Kolmogorov-Smirnov test with $\alpha = 0.05$.

Table 3 Mean validation accuracy and mean expected deviation from optimality, ρ, for all CDR models on the Pareto front from Fig. 7.

Problem	PREF NrFeat.Model	Accuracy (%) Optimality	Classification	ρ (%)	Pareto
$\mathscr{P}_{j.rnd}^{10\times10}$	**3.524**	91.55	62.57	**12.67**	▲
	3.358	91.82	62.74	12.90	▲
	3.355	91.90	62.71	12.92	▲
	2.69	91.02	61.41	12.92	
	1.11	80.77	55.78	21.63	
	1.13	85.26	57.17	22.79	
	16.1	**92.24**	**63.61**	30.47	▲
	2.111	91.52	59.69	32.68	
	1.6	89.85	58.33	33.08	
	1.3	89.86	58.34	33.41	
	3.300	91.91	60.05	51.87	
$\mathscr{P}_{j.rndn}^{10\times10}$	**3.281**	86.24	60.34	**12.89**	▲
	3.231	86.52	58.92	12.98	▲
	3.222	86.69	58.86	13.23	▲
	2.68	86.19	59.27	13.34	
	3.223	86.73	58.80	13.44	▲
	3.528	86.84	59.49	13.61	▲
	2.52	86.47	59.16	13.65	
	2.73	86.55	59.26	13.67	
	3.159	86.88	58.87	13.91	▲
	3.263	86.95	59.20	14.06	▲
	3.162	86.92	58.97	14.06	▲
	2.51	86.65	58.90	14.06	
	3.147	87.18	58.88	14.29	▲
	3.148	87.45	59.24	14.79	▲
	2.75	87.11	**60.45**	15.30	
	3.418	87.75	59.57	16.22	▲
	1.13	86.22	58.04	19.21	
	2.91	87.12	60.17	19.48	
	3.139	87.81	59.09	29.00	▲
	3.237	88.07	59.40	32.69	▲
	16.1	**88.86**	60.17	42.88	▲
$\mathscr{P}_{f.rnd}^{10\times10}$	**3.539**	95.22	64.97	**16.40**	▲
	3.151	96.06	64.31	16.75	▲
	3.216	96.28	**71.12**	16.78	▲
	2.94	92.79	63.12	16.88	
	3.213	96.30	71.05	17.22	▲
	2.111	94.16	65.07	17.73	
	2.51	95.83	64.21	17.95	
	1.7	87.59	61.74	19.05	
	1.6	92.61	62.91	19.18	
	16.1	**96.67**	70.58	22.50	▲

For $\mathscr{P}_{j.rnd}^{10\times10}$ there is no statistical difference between models (2.69, 3.355, 3.358, 3.524), w.r.t. ρ and the latter three w.r.t. accuracy. These models are therefore equivalently 'best' for the problem space. As Fig. 8 shows, ϕ_3, ϕ_7 and ϕ_{11} are similar in value, and in the case of 3.358, then ϕ_9 has similar contribution as ϕ_3 for the other models. Which, as standalone models are 1.6 and 1.3, respectively, and yield equivalent mean ρ and accuracy. 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.[6] Most likely, the equivalence of these features is indicating that the schedules are rarely able to dispatch in earlier slots, i.e., $\phi_3 \approx \phi_9$.

In addition, (2.111, 3.300) and (16.1, 3.355) are statistically insignificant w.r.t. validation accuracy for $\mathscr{P}^{10\times10}_{j.rnd}$. However, they have considerable performance difference w.r.t. ρ ($\Delta\rho \approx 18\%$). So even looking at stepwise optimality by itself is very fickle, because slight variations can be quite dramatic to the end result.

The solutions on the Pareto front for $\mathscr{P}^{10\times10}_{j.rndn}$ are quite a few models with no (or minimal) statistical difference w.r.t. ρ, and considerably more w.r.t. validation accuracy. Most notably are (3.281, 2.73, 2.75, 1.13), (note, first one has the lowest mean ρ) which are all statistically insignificant w.r.t. validation accuracy yet none w.r.t. ρ, with difference up to $\Delta\rho = 6.32\%$.

For $\mathscr{P}^{10\times10}_{f.rnd}$ almost all models are statistically different w.r.t. ρ, only exception is (1.6, 1.7). Although, w.r.t. validation accuracy, there are a few equivalent models, namely, (3.151, 2.51), (2.94, 1.6) and (3.216, 3.213, 16.1), with 1.2%, 2.3% and 5.75% difference in mean ρ, respectively.

It's interesting to inspect the full model for $\mathscr{P}^{10\times10}_{f.rnd}$, 16.1. Despite having similar contributions, yet statistically significantly different, as all the active features as (3.213, 3.216), then the (slight) interference from of other features, hinders the full model from achieving a low ρ. Only considering ϕ_8 and ϕ_{12} with either ϕ_3 and ϕ_9, boosts performance by 5.28% and 5.72%, respectively. Thereby stressing the importance of feature selection, to steer clear of over-fitting. Note, unlike $\mathscr{P}^{10\times10}_{j.rnd}$, now ϕ_3 differs from ϕ_9, indicating that there are some slots created, which could be better utilised. Moreover, looking at model 2.111 for $\mathscr{P}^{10\times10}_{f.rnd}$, which has similar contributions as the best model, 3.539. Then introducing a third feature, ϕ_{11}, is the key to the success of the CDR, with a boost of ρ performance by 1.33%.

Note, for both $\mathscr{P}^{10\times10}_{j.rnd}$ and $\mathscr{P}^{10\times10}_{j.rndn}$, model 1.13 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. 8). Revisiting Eq. (10), we observe that this implies the learning model was able to discover MWR as one of the Pareto solutions, and as is expected, there is no statistical difference to between 1.13 and MWR.

As one can see from Fig. 7, 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 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. 9, again using probability of randomly guessing an optimal move from Fig. 3 (denoted RND) as a benchmark. As one can see for both $\mathscr{P}^{10\times10}_{j.rnd}$ and $\mathscr{P}^{10\times10}_{j.rndn}$, despite having a higher mean validation accuracy overall, the probabil-

[6] Note, $\phi_3 \leq \phi_9$, where $\phi_3 = \phi_9$ when J_j is the latest job on M_a, otherwise J_j is placed in a previously created slot on M_a.

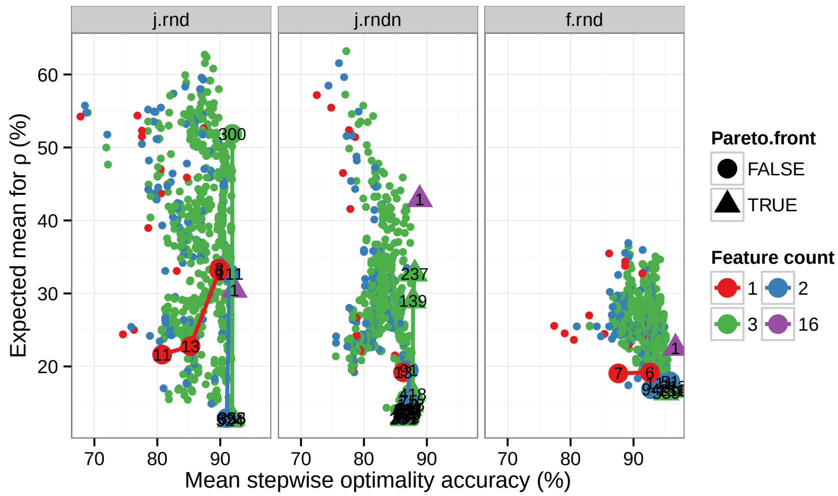

Fig. 7 Scatter plot for validation accuracy (%) against its corresponding mean expected ρ (%) for all 697 linear models, based on either one, two, three or all d combinations of features. Pareto fronts for each active feature count based on maximum validation accuracy and minimum mean expected ρ (%), and labelled with their model ID. Moreover, actual Pareto front over all models is marked with triangles.

ities 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.[7] 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 5.3 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 4.2 and compare the best CDR models, a box-plot for ρ is depicted in Fig. 10. 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] It's almost illegible to notice this shift directly from Fig. 9, as the difference between the two best models is oscillating up to only 3% at any given step. In fact $\mathscr{P}^{10\times10}_{j.rndn}$ has the most clear difference w.r.t. classification accuracy of indicating when a minimum ρ model excels at choosing the preferred move.

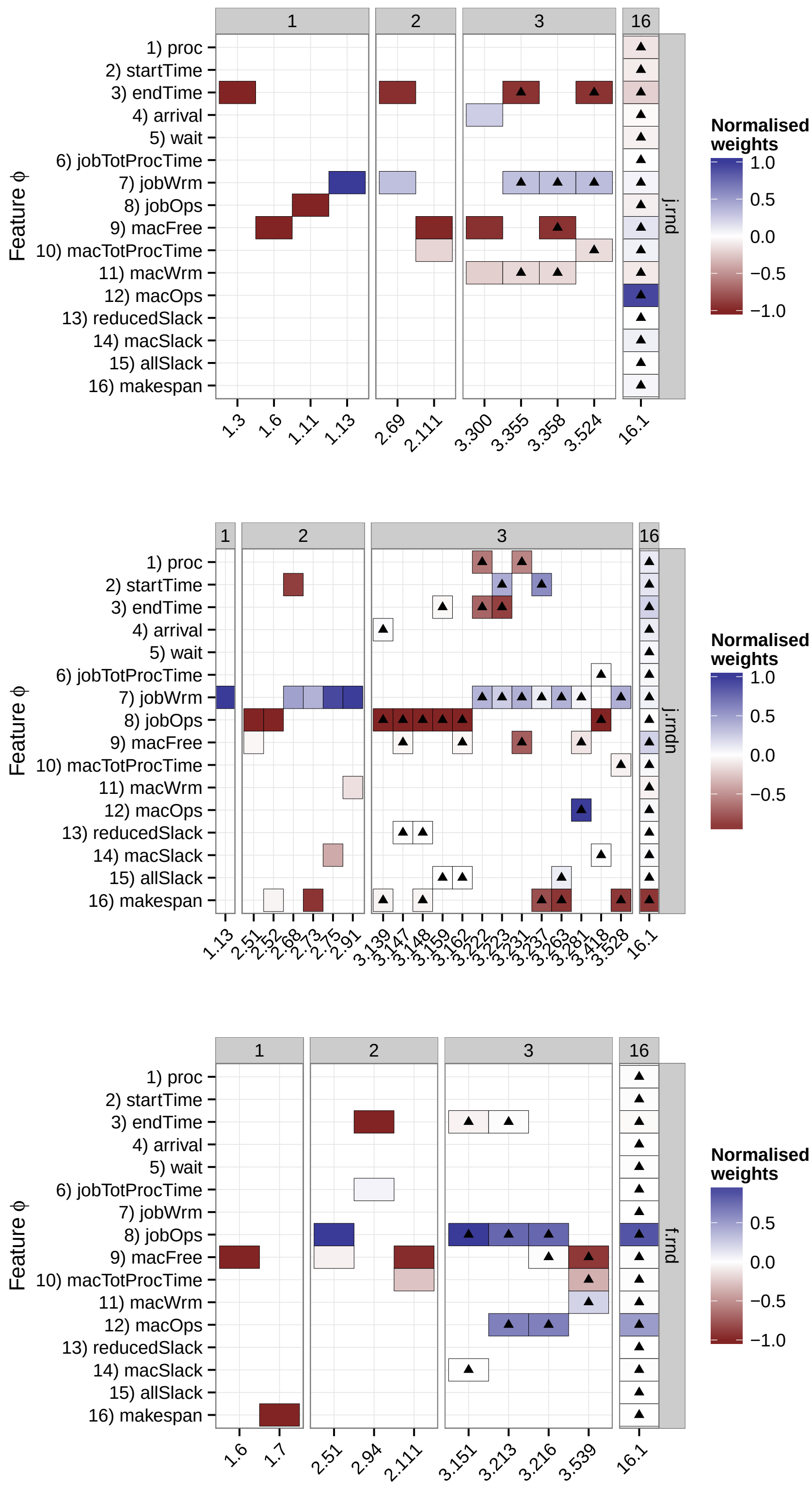

Fig. 8 Normalised weights for CDR models from Table 3, models are grouped w.r.t. its dimensionality, d. Note, a triangle indicates a solution on the Pareto front.

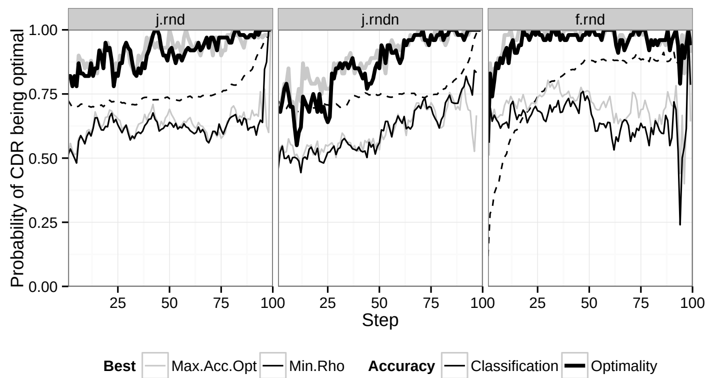

Fig. 9 Probability of choosing optimal move for models corresponding to highest mean validation accuracy (grey) and lowest mean deviation from optimality, ρ, (black) compared to the baseline of probability of choosing an optimal move at random (dashed).

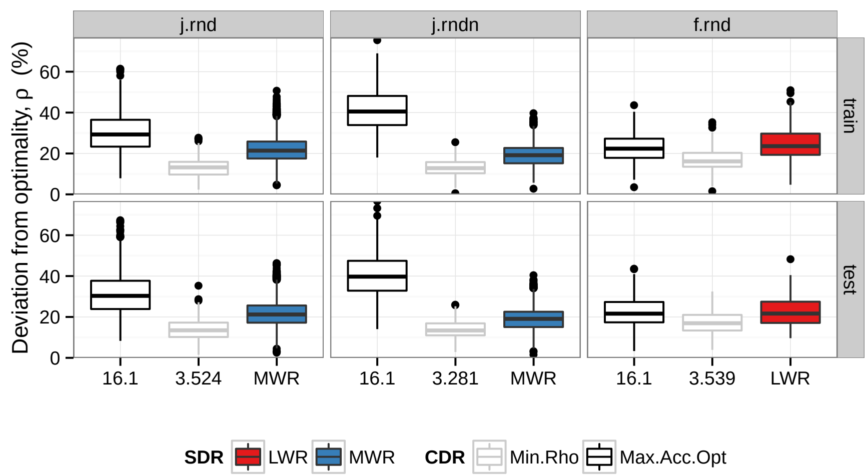

Fig. 10 Box-plot for deviation from optimality, ρ, (%) for the best CDR models (cf. Table 3) and compared against the best SDRs from Section 4.2, both for training and test sets.

6 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 priority dispatching rules that are likely to be successful.

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 5 was limited to up to three active features, as the full feature set was clearly suboptimal 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 priority 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 5.4 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 suboptimality. 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. 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.

Could discuss new sampling strategies, e.g., proportional to best/worst case, optimality, etc. – have done some experiments, but not clear what strategy is best, so only equal probability reported

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 [34, 35], 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 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.

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.

Not done, but possible

References

1. Ak, B., Koc, E.: A Guide for Genetic Algorithm Based on Parallel Machine Scheduling and Flexible Job-Shop Scheduling. Procedia - Social and Behavioral Sciences **62**, 817–823 (2012)
2. Andresen, M., Engelhardt, F., Werner, F.: LiSA - A Library of Scheduling Algorithms (version 3.0) [software] (2010). URL `http://www.math.ovgu.de/Lisa.html`
3. Burke, E., Petrovic, S., Qu, R.: Case-based heuristic selection for timetabling problems. Journal of Scheduling **9**, 115–132 (2006)
4. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (2013)
5. Chen, T., Rajendran, C., Wu, C.W.: Advanced dispatching rules for large-scale manufacturing systems. The International Journal of Advanced Manufacturing Technology (2013)
6. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithmsI. Representation. Computers & Industrial Engineering **30**(4), 983–997 (1996)
7. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Computers & Industrial Engineering **36**(2), 343–364 (1999)
8. Dhingra, A., Chandna, P.: A bi-criteria M-machine SDST flow shop scheduling using modified heuristic genetic algorithm. International Journal of Engineering, Science and Technology **2**(5), 216–225 (2010)
9. 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)
10. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
11. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2), 43–62 (2001)
12. Gurobi Optimization, Inc.: Gurobi optimization (version 6.0.0) [software] (2014). URL `http://www.gurobi.com/`
13. Haupt, R.: A survey of priority rule-based scheduling. OR Spectrum **11**, 3–16 (1989)
14. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. GECCO '10: Proceedings of the 12th annual conference on Genetic and evolutionary computation pp. 257–264 (2010)
15. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: C.A. Coello (ed.) Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 6683, pp. 263–277. Springer Berlin Heidelberg (2011). DOI 10.1007/978-3-642-25566-3_20. URL `http://dx.doi.org/10.1007/978-3-642-25566-3_20`

16. Ingimundardottir, H., Runarsson, T.P.: Evolutionary learning of weighted linear composite dispatching rules for scheduling. In: International Conference on Evolutionary Computation Theory and Applications (ECTA). SCITEPRESS (2014)
17. Ingimundardttir, H., Philip Rnarsson, T.: Generating training data for learning linear composite dispatching rules for scheduling. In: C. Dhaenens, L. Jourdan, M.E. Marmion (eds.) Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 8994, pp. 236–248. Springer International Publishing (2015). DOI 10.1007/978-3-319-19084-6_22. URL `http://dx.doi.org/10.1007/978-3-319-19084-6_22`
18. 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)
19. Qing-dao-er ji, R., Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research **39**(10), 2291–2299 (2012)
20. Kalyanakrishnan, S., Stone, P.: Characterizing reinforcement learning methods through parameterized learning problems. Machine Learning **84**(1-2), 205–247 (2011)
21. 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)
22. Koza, J.R., Poli, R.: Genetic programming. In: E. Burke, G. Kendal (eds.) Introductory Tutorials in Optimization and Decision Support Techniques, chap. 5. Springer (2005)
23. Li, X., Olafsson, S.: Discovering dispatching rules using data mining. Journal of Scheduling **8**, 515–527 (2005)
24. Lu, M.S., Romanowski, R.: Multicontextual dispatching rules for job shops with dynamic job arrival. The International Journal of Advanced Manufacturing Technology (2013)
25. Malik, A.M., Russell, T., Chase, M., Beek, P.: Learning heuristics for basic block instruction scheduling. Journal of Heuristics **14**(6), 549–569 (2008)
26. Meeran, S., Morshed, M.: A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. Journal of intelligent manufacturing **23**(4), 1063–1078 (2012)
27. 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)
28. 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)
29. 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)
30. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Operations Research **25**(1), 45–61 (1977)
31. Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B.: Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. International Journal of Production Economics **145**(1), 67–77 (2013)
32. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3 edn. Springer Publishing Company, Incorporated (2008)
33. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
34. 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)
35. 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)
36. 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)
37. Runarsson, T., Schoenauer, M., Sebag, M.: Pilot, rollout and monte carlo tree search methods for job shop scheduling. In: Y. Hamadi, M. Schoenauer (eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp. 160–174. Springer Berlin Heidelberg (2012)
38. Russell, T., Malik, A.M., Chase, M., van Beek, P.: Learning heuristics for the superblock instruction scheduling problem. IEEE Trans. on Knowl. and Data Eng. **21**(10), 1489–1502 (2009)

39. Stafford, E.F.: On the Development of a Mixed-Integer Linear Programming Model for the Flowshop Sequencing Problem. Journal of the Operational Research Society **39**(12), 1163–1174 (1988)
40. 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)
41. Tsai, J.T., Liu, T.K., Ho, W.H., Chou, J.H.: An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. The International Journal of Advanced Manufacturing Technology **38**(9-10), 987–994 (2007)
42. Vázquez-Rodríguez, J.A., Petrovic, S.: A new dispatching rule based genetic algorithm for the multi-objective job shop problem. Journal of Heuristics **16**(6), 771–793 (2009)
43. 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)
44. 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)
45. Zhang, W., Dietterich, T.G.: A reinforcement learning approach to job-shop scheduling. In: Proceedings of the 14th international joint conference on Artificial Intelligence, *IJCAI'95*, vol. 2, pp. 1114–1120. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1995)