Todo list

1.2. dispatching rules are a very useful approach to dealing with these environments because they are easy to implement (by computers and shop floor operators) and can cope with dynamic changes. 6

1.2. Keane (2015) used GP to creates features for Case Based Reasoning, which were hard do understand and cumbersome in implementation due to their complexity. In order to mediate the process, the *Espresso Algorithm* from logic circuit design was used for feature selection, as 'espresso' summarises the evolved features obtained by GP, yielding a much simpler form that is more comprehensible for the end-user. The motivation for easily interpretable models, is particularly appealing, even necessary in some cases. Because in some paradigms they become essential for getting them sanctioned, e.g., due to legislation for implementation of uninhabited aerial vehicles (i.e. drones). 6

1.2. More learning methods for JSP 8

1.2. Relocate somewhere else 8

1.2. Add multiobjective job-shop citations 9

1.3. ξ_π can be interpreted as training accuracy.
Post-processing: apply analysis on $\hat{\pi}$ (learned policy), i.e., $\xi_{\hat{\pi}}$ and $\zeta_{\hat{\pi}}$. 10

1.4. Update outline once draft is ready... 11

1.4. our ability to learn about the strengths and weaknesses of algorithms from empirical evidence 12

4.4. Make sure Section 6.6 reference Tables 4.2 to 4.7. 50

6.3. Check if Fig. 6.4 are part of *d* choose 1 pref-models feature selection . . . 61

Figure: Easy and Hard simultaneous corrolation 71

Figure: Easy corrolation 71

Figure: Hard corrolation 71

6.7. Check with best/worst case scenario 72

7.1. Which is being used, ordinal or logistic regression? 74

7.3. No longer one model for all steps 75

7.5. No longer the case, one model instead of K stepwise-models 81

8.1. Throughout a Kolmogorov-Smirnov test with $\alpha = 0.05$ is applied to determine statistical significance between methodologies. 84
8.4. A prevalent approach to solving job shop scheduling problems is to combine several relatively simple dispatching rules such that they may benefit each other for a given problem space. Generally, this is done in an ad-hoc fashion, requiring expert knowledge from heuristics designers, or extensive exploration of suitable combinations of heuristics. The approach here is to automate that selection by translating dispatching rules into measurable features and optimising what their contribution should be via evolutionary search. The framework is straight forward and easy to implement and shows promising results. Various data distributions are investigated for both job shop and flow shop problems, as is scalability for higher dimensions. Moreover, the study shows that the choice of objective function for evolutionary search is worth investigating. Since the optimisation is based on minimising the expected mean of the fitness function over a large set of problem instances which can vary within the set, then normalising the objective function can stabilise the optimisation process away from local minima. 92
9.0. Compare CMA-ES to PREF models 93
9.0. When applying rollout features from **??**, then is sensible to keep track of the best solution found (even though they hadn't been followed), they will referred to as *fortified* solution. 93
Figure: content... 94
Figure: content... 94
10.0. Write overall conclusions of dissertation! 95
IV.0. Find second publication for Paper IV – Selected papers ISI? 119
VI.0. Paper VI not yet submitted 123
VII.0. Paper VII not yet submitted 125

This page is intentionally left blank.

Listing of Publications

This dissertation is based on the following publications, listed in chronological order:

Paper I Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling

Paper II Sampling Strategies in Ordinal Regression for Surrogate Assisted Evolutionary Optimization

Paper III Determining the Characteristic of Difficult Job Shop Scheduling Instances for a Heuristic Solution Method

Paper IV Evolutionary Learning of Weighted Linear Composite Dispatching Rules for Scheduling

Paper V Generating Training Data for Learning Linear Composite Dispatching Rules for Scheduling

Paper VI Supervised Learning Linear Composite Dispatch Rules for Scheduling

Paper VII Imitation Learning Linear Composite Dispatch Rules for Scheduling

These publications will be referenced throughout using their Roman numeral. The thesis is divided into two parts: *Prologue*, and *Papers*. Prologue gives a coherent connection for the publications, and elaborates on chosen aspects. Whereas, Papers contains copies of the publications reprinted with permission from the publishers.

Table 1: Summary of experimental designs in Part II.

Paper	Problem	Model	Model parameters	|Model|*
I	JSP	PREF	$\Phi^{\text{OPT}}, \Psi_b$	K
II	$\mathbb{R}$-functions	CMA-ES	surrogate sampling strategies	1
III	JSP	SDR	MWR	1
IV	JSP, FSP	CMA-ES	$\min \mathbb{E}\left[C_{\max}\right], \min \mathbb{E}\left[\rho\right]$	1
V	JSP	PREF	$\{\Phi^{\pi} \mid \pi \in \{\text{OPT}, \text{SDR}, \text{ALL}\}\}$ $\{\Psi_r \mid r \in \{b, f, p\}\}$	K
VI	JSP, FSP	PREF	$\Phi^{\text{OPT}}, \Psi_p$	1
VII	JSP, FSP	PREF	$\{\Phi^{\pi} \mid \pi \in \{\text{OPT}, \text{OPT}\varepsilon, \text{DA}i\}\}$ Ψ_p	1

*Models are either stepwise (i.e. total of K models) or fixed throughout the dispatching process.

Mapping between Part I and Part II

The prologue will be addressing the job-shop scheduling problem, detailed in Chapter 2 and correspond to the application in Papers I and III to VII. The problem generators used are subsequently described in Chapter 3. From there, we try do define problem difficulty in Chapter 4, improving upon the ad-hoc definition from Paper III. There will be two algorithms considered: *i)* preference learning in Chapter 7, which is a tailored algorithm, and *ii)* evolutionary search in Chapter 8, which is a general algorithm.

The latter was implemented in Paper IV, which could be improved by incorporating the methodology from Paper II. Preference models on the other hand, are highly dependent on training data, whose collection is addressed in Chapter 5, and were the topic of discussion in Paper V and Paper VII. Moreover, the training data contains an abundance of information that can be used to determine a algorithm's footprint in instance space, which was done for optimal solutions in Paper VI, and in addition to that SDR based trajectories were inspected in Chapter 6 along with tying together the preliminary work in Paper III. Finally, Chapter 9 compares to two methodologies, as the preference models had been significantly improved since Paper IV. An overview of experimental settings in Part II is given in Table 1.

Part I

Prologue

Begin at the beginning and go on till you come to the end: then stop.

The King

1 Introduction

Hand crafting heuristics for NP-hard problems is a time consuming trial-and-error process, requiring inductive reasoning or problem specific insights from their human designers. Furthermore, within a problem class (such as scheduling) it is possible to construct problem instances where one heuristic would outperform another.

Each heuristic performs distinctly to others depending on the underlying data distribution of the problem. Because any algorithm which has superior performance in one class of problems is inevitably inferior over another class, i.e., *no free lunch* theorem (Wolpert and Macready, 1997). The success of a heuristic is how it manages to deal with and manipulate the characteristics of its given problem instance. Thus, in order to understand more fully how a heuristic will eventually perform, one needs to look into what kind of problem instances are being introduced to the system. What defines a problem instance, e.g., what are its key features? And how can they help with designing better heuristics? Once the problem instances are fully understood, an appropriate learning algorithm can be implemented in order to create heuristics that are self-adapting to its those instances.

Given the ad-hoc nature of the heuristic design process, there is clearly room for improvement. A number of attempts have been made to automate heuristic design, and it is the ultimate goal of this dissertation to automate optimisation heuristics via ordinal regression. The focal point will be based on scheduling processes named job-shop scheduling problem (JSP), and one of its subclasses, the flow-shop scheduling problem (FSP).

There are two main viewpoints on how to approach scheduling problems, namely,

> **Tailored algorithms** or constructive methods,
> by building schedules for one problem instance at a time.
>
> **General algorithms** or iterative methods,
> by building schedules for all problem instances at once.

For tailored algorithm construction: *i)* a simple construction heuristic is applied; *ii)* the schedule's features are collected at each dispatch iteration, and *iii)* from which a learning model will inspect the feature set to discriminate which operations are preferred to others via ordinal regression. The focus is essentially on creating a meaningful preference set composed of features and their ranks, as the learning algorithm is only run *once* to find suitable operators for the value function. However, for general algorithm construction, there is no feature set collected beforehand, since the learning model is optimised directly via evolutionary search. This requires numerous costly value function evaluations. In fact, it involves an indirect method of evaluation whether one learning model is preferable to another w.r.t. which one yields the better expected mean. Evolutionary search only requires the rank of the candidates, and therefore it is appropriate to retain a sufficiently accurate surrogate for the value function during evolution in order to reduce the number of costly true value function evaluations. In this paradigm, ordinal regression can be used for surrogate assisted evolutionary optimisation, where models are ranked – whereas for tailored algorithms, features were ranked.

1.1 Rice's framework for algorithm selection

The aim of this dissertation is to understand what underlying characteristics of the problem instances distinguish *good* and *bad* solutions when implementing a particular algorithm. Smith-Miles and Lopes (2011) were interested in discovering whether synthetic instances were in fact similar to real-world instances for timetabling scheduling. Moreover, Smith-Miles and Lopes focused on how varying algorithms perform on different data distributions. Hence, the investigation of heuristic efficiency is closely intertwined with problem generation. The relation between problem structure and heuristic efficiency, called *footprints in instance space*, will be addressed in Chapters 4 and 6. In order to formulate the relationship for footprints, one can utilise Rice's framework for algorithm selection problem from 1976. The framework consists of four fundamental components:

> **Problem space or instance space** $\mathcal{P}$,
> set of problem instances;

Feature space $\mathcal{F}$,
measurable properties of the instances in $\mathcal{P}$;

Algorithm space $\mathcal{A}$,
set of all algorithms under inspection;

Performance space $\mathcal{Y}$,
the outcome for $\mathcal{P}$ using an algorithm from $\mathcal{A}$.

For a given problem instance $\mathbf{x} \in \mathcal{P}$ with d features $\boldsymbol{\varphi}(\mathbf{x}) = \left[\varphi_1(\mathbf{x}), ..., \varphi_d(\mathbf{x})\right]^T \in \mathcal{F}$ and using algorithm $a \in \mathcal{A}$ the performance is $y = \Upsilon(a, \boldsymbol{\varphi}(\mathbf{x})) \in \mathcal{Y}$, where $\Upsilon : \mathcal{A} \times \mathcal{F} \mapsto \mathcal{Y}$ is the mapping for algorithm and feature space onto the performance space. This data collection is often referred to as meta-data.

In the context of Rice's framework, the aforementioned approaches to scheduling problems are to maximise its expected performance:

Tailored algorithms

$$\max_{\mathcal{F}' \subset \mathcal{F}} \mathbb{E}\left[\Upsilon\left(a, \boldsymbol{\varphi}(\mathbf{x})\right)\right] \tag{1.1}$$

The focal point is only using problem instances that represent the problem space, $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$, in addition finding a suitable subset of the feature space, $\mathcal{F}' \subset \mathcal{F}|_{\mathcal{P}'}$. If done effectively, then the resulting learning model $a \in \mathcal{A}$ needs only be run once via ordinal regression.

General algorithms

$$\max_{a \in \mathcal{A}} \mathbb{E}\left[\Upsilon\left(a, \boldsymbol{\varphi}(\mathbf{x})\right)\right] \tag{1.2}$$

This is straightforward approach as the algorithm $a \in \mathcal{A}$ is optimised directly given the entire instances space $\mathbf{x} \in \mathcal{P}$ dedicated for training. Alas, this comes at a great computational cost.

Note, the mappings $\boldsymbol{\varphi} : \mathcal{P} \mapsto \mathcal{F}$ and $\Upsilon : \mathcal{A} \mapsto \mathcal{Y}$ are the same for both paradigms.

A schematic flow-chart of the model selection process is illustrated in Fig. 1.1. Meta-data is analysed to investigate problem structure and heuristic effectiveness, i.e., its footprint. Moreover, the schematic details how the preference model, which is a tailored algorithm, from Chapter 7 will come into play in the framework.

1.2 Previous work

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

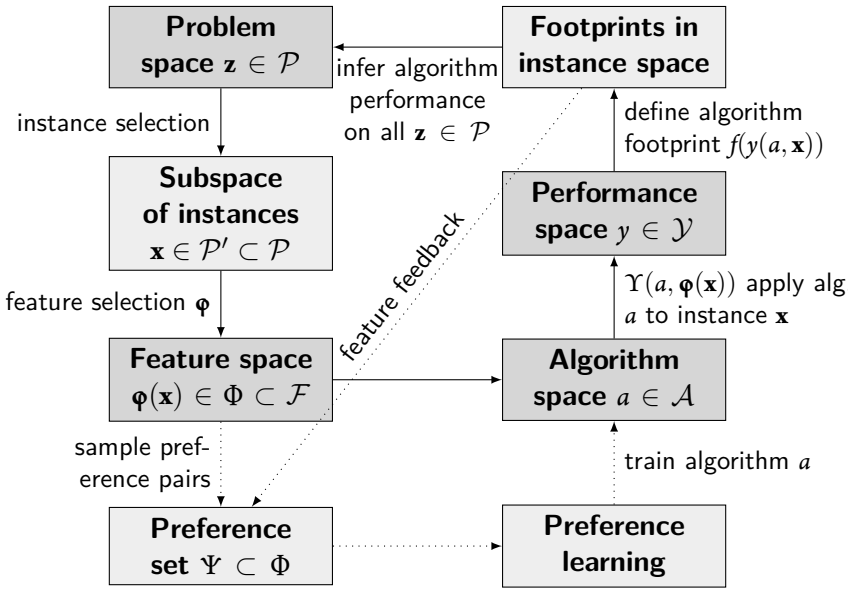


Figure 1.1: Flow-chart for Rice's framework for algorithm selection

solution, however, job-shop scheduling is strongly NP-hard* (Garey et al., 1976). 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. Heuristics are generally more time efficient, but do not necessarily attain the global optimum. Therefore, JSP 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 (Meeran and Morshed, 2012), all from simple and straight forward dispatching rules to highly sophisticated frameworks. Figure 1.2 summarise the main techniques applied to solve JSP. The figure is based on Fig. 1 from Jain and Meeran (1999), however, updated to reflect the previous work relevant to this dissertation.

1.2. dispatching rules are a very useful approach to dealing with these environments because they are easy to implement (by computers and shop floor operators) and can cope with dynamic changes.

*NP stands for Non-deterministic Polynomial-time. If P≠NP, then NP-hard problems cannot be solved by a deterministic Turing machine in polynomial time.

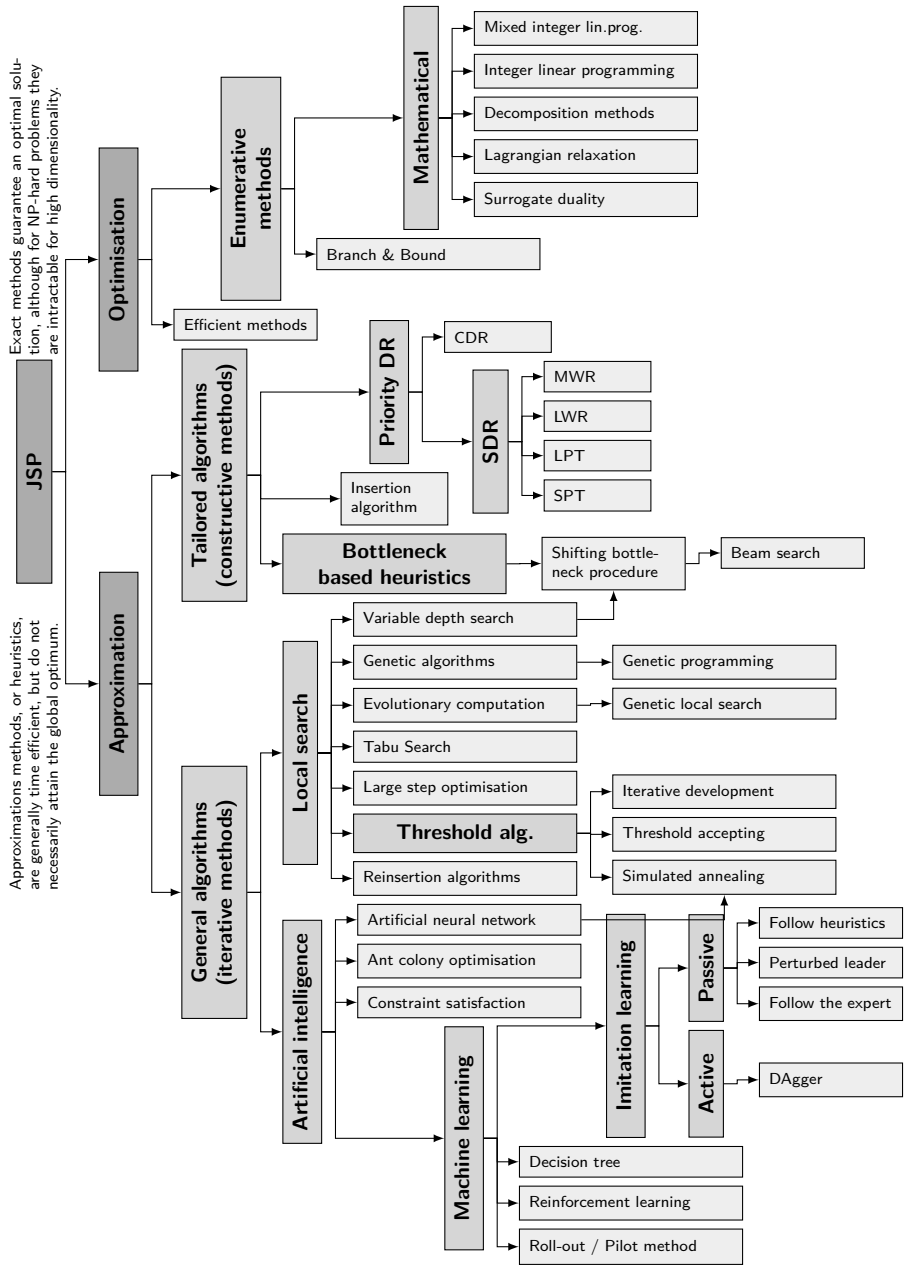


Figure 1.2: Various methods for solving JSP (based on Fig. 1 from Jain and Meeran, 1999)

1.2. Keane (2015) used GP to creates features for Case Based Reasoning, which were hard do understand and cumbersome in implementation due to their complexity. In order to mediate the process, the *Espresso Algorithm* from logic circuit design was used for feature selection, as 'espresso' summarises the evolved features obtained by GP, yielding a much simpler form that is more comprehensible for the end-user. The motivation for easily interpretable models, is particularly appealing, even necessary in some cases. Because in some paradigms they become essential for getting them sanctioned, e.g., due to legislation for implementation of uninhabited aerial vehicles (i.e. drones).

In the field of Artificial Intelligence, Meeran and Morshed (2012) point out that despite their 'intelligent' solutions, the effectiveness of finding the optimum has been rather limited. However, combined with local-search methodologies, they can be improved upon significantly, as Meeran and Morshed showed with the use of a hybrid method involving 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 the intensified local search capabilities of TS. Unfortunately, hybridisation of global and local methodologies is non-trivial. In general, combination of the two improves performance. Unfortunately, they often come at a great computational cost.

1.2. More learning methods for JSP

Various *learning* approaches have been applied to solving job-shop scheduling, such as: *i)* reinforcement learning (Zhang and Dietterich, 1995); *ii)* evolutionary learning (Tay and Ho, 2008), and *iii)* supervised learning (Li and Olafsson, 2005, Malik et al., 2008). The approach taken in this dissertation is a supervised learning classifier using ordinal regression.

A common way of finding a good feasible solution for JSP is applying construction heuristics with some dispatching rules, e.g., choosing a task corresponding to: *i)* longest or shortest processing time; *ii)* most or least successors (i.e. operation number), or *iii)* ranked positional weight, i.e., sum of processing times of its predecessors or successors. Ties are broken in an arbitrary fashion or by another heuristic rule. A summary of over 100 classical dispatching rules for scheduling can be found in Panwalkar and Iskander (1977), and it is noted that these classical dispatching rules are continually used in research. There is no dominant rule, but the most effective have been single priority dispatching rules based on job processing attributes (Haupt, 1989). Tay and Ho (2008) showed that combining dispatching rules, with the aid of genetic programming, is promising. However, there is large number of rules to choose from, thus their combinations require expert knowledge or extensive trial-and-error process.

The literature in scheduling mainly focuses on different objectives,

1.2. Relocate somewhere else

e.g., Chang (1996) minimised the due-date tightness and Drobouchevitch and Strusevich (2000), Gao et al. (2007) looked into solving for bottleneck machines. In this dissertation only minimisation of the makespan will be considered, thus ignoring all due-date constraints. Model assumptions (i.e. shop floor constraints) can also vary, e.g., Thiagarajan and Rajendran (2005) incorporate different earliness, tardiness and holding costs. Brandimarte (1993), Pezzella et al. (2008), Xia and Wu (2005) extend the classical JSP set-up, called *flexible* job-shop, by allowing tasks to be processed by any machine from a given set, i.e., adding assignment of operations to the constraints. Moreover, it is possible to reduce JSP to a FSP, since in practice, most jobs in the job-shop use the machines in the same order (Guinet and Legrand, 1998, Ho et al., 2007).

1.2. Add multiobjective job-shop citations

Instead of using construction heuristics that creates job-shop schedules by sequentially dispatching one job at a time, one could work with complete feasible schedules and iteratively repairing them for a better result. Such was the approach by Zhang and Dietterich (1995) who studied space shuttle payload processing by using reinforcement learning, 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. This approach of a two phased process of construction and improvement is also implemented in timetable scheduling, e.g., Asmuni et al. (2009) used a fuzzy approach in considering multiple heuristic ordering in the construction process, and only allowed feasible schedules to be passed to the improvement phase.

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 Li and Olafsson (2005) for job-shop where a LPT-heuristic is applied. Afterwards, a decision tree is used to create a dispatching rule with similar logic. However, this method cannot outperform the original LPT-heuristic used to guide the search. For instruction scheduling, this drawback is confronted in Malik et al. (2008), Olafsson and Li (2010), Russell et al. (2009), by using an optimal scheduler, computed offline. The optimal solutions are used as training data and a decision tree learning algorithm is 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. A similar approach is taken for timetable scheduling in Burke et al. (2006), using case based reasoning, where training data is guided by the two best heuristics in the field. Burke et al. point out that in order for their framework to be

successful, problem features need to be sufficiently explanatory and training data needs to be selected carefully so they can suggest the appropriate solution for a specific range of new cases. Again, stressing the importance of meaningful feature selection.

1.3 Contributions

The initial goal of the Ph.D. project was to use sophisticated algorithms for preference learning on hard problems, in particular job-shop scheduling, and find ways to mediate the computational effort that they require. After painstaking parameter tuning, and managing to find complex models with high training accuracy. Alas, severely overfitted to the training instances – a simple linear model would suffice with similar performance, and for much less overhead! Also, linear models come with the added benefit of easy interpretability.

Unfortunately, there is not much said about algorithms that fail (Smith-Miles and Bowly, 2015), as the focus tend to be on claiming superiority in performance to some previous approach. So to quote a pioneer in scheduling,

> "The only real mistake is the one from which we learn nothing"
>
> Henry Ford

In order to make the best of a bad situation, this derailment* designed the course of the body of work presented in this dissertation. Namely, by dwelling on optimal solutions and try to understand their fundamental building blocks, and applying what you learn on *simple* models, before investing valuable time and resources in implementing the current state-of-the-art algorithms. The research questions that are put forth are: *i)* how are optimal solutions *supposed* to behave – what are the key indicators? *ii)* Where and when should there be emphasis on learning? And ultimately, *iii)* what states of our problem are worth investigating further to achieve the desired result?

Hopefully, this preparatory work helps recognising any limitations, and will lead to better algorithm design, or at least improved understanding of *why* the models are performing in the way that they do. It's the believe of the author, that the methodology of going about this, which is roughly described in Alg. 1, can be applied to any kind of optimisation problem. As such, then it's suitable to name the framework: *Analysis & Learning in Complex Environments*, or ALICE** for short. For demonstration purposes, this dissertation will solely be focusing on applying ALICE to dispatching rules for job-shop scheduling.

*This explains why Paper II is completely different from the other publications.

**The hopefully catchy and very deliberate 'backronym,' pays homage to the wonderful literary character, Alice in Wonderland–a personal favourite of the author.

Algorithm 1 Analysis & Learning in Complex Environments (ALICE) framework, given a problem space $\mathcal{P}$, an expert policy $\pi_\star$, and set of benchmark algorithms $\mathcal{A}$.

procedure ALICE($\mathcal{P}, \pi_\star, \mathcal{A}$)
  $Y^{\pi_\star} \leftarrow \{Y(\pi_\star, \boldsymbol{\varphi}(\mathbf{x})) \mid \mathbf{x} \in \mathcal{P}\}$ ▷ collect optimal solutions
  $\Phi^{\pi_\star} \leftarrow \left\{\boldsymbol{\varphi}_{\pi^\star}(\mathbf{x}) \;\middle|\; \mathbf{x} \in \mathcal{P}\right\}$ ▷ collect optimal meta-data
  for all $\pi \in \mathcal{A}$ **do** ▷ for each algorithm
    $Y^{\pi} \leftarrow \{Y(\pi, \boldsymbol{\varphi}(\mathbf{x})) \mid \mathbf{x} \in \mathcal{P}\}$ ▷ collect solutions
    $\Phi^{\pi} \leftarrow \left\{\boldsymbol{\varphi}_{\pi}(\mathbf{x}) \;\middle|\; \mathbf{x} \in \mathcal{P}\right\}$ ▷ collect meta-data
    $\xi_\pi \leftarrow \mathbb{E}\left[\pi_\star = \pi \mid \pi_\star\right]$ ▷ optimality of π (i.e. when $Y^{\pi_\star} = Y^{\pi}$)
    $\zeta_\pi \leftarrow \textsc{Analyse}(\xi_\pi \mapsto Y^{\pi})$ ▷ relation between optimality and end-result
    $\Phi^{\pi} \leftarrow \textsc{Sample}(\Phi_\pi, \zeta_\pi)$ ▷ adjust set w.r.t. analysis
  end for
  $\Phi \leftarrow \{\Phi^{\pi} \mid \pi \in \{\mathcal{A} \cup \pi_\star\}\}$ ▷ training set
  $\hat{\pi} \leftarrow \textsc{Train}(\Phi)$ ▷ apply learning algorithm
  return $\hat{\pi}$ ▷ learned policy
end procedure

1.3. ξ_π can be interpreted as training accuracy.
Post-processing: apply analysis on $\hat{\pi}$ (learned policy), i.e., $\xi_{\hat{\pi}}$ and $\zeta_{\hat{\pi}}$.

1.4 Outline

1.4. Update outline once draft is ready...

An approach based on supervised learning, mostly on optimal schedules will be investigated and its effectiveness illustrated by improving upon well known dispatch rules for job-shop scheduling in Chapter 7. The method of generating training data is shown to be critical for the success of the method, as shown in **??**. Moreover the choice of problem instances under consideration is worth considering, as discussed in Chapter 3, and will be used throughout in the subsequent chapters.

The preliminary experiments done in Paper III investigated the characteristics of difficult job-shop schedules for a single heuristic, continuing with that research, Chapter 4 compares a set of widely used dispatching rules on different problem spaces in the hopes of extrapolating where an algorithm excels in order to aid its failing aspects, which will be beneficial information for the creation of learning models in Chapter 7, as they are dependant on features based on those same dispatching rules under investigation.

Due to scarcity of real-world data, we let random problem generators suffice, that are described in Chapter 5. Moreover, the traditional OR-Library benchmark instances are

similarly created, although for a greater variety of problem sizes. Smith-Miles and Bowly (2015) warn that general practice in the OR-community is over-tuning of algorithms to a relatively small set of aging* instances. Obviously, the choice of data set has a direct influence of the proposed algorithm. As they are developed with them specifically in mind. This is why robustness towards different problem spaces, than initially trained on, is of so much value, as it indicates how applicable our model is for real-world deployment

1.4. our ability to learn about the strengths and weaknesses of algorithms from empirical evidence

1.5 Supplementary material

The Prologue will mostly focus on traditional job-shop problem instances. However, in Chapter 3 there is a greater variety of problem spaces introduced, and when seen fit some of them will be investigated as well in the subsequent chapters. Since most experiments have been run on all problem spaces, they can be inspected in the supplementary Shiny application written in R. In addition, all source code and data is freely distributed from:

`https://github.com/ALICE-InRu/`

under the permissive creative commons share-alike licence.**

*The OR-Library problem instances are mostly from the 1980s and 1990s, or earlier (cf. Table 3.3).

**Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)

1.5. SUPPLEMENTARY MATERIAL

Some notes

Local Search Vaessens et al. (1996) divide approximations methods to:

i) iterative methods, which starts with some initial feasible solution, and its neighbourhood is searched for one with lower cost. If such a solution is found, the algorithm is continued from there; otherwise, a local minimum has been found

ii) constructive methods, construct a complete schedule and apply local search to partial schedules on the way .

Beam search by Sabuncuoglu and Bayiz (1999) is an adaptation of the branch and bound method in which only some nodes are evaluated in the search tree. At any level, only the promising nodes are kept for further branching and remaining nodes are pruned off permanently.

Insertion Algorithm Werner and Winkler (1995)

Read the directions and directly you will be directed in the right direction.

Doorknob

2

Job-shop Scheduling Problem

Scheduling problems, which occur frequently in practice, are a category within combinatorial optimisation problems. A subclass of scheduling problems is job-shop (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. Job-shop's 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.

Deterministic JSP is the most *general* case for classical scheduling problems (Jain and Meeran, 1999). Many other scheduling problems can be reformulated as JSP. For instance, the *travelling salesman problem** can be contrived as JSP: with the salesman as a single machine in use; the cities to be visited are the jobs to be processed, and distance is sequence dependent set-up time. The general form of JSP assumes that each job can have its own distinctive flow pattern through the machines, which is independent of the other jobs. In the case where all jobs share the same permutation route, job-shop is reduced to a flow-shop scheduling problem (FSP) (Guinet and Legrand, 1998, Tay and Ho, 2008).

*The travelling salesman problem (TSP) was formulated in the 1800s by the mathematicians W.R. Hamilton and Thomas Kirkman (Biggs et al., 1986). The salesman has to visit a set of cities exactly once (i.e. Hamiltonian path), with the objective of minimising the route, in terms of distance, between them.

Therefore, without loss of generality, this dissertation is structured around JSP.

Remark: Throughout the dissertation the FSP variation will *not* be a commonly used permutation flow-shop (PFSP) from the literature,* which has the added constraints of not allowing any jobs to pass one another. Here, the jobs have to be processed in the same machine order. However, machines do not necessarily need to process jobs in the same order, as is implied in PFSP. For PFSP the Manne (1960) model would be more appropriate, rather than the one described in the following section.

2.1 Mathematical formulation

Job-shop considered for this dissertation is when n jobs, $\mathcal{J} = \{J_j\}_{j=1}^{n}$, are scheduled on a finite set, $\mathcal{M} = \{M_a\}_{a=1}^{m}$, of m machines, subject to the constraint that each job J_j must follow a predefined machine order (a chain 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. The objective is to schedule jobs in such a manner as to minimise the maximum completion times for all tasks, which is also known as the makespan, $C_{\max}$.

A common notation for scheduling problems (cf. Chapter 2 in Pinedo, 2008) is given by a triplet $\alpha|\beta|\gamma$, where: α describes the machine environment; β details any additional processing characteristics and/or constraints, and finally γ lists the problem's objective. Hence our family of scheduling problems, i.e., a m machine JSP and FSP w.r.t. minimising makespan, is $Jm||C_{\max}$ and $Fm||C_{\max}$, respectively. An additional constraint commonly considered are job release-dates and due-dates, and then the objective is generally minimising the maximum lateness, denoted $Jm|r_j, d_j|L_{\max}$. However, those shop-requirements will not be considered here.

Henceforth, the index j refers to a job $J_j \in \mathcal{J}$, while the index a refers to a machine $M_a \in \mathcal{M}$. 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. Moreover, index k will denote the time step of the operation. Note that once an operation is started, it must be completed uninterrupted, i.e., pre-emption is not allowed. Moreover, there are no sequence dependent set-up times.

For any given JSP, then each job J_j has an indivisible processing 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 completion or end time is denoted $x_e(j, a)$ where,

$$x_e(j, a) := x_s(j, a) + p_{ja} \tag{2.1}$$

*Paper III wrongly states that it is used PFSP problem instances, it was in fact FSP.

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.,

$$x_s(j, \boldsymbol{\sigma}_j(a)) \geq x_e(j, \boldsymbol{\sigma}_j(a-1)) \tag{2.2}$$

for all $J_j \in \mathcal{J}$ 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 permutation route, JSP is reduced to a FSP.

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

$$x_s(j, a) \geq x_e(j', a) \quad \text{or} \quad x_s(j', a) \geq x_e(j, a) \tag{2.3}$$

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

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,

$$C_{\max} := \max \left\{ x_e(j, \boldsymbol{\sigma}_j(m)) \;\middle|\; J_j \in \mathcal{J} \right\}. \tag{2.4}$$

Clearly, w.r.t. minimum makespan, it is preferred that schedules are non-delay, i.e., the machines are not kept idle. The time in which machine M_a is idle between consecutive jobs J_j and $J_{j'}$ is called idle time, or slack,

$$s(a, j) := x_s(j, a) - x_e(j', a) \tag{2.5}$$

where J_j is the immediate successor of $J_{j'}$ on M_a. Although this is not a variable directly needed to construct a schedule for JSP, it is a key attribute in order to measure the quality of the schedule.

Note, from a job-oriented viewpoint, for a job already dispatched $J_j \in \mathcal{J}$ the corresponding set of machines already processed is $\mathcal{M}_j \subset \mathcal{M}$. Similarly from the machine-oriented viewpoint, $M_a \in \mathcal{M}$ with corresponding $\mathcal{J}_a \subset \mathcal{J}$.

2.2 Construction heuristics

Construction heuristics are designed in such a way that it limits the search space in a logical manner. Preferably without excluding the true optimum. Here, the construction heuristic, Υ, is to schedule the dispatches as closely together as possible, i.e., minimise the schedule's idle times. More specifically, once an operation (j, a) has been chosen from the job-list, $\mathcal{L}$, 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 Ineq. (2.2). However, to guarantee that Ineq. (2.3) is not violated, idle times M_a are inspected, as they create a slot which in 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. (2.5) directly, instead it has to be updated as follows,

$$\tilde{s}(a, j') := x_s(j'', a) - \max\{x_e(j', a),\ x_e(j, \boldsymbol{\sigma}_j(a-1))\} \tag{2.6}$$

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

$$\tilde{S}_{ja} := \left\{ J_{j'} \in \mathcal{J}_a \;\middle|\; \tilde{s}(a, j') \geq p_{ja} \right\}. \tag{2.7}$$

There are several heuristic methods for selecting a slot from Eq. (2.7), 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. However, dispatching J_j in the first slot would result in its earliest possible release time, which would be beneficial for subsequent dispatches for J_j. Experiments favoured dispatching in the earliest slot,* thus used throughout.

Note that the choice of slot is an intrinsic heuristic within Υ. The focus of this dissertation, however, is on learning the priority of the jobs on the job-list, for a fixed construction heuristic. Hence, there could be some problem instances in which the optimum makespan cannot be achieved, due to the limitations of Υ of not being properly able to differentiate between which slot from Eq. (2.7) is the most effective. Instead, hopefully, the learning algorithm will be able to spot these problematic situations, should they arise, by inspecting the schedule's features and translate that into the jobs' priorities.

Dispatching rules

Dispatching rules (DR) are an integral part of a construction heuristics, as it determines the priorities of the job-list, i.e., the jobs who still have operations unassigned. Starting with an empty schedule, and sequentially adding one operation (or task) at a time. Then, for each time step k, an operation is dispatched which has the highest priority of the job-list, $\mathcal{L}^{(k)} \subset \mathcal{J}$. If there is a tie, some other priority measure is used. However, let's assume that ties are broken randomly. Algorithm 2 outlines the pseudo code for the entire dispatching process of a JSP problem instance.

*Preliminary experiments of 500 JSP instances where inspected: First slot chosen could always achieve its known optimum by implementing Alg. 2, however, only 97% of instances when choosing the smallest slot.

Algorithm 2 Pseudo code for constructing a JSP sequence using a deterministic scheduling policy (or dispatching rule), π, for a fixed construction heuristic, Υ.

procedure SCHEDULEJSP(π, Υ)
  $\boldsymbol{\chi} \leftarrow \emptyset$ ▷ initial current dispatching sequence
  for $k \leftarrow 1$ **to** $l = n \cdot m$ **do** ▷ at each dispatch iteration
    for all $J_j \in \mathcal{L}^{(k)} \subset \mathcal{J}$ **do** ▷ inspect job-list
      $\boldsymbol{\chi}^j \leftarrow \{\chi_i\}_{i=1}^{k-1} \cup J_j$ ▷ partial temporal schedule
      $\boldsymbol{\varphi}^j \leftarrow \boldsymbol{\varphi} \circ \Upsilon\left(\boldsymbol{\chi}^j\right)$ ▷ features for post-descision state
      $I_j^\pi \leftarrow \pi\left(\boldsymbol{\varphi}^j\right)$ ▷ priority for J_j
    end for
    $j^* \leftarrow \operatorname{argmax}_{j \in \mathcal{L}^{(k)}} \{I_j^\pi\}$ ▷ choose highest priority
    $\chi_k \leftarrow J_{j^*}$ ▷ dispatch j^*
  end for
  return $C_{\max}^\pi \leftarrow \Upsilon(\boldsymbol{\chi})$ ▷ makespan and final schedule
end procedure

Henceforth, we will adopt the following terminology: a *sequence* will refer to the sequential ordering of the dispatches* of tasks to machines, namely,

$$\boldsymbol{\chi} = \{\chi_k\}_{k=1}^K = \left\{(j,a) \;\middle|\; J_j \in \mathcal{L}^{(k)}\right\}_{k=1}^K \tag{2.8}$$

The collective set of allocated tasks to machines, which is interpreted by its sequence, is referred to as a *schedule*; and a *scheduling policy* (or dispatching rule) π will pertain to the manner in which the sequence is manufactured: be it a SDR such as SPT or some other heuristic. Sequence and schedule are often used interchangeably, as they are closely related. A complete schedule is also known as K-solution** (Bertsekas et al., 1997).

2.3 Example

There are many examples of job-shop for real-world application. For demonstration purposes, let's examine a hypothetical problem from the 18th century. Assume we are invited to the Mad Hatter's Tea Party in Wonderland, illustrated in Fig. 2.1. There are four guests attending: J_1) Alice; J_2) March Hare; J_3) Dormouse, and of course our host J_4) Mad Hatter. During these festivities, there are several things each member of the party has to perform. They all have to: M_1) have wine or pour tea; M_2) spread butter; M_3) get a haircut; M_4) check the time of the broken watch for themselves, and M_5) say what they

*Note, only a sequence of J_j is needed, since the corresponding M_a can be obtained by reading $\boldsymbol{\sigma}$.
**A partial schedule, at step k, is called k-solution.

Table 2.1: Example of 4×5 JSP

Guest	Job	Machine ordering $\boldsymbol{\sigma}$					Processing times $\mathbf{p}$				
Alice	J_1	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~26~~	~~25~~	~~40~~	~~15~~	~~42~~
March Hare	J_2	~~1~~	**2**	3	4	5	~~18~~	**86**	86	68	84
Dormouse	J_3	~~1~~	~~3~~	**2**	4	5	~~20~~	**59**	~~23~~	33	96
Mad Hatter	J_4	~~4~~	~~3~~	**1**	5	2	**40**	47	~~55~~	~~13~~	99

mean, e.g., asking a riddle or reciting a poem to the group. The guests are very particular creatures, and would like to do these task in a very specific order, e.g., March Hare insists on doing them alphabetically. Each would rather wait than breaking their habit. They tend to be absent-minded, so each task takes them a different amount of time. Let's assume their processing times and ordering are given in Table 2.1.

Unfortunately, Alice can't stay long. She must leave as soon as possible to play croquet with the Red Queen, and she mustn't be late for that very important date. Otherwise, it's off with someone's head! However, Alice, had a proper upbringing and won't leave the table until everyone has finished their tasks. How should the guests go about their tea-party, in order for Alice to be on-time?

Figure 2.1: The Mad Hatter's Tea Party, from Alice's Adventures in Wonderland by Carroll (1865). Illustration by John Tenniel (1820-1914).

The problem faced by Alice and her new friends is in what order should they rotate their tasks between themselves so that they all finish as soon as possible? This can be considered as is a typical four-job and five-machine job-shop, where: our guests are the jobs; their tasks are the machines, and our objective is to minimise $C_{\max}$, i.e., when Alice can leave.

Let's assume we've come to the party, after 10 operations have already been made (i.e. strikeout entries in Table 2.1), by using the following job sequence,*

$$\boldsymbol{\chi} = \{\chi_i\}_{i=1}^{k-1} = \{J_4, J_2, J_3, J_3, J_1, J_1, J_1, J_1, J_1, J_4\} \tag{2.9}$$

hence currently, at step $k = 11$, the job-list is $\mathcal{L}^{(k)} = \{J_2, J_3, J_4\}$ indicating the 3 potential** jobs (i.e. denoted in bold in Table 2.1) to be dispatched, i.e., $\chi_k \in \mathcal{L}^{(k)}$.

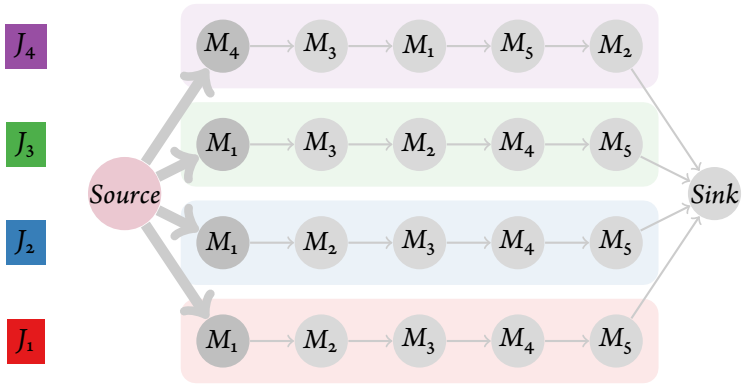
This is a very compact form for the current partial solution, it's easiest to comprehend it via disjunctive graph (Roy and Sussmann, 1964) to model the work-flow of tasks to be scheduled. Let's encode: *i)* the operations as vertices; *ii)* horizontally aligning them w.r.t. each job J_j; *iii)* connect vertices with directed edges according the Ineq. (2.2), and *iv)* by introducing dummy vertices before and after, then the goal is to visit each vertex exactly once, or *Hamiltonian* path: starting at the 'source' (i.e. empty schedule), and finishing at the sink (i.e. complete schedule). The path gives the prescription of the order in which the jobs rotate between machines. Figure 2.2 depicts the path generation at the beginning, midway, and final stages for our Tea Party.: *i)* gray vertices are operations that haven't yet been dispatched; *ii)* pink vertices are the ones that correspond to $\boldsymbol{\chi}$, and *iii)* pink directed edges indicate the current partial Hamiltonian path.

Now we're interested to know when each guest should start their task, i.e., the project schedule. Figure 2.3 illustrates the temporal partial schedule (or k-solution) of Eq. (2.9) as a Gantt-chart: *i)* numbers in the boxes represent the job identification j; *ii)* the width of the box illustrates the processing times for a given job for a particular machine M_a (on the vertical axis); *iii)* the dashed boxes represent the resulting $(k+1)$-solution for when a particular job is scheduled next, and *iv)* the current $C_{\max}$ is denoted with a dotted line. Note, the disjunctive graph from Fig. 2.2b gives the schedule in Fig. 2.3.

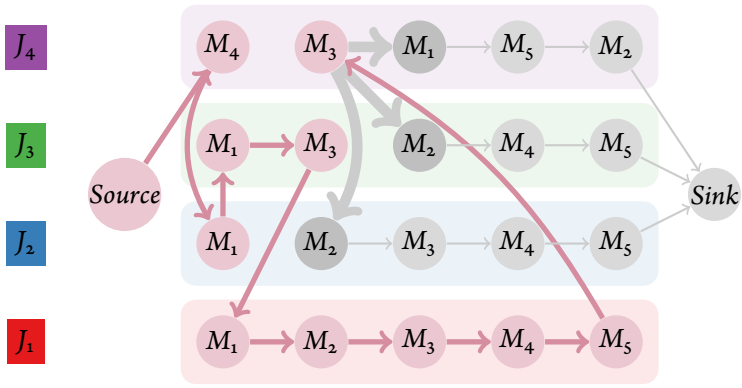
If the job with the shortest processing time were to be scheduled next, i.e., applying SPT-rule, then J_4 would be dispatched. Similarly, for LPT-rule (largest processing time) then J_2 would be dispatched. Other DRs use features not directly observable from looking at the k-solution (but easy to keep record of), e.g., by assigning jobs with most or least total processing time remaining, i.e., MWR and LWR heuristics, who would yield J_2 and J_4, respectively.

*In fact this is the sequence resulting from 10 dispatches following the SPT-rule, to be defined shortly.

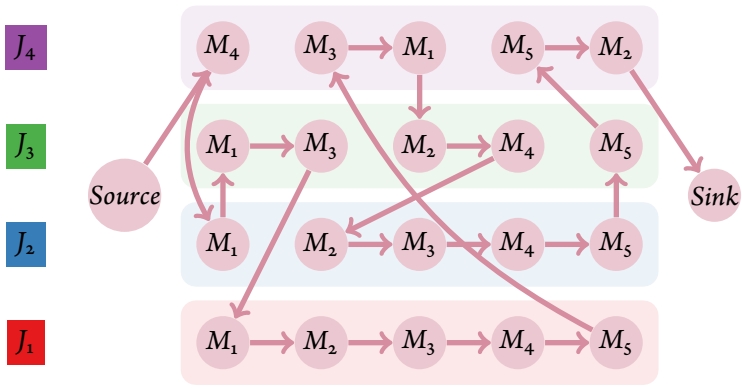
**Alice is quite anxious to leave, so she has already completed everything, and therefore $J_1 \notin \mathcal{L}^{(11)}$.



(a) $k = 0$



(b) $k = K/2$



(c) $k = K$

Figure 2.2: Graph representation of a 4×5 job-shop, where pink vertices are completed tasks, and grey are unassigned. Moreover, grey arrows point to the operations that are next on the job-list, $\mathcal{L}^{(k+1)}$, and pink arrows (traversing from source towards sink) yield the sequence of operations for the schedule, i.e., $\boldsymbol{\chi}$.

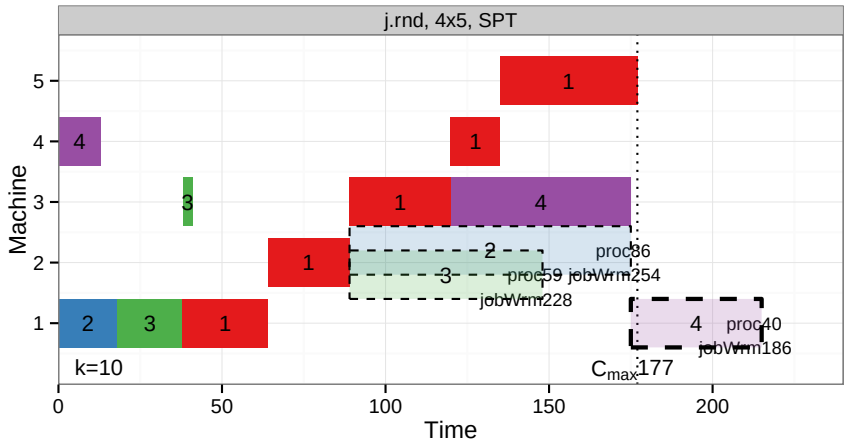


Figure 2.3: Gantt chart of a partial JSP schedule after 10 dispatches: Solid and dashed boxes represent χ and $\mathcal{L}^{(11)}$, respectively. Current $C_{\max}$ denoted as dotted line.

2.4 Single priority based dispatching rules

A *single priority dispatching rule* (SDR) is a function of attributes, or features, of the jobs and/or machines of the schedule. The features can be constant or vary throughout the scheduling process. For instance, priority may depend on job processing attributes, such as which job has,

Shortest immediate processing time (SPT)
greedy approach to finish shortest tasks first,

Longest immediate processing time (LPT)
greedy approach to finish largest tasks first,

Least work remaining (LWR)
whose intention is to complete jobs advanced in their progress, i.e., minimising $\mathcal{L}$,

Most work remaining (MWR)
whose intention is to accelerate the processing of jobs that require a great deal of work, yielding a balanced progress for all jobs during dispatching. However, in-process inventory can be high.

These rules are the ones most commonly applied in the literature due to their simplicity and surprising efficiency. Therefore, they will be referenced throughout the dissertation. However, there are many more available, e.g., randomly selecting an operation with equal

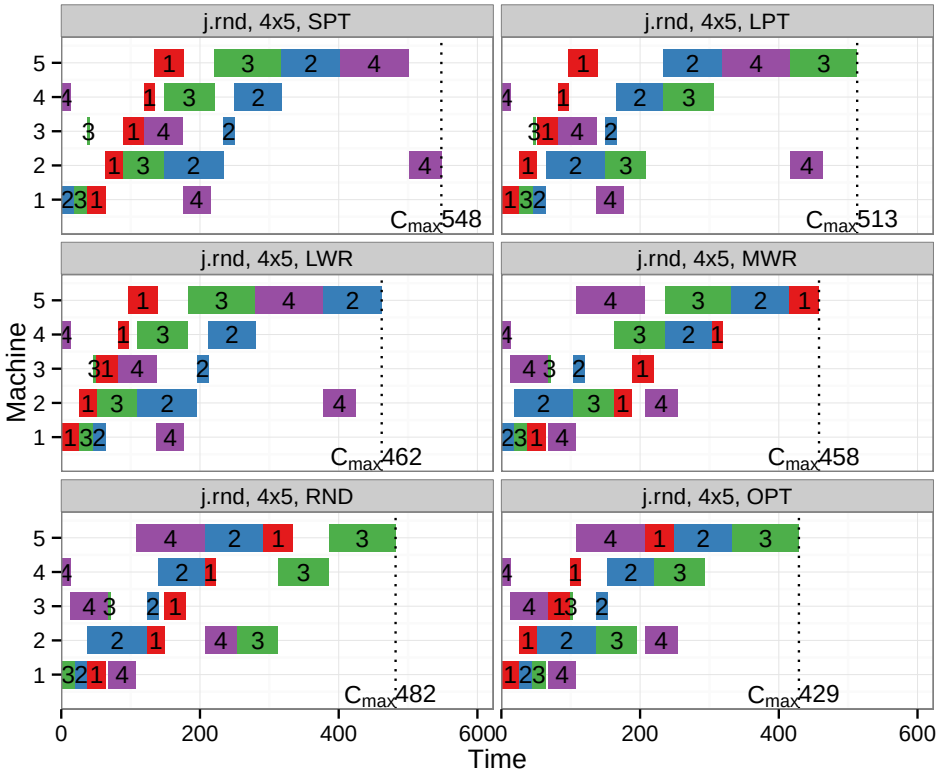


Figure 2.4: SDRs applied to the Tea-party example in Section 2.3. A possible optimal solution is shown in the lower right corner as a reference.

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 Panwalkar and Iskander (1977). However, the reader is referred to an in-depth survey for SDRs by Haupt (1989).

To summarise, SDRs assign an index to each job of the job-list waiting to be scheduled, and are generally only based on few features and simple mathematical operations. Continuing with the example from Section 2.3, the final schedules for these main SDRs (and a possible optimal schedule for reference) are depicted in Fig. 2.4. As we can see, MWR would have been the best strategy for Alice and company, since it has the makespan closest to the optimum.

2.5 Features for job-shop

A DR may need to perform a one-step look-ahead, and observe features of the partial schedule to make a decision. For example by observing the resulting temporal makespan. These emanated observed features are sometimes referred to as an *after-state* or *post-decision state*. A k-solution is denoted $\boldsymbol{\chi}^j$ where J_j is the latest dispatch, i.e., $\chi_k = J_j$, and its resulting features is denoted,

$$\boldsymbol{\varphi}^j := \boldsymbol{\varphi}(\boldsymbol{\chi}^j). \tag{2.10}$$

Features are used to grasp the essence of the current state of the schedule. Temporal scheduling features applied in this dissertation are given in Table 2.2.

The features of particular interest were obtained from inspecting the aforementioned SDRs from Section 2.4: namely φ_1 and φ_7. Moreover, φ_1-φ_8 and φ_9-φ_{16} are job-related and machine-related attributes of the current schedule, respectively.

Some features are directly observed from the k-solution, such as the job- and machine-related features, namely, φ_1-φ_{16} and they are only based on the current step of the schedule, i.e., schedule's *local features*, and might not give an accurate indication of how it will effect the schedule in the long run. Therefore, a set of features are needed to estimate the schedule's overall performance, referred to as its *global features*.

The approach here is to use well known SDRs, φ_{17}-φ_{20}, as a benchmark by retrieving what would the resulting $C_{\max}$ be given if that SDR would be implemented from that point forward. Moreover, random completion of the k-solution are implemented, here φ_{21}-φ_{24} corresponds to statistics from 100 random roll-outs, which can be used to identify which features $\boldsymbol{\varphi}$ are promising on a long-term basis. *Roll-out algorithms* (Bertsekas et al., 1997), also known as *Pilot Method* (Duin and Voß, 1999), for combinatorial optimisation aim to improve performance by sequential application of a pilot heuristic, which completes the remaining $K - k$ steps. Roll-outs for JSP have been conducted by Runarsson et al. (2012). Continuing with that work, Geirsson (2012) compares several pilot heuristics, e.g., *Randomly Chosen Dispatch Rules* which is similar to φ_{17}-φ_{20} (but here one roll-out per fixed SDR). The motivation being, that a SDR-based roll-out are of higher quality than random ones, requiring less computational budget. However, Geirsson notes that performance w.r.t. traditional random roll-outs is statistically insignificant, and not worth the overhead of implementing various SDRs beforehand.

Geirsson reworks the roll-out algorithm as an $|\mathcal{L}|$-armed bandit,* i.e., each job of the

*In probability theory, the multi-armed bandit problem (Berry and Fristedt, 1985) describes a gambler at a row of slot machines, who has to decide which machines to play, i.e., pull its lever, in order to maximise his rewards, that are specific to each machine. The gambler also has to decide how many times to play each machine and in which order to play them. The gambler's actions are referred to as *pilot-heuristic*.

job-list are the levers. Since the best job, j^*, to dispatch at step k, is not known beforehand, therefore all available jobs are evaluated, using roll-outs. As a result, using the features φ_{21}-φ_{24}, the weights $\mathbf{w}$ yield the deterministic pilot heuristic. Although in Geirsson's work, other statistics are used for guidance, e.g., quartile and octile.

It's noted, that the roll-outs considered in this dissertation, are with a very frugal budget, only 100 roll-outs per lever is considered – all evenly distributed between levers. But using the multi-armed bandit paradigm, it's possible to allocate roll-outs originating from the job-list with bias towards more promising levers.

2.6 Composite dispatching rules

Priority dispatching rules were originally introduced in Giffler and Thompson (1960) to resolve conflicts of the job-list, and have made great headway since. They are especially attractive since they are relatively simple to implement, fast and find good schedules. In addition, they are easy to interpret, which makes them desirable for the end-user (i.e. shop floor operators). However, they can also fail unpredictably. Jayamohan and Rajendran (2004) showed that a careful combination of dispatching rules can perform significantly better. These are referred to as *composite dispatching rules* (CDR), where the priority ranking is an expression of several DRs.

For instance, optimising $J1||L_{\max}$ (Pinedo, 2008, see. chapter 14.2), one can combine SDRs that are optimal for a different criteria of problem instances, which complement each other as a CDR, e.g., combining the SDRs: *i)* WSPT * (SPT weighted w.r.t. $\mathcal{J}$), and *ii)* minimum slack first (MS),** yields the CDR *Apparent Tardiness Cost*, which can work well on a broader set of problem instances than the original SDRs by themselves.

CDRs can deal with a greater number of more complicated functions 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 $J_j \in \mathcal{L}^{(k)}$ using π is,

$$I_j^{\pi} = \sum_{i=1}^{d} w_i \pi_i(\boldsymbol{\chi}^j) \tag{2.11}$$

where $w_i > 0$ and $\sum_{i=0}^{d} w_i = 1$, then w_i gives 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 k-solution $\boldsymbol{\chi}^j$.

*WSPT is optimal when all release dates and due dates are zero.

**MS is optimal when all due dates are sufficiently loose and spread out.

Table 2.2: Feature space $\mathcal{F}$ for JSP where job J_j on machine M_a given the resulting temporal schedule after dispatching (j, a).

φ	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	$\sum_{a\in\mathcal{M}} p_{ja}$	jobTotProcTime
φ_7	total work remaining for job	$\sum_{a'\in\mathcal{M}\setminus\mathcal{M}_j} p_{ja'}$	jobWrm
φ_8	number of assigned operations for job	$\lvert\mathcal{M}_j\rvert$	jobOps
	machine related		
φ_9	when machine is next free	$\max_{j'\in\mathcal{J}_a}\{x_e(j',a)\}$	macFree
φ_{10}	total processing time for machine	$\sum_{j\in\mathcal{J}} p_{ja}$	macTotProcTime
φ_{11}	total work remaining for machine	$\sum_{j'\in\mathcal{J}\setminus\mathcal{J}_a} p_{j'a}$	macWrm
φ_{12}	number of assigned operations for machine	$\lvert\mathcal{J}_a\rvert$	macOps
φ_{13}	change in idle time by assignment	$\Delta s(a,j)$	reducedSlack
φ_{14}	total idle time for machine	$\sum_{j'\in\mathcal{J}_a} s(a,j')$	macSlack
φ_{15}	total idle time for all machines	$\sum_{a'\in\mathcal{M}}\sum_{j'\in\mathcal{J}_{a'}} s(a',j')$	allSlack
φ_{16}	current makespan	$\max_{(j',a')\in\mathcal{J}\times\mathcal{M}_{j'}}\{x_f(j',a')\}$	makespan
	final makespan related		
φ_{17}	final makespan using SPT	$C_{\max}^{\text{SPT}}$	SPT
φ_{18}	final makespan using LPT	$C_{\max}^{\text{LPT}}$	LPT
φ_{19}	final makespan using LWR	$C_{\max}^{\text{LWR}}$	LWR
φ_{20}	final makespan using MWR	$C_{\max}^{\text{MWR}}$	MWR
φ_{RND}	final makespans using 100 random rollouts	$\{C_{\max}^{\text{RND}}\}_{i=1}^{100}$	
φ_{21}	mean for φ_{RND}	$\mathbb{E}\left[\varphi_{\text{RND}}\right]$	RNDmean
φ_{22}	standard deviation for φ_{RND}	$\sqrt{\mathbb{E}\left[\varphi_{\text{RND}}^2\right] - \mathbb{E}\left[\varphi_{\text{RND}}\right]^2}$	RNDstd
φ_{23}	minimum value for φ_{RND}	$\min\{\varphi_{\text{RND}}\}$	RNDmin
φ_{24}	maximum value for φ_{RND}	$\max\{\varphi_{\text{RND}}\}$	RNDmax

The composite priority dispatching rule presented in Eq. (2.11) can be considered as a special case of a the following general linear value function,

$$\pi(\boldsymbol{\chi}^j) = \sum_{i=1}^{d} w_i \varphi_i(\boldsymbol{\chi}^j) \overset{(2.10)}{=} \left\langle \mathbf{w} \cdot \boldsymbol{\varphi}^j \right\rangle. \tag{2.12}$$

when $\pi_i(\cdot) = \varphi_i(\cdot)$, i.e., a composite function of the features from Table 2.2.

Finally, the job to be dispatched, J_{j^*}, corresponds to the one with the highest value, i.e.,

$$J_{j^*} = \underset{J_j \in \mathcal{L}}{\operatorname{argmax}} \ \pi(\boldsymbol{\varphi}^j) \tag{2.13}$$

Since we're using a feature space based on job-attributes, then it's trivial to interpret Eq. (2.12) as the SDRs from Section 2.4. Then for $i \in \{1, \ldots, d\}$, they're simply,

$$\text{SPT:} \qquad w_i = \begin{cases} -1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.14a}$$

$$\text{LPT:} \qquad w_i = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.14b}$$

$$\text{MWR:} \qquad w_i = \begin{cases} 1 & \text{if } i = 7 \\ 0 & \text{otherwise} \end{cases} \tag{2.14c}$$

$$\text{LWR:} \qquad w_i = \begin{cases} -1 & \text{if } i = 7 \\ 0 & \text{otherwise} \end{cases} \tag{2.14d}$$

Automated discovery of CDRs

Generally the weights $\mathbf{w}$ in Eq. (2.12) are chosen by the algorithm designer a priori. A more sophisticated approach would have the algorithm discover these weights autonomously. For instance via preference-based imitation learning or evolutionary search, to be discussed in Chapter 7 and Chapter 8, respectively.

Mönch et al. (2013) stress the importance of automated discovery of DRs and named several successful such implementations in the field of semiconductor wafer fabrication facilities. However, Mönch et al. note that 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 Chen et al. (2013) 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.

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 creation of optimisation heuristics for scheduling.

With meta heuristics one can use existing DRs and use for example portfolio-based algorithm selection either based on a single instance (Gomes and Selman, 2001, Rice, 1976) or class of instances (Xu et al., 2007) 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* (Vilalta and Drissi, 2002), 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.

Kalyanakrishnan and Stone (2011) 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.

Nguyen et al. (2013) proposed a novel iterative dispatching rules 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 Nguyen et al. stress that there still remains room for improvement.

Korytkowski et al. (2013) implemented ant colony optimisation to select the best DR from a selection of 9 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 all of the DRs to choose from rather than just a single DR at a time.

Similarly, Lu and Romanowski (2013) investigate 11 SDRs for JSP to create a pool of 33 CDRs that strongly outperformed the ones they were based on. The CDRs were created with multi-contextual functions based either on machine idle time or job waiting time (similar to φ_5 and φ_{14} in Table 2.2), creating CDRs thate are a combination of those two

key features of the schedule and then the basic DRs. However, there are no combinations of the basic DR explored, only machine idle time and job waiting time.

Yu et al. (2013) used priority rules to combine 12 existing DRs 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 DRs is found.

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 aforementioned approaches introduce fairly ad-hoc solutions and there is no guarantee the optimal combination of dispatching rules were found.

2.7 Rice's framework for job-shop

Rice's framework for algorithm selection (discussed in Section 1.1) has already been formulated for job-shop (cf. Smith-Miles and Lopes (2011), Smith-Miles et al. (2009) and Paper III), as follows,

Problem space $\mathcal{P}$ is defined as the union of N problem instances consisting of processing time and ordering matrices, $\mathbf{x} = (\mathbf{p}, \boldsymbol{\sigma})$, for n-jobs and m-machines,

$$\mathcal{P} = \{\mathbf{x}_i \mid n \times m\}_{i=1}^{N} \tag{2.15}$$

Problem generators for $\mathcal{P}$ are given in Chapter 3.

Feature space $\mathcal{F}$ which was outlined in Section 2.5. Note, these are not the only possible set of features. However, the local feature, φ_1-φ_{16}, are built on the work by Smith-Miles et al. (2009) and Paper I and deemed successful in capturing the essence of a job-shop data structure;

Algorithm space $\mathcal{A}$ is simply the scheduling policies under consideration, e.g., SDRs from Section 2.4,

$$\mathcal{A} = \{\text{SPT, LPT, LWR, MWR, RND, } \dots\}. \tag{2.16}$$

Performance space $\mathcal{Y}$ is based on the resulting $C_{\max}$, defined by Eq. (2.4). The optimum makespan is denoted $C^{\pi_\star}_{\max}$, i.e., following the expert policy $\pi_\star$, and the makespan obtained from the scheduling policy $\pi \in \mathcal{A}$ under inspection by $C^{\pi}_{\max}$. Since the optimal makespan varies between problem instances the performance

measure is the following,

$$\rho = \frac{C^{\pi}_{\max} - C^{\pi_\star}_{\max}}{C^{\pi_\star}_{\max}} \cdot 100\% \tag{2.17}$$

which indicates the deviation from optimality, ρ. Thus $\mathcal{Y}$ is given as,

$$\mathcal{Y} = \left\{\rho_i\right\}_{i=1}^{N} \tag{2.18}$$

The mapping $\Upsilon : \mathcal{A} \times \mathcal{F} \mapsto \mathcal{Y}$ is the step-by-step construction heuristic in Alg. 2.

If it had grown up, it would have made a dreadfully ugly child; but it makes rather a handsome pig, I think.

Alice

3 Problem generators

Synthetic problem instances for JSP and FSP will be used throughout this dissertation. The problem spaces are detailed in the Sections 3.1 and 3.2 for JSP and FSP, respectively. Moreover, a brief summary is given in Table 3.2. Following the approach in Watson et al. (2002), difficult problem instances are not filtered out beforehand. The problem spaces for Part II are summarised in Table 3.1. Note, that the problem generators in Papers IV to VII are the same as described here.

Although real-world instances are desirable, unfortunately they are scarce, hence in some experiments, problem instances from OR-Library maintained by Beasley (1990) will be used as benchmark problems, and detailed in Section 3.3.

Table 3.1: JSP and FSP problems spaces used in Part II

Paper	Problem	$I = [u_1, u_2]$*	size $(n \times m)$	name
I	JSP	$[1, 100], [50, 100]$	6×6	j.rnd, j.rndn
III	JSP	$[1, 200]$	6×6	j.rnd
IV	JSP, FSP	$[1, 99], [45, 55]$	$6 \times 5, 10 \times 10$	j.rnd, j.rndn, f.rnd, f.rndn, f.jc
V	JSP	$[1, 99], [45, 55]$	6×5	j.rnd, j.rndn
VI	JSP, FSP	$[1, 99], [45, 55]$	10×10	j.rnd, j.rndn, f.rnd
VII	JSP, FSP	$[1, 99], [45, 55]$	$6 \times 5, 10 \times 10$	j.rnd, j.rndn, f.rnd

*Processing times are uniformly distributed from an interval $I = [u_1, u_2]$, i.e., $\mathbf{p} \sim \mathcal{U}(u_1, u_2)$.

It is noted, that some of the instances are also simulated, but the majority are based on real-world instances, albeit sometimes simplified.

3.1 Job-shop

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. Two different processing times distributions were explored, namely,

JSP random $\mathcal{P}^{n\times m}_{j.rnd}$
where $\mathbf{p} \sim \mathcal{U}(1, 99)$;

JSP random-narrow $\mathcal{P}^{n\times m}_{j.rndn}$
where $\mathbf{p} \sim \mathcal{U}(45, 55)$.

The machine ordering is a random permutation of all of the machines in the job-shop. For each JSP class N_{train} and N_{test} instances were generated for training and testing, respectively. Values for N are given in Table 3.2.

Although in the case of $\mathcal{P}^{n\times m}_{j.rnd}$ this may be an excessively large range for the uniform distribution, it is however, chosen in accordance with the literature (Demirkol et al., 1998) for creating synthesised $Jm||C_{\max}$ problem instances.

In order to inspect the impact of any slight change within the problem spaces, two mutated versions were created based on $\mathcal{P}^{n\times m}_{j.rnd}$, namely,

JSP random with job variation $\mathcal{P}^{n\times m}_{j.rnd,J_1}$
where the first job, J_1, is always twice as long as its random counterpart, i.e., $\tilde{p}_{1a} = 2 \cdot p_{1a}$, where $p \in \mathcal{P}^{n\times m}_{j.rnd}$, for all $M_a \in \mathcal{M}$.

JSP random with machine variation $\mathcal{P}^{n\times m}_{j.rnd,M_1}$
where the first machine, M_1, is always twice as long as its random counterpart, i.e., $\tilde{p}_{j1} = 2 \cdot p_{j1}$, where $p \in \mathcal{P}^{n\times m}_{j.rnd}$, for all $J_j \in \mathcal{J}$.

Therefore making job J_1 and machine M_1 bottlenecks for $\mathcal{P}^{n\times m}_{j.rnd,J_1}$ and $\mathcal{P}^{n\times m}_{j.rnd,M_1}$, respectively.

Hildebrandt et al. (2010) argue 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-dates, yielding schedules that are overloading in the beginning phases. However, as stated in Chapter 2, release-dates constraints won't be considered here. 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 $Jm|2\text{sets}|C_{\max}$ problems, but as discussed in Storer et al. (1992) 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 $Jm||C_{\max}$ set-up. This is in stark contrast to Watson et al. (2002) whose findings showed that structured $Fm||C_{\max}$ were much 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. Whereas, structured problem subclasses will be explored for FSP.

3.2 Flow-shop

Problem instances for FSP are generated using Watson et al. (2002) problem generator*. There are two fundamental types of problem classes: non-structured versus structured.

Firstly, there are two 'conventional' random, i.e., non-structured, problem classes for FSP where processing times are i.i.d. and uniformly distributed,

FSP random $\mathcal{P}^{n\times m}_{f.rnd}$
where $\mathbf{p} \sim \mathcal{U}(1, 99)$ whose instances are equivalent to Taillard (1993)**;

FSP random narrow $\mathcal{P}^{n\times m}_{f.rndn}$
where $\mathbf{p} \sim \mathcal{U}(45, 55)$.

In the JSP context $\mathcal{P}^{n\times m}_{f.rnd}$ and $\mathcal{P}^{n\times m}_{f.rndn}$ are analogous to $\mathcal{P}^{n\times m}_{j.rnd}$ and $\mathcal{P}^{n\times m}_{j.rndn}$, respectively.

Secondly, there are three structured problem classes of FSP which are modelled after real-world *characteristics* in flow-shop manufacturing, namely,

FSP job-correlated $\mathcal{P}^{n\times m}_{f.jc}$
where $\mathbf{p}$ is dependent on job index, however, independent of machine index.

FSP machine-correlated $\mathcal{P}^{n\times m}_{f.mc}$
where $\mathbf{p}$ is dependent on machine index, however, independent of job index.

FSP mixed-correlated $\mathcal{P}^{n\times m}_{f.mxc}$
where $\mathbf{p}$ is dependent on machine and job indices.

*Both code, written in C++, and problem instances used in their experiments can be found at: `http://www.cs.colostate.edu/sched/generator/`

**Taillard's generator is available from the OR-Library.

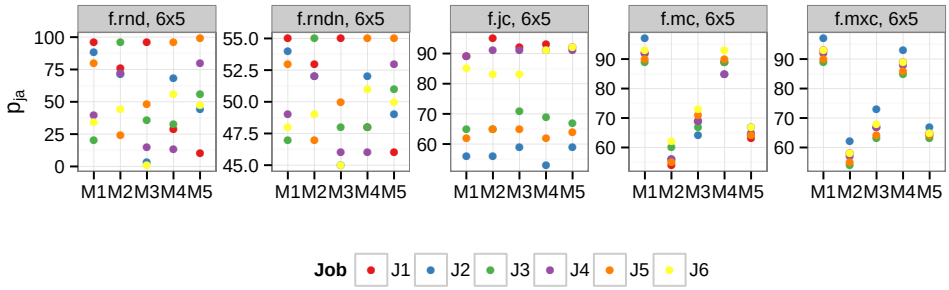


Figure 3.1: Examples of job processing times for 6×5 of different FSP structures.

In all cases, the (job, machine or mixed) correlation can be of degree $0 \leq a \leq 1$. When $a = 0.0$ the problem instances closely correspond to $\mathcal{P}^{n\times m}_{f.rnd}$, hence the degree of a controls the transition of random to structured. Let's assume $a = 1$.

An example of distribution of processing times are depicted in Fig. 3.1, where machine indices are on the horizontal axis, job indices are colour-coded, and their corresponding processing times, p_{ja}, are on the vertical axis.

For each FSP class N_{train} and N_{test} instances were generated for training and testing, respectively. Values for N are given in Table 3.2.

3.3 Benchmark problem suite

A total of 82 and 31 benchmark problems for JSP and FSP, respectively, were obtained from the Operations Research Library (OR-Library) maintained by Beasley (1990) and summarised in Table 3.3. Given the high problem dimensions of some problems, the optimum is not known, hence in those instances Eq. (2.17) will be reporting deviation from the latest best known solution (BKS) from the literature, reported by Banharnsakun et al. (2012), Jain and Meeran (1999), and for FSP consult Ancău (2012).

Job-shop OR-Library

Fisher and Thompson (1963) had one of the more notorious benchmark problems for JSP, and computationally expensive. However, now these instances have been solved to optimality. Similar to the synthetic JSP problem spaces discussed earlier, Adams et al. (1988) introduce five JSP instances with a random machine ordering and processing times $\mathbf{p} \sim \mathcal{U}(50, 100)$, for dimensions 10×10 and 20×15. Likewise, Yamada and Nakano (1992) consists of four 20×20 random problem instances, where $\mathbf{p} \sim \mathcal{U}(10, 50)$. Storer

Table 3.2: Problem space distributions used in experimental studies.

	name	size $(n \times m)$	N_{train}	N_{test}	note
JSP	$\mathcal{P}^{6\times5}_{j.rnd}$	6×5	500	500	random
	$\mathcal{P}^{6\times5}_{j.rndn}$	6×5	500	500	random-narrow
	$\mathcal{P}^{6\times5}_{j.rnd,J_1}$	6×5	500	500	random with job variation
	$\mathcal{P}^{6\times5}_{j.rnd,M_1}$	6×5	500	500	random with machine variation
	$\mathcal{P}^{10\times10}_{j.rnd}$	10×10	300	200	random
	$\mathcal{P}^{10\times10}_{j.rndn}$	10×10	300	200	random-narrow
	$\mathcal{P}^{10\times10}_{j.rnd,J_1}$	10×10	300	200	random with job variation
	$\mathcal{P}^{10\times10}_{j.rnd,M_1}$	10×10	300	200	random with machine variation
FSP	$\mathcal{P}^{6\times5}_{f.rnd}$	6×5	500	500	random
	$\mathcal{P}^{6\times5}_{f.rndn}$	6×5	500	500	random-narrow
	$\mathcal{P}^{6\times5}_{f.jc}$	6×5	500	500	job-correlated
	$\mathcal{P}^{6\times5}_{f.mc}$	6×5	500	500	machine-correlated
	$\mathcal{P}^{6\times5}_{f.mxc}$	6×5	500	500	mixed-correlation
	$\mathcal{P}^{10\times10}_{f.rnd}$	10×10	300	200	random

et al. (1992) introduce a set of JSP problems where $\mathbf{p} \sim \mathcal{U}(1, 100)$. There are a total of five problems in four dimension classes: *i)* 20×10; *ii)* 20×15; *iii)* 50×10, and *iv)* 50×10. Where the first three classes are considered 'hard' and the last one as 'easy'. Easy problems are ones corresponding to random machine ordering, whereas hard problems are partitioned in such a way the jobs must be processed on the first half of the machines before starting on the second half, i.e., $Jm|2\text{sets}|C_{\max}$. Applegate and Cook (1991) introduced ten problem instances of 10×10 JSP where generated such that the machine ordering was chosen by random users in order to make them 'difficult.' Moreover, the processing times were drawn at random, and the distribution that had the greater gap between its optimal value and standard lower bound was chosen.

Table 3.3: Benchmark problems from OR-Library used in experimental studies.

	name	$n \times m$	N_{test}	note	shorthand
JSP	$\mathcal{P}_{ft}$	various	3	Fisher and Thompson (1963)	`ft06,ft10,ft20`
	$\mathcal{P}_{la}$	various	40	Lawrence (1984)	`la01-la40`
	$\mathcal{P}_{abz}$	various	5	Adams et al. (1988)	`abz05-abz09`
	$\mathcal{P}_{orb}$	10×10	10	Applegate and Cook (1991)	`orb01-orb10`
	$\mathcal{P}_{swv}$	various	20	Storer et al. (1992)	`swv01-swv20`
	$\mathcal{P}_{yn}$	20×20	4	Yamada and Nakano (1992)	`yn01-yn04`
FSP	$\mathcal{P}_{car}$	various	8	Carlier (1978)	`car1-car8`
	$\mathcal{P}_{hel}$	various	2	Heller (1960)	`hel1,hel2`
	$\mathcal{P}_{reC}$	various	21	Reeves (1995)*	`reC01-reC42`

*Only odd-numbered instances in `rec01-rec42` are given, since the even-numbered instances are obtained from the previous instance by just reversing the processing order of each job; the optimal value of each odd-numbered instance and its even-numbered counterpart is the same.

Flow-shop OR-Library

For the FSP benchmarks, Heller (1960) introduces two deterministic instances based on 'many-machine version of book-printing,' where processing times for $n \in \{20, 100\}$ jobs and $m = 10$ machines are relatively short, i.e., $p_{ja} \in \{0, .., 9\}$. Carlier (1978) however, comprises of eight problems (of various dimension) where there is high variance in processing times, presumably $\mathbf{p} \sim \mathcal{U}(1, 1000)$. Reeves (1995) argue that completely random problem instances are unlikely to occur in practice. However, only the random instances they used (type C) are reported in the OR-Library, for a total of 42 problem instances with processing times following a uniform distribution, $\mathbf{p} \sim \mathcal{U}(1, 100)$, of dimensions varying from 20×5 to 75×20, although Ancău (2012) omitted $\mathcal{P}_{reC}^{75\times 20}$ instances in their comparison.

Sentence first – verdict afterwards.

The Queen

4

Problem difficulty

Problem structure and heuristic effectiveness are closely intertwined. When investigating the relation between the two, one can research what Corne and Reynolds (2010) call *footprints*, which is an indicator how an algorithm generalises over a given instance space. This sort of investigation has also been conducted by Pfahringer et al. (2000) under the alias *landmarking*. From experiments performed by Corne and Reynolds, it is evident that one-algorithm-for-all problem instances is not ideal, in accordance with no free lunch theorem (Wolpert and Macready, 1997). An algorithm may be favoured for its best overall performance, however, it is rarely the best algorithm available over various subspaces of the instance space. Therefore, when comparing different algorithms one needs to explore how they perform w.r.t. the instance space, i.e., their footprint. That is to say, one can look at it as finding which footprints correspond to a subset of the instance space that works *well* for a given algorithm, and similarly finding which footprints correspond to a subset of the instance space that works *poorly* for a given algorithm.

In the context of job-shop this corresponds to finding *good* (makespan close to its optimum) and *bad* (makespan far off its optimum) schedules. Note, good and bad schedules are interchangeably referred to as *easy* and *hard* schedules (pertaining to the manner they are achieved), respectively.

Smith-Miles and Lopes (2011) also investigate algorithm performance in instance space using footprints. The main difference between Corne and Reynolds and Smith-

Miles and Lopes is how they discretise the instance space. In the case of Corne and Reynolds they use job-shop and discretise manually between different problem instances; on one hand w.r.t. processing times, e.g., $\mathbf{p} \sim \mathcal{U}(10, 20)$ versus $\mathbf{p} \sim \mathcal{U}(20, 30)$ etc., and on the other hand w.r.t. number of jobs, n. They warn that footprinting can be uneven, so great care needs to be taken in how to discretise the instance space into subspaces. This is why we consider the random vs. random-narrow problem spaces in Sections 3.1 and 3.2.

On the other hand, Smith-Miles and Lopes use a completely automated approach. Using timetabling instances, they implement a self-organizing map (SOM) on the feature space to group similar problem instances together, that were both real world instances and synthetic ones using different problem generators. That way it was possible to plot visually the footprints for several algorithms.

Going back to the job-shop paradigm, then the interaction between processing time distribution and its permutation is extremely important, because it introduces hidden properties in the data structure making it *easy* or *hard* to schedule for the given algorithm. These underlying characteristics (i.e. features), define its data structure. A more sophisticated way of discretising the instance space is grouping together problem instances that show the same kind of feature behaviour, especially given the fact the learning models in Chapter 7 will be heavily based on feature pairs. Thereby making it possible to infer what sort of feature behaviour distinguishes between *good* and *bad* schedules.

Instead of searching through a large set of algorithms and determining which algorithm is the most suitable for a given subset of the instance space, i.e., creating an algorithm portfolio, as is generally the focus in the current literature (Corne and Reynolds, 2010, Smith-Miles and Lopes, 2011, Smith-Miles et al., 2009), the focus of the experimental study in the subsequent sections is rather on few simple algorithms, namely the SDRs described in Section 2.4, i.e., we will limit the algorithm space to,

$$\mathcal{A} := \{\text{SPT, LPT, LWR, MWR}\} \tag{4.1}$$

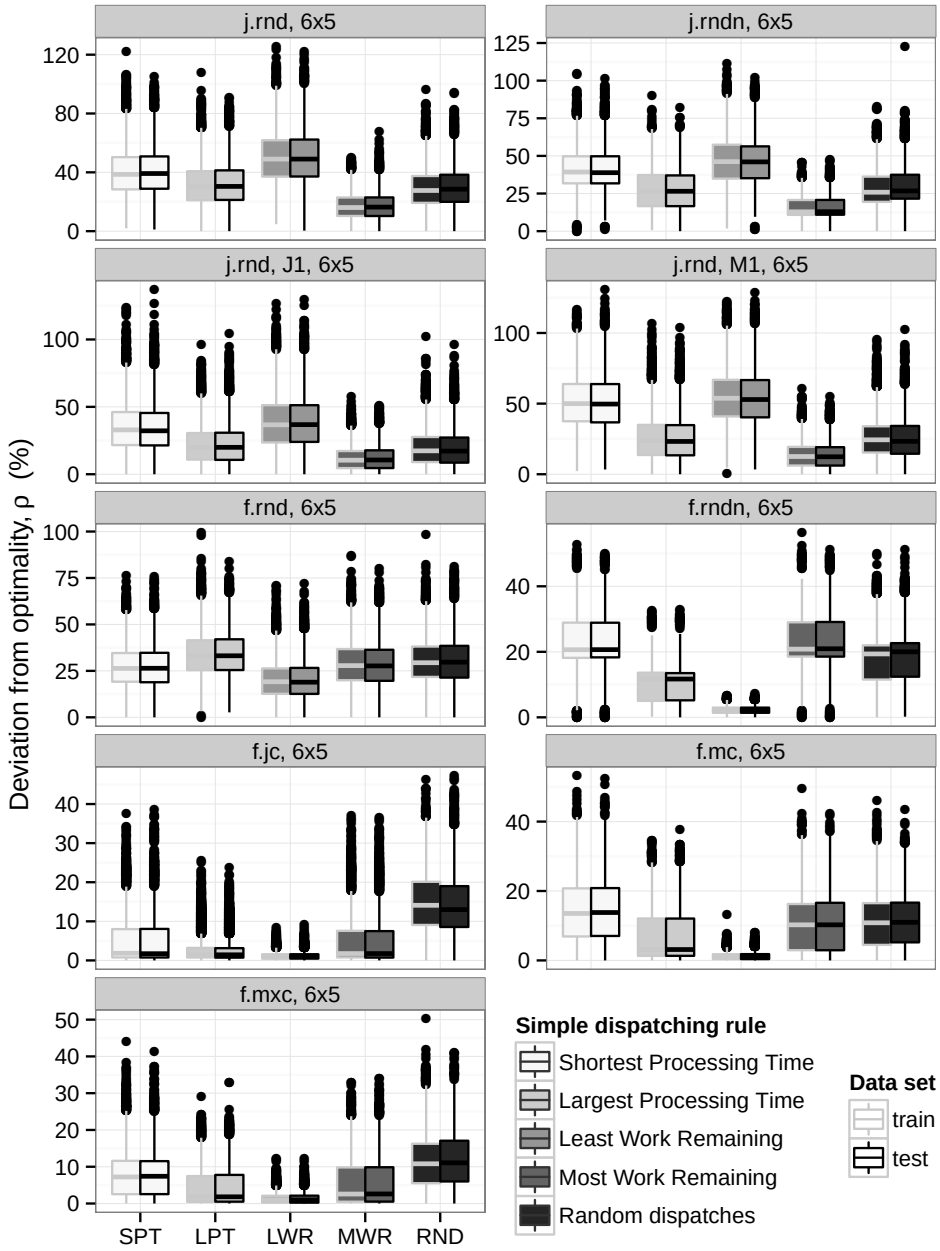
and try to understand *how* they work on the instance space, similar to Watson et al. (2002), who analysed the fitness landscape of several problem classes for a fixed algorithm.

Depending on the data distribution, dispatching rules perform differently. A box-plot for deviation from optimality, ρ, defined by Eq. (2.17), using all problem spaces from Table 3.2 are depicted in Chapter 4. As one can see, there is a staggering difference between the interaction of SDRs and their problem space. MWR is by far the best out of the four SDRs inspected for JSP – not only does it reach the known optimum most often but it also has the lowest worst-case factor from optimality. Similarly LWR for FSP.

Although the same processing time distribution is used, there are some inherent structure in which MWR and LWR can exploit for JSP and FSP, respectively, whereas the other SDRs cannot. However, *all* of these dispatching rules are considered good and commonly used in practice and no one is better than the rest (Haupt, 1989), it simply depends on the data distribution at hand. This indicates that some distributions are harder than others, and these JSP problem generators simply favours MWR, whereas the FSP problem generators favours LWR.

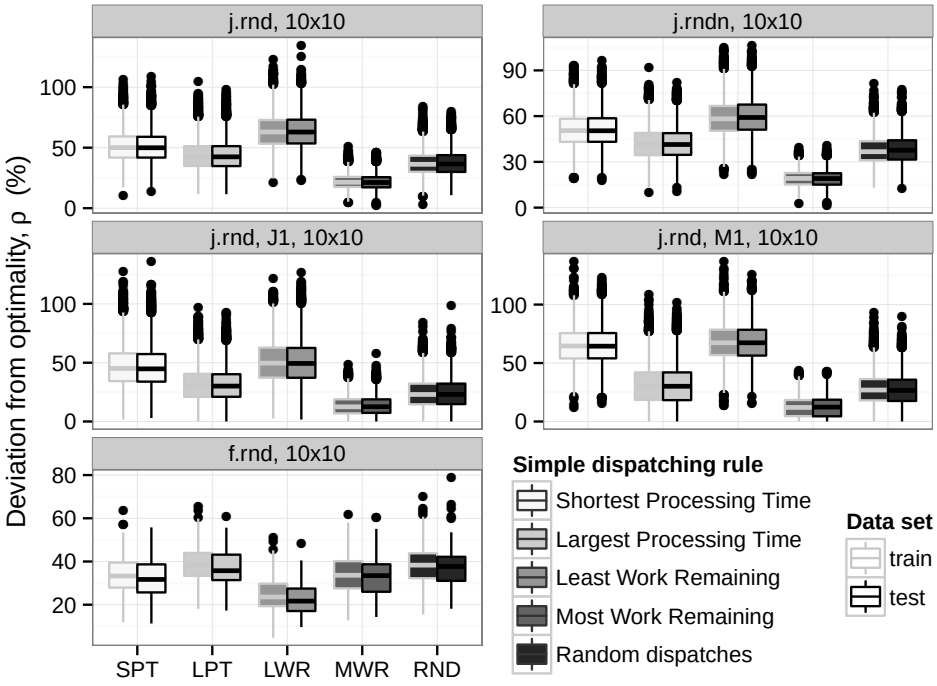
4.1 Distribution difficulty

In Paper III, a single problem generator was used to create $N = 1,500$ synthetic 6×6 job-shop problem instances, where $\mathbf{p} \sim \mathcal{U}(1, 200)$ and $\boldsymbol{\sigma}$ was a random permutation. The experimental study showed that MWR works either well or poorly on a subset of the instances, in fact 18% and 16% of the instances were classified as *easy* and *hard* for MWR, respectively. Since the problem instances were naïvely generated, not to mention given the high variance of the data distribution, it is intuitive that there are some inherent structural qualities that could explain this difference in performance. The experimental study investigated the feature behaviours for these two subsets, namely, the easy and hard problem instances. For some features, the trend was more or less the same, which are explained by the common denominating factor, that all instances were sampled from the same problem generator. Whereas, those features that were highly correlated with the end-result, i.e., the final makespan, which determined if an instance was labelled easy or hard, then the significant features varied greatly between the two difficulties, which imply the inherent difference in data structure. Moreover, the study in gives support to that random problem instance generators are *too* general and might not suit real-world applications. Watson et al. (2002) argue that problem instance generator should be more structured, since real-world manufacturing environment is not completely random, but rather structured, e.g., job's tasks can be correlated or machines in the shop. Watson et al. propose a problem instance generator that relates to real-world flow-shop attributes, albeit not directly modelled after real-world flow-shop due to the fact that deterministic $Fm||C_{\max}$ is seldom directly applicable in practice (Dudek et al., 1992). This is why $\mathcal{P}_{f.jc}^{n\times m}$, $\mathcal{P}_{f.mc}^{n\times m}$ and $\mathcal{P}_{f.mxc}^{n\times m}$ are also taken into consideration in Section 3.2.



(a) 6×5

Figure 4.1: Box-plots of deviation from optimality, ρ, when applying SDRs for all problem spaces in Chapter 3



(b) 10×10

Figure 4.1 (cont.)

4.2 Defining *easy* versus *hard* schedules

It's relatively ad-hoc how to define what makes a schedule 'difficult'. For instance, it could be sensible to define it in terms of how many Simplex iterations are needed to find an optimal schedule, using *Branch and Bound*.* However, preliminary experiments showed that an increased amount of Simplex iterations didn't necessarily transcend to high ρ. If anything, it means there are many optimal (or near-optimal) solutions available, which causes the slow process of pruning branches of the tree, before reaching to a final incumbent solution. If that's the case, than that's promising for our instance, as it's likelier for an arbitrary algorithm to find a good solution.

*Branch and bound (Land and Doig, 1960) is a methodology in integer linear programming, where the original problem is branched into smaller sub-problems until it becomes easily solvable. Each sub-problem has a lower bound on its solution, found with LP-relaxation. Depending on the lower bound, sub-branches are systemically discarded, since they cannot contain the optimal solution.

Table 4.1: Threshold for ρ for easy and hard schedules, i.e., $\rho < \rho^{1\text{st Qu.}}$ and $\rho > \rho^{3\text{rd Qu.}}$ are classified as easy and hard schedules, respectively. Based on Table 3.2 training sets.

(a) 6×5

Problems	Q1	Q3
$\mathcal{P}^{6\times5}_{j.rnd}$	19.91	47.21
$\mathcal{P}^{6\times5}_{j.rndn}$	16.63	45.01
$\mathcal{P}^{6\times5}_{j.rnd,J_1}$	11.85	38.53
$\mathcal{P}^{6\times5}_{j.rnd,M_1}$	16.35	53.19
$\mathcal{P}^{6\times5}_{f.rnd}$	18.46	35.52
$\mathcal{P}^{6\times5}_{f.rndn}$	3.39	21.07
$\mathcal{P}^{6\times5}_{f.jc}$	0.64	3.34
$\mathcal{P}^{6\times5}_{f.mc}$	1.04	13.40
$\mathcal{P}^{6\times5}_{f.mxc}$	0.46	3.67

(b) 10×10

Problems	Q1	Q3
$\mathcal{P}^{10\times10}_{j.rnd}$	29.27	58.45
$\mathcal{P}^{10\times10}_{j.rndn}$	26.74	57.17
$\mathcal{P}^{10\times10}_{j.rnd,J_1}$	17.90	50.29
$\mathcal{P}^{10\times10}_{j.rnd,M_1}$	18.00	65.79
$\mathcal{P}^{10\times10}_{f.rnd}$	26.13	39.27

Intuitively, it's logical to use the schedule's objective to define the difficulty directly, i.e., inspecting deviation from optimality, ρ. Moreover, since the SDRs from Eq. (4.1) will be used throughout as a benchmark for subsequent models, the quartiles for ρ, using the SDRs on their training set will be used to differentiate between easy and hard instances. In particular, the classification is defined as follows,

Easy schedules belong to the first quartile, i.e.,

$$\mathcal{E}(a) := \left\{ \mathbf{x} \;\middle|\; \rho = \Upsilon(a, \mathbf{x}) < \rho^{1\text{st. Qu.}} \right\} \tag{4.2}$$

Hard schedules belong to the third quartile, i.e.,

$$\mathcal{H}(a) := \left\{ \mathbf{x} \;\middle|\; \rho = \Upsilon(a, \mathbf{x}) > \rho^{3\text{rd. Qu.}} \right\} \tag{4.3}$$

where $\mathbf{x} \in \mathcal{P}_{\text{train}}$ for a given $a \in \mathcal{A}$ from Eq. (4.1). Table 4.1 reports the first and third quartiles for each problem space, i.e., the cut-off values that determine the SDRs difficulty, whose division, defined as percentage of problem instances, i.e.,

$$\frac{|\mathcal{E}(a)|}{N_{\text{train}}} \cdot 100\% \quad \text{and} \quad \frac{|\mathcal{H}(a)|}{N_{\text{train}}} \cdot 100\% \tag{4.4}$$

for each $a \in \mathcal{A}$, are given in Tables 4.2 and 4.3, respectively.

Table 4.2: Percentage (%) of 6×5 training instances classified as easy and hard schedules, defined by Eq. (4.4). Note, each problem space consists of $N_{\text{train}} = 500$.

(a) $\mathcal{P}^{6\times5}_{j.rnd}$

SDR	Easy	Hard
SPT	8.90	30.38
LPT	22.06	15.24
LWR	3.64	54.18
MWR	65.30	0.20

(b) $\mathcal{P}^{6\times5}_{j.rndn}$

SDR	Easy	Hard
SPT	2.88	37.54
LPT	24.42	9.70
LWR	2.10	52.82
MWR	70.70	0.06

(c) $\mathcal{P}^{6\times5}_{j.rnd,J_1}$

SDR	Easy	Hard
SPT	8.22	38.20
LPT	27.92	14.18
LWR	7.80	46.70
MWR	56.00	0.92

(d) $\mathcal{P}^{6\times5}_{j.rnd,M_1}$

SDR	Easy	Hard
SPT	2.28	43.08
LPT	31.68	5.72
LWR	1.10	51.12
MWR	64.96	0.10

(e) $\mathcal{P}^{6\times5}_{f.rnd}$

SDR	Easy	Hard
SPT	23.02	22.90
LPT	8.44	41.82
LWR	47.60	7.50
MWR	20.94	27.82

(f) $\mathcal{P}^{6\times5}_{f.rndn}$

SDR	Easy	Hard
SPT	0.94	44.38
LPT	13.22	7.28
LWR	85.18	0
MWR	0.48	48.42

(g) $\mathcal{P}^{6\times5}_{f.jc}$

SDR	Easy	Hard
SPT	22.14	36.44
LPT	21.52	24.08
LWR	35.64	2.80
MWR	21.38	36.70

(h) $\mathcal{P}^{6\times5}_{f.mc}$

SDR	Easy	Hard
SPT	10.64	49.20
LPT	18.46	18.98
LWR	49.04	0
MWR	21.46	31.76

(i) $\mathcal{P}^{6\times5}_{f.mxc}$

SDR	Easy	Hard
SPT	12.58	45.16
LPT	26.30	24.78
LWR	31.60	7.68
MWR	29.66	22.48

Table 4.3: Percentage (%) of 10×10 training instances classified as easy and hard schedules, defined by Eq. (4.4). Note, each problem space consists of $N_{\text{train}} = 300$.

(a) $\mathcal{P}^{10\times10}_{j.rnd}$

SDR	Easy	Hard
SPT	2.67	27.00
LPT	10.33	13.67
LWR	0.67	59.33
MWR	86.33	0

(b) $\mathcal{P}^{10\times10}_{j.rndn}$

SDR	Easy	Hard
SPT	1.00	31.67
LPT	6.67	9.33
LWR	0	59.00
MWR	92.33	0

(c) $\mathcal{P}^{10\times10}_{j.rnd,J_1}$

SDR	Easy	Hard
SPT	3.33	40.00
LPT	21.67	11.33
LWR	3.67	48.67
MWR	71.33	0

(d) $\mathcal{P}^{10\times10}_{j.rnd,M_1}$

SDR	Easy	Hard
SPT	0	44.33
LPT	25.33	3.33
LWR	0	52.33
MWR	74.67	0

(e) $\mathcal{P}^{10\times10}_{f.rnd}$

SDR	Easy	Hard
SPT	20.15	20.90
LPT	4.10	49.25
LWR	58.58	5.60
MWR	17.16	24.63

4.3 Consistency of problem instances

The intersection of pairwise SDRs being simultaneously easy or hard are given in Tables 4.4 to 4.7, i.e.,

$$\frac{|\mathcal{E}(a_i) \cap \mathcal{E}(a_j)|}{N_{\text{train}}} \cdot 100\% \quad \text{or} \quad \frac{|\mathcal{H}(a_i) \cap \mathcal{H}(a_j)|}{N_{\text{train}}} \cdot 100\% \tag{4.5}$$

where $a_i, a_j \in \mathcal{A}$. Note, when $a_i = a_j$ then Eq. (4.5) is equivalent to Eq. (4.4).

Even though this is a naïve way to inspect difference between varying SDRs, it's does give some initial insight of the potential of improving dispatching rules; a sanity check before going into extensive experiments, as will be done in Section 6.6.

For the corresponding 10×10 training set (cf. Tables 4.6 and 4.7), the intersections between SDRs from 6×5 (cf. Tables 4.4 and 4.5) seem to hold. However, by going to a higher dimension, the performance edge between SDRs becomes more apparent, e.g., in JSP when there was a slight possibility of LWR being simultaneously easy as other SDRs ($5\% <$ chance), it becomes almost impossible for 10×10. Making LWR a clear underdog. Despite that, for FSP the tables turn; now LWR has the performance edge. For instance, for $\mathcal{P}^{6\times5}_{f.rndn}$ the second best option is to apply LPT (13.22%), however, there is a quite high overlap with LWR (11.74%), and since LWR is easier significantly more often (85.18%), the choice of SDR is quite obvious. Although, it goes to show that there is the possibility of improving LWR by sometimes applying LPT-based insight; by seeing what sets apart the intersection of their easy training sets.

Similarly for every 10×10 JSP (cf. Table 4.6), almost all easy LPT schedules are also easy for MWR ($< 1\%$ difference), as is to be expected as MWR is the more sophisticated counterpart for LPT (like LWR is for SPT). However, the greedy approach here is not gaining any new information on how to improve MWR. In fact, MWR is never considered hard for any of the JSP (cf. Table 4.7), therefore no intersection with any hard schedules. But the LPT counterpart has a relatively high occurrence rate (3-14%), so due to the similarity of the dispatching rules, the denominating factor between LPT and MWR can be an indicator for explaining some of MWR's pitfalls. That is to say, why aren't all of the job-shop schedules easy when applying MWR?

Table 4.4: Percentage (%) of 6×5 training instances classified as easy simultaneously, defined by Eq. (4.5). Note, each problem space consists of $N_{\text{train}} = 500$.

(a) $\mathcal{P}^{6\times5}_{j.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	8.90	2.04	1.02	5.44
LPT	2.04	22.06	1.14	17.46
LWR	1.02	1.14	3.64	2.12
MWR	5.44	17.46	2.12	65.30

(b) $\mathcal{P}^{6\times5}_{j.rndn}$

SDR	SPT	LPT	LWR	MWR
SPT	2.88	0.82	0.34	2.12
LPT	0.82	24.42	0.54	18.96
LWR	0.34	0.54	2.10	1.46
MWR	2.12	18.96	1.46	70.70

(c) $\mathcal{P}^{6\times5}_{j.rnd,J_1}$

SDR	SPT	LPT	LWR	MWR
SPT	8.22	3.20	2.46	5.12
LPT	3.20	27.92	3.22	22.10
LWR	2.46	3.22	7.80	4.94
MWR	5.12	22.10	4.94	56.00

(d) $\mathcal{P}^{6\times5}_{j.rnd,M_1}$

SDR	SPT	LPT	LWR	MWR
SPT	2.28	0.60	0.24	1.20
LPT	0.60	31.68	0.36	26.60
LWR	0.24	0.36	1.10	0.64
MWR	1.20	26.60	0.64	64.96

(e) $\mathcal{P}^{6\times5}_{f.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	23.02	2.76	15.00	4.90
LPT	2.76	8.44	6.12	4.02
LWR	15.00	6.12	47.60	7.46
MWR	4.90	4.02	7.46	20.94

(f) $\mathcal{P}^{6\times5}_{f.rndn}$

SDR	SPT	LPT	LWR	MWR
SPT	0.94	0.30	0.88	0.06
LPT	0.30	13.22	11.74	0.16
LWR	0.88	11.74	85.18	0.36
MWR	0.06	0.16	0.36	0.48

(g) $\mathcal{P}^{6\times5}_{f.jc}$

SDR	SPT	LPT	LWR	MWR
SPT	22.14	4.24	21.44	3.88
LPT	4.24	21.52	5.78	15.38
LWR	21.44	5.78	35.64	4.62
MWR	3.88	15.38	4.62	21.38

(h) $\mathcal{P}^{6\times5}_{f.mc}$

SDR	SPT	LPT	LWR	MWR
SPT	10.64	5.28	3.74	7.96
LPT	5.28	18.46	8.16	10.08
LWR	3.74	8.16	49.04	4.34
MWR	7.96	10.08	4.34	21.46

(i) $\mathcal{P}^{6\times5}_{f.mxc}$

SDR	SPT	LPT	LWR	MWR
SPT	12.58	0.82	12.42	0.76
LPT	0.82	26.30	1.08	25.10
LWR	12.42	1.08	31.60	0.98
MWR	0.76	25.10	0.98	29.66

Table 4.5: Percentage (%) of 6×5 training instances classified as hard simultaneously, defined by Eq. (4.5). Note, each problem space consists of $N_{\text{train}} = 500$.

(a) $\mathcal{P}^{6\times5}_{j.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	30.38	5.24	21.08	0.04
LPT	5.24	15.24	9.78	0.10
LWR	21.08	9.78	54.18	0.08
MWR	0.04	0.10	0.08	0.20

(b) $\mathcal{P}^{6\times5}_{j.rndn}$

SDR	SPT	LPT	LWR	MWR
SPT	37.54	4.46	25.56	0.02
LPT	4.46	9.70	6.18	0.04
LWR	25.56	6.18	52.82	0.06
MWR	0.02	0.04	0.06	0.06

(c) $\mathcal{P}^{6\times5}_{j.rnd,J_1}$

SDR	SPT	LPT	LWR	MWR
SPT	38.20	7.34	26.46	0.40
LPT	7.34	14.18	9.10	0.46
LWR	26.46	9.10	46.70	0.48
MWR	0.40	0.46	0.48	0.92

(d) $\mathcal{P}^{6\times5}_{j.rnd,M_1}$

SDR	SPT	LPT	LWR	MWR
SPT	43.08	3.00	31.42	0.04
LPT	3.00	5.72	3.62	0
LWR	31.42	3.62	51.12	0.04
MWR	0.04	0	0.04	0.10

(e) $\mathcal{P}^{6\times5}_{f.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	22.90	11.70	3.74	6.24
LPT	11.70	41.82	5.64	16.14
LWR	3.74	5.64	7.50	1.16
MWR	6.24	16.14	1.16	27.82

(f) $\mathcal{P}^{6\times5}_{f.rndn}$

SDR	SPT	LPT	LWR	MWR
SPT	44.38	3.48	0	22.20
LPT	3.48	7.28	0	3.90
LWR	0	0	0	0
MWR	22.20	3.90	0	48.42

(g) $\mathcal{P}^{6\times5}_{f.jc}$

SDR	SPT	LPT	LWR	MWR
SPT	36.44	12.48	2.74	18.22
LPT	12.48	24.08	0.94	14.28
LWR	2.74	0.94	2.80	0.90
MWR	18.22	14.28	0.90	36.70

(h) $\mathcal{P}^{6\times5}_{f.mc}$

SDR	SPT	LPT	LWR	MWR
SPT	49.20	12.94	0	23.16
LPT	12.94	18.98	0	9.76
LWR	0	0	0	0
MWR	23.16	9.76	0	31.76

(i) $\mathcal{P}^{6\times5}_{f.mxc}$

SDR	SPT	LPT	LWR	MWR
SPT	45.16	12.24	7.48	11.34
LPT	12.24	24.78	0.52	14.10
LWR	7.48	0.52	7.68	0.26
MWR	11.34	14.10	0.26	22.48

Table 4.6: Percentage (%) of 10 × 10 training instances classified as easy simultaneously, defined by Eq. (4.5). Note, each problem space consists of $N_{\text{train}} = 300$.

(a) $\mathcal{P}^{10\times10}_{j.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	2.67	0.33	0	2.33
LPT	0.33	10.33	0	10.33
LWR	0	0	0.67	0.33
MWR	2.33	10.33	0.33	86.33

(b) $\mathcal{P}^{10\times10}_{j.rndn}$

SDR	SPT	LPT	LWR	MWR
SPT	1.00	0.33	0	1.00
LPT	0.33	6.67	0	5.00
LWR	0	0	0	0
MWR	1.00	5.00	0	92.33

(c) $\mathcal{P}^{10\times10}_{j.rnd,J_1}$

SDR	SPT	LPT	LWR	MWR
SPT	3.33	1.00	1.33	3.00
LPT	1.00	21.67	1.67	20.33
LWR	1.33	1.67	3.67	3.67
MWR	3.00	20.33	3.67	71.33

(d) $\mathcal{P}^{10\times10}_{j.rnd,M_1}$

SDR	SPT	LPT	LWR	MWR
SPT	0	0	0	0
LPT	0	25.33	0	25.00
LWR	0	0	0	0
MWR	0	25.00	0	74.67

(e) $\mathcal{P}^{10\times10}_{f.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	20.15	1.49	15.30	1.87
LPT	1.49	4.10	2.99	0.75
LWR	15.30	2.99	58.58	7.09
MWR	1.87	0.75	7.09	17.16

Table 4.7: Percentage (%) of 10 × 10 training instances classified as hard simultaneously, defined by Eq. (4.5). Note, each problem space consists of $N_{\text{train}} = 300$.

(a) $\mathcal{P}^{10\times10}_{j.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	27.00	4.67	17.67	0
LPT	4.67	13.67	9.00	0
LWR	17.67	9.00	59.33	0
MWR	0	0	0	0

(b) $\mathcal{P}^{10\times10}_{j.rndn}$

SDR	SPT	LPT	LWR	MWR
SPT	31.67	3.00	23.33	0
LPT	3.00	9.33	5.33	0
LWR	23.33	5.33	59.00	0
MWR	0	0	0	0

(c) $\mathcal{P}^{10\times10}_{j.rnd,J_1}$

SDR	SPT	LPT	LWR	MWR
SPT	40.00	7.00	27.00	0
LPT	7.00	11.33	9.67	0
LWR	27.00	9.67	48.67	0
MWR	0	0	0	0

(d) $\mathcal{P}^{10\times10}_{j.rnd,M_1}$

SDR	SPT	LPT	LWR	MWR
SPT	44.33	1.67	28.00	0
LPT	1.67	3.33	2.00	0
LWR	28.00	2.00	52.33	0
MWR	0	0	0	0

(e) $\mathcal{P}^{10\times10}_{f.rnd}$

SDR	SPT	LPT	LWR	MWR
SPT	20.90	12.31	2.61	4.85
LPT	12.31	49.25	5.22	14.93
LWR	2.61	5.22	5.60	1.49
MWR	4.85	14.93	1.49	24.63

4.4 Discussion and Conclusion

These have up until now all been speculations about how SDRs differ. One thing is for certain, the underlying problem space plays a great role on a SDR's success. Even slight variations to one job or machine, i.e., $\mathcal{P}^{10\times10}_{j.rnd,J_1}$ and $\mathcal{P}^{10\times10}_{j.rnd,M_1}$, shows significant change in performance. Due to the presence of bottleneck, MWR is able to detect it and thus becomes the clear winner. Even outperforming the original $\mathcal{P}^{10\times10}_{j.rnd}$ which they're based on, despite having processing times doubled for a single job or machine, with approximately 10% lower first quartile (cf. Table 4.1b) in both cases.

As the objective of this dissertation is not to choose which DR is best to use for each problem instance. The focus is set on finding what characterises of job-shop overall, are of value (i.e. feature selection), and create a new model that works well for the problem space to a great degree. Namely, by exploiting feature behaviour that is considered more favourable. The hypothesis being that features evolutions of easy schedules greatly differ from features evolutions corresponding to hard schedules, and Section 6.6 will attempt to explain the evidence show in Tables 4.2 to 4.7.

4.4. Make sure Section 6.6 reference Tables 4.2 to 4.7.

Note, this section gave the definition of what constitutes an 'easy' and 'hard' schedule. Since these are based on four SDRs (cf. Eq. (4.1)) the training data for the experiments done in this chapter is based on $4N_{\text{train}}$ problem instances, per problem space, therefore,

$$\sum_{a\in\mathcal{A}}|\mathcal{E}(a)| \approx N_{\text{train}} \quad \text{and} \quad \sum_{a\in\mathcal{A}}|\mathcal{H}(a)| \approx N_{\text{train}} \tag{4.6}$$

due to the fact Eqs. (4.2) and (4.3) are based on the first and third quartiles of the entire training set. Now, as the SDRs vary greatly in performance, the contribution of a SDR to Eq. (4.6) varies, resulting in an unbalanced sample size when restricted to a single SDR.

Despite problem instances being created by the same problem generator, they vary among one another enough. As a result, all instances are not created equal; some are always hard to solve, others always easy. Since the description of the problem space isn't enough to predict its performance, we need a measure to understand what's going on. Why are some instances easier to find their optimum (or close enough)? That is to say, what's their secret? This is where their feature evolution comes into play. By using schedules obtained by applying SDRs we have the ability to get some insight into the matter.

Well! I've often seen a cat without a grin; but a grin without a cat!
It's the most curious thing I ever say in my life!

Alice

5 Generating Training Data

When building a complete job-shop schedule, $K = n \cdot m$ dispatches must be made sequentially. A job is placed at the earliest available time slot for its next machine, whilst still fulfilling constraints Ineqs. (2.2) and (2.3). Unfinished jobs are dispatched one at a time according to some heuristic, or policy π. After each dispatch* the schedule's current features (cf. Table 2.2) are updated based on the half-finished schedule. Namely, when implementing Alg. 2, a training set will consist of all features from Table 2.2 at every post-decision state visited in line 6. These collected features are denoted Φ, where,

$$\Phi := \bigcup_{i=1}^{N_{\text{train}}} \bigcup_{k=1}^{K} \bigcup_{J_j \in \mathcal{L}^{(k)}} \left\{ \boldsymbol{\varphi}_j \;\middle|\; \mathbf{x}_i \in \mathcal{P}_{\text{train}}^{n \times m} \right\}. \tag{5.1}$$

5.1 Job-shop tree representation

Continuing with the example from Section 2.3, Fig. 5.1 shows how the first two dispatches could be executed for a 4×5 job-shop from Section 2.3. In the top layer one can see an empty schedule. In the middle layer one of the possible dispatches from the layer above is fixed, and one can see the resulting schedule, i.e., what are the next possible dispatches

*The terms dispatch (iteration) and time step are used interchangeably.

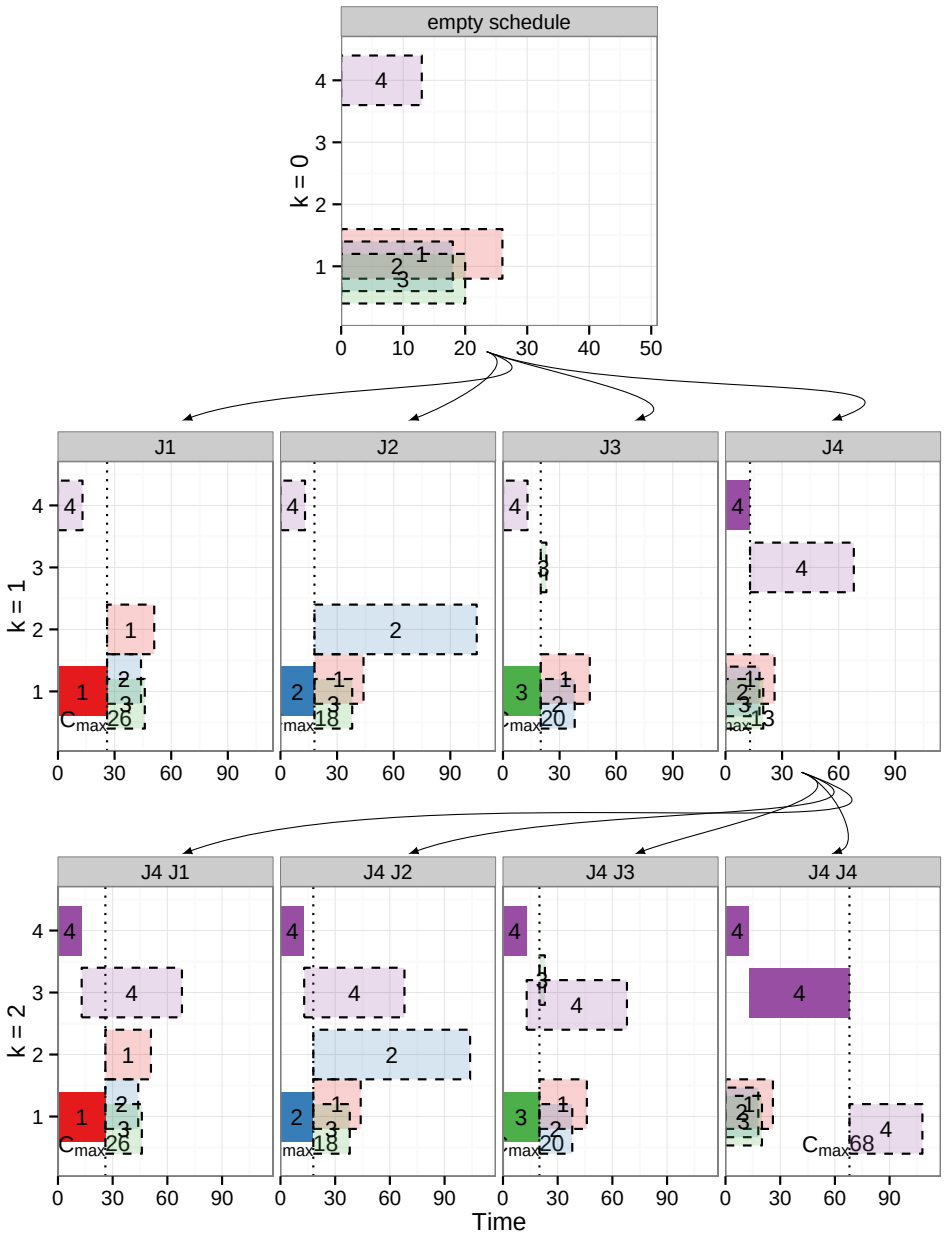


Figure 5.1: Partial Game Tree for job-shop for the first two dispatches. Top layer depicts all possible dispatches (dashed) for an empty schedule. Middle layer depicts all possible dispatches given that one of the dispatches from the layer above has been executed (solid). Bottom layer depicts when job J_4 on machine M_4 has been chosen to be dispatched from the previous layer, moreover it depicts all possible next dispatches from that scenario.

given this scenario? Assuming J_4 would be dispatched first, the bottom layer depicts all the next possible partial schedules.

This sort of tree representation is similar to *game trees* (cf. Rosen, 2003) where the root node denotes the initial (i.e. empty) schedule and the leaf nodes denote the complete schedule (resulting after $n \cdot m$ dispatches, thus height of the tree is K), therefore the distance k from an internal node to the root yields the number of operations already dispatched. Traversing from root to leaf node one can obtain a sequence of dispatches that yielded the resulting schedule, i.e., the sequence indicates in which order the tasks should be dispatched for that particular schedule.

5.2 Labelling schedules w.r.t. optimal decisions

One can easily see that sequence $\boldsymbol{\chi}$ from Eq. (2.8) for task assignments is by no means unique. Inspecting a partial schedule further along in the dispatching process such as in Fig. 2.3, then let's say J_2 would be dispatched next, and in the next iteration J_4. Now this sequence would yield the same schedule as if J_4 would have been dispatched first and then J_2 in the next iteration. This is due to the fact they have non-conflicting machines, which indicates that some of the nodes in game tree can merge. Meanwhile, the states of the schedule are different and thus their features, although they manage to yield with the same (partial) schedule at a later date. In this particular instance one can not infer that choosing J_2 is better and J_4 is worse (or vice versa) since they can both yield the same solution.

Furthermore, 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. In addition, varying permutations of 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 Eq. (2.17) (which is the measure under consideration). Care must be taken in this case that neither resulting features are labelled as undesirable. Only the features from a dispatch yielding a truly suboptimal solution should be labelled undesirable.

5.3 Computational growth

The creation of the game tree for job-shop can be done recursively for all possible permutation of dispatches, in the manner described above, resulting in a full n-ary tree (since $|\mathcal{L}| \leq n$) of height K. Such an exhaustive search would yield at the most n^K leaf nodes (worst case scenario being that no sub-trees merge). Now, since the internal vertices, i.e.,

partial schedules, are only of interest to learn,* the number of those can be at the most $^{n^K-1}/_{n-1}$. Even for small dimensions of n and m the number of internal vertices are quite substantial and thus too computationally expensive to investigate them all. Not to mention that this is done iteratively for all N_{train} problem instances.

Since we know that once a job is processed on all of its machines, then it stops being a contender for future dispatches, therefore the all possible assignments of operations for an $n \times m$ JSP would require an examination of $(n!)^m$ (Giffler and Thompson, 1960), thus a 6×5 problem may have at most $1.93 \cdot 10^{14}$ possible solutions, and for 10×10 problem then it's $3.96 \cdot 10^{65}$ solutions! Thus the factorial growth makes it infeasible for exploring all nodes to completion. However, our training data consist of relatively large N_{train}, so even though we will only pursue one trajectory per instance, then the aggregated training data will give it variety.

5.4 Trajectory sampling strategies

For each feature in Eq. (5.1) we need to keep track of the resulting makespan for its dispatched job. As a result, we obtain the meta-data $\{\Phi, \mathcal{Y}\}$ from Fig. 1.1 as follows,

$$\left\{ \{\boldsymbol{\varphi}_j, C^{\mathbf{x}^j}_{\max}\} \;\middle|\; J_j \in \mathcal{L}^{(k)} \right\}_{k=1}^{K} \in \Phi \times \mathcal{Y} \tag{5.2}$$

for a single problem instance $\mathbf{x} \in \mathcal{P}_{\text{train}}$.

Due to superabundant possible solutions for a single problem instance, there needs to be some logic based on how to sample the state-space for a valuable outcome. Especially considering the cost of correctly labelling** each dispatch that is encountered.† Obviously we'd like to inspect optimal solutions as they are what we'd like to mimic. Moreover, since we'd like to infer the footprints in instance space for the SDRs we started doing in Chapter 4, then we will consider them also. Similarly, the weights for Eq. (2.12) that were optimised directly using from evolutionary search (cf. Chapter 8) will also be used.

To clarify, the trajectory sampling strategies for collecting a feature set and its corresponding labelling for Eq. (5.2) are the following:

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

**Optimal solutions can be obtained by using a commercial software package by Gurobi Optimization, Inc. (2014), which has a free academic licence. However, GLPK by Free Software Foundation, Inc. (2014) has a free licence. Alas, GLPK has a lacklustre performance w.r.t. speed for solving 10×10 JSP.

†Generally it takes only several hours to collect $N^{6\times5}_{\text{train}} = 500$. Alas, when going to higher dimension, $N^{10\times10}_{\text{train}} = 300$ really becomes an issue, as $\mathcal{P}^{10\times10}_{j.rnd}$ needs a few days, and $\mathcal{P}^{10\times10}_{j.rndn}$ or $\mathcal{P}^{10\times10}_{f.rnd}$ require several weeks!

Optimum trajectory, $\Phi^{\mathbf{OPT}}$, at each dispatch some (random) optimal task is dispatched. This is also referred to following the expert policy, $\pi_\star$.

SPT trajectory, $\Phi^{\mathbf{SPT}}$, at each dispatch the task corresponding to shortest processing time is dispatched, i.e., following single priority dispatching rule SPT.

LPT trajectory, $\Phi^{\mathbf{LPT}}$, at each dispatch the task corresponding to largest processing time is dispatched, i.e., following single priority dispatching rule LPT.

LWR trajectory, $\Phi^{\mathbf{LWR}}$, at each dispatch the task corresponding to least work remaining is dispatched, i.e., following single priority dispatching rule LWR.

MWR trajectory, $\Phi^{\mathbf{MWR}}$, at each dispatch the task corresponding to most work remaining is dispatched, i.e., following single priority dispatching rule MWR.

Random trajectory, $\Phi^{\mathbf{RND}}$, at each dispatch some random task is dispatched.

CMA-ES trajectory, $\Phi^{\mathbf{CMA}}$, at each dispatch the task corresponding to highest priority, computed with fixed weights $\mathbf{w}$, which were obtained by optimising the mean for deviation from optimality, ρ, defined by Eq. (2.17), with CMA-ES.

All trajectories, $\Phi^{\mathbf{ALL}}$, denotes all aforementioned trajectories were explored, i.e.,

$$\Phi^{\mathrm{ALL}} := \left\{ \Phi^A \;\middle|\; \forall A \in \left\{ \mathrm{OPT}, \mathrm{CMA}, \mathrm{SPT}, \mathrm{LPT}, \mathrm{LWR}, \mathrm{MWR}, \mathrm{RND} \right\} \right\}. \quad (5.3)$$

When following optimal trajectory, then due to the nature of the sequence representation (i.e. $\boldsymbol{\chi}$), the earlier stages for $\mathcal{P}_{j.rnd}$ of the dispatching are more or less equivalent and thus irrelevant (cf. Fig. 6.3). Hence it is appropriate to follow some random optimal path to begin with and then go after some (if not all possible) optimal paths until completion at step K.

In the case of the $\Phi^{\langle \mathrm{SDR} \rangle}$ trajectories it is sufficient to explore each trajectory exactly once for each problem instance. Whereas, for Φ^{OPT} and Φ^{RND} there can be several trajectories worth exploring, however, only one is chosen (at random). It is noted that since the number of problem instances, N_{train}, is relatively large, it is deemed sufficient to explore one trajectory for each instance, in those cases as well.

These trajectory strategies were initially introduced in Paper V. However, more SDR-based trajectories are now addressed since, e.g., LWR is considered more favourable for flow-shop rather than MWR (cf. Chapter 4).

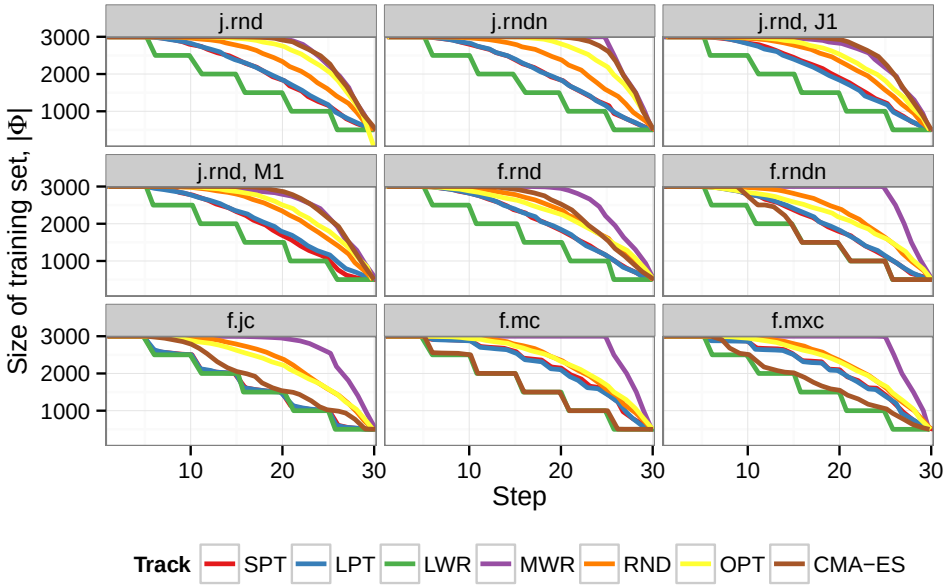


Figure 5.2: Size of training set, $|\Phi|$, for different trajectory strategies ($N_{\text{train}} = 500$)

The number of features that were collected on a step-by-step basis for $\mathcal{P}^{6\times5}_{\text{train}}$ in Table 3.2 is illustrated in Fig. 5.2. There is an apparent stair-like structure for LWR, in accordance with its motivation (cf. Section 2.4), which is completing jobs advanced in their progress, that is to ay minimising $\mathcal{L}$ and from Eq. (5.1) we have $|\Phi| \propto |\mathcal{L}|$. Whereas MWR tries to keep the jobs more balanced, hence more steady $|\mathcal{L}|$, until at $k > (K - n)$ then $|\mathcal{L}| \lesssim (K - k)$, which explains the sharp decent near the end for MWR. Table 5.1 gives the total size for $|\Phi|$, which gives the number of optimisations needed for obtaining $\mathcal{Y}$.

Table 5.1: Total number of features in Φ for all K steps

(a) $\mathcal{P}^{6\times5}_{\text{train}}$, $N_{\text{train}} = 500$

Track	j.rnd	j.rndn	j.rnd,J1	j.rnd,M1	f.rnd	f.rndn	f.jc	f.mc	f.mxc
SPT	63197	63074	64560	61320	63287	63123	53678	66995	66216
LPT	63516	63374	63595	62864	63535	63320	53746	66356	65662
LWR	52500	52500	52500	52500	52500	52500	52500	52500	52500
MWR	79230	82500	78327	77934	79288	82500	80546	82498	82485
RND	71390	71608	71445	71463	71427	71945	71558	71456	71649
OPT	76592	78176	74109	74069	70037	69180	69716	71602	71102
CMA	79319	81423	79070	78501	72866	56428	60049	52710	57177

(b) $\mathcal{P}^{10\times10}_{\text{train}}$, $N_{\text{train}} = 300$

Track	j.rnd	j.rndn	f.rnd
SPT	211351	–	–
LPT	210490	–	–
LWR	165000	–	–
MWR	280739	–	–
RND	252515	–	–
OPT	272858	245313	211763

I don't believe there's an atom of meaning in it.

Alice

6

Analysing Solutions

It is interesting to know if the difference in the structure of the schedule is time dependent, e.g., is there a clear time of divergence within the scheduling process? Moreover, investigation of how sensitive is the difference between two sets of features, e.g., can two schedules with similar feature values yield: *i)* completely contradictory outcomes (i.e. one poor and one good schedule)? Or *ii)* will they more or less follow the their predicted trend? If the latter is the prevalent case, then instances need to be segregated w.r.t. their difficulty, where each has their own learning algorithm implemented, for a meaningful outcome overall.

This also, essentially, answers the question of whether it is in fact feasible to discriminate between *good* and *bad* schedules using the currently selected features as a measure of the quality of a solution. If results are contradictory, it is an indicator the features selected are not robust enough to capture the essence of the data structure and some key features are missing from the feature set that could be able to discriminate between *good* and *bad* schedules. Additionally, there is also the question of how can one define 'similar' schedules, and what measures should be used? This chapter describes some preliminary experiments with the aim of investigating the feasibility of finding distinguishing features corresponding to *good* and *bad* schedules in job-shop. To summarise: *i)* is there a time of divergence? *ii)* what are 'similar' schedules? *iii)* do similar features yield contradictory outcomes? *iv)* are extra features needed? And *v)* what can be learned from feature behaviour?

Remark: Figures 6.1 and 6.2 depict the mean over all the training data, which are quite noisy functions. Paper VI depicts the mean as is, albeit only for 10×10 problem spaces. Thus, for clarity purposes, they are fitted with local polynomial regression, making the boundary points sometimes biased. However, Figs. 6.3 to 6.5 and 6.7 depict the mean as is.

6.1 Making optimal decisions

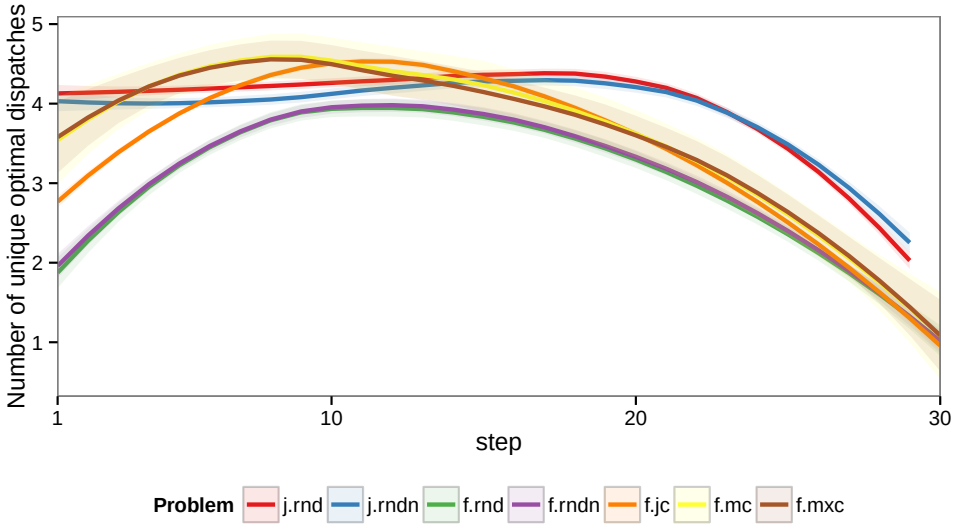
In order to create successful dispatching rule, a good starting point is to investigate the properties of optimal solutions and hopefully be able to learn how to mimic such 'good' behaviour. For this, we follow an optimal solution (cf. Φ^{OPT} in Section 5.4), and inspect the evolution of its features (defined in Table 2.2) throughout the dispatching process, which is detailed in Chapter 5. 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_{train} independent instances which gives the training data variety.

Firstly, we can observe that on a step-by-step basis there are several optimal dispatches to choose from. Figure 6.1 depicts how the number of optimal dispatches evolve at each dispatch iteration. Note, that only one optimal trajectory is pursued (chosen at random), hence this is only a lower bound of uniqueness of optimal solutions. As the number of possible dispatches decrease over time, Fig. 6.2 depicts the probability of choosing an optimal dispatch at each iteration.

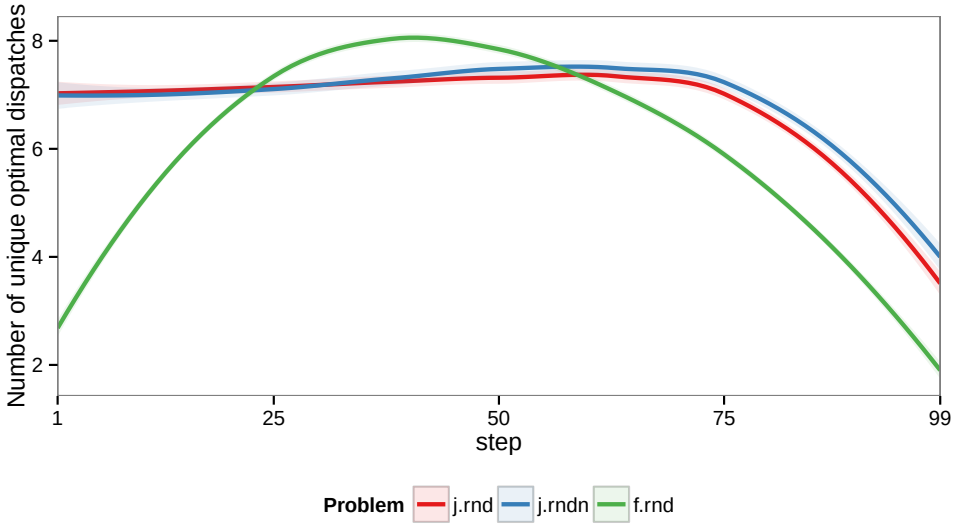
6.2 Making suboptimal decisions

Looking at Fig. 6.2, $\mathcal{P}^{10\times10}_{j.rnd}$ 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 interaction, Fig. 6.3 depicts the worst and best case scenario of 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 many mistakes made, and then the compound effects of making suboptimal decisions really start adding up. In fact, Fig. 6.5 shows the probability of optimality when following a fixed SDR.

It is interesting that for JSP, that over time making suboptimal decisions make more of an impact on the resulting makespan. This is most likely due to the fact that if a 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

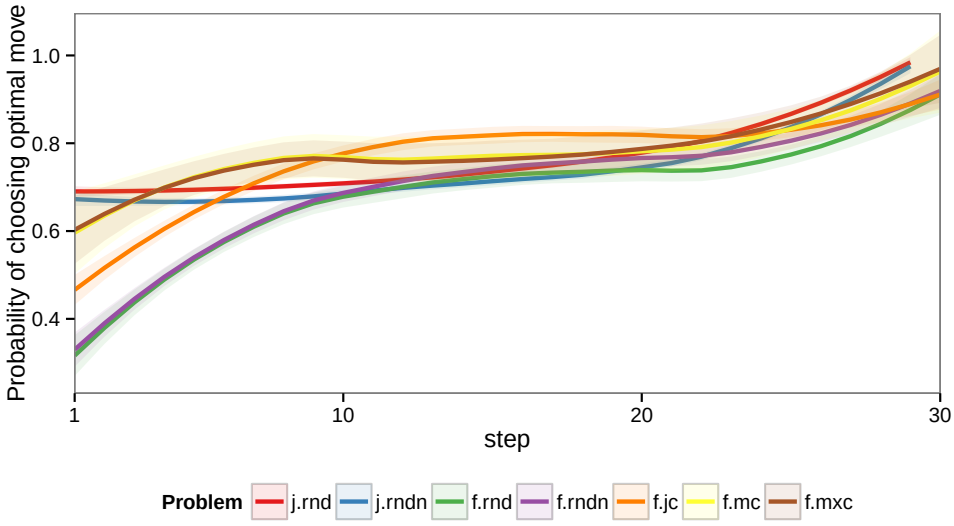


(a) 6×5

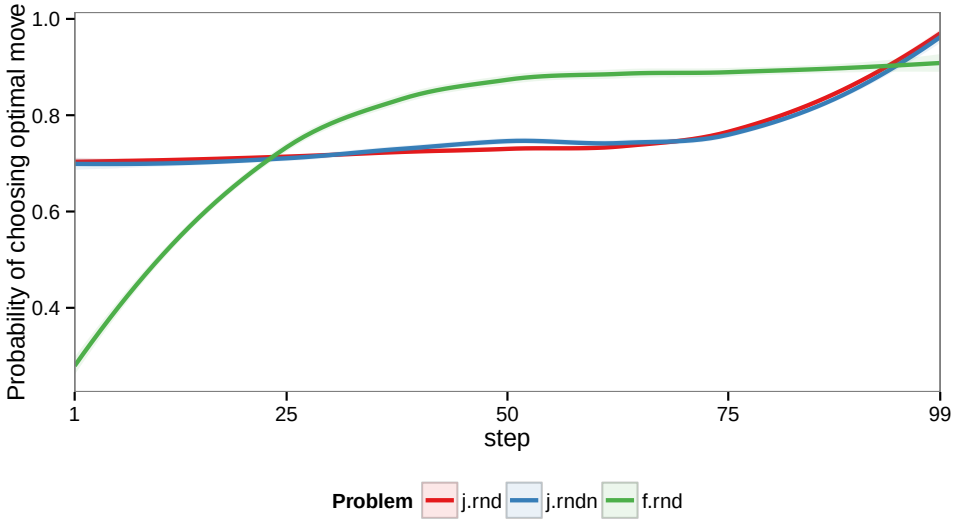


(b) 10×10

Figure 6.1: Number of unique optimal dispatches (lower bound).



(a) 6×5



(b) 10×10

Figure 6.2: Probability of choosing optimal move (at random)

the damage is already been inflicted upon the schedule. However, for FSP, the case is the exact opposite. Under those circumstances it's imperative to make good decisions right from the get-go. This is due to the major structural differences between job-shop and flow-shop, namely the latter having a homogeneous machine ordering, constricting the solution immensely. Luckily, this does have the added benefit of making flow-shop less vulnerable for suboptimal decisions later in the decision process.

6.3 Optimality of extremal features

The probability of optimality of the aforementioned features from Table 2.2, yet still maintaining our optimal trajectory, i.e., the probability of a job chosen by an extremal value for a feature being able to yield an optimal makespan on a step-by-step basis, is depicted in Fig. 6.4, for both $\mathcal{P}^{6\times5}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rnd}$. Moreover, the dashed line represents the benchmark of randomly guessing the optimum (cf. Fig. 6.2). Furthermore, the figures are annotated with the corresponding mean deviation from optimality, ρ, for the training set if it were scheduled solely w.r.t. that extremal feature.

Generally, a high stepwise optimality means a low ρ, e.g., φ_{17}-φ_{24}, save for φ_{22}.* Unfortunately, it's not always so predictable. Take for instance φ_1, then the minimum value gives a better ρ, even though it's unlikelier to be optimal than it's maximum counterpart.

Before inspecting the local based features further. Notice that the staggering performance edge for φ_{23} is lost when going to a higher dimension (cf. φ_{23} in Fig. 6.4a has $\rho = 1.3\%$ and increases to 8.8% in Fig. 6.4b), implying that 100 random roll-outs for are not sufficient for fully exploring 10×10 state-space, yet highly competitive for 6×5.

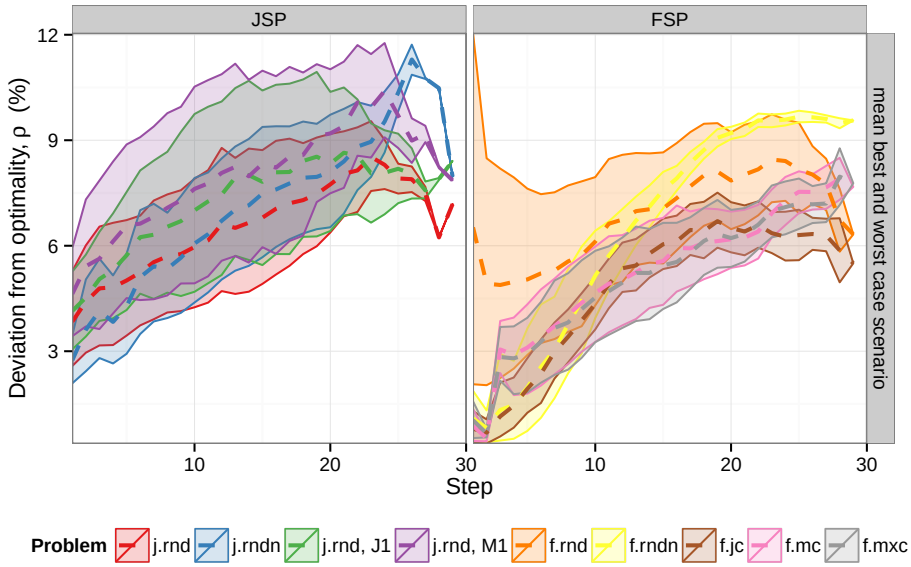
6.3. Check if Fig. 6.4 are part of *d* choose 1 pref-models feature selection

6.3.1 Optimality of SDRs

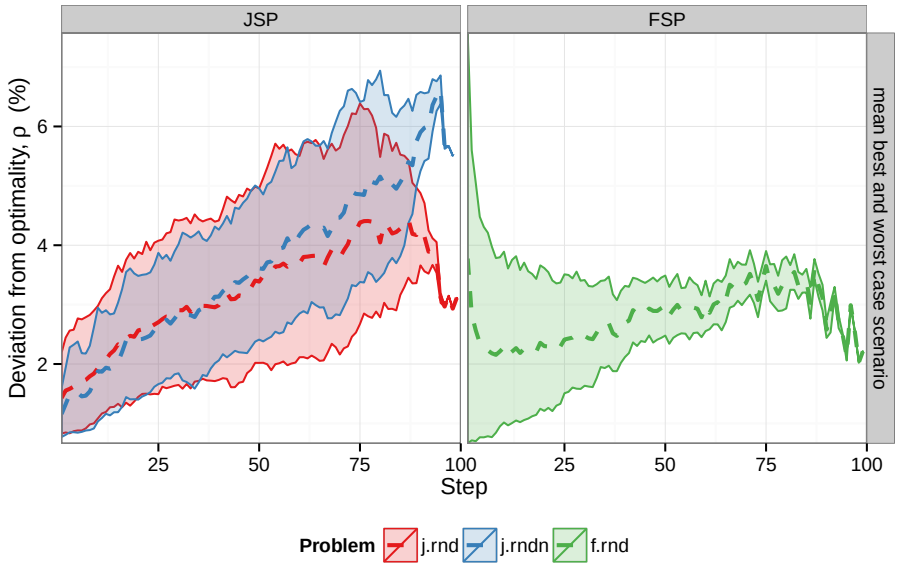
Let's limit ourselves to only features that correspond to SDRs from Section 2.4. Namely, Eq. (2.14) yield: *i)* φ_1 for SPT and LPT, and *ii)* φ_7 for LWR and MWR. By choosing the lowest value for the first SDR, and highest value for the latter SDR, i.e., the extremal values for those given features. Figure 6.5 depicts the corresponding probabilities from Fig. 6.4 in one graph, for all problem spaces in Table 3.2.

Now, let's bare in mind deviation from optimality, ρ, of applying SDRs throughout the dispatching process (cf. box-plots of which in Chapter 4), then there is a some correspondence between high probability of stepwise optimality and low ρ. Alas, this isn't always the case, for $\mathcal{P}^{10\times10}_{j.rnd}$ SPT always outperforms LPT w.r.t. stepwise optimality, however,

*Note, φ_{22} is non-informative on its own, as a tight standard deviation implies either consistently high *or* low C_{max} from the roll-outs.

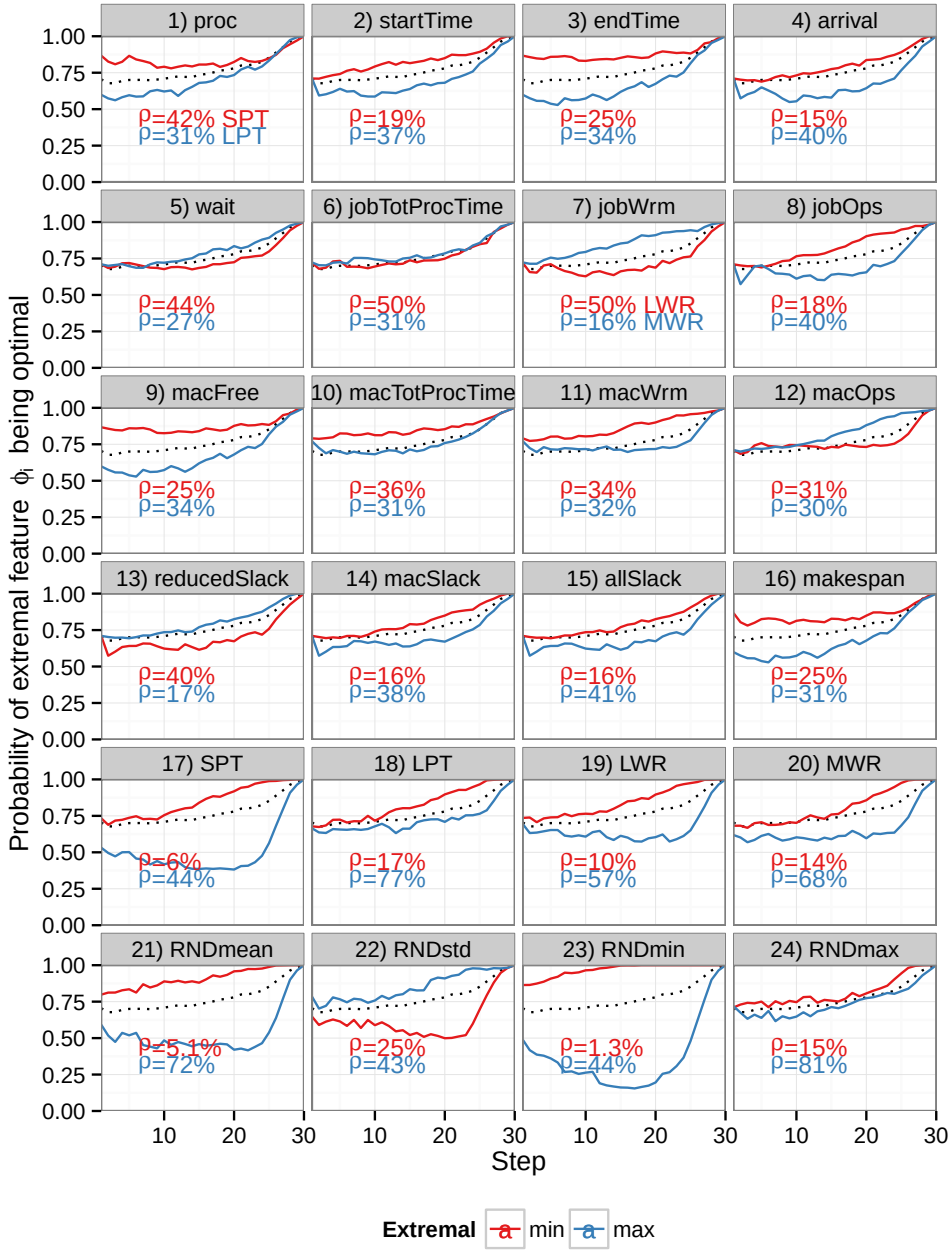


(a) 6×5



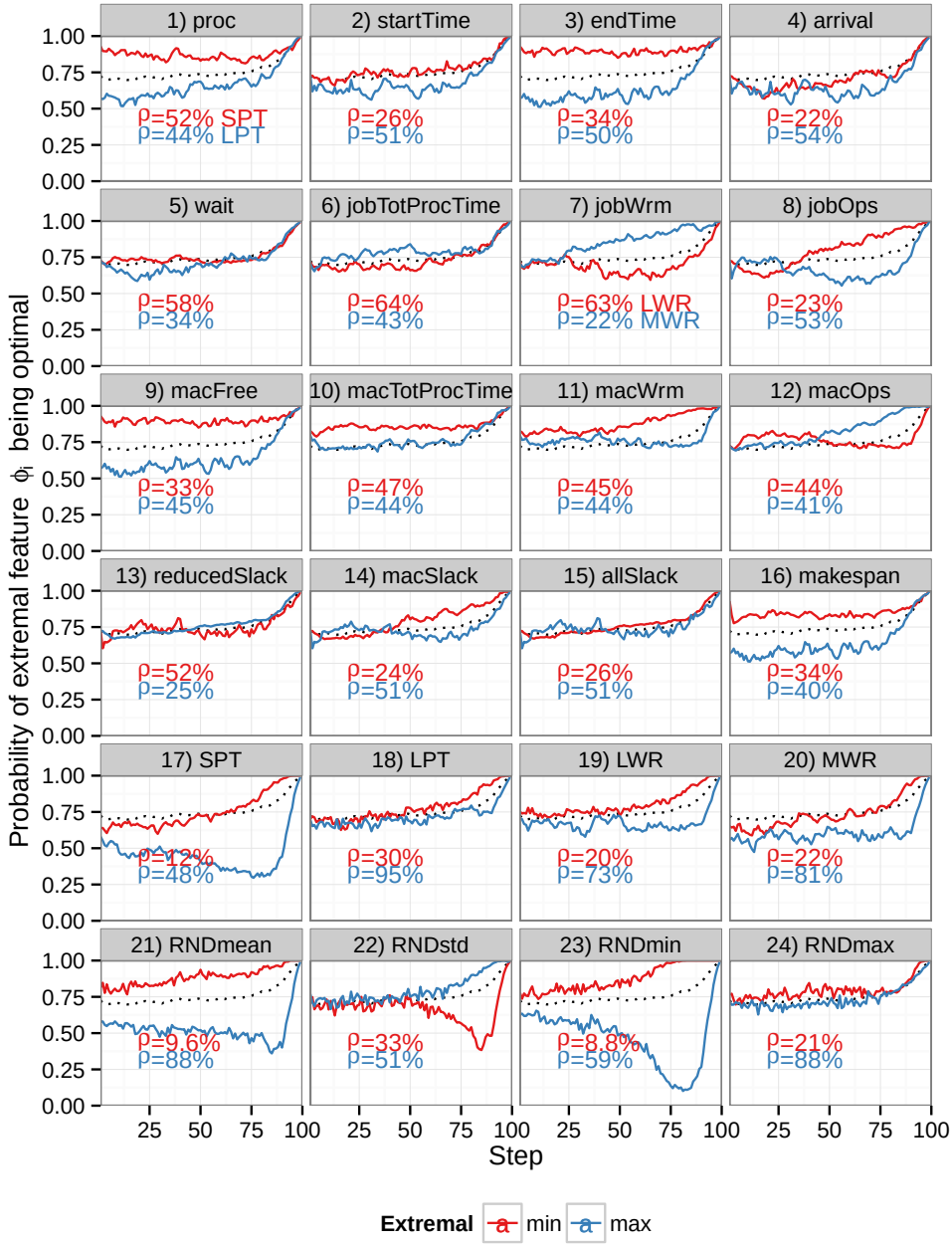
(b) 10×10

Figure 6.3: Mean deviation from optimality, ρ, for best and worst case scenario of when making *one* sub-optimal dispatch, depicted as lower and upper bound, respectively. Moreover, mean suboptimal move is given as dashed line.



(a) $\mathcal{P}^{6\times5}_{j.rnd}$

Figure 6.4: Probability of extremal feature being optimal



(b) $\mathcal{P}^{10\times10}_{j.rnd}$

Figure 6.4 (cont.)

this does not transcend to SPT having a lower ρ value than LPT. Hence, it's not enough to just learn optimal behaviour, one needs to investigate what happens once we encounter suboptimal state spaces.

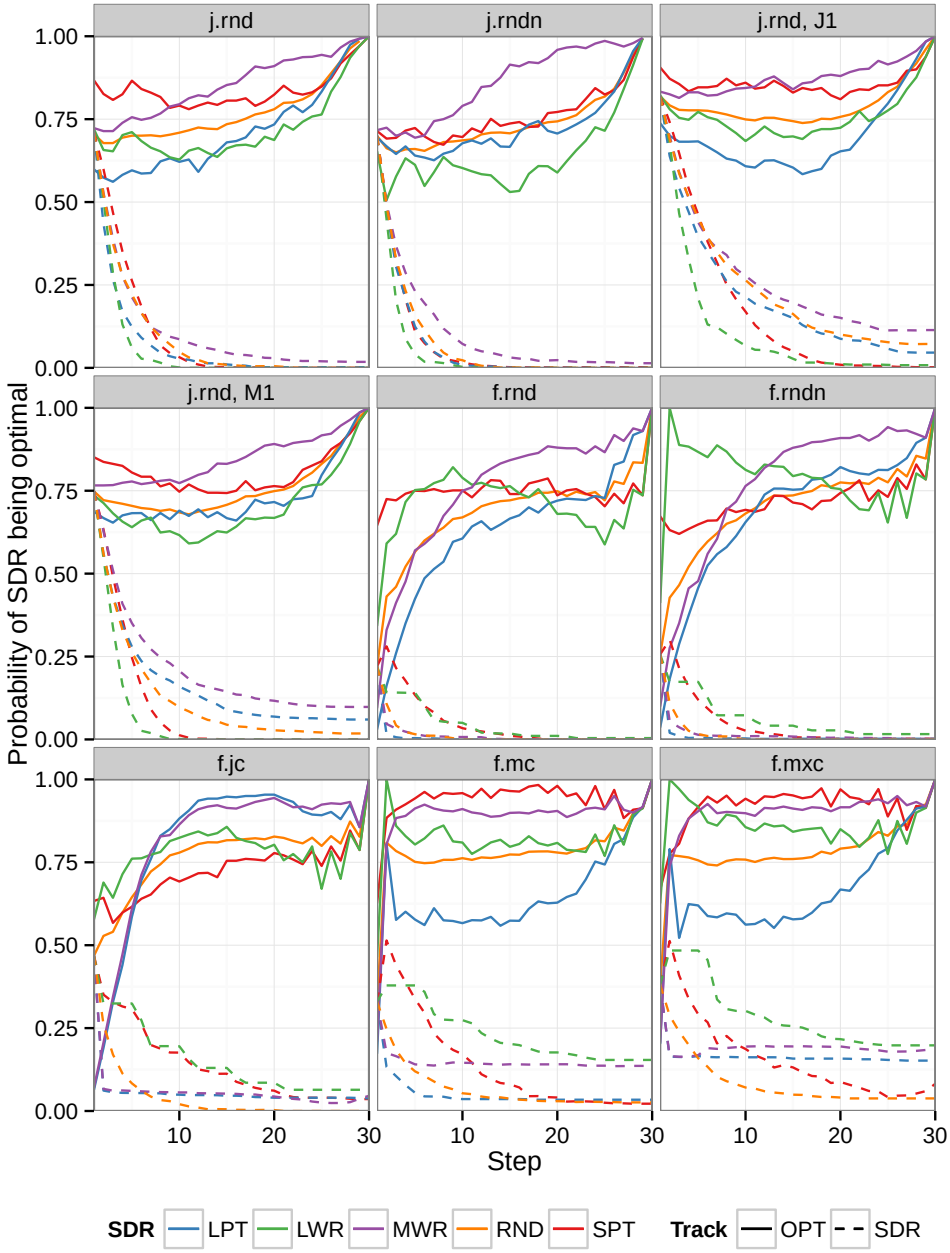
6.4 Simple blended dispatching rule

As stated before, the goal of this chapter is to utilise feature behaviour to motivate new, and *hopefully* better, dispatching rules. A naïve approach would be creating a simple blended dispatching rule (BDR) which would be for instance switch between two SDRs at a predetermined time point. Hence, going back to Fig. 6.5 a presumably good BDR for $\mathcal{P}^{10\times10}_{j.rnd}$ would be starting with SPT and then switching over to MWR at around time step $k = 40$, where the SDRs change places in outperforming one another. Note, MWR∩SPT hardly ever coincide for easy or hard schedules (cf. Tables 4.6 and 4.7, so its reasonable to believe they could complement one another. A box-plot for deviation from optimality, ρ, for all 10×10 problem spaces is depicted in Fig. 6.6. This little manipulation between SDRs does outperform SPT immensely, yet doesn't manage to gain the performance edge of MWR, save for $\mathcal{P}^{10\times10}_{f.rnd}$. This gives us insight that for job-shop, the attribute based on MWR is quite fruitful for good dispatches, whereas the same cannot be said about SPT – a more sophisticated DR 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. 6.5 shows, the inherent structure is already taking place, and might make it hard to come across optimal moves by simple methods. Therefore it's by no means guaranteed that by simply swapping over to MWR will handle the situation that applying SPT has already created. Figure 6.6 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 $\mathcal{P}^{10\times10}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rndn}$ the fact remains that the schedules have diverged too far from what MWR would have been able to achieve on its own. Preferably the blended dispatching rule should use best of both worlds, and outperform all of its inherited DRs, otherwise it goes without saying, one would simply keep on still using the original DR that achieved the best results.

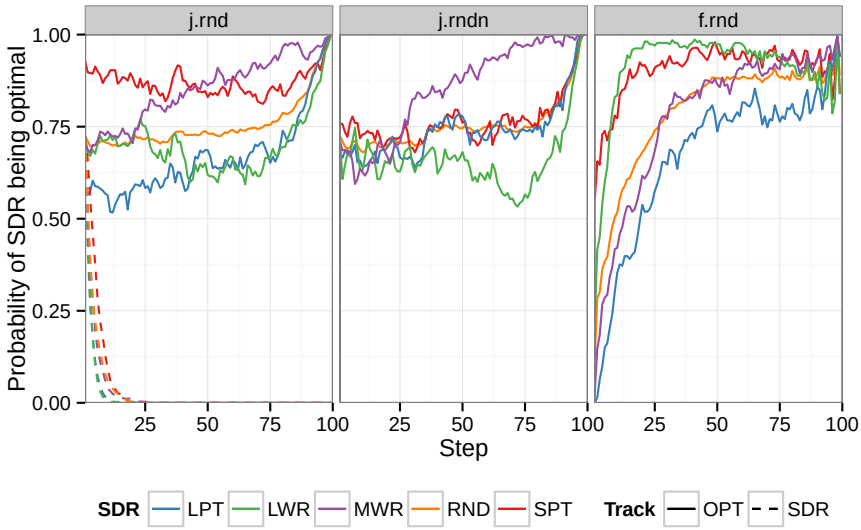
6.5 Feature evolution

In order to put the extremal features from Fig. 6.4 into perspective, it's worth comparing them with how the evolution of the features are over time, depicted in Fig. 6.7. Note that



(a) 6×5

Figure 6.5: Probability of SDR being optimal move. Both optimal (solid) and SDR-based (dashed) trajectories are inspected.



(b) 10 × 10

Figure 6.5 (cont.): Note, due to computational complexity, only $\mathcal{P}^{10\times10}_{j.rnd}$ has SDR-based trajectories also inspected. Otherwise, only optimal is pursued.

the optimal trajectory describes how 'good' features should aspire to be. We can also notice that the relative ranking in φ_{17}-φ_{24} is proportional their expected mean deviation from optimality, ρ. Although $(K-k)$-step lookahead give consistently the best (single) indicators for finding good solutions. Sadly, they are not practical features for high dimensional data due to computational cost. Nevertheless, bearing Fig. 6.3b in mind then it might be sufficient to do only a few steps lookahead at some key times in the dispatching process. For instance, let the computational budget for $\mathcal{P}^{10\times10}_{f.rnd}$ roll-outs be full K-solutions in the beginning phases, as that's when the problem space is most susceptible to bad moves. Then gradually decrease to only a few step lookahead, as flow-shop is then relatively stable. Conversely for $\mathcal{P}^{10\times10}_{j.rnd}$, start with a few step lookahead, and then expand the horizon as time goes by. Alternatively, when there aren't that many dispatches left, it might be worth developing a hybrid approach where the remaining dispatches from that point are optimised with some exact methods.

6.6 Emergence of problem difficulty

The main focus now is on knowing *when* during the scheduling process easy and hard problems diverge and explore in further detail *why* they diverged. Rather than visualising

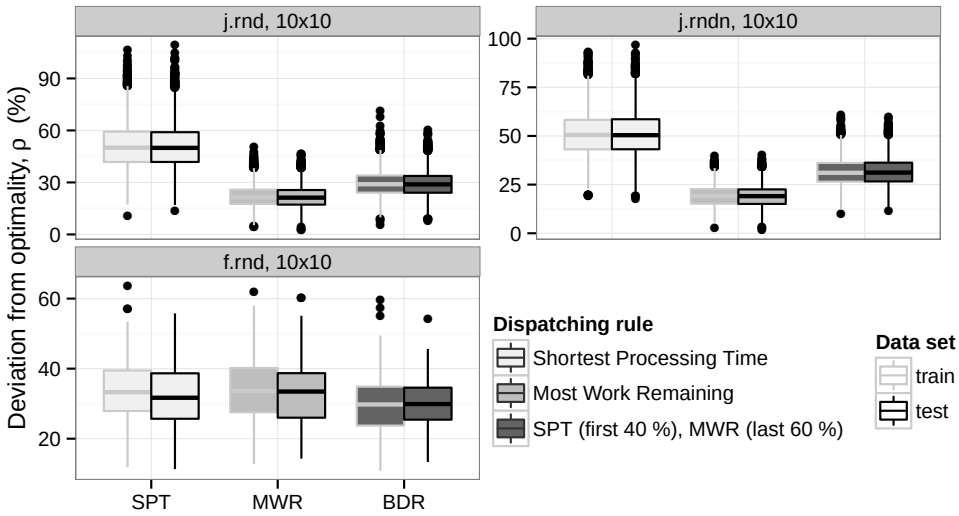


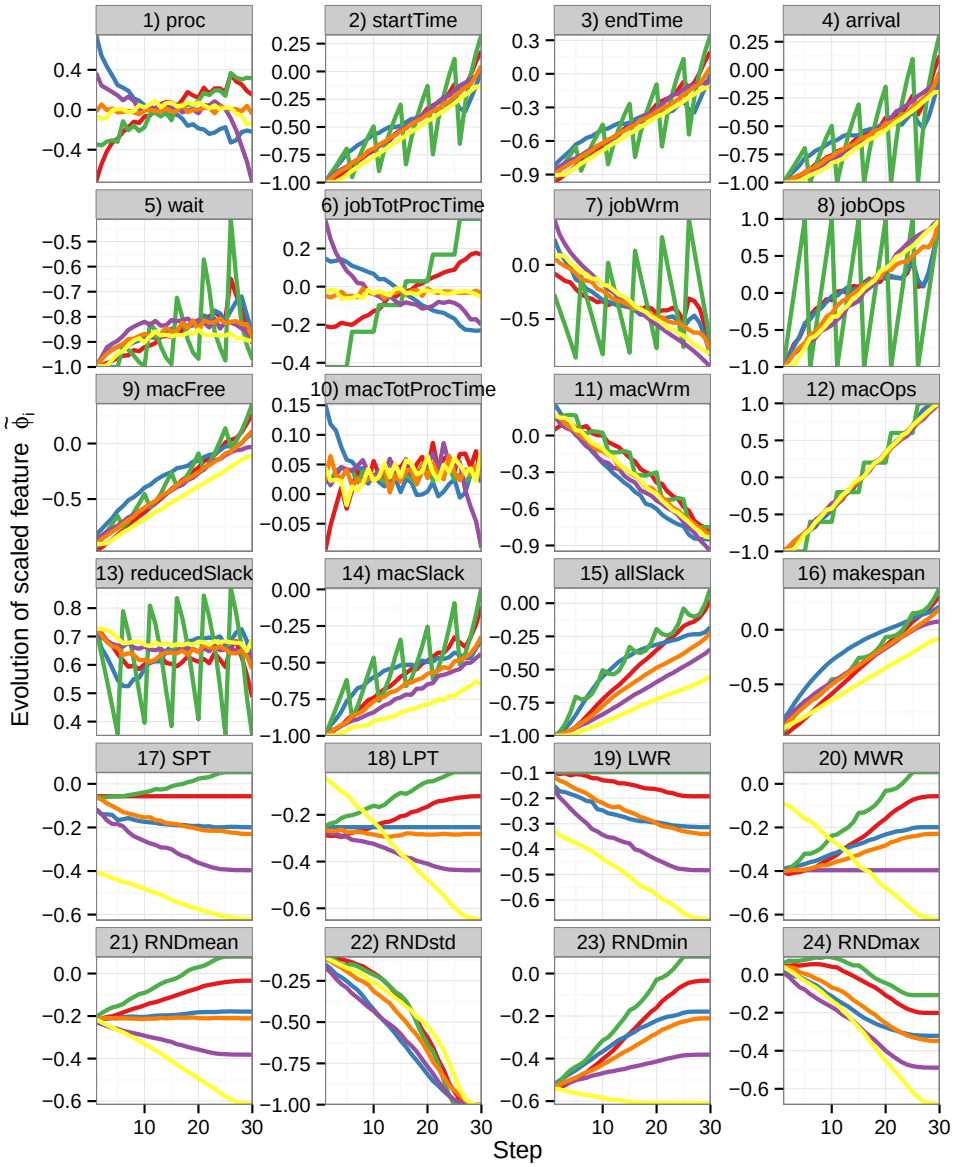
Figure 6.6: Box plot for deviation from optimality, ρ, for BDR where SPT is applied for the first 40% of the dispatches, followed by MWR.

high-dimensional data projected onto two dimensional space (as was the focus in Smith-Miles and Lopes (2011) with SOM), instead appropriate statistical tests with a significance level $\alpha = 0.05$ is applied to determine if there is any difference between different data distributions.

For this the two-sample Kolmogorov–Smirnov test (K-S test) is used to determine whether two underlying one-dimensional probability distributions differ. Furthermore, in order to find defining characteristics for easy or hard problems, a (linear) correlation is computed between features to the resulting deviation from optimality, ρ.

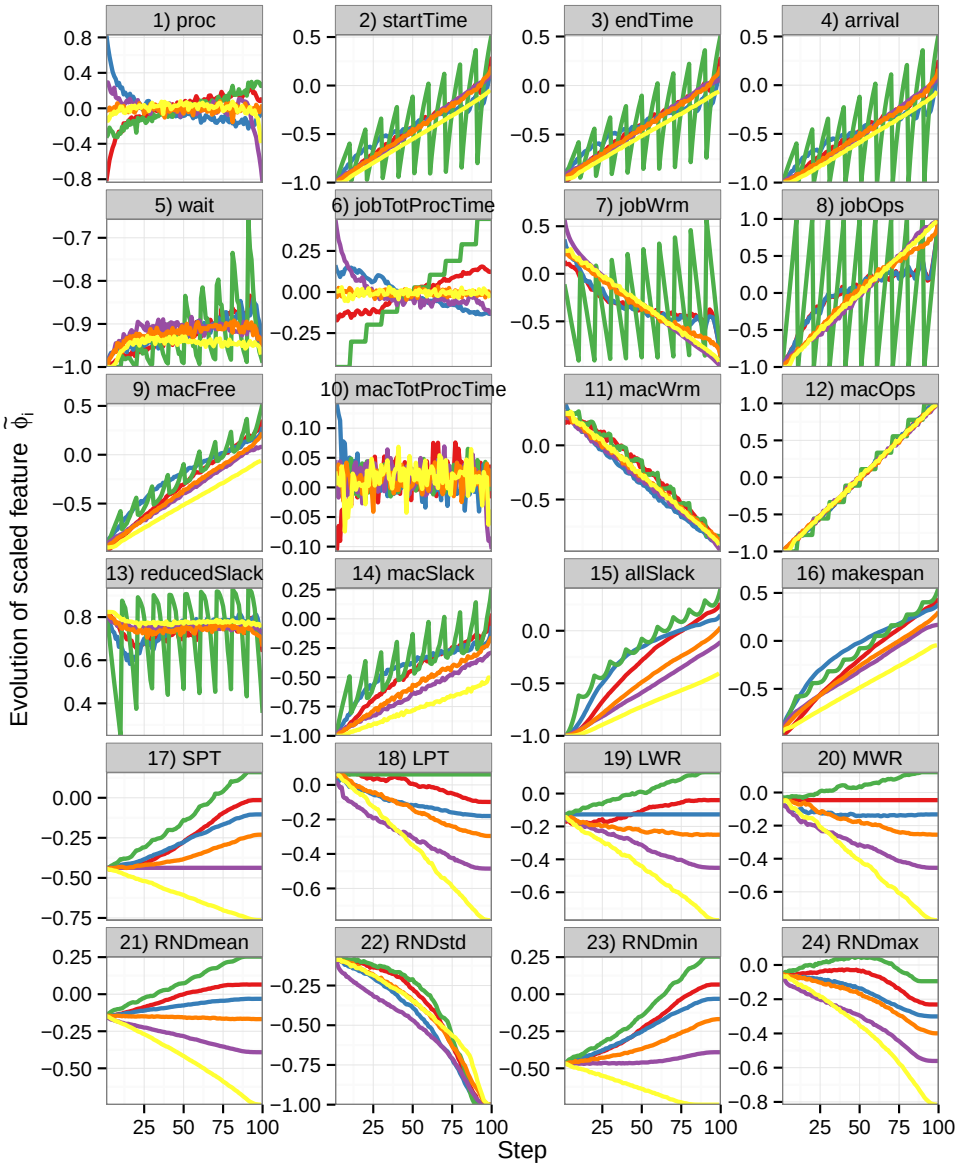
Note, when inspecting any statistical difference between data distribution of the features on a step by step basis, the features at step $k + 1$ are of course dependant on all previous k steps. This results in repetitive statistical testing, therefore a Bonferroni adjustment is used to counteract the multiple comparisons, i.e., each stepwise comparison has the significant level $\alpha_k = \frac{\alpha}{K}$, and thus maintaining the $\sum_{k=1}^{K} \alpha_k = \alpha$ significance level.

From the experimental study it is apparent that features have different correlation with the resulting schedule depending in what stage it is in the scheduling process, implying that their influence varies over the dispatching sequencing. Moreover, features constant throughout the scheduling process are not correlated with the end-result. There are some common features for both difficulties considered which define job-shop on a whole. However, the significant features are quite different across the two difficulties, implying there is



(a) $\mathcal{P}^{6\times5}_{j.rnd}$

Figure 6.7: Mean stepwise evolution of $\tilde{\boldsymbol{\varphi}}$, which is scaled according to Eq. (A.13)



(b) $\mathcal{P}^{10\times10}_{j.rnd}$

Figure 6.7 (cont.)

Table 6.1: Features for easy and hard problems are drawn from the same data distribution (denoted by ·)

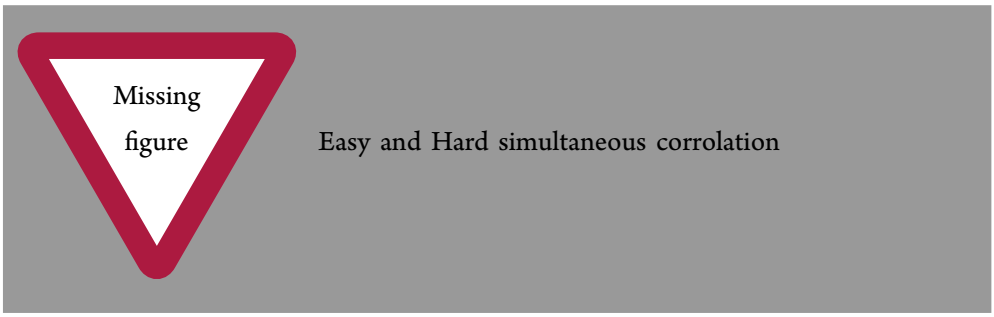
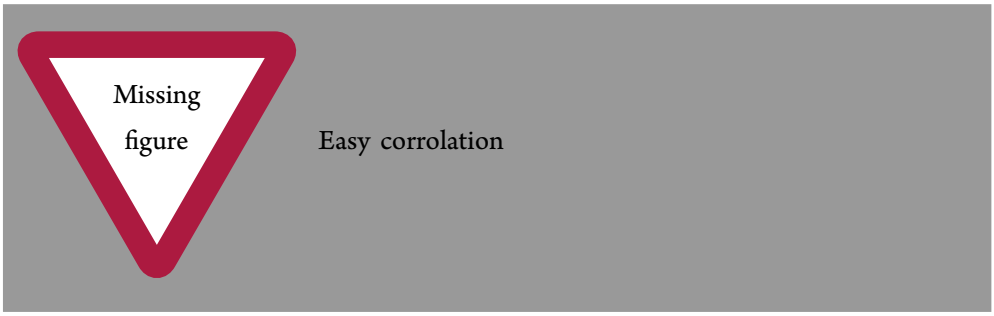
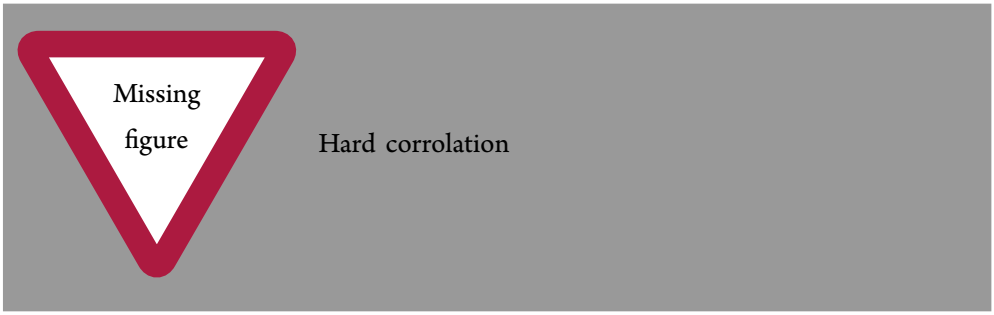


Table 6.2: Stepwise significant correlation (denoted by ·) for features φ and resulting deviation from optimality, ρ. Commonly significant features across the tables are denoted by •.

(a) Easy problems



(b) Hard problems



a clear difference in their data structure. The amount of significant features were considerably more for easy problems, indicating their key elements had been found. However, the features distinguishing hard problems were scarce. Most likely due to their more complex data structure their key features are of a more composite nature. As a result, new 'global' features were introduced.

It is possible for a JSP schedule to have more than one sequential dispatching representation. It is especially w.r.t. the initial dispatches. Revisiting Fig. 2.3, if we were to dispatch J_2 first and then J_4, then that would be the same equivalent temporal schedule if we did it the other way around. This is because they don't create a conflict for one another (as is the case for jobs J_2 and J_3). This drawback of non-uniqueness of sequential dispatching representation explains why there is hardly any significant feature for the initial steps of the scheduling process (cf. Tables 6.2a and 6.2b).

6.7 Summary and conclusions

Since feature selection is of paramount importance in order for algorithms to become successful, one needs to give great thought to how features are selected. What kind of features yield *bad* schedules? And can they be steered onto the path of more promising feature characteristics? This sort of investigation can be an indicator how to create meaningful problem generators. On the account that real-world problem instances are scarce, their hidden properties need be drawn forth in order to generate artificial problem instances from the same data distribution.

The feature attributes need to be based on statistical or theoretical grounds. Scrutiny in understanding the nature of problem instances therefore becomes of paramount importance in feature engineering for learning, as it yields feedback into what features are important to devote more attention to, i.e., features that result in a failing algorithm. For instance, in Table 6.1 the slack features have the same distribution in the initial stages of the scheduling process. However, there is a clear point of divergence which needs to be investigate why the sudden change? In general, this sort of analysis can undoubtedly be used in better algorithm design which is more equipped to deal with varying problem instances and tailor to individual problem instance's needs, i.e., a footprint-oriented algorithm.

6.7. Check with best/worst case scenario

Although this methodology was only implemented on a set of simple single-priority dispatching rules, the methodology is easily adaptable for more complex algorithms, such as the learned preference models in Chapter 7. The main objective of this work is to illustrate the interaction of a specific algorithm on a given problem structure and its properties.

It was much pleasanter at home, when one wasn't always growing larger and smaller, and being ordered about by mice and rabbits.

Alice

7 Preference Learning of CDRs

Learning models considered in this dissertation 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 Runarsson (2006), and given in Appendix A for completeness.

7.1 Ordinal regression for job-shop

Let $\boldsymbol{\varphi}_o \in \mathcal{F}$ 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 $\boldsymbol{\varphi}_s \in \mathcal{F}$. One could label which feature sets were considered optimal, $\mathbf{z}_o = \boldsymbol{\varphi}_o - \boldsymbol{\varphi}_s$, and suboptimal, $\mathbf{z}_s = \boldsymbol{\varphi}_s - \boldsymbol{\varphi}_o$ 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,

$$S := \bigcup_{\{\mathbf{x}_i\}_{i=1}^{N_{\text{train}}}} \left\{ \{\mathbf{z}_o, +1\}, \{\mathbf{z}_s, -1\} \;\middle|\; \forall (o,s) \in \mathcal{O}^{(k)} \times \mathcal{S}^{(k)} \right\}_{k=1}^{K} \subset \Phi \times Y \qquad (7.1)$$

where $\Phi \subset \mathcal{F}$ is the training set of d features, $Y = \{-1, +1\}$ is the outcome space, and

at dispatch $k \in \{1, .., K\}$, $o \in \mathcal{O}^{(k)}$, $s \in \mathcal{S}^{(k)}$ are optimal and suboptimal dispatches, respectively. Note, $\mathcal{O}^{(k)} \cup \mathcal{S}^{(k)} = \mathcal{L}^{(k)}$, and $\mathcal{O}^{(k)} \cap \mathcal{S}^{(k)} = \emptyset$.

For job-shop there are $d = 16$ features (cf. the local features from Table 2.2), and the training set is created in the manner described in Chapter 5.

7.1. Which is being used, ordinal or logistic regression?

Logistic regression makes decisions regarding optimal dispatches and at the same time efficiently estimates a posteriori probabilities. When using linear classification model (cf. Appendix A.2) for Eq. (2.12), then the optimal $\mathbf{w}^*$ obtained from the preference set can be used on any new data point (i.e. partial schedule), $\boldsymbol{\chi}$, and their inner product is proportional to probability estimate **??**. Hence, for each job on the job-list, $J_j \in \mathcal{L}$, let $\boldsymbol{\varphi}_j$ 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.,

$$J_{j^*} = \underset{J_j \in \mathcal{L}}{\operatorname{argmax}} \ \hat{\pi}(\boldsymbol{\varphi}_j) \tag{7.2}$$

where $\hat{\pi}(\cdot)$ is the classification model obtained by the preference set, S, defined by Eq. (7.1).

7.2 Interpreting linear classification models

When using a feature space based on SDRs, the linear classification models can very easily be interpreted as composite dispatching rules with predetermined weights. Looking at the features description in Table 2.2 it is possible for the ordinal regression to 'discover' the weights $\mathbf{w}$ in order for Eq. (2.12) corresponding applying a single priority dispatching rule from Section 2.4.

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

The approach taken here is to verify analytically, at each time step, by retaining the current temporal schedule as an initial state, whether it can indeed *somehow* yield an optimal schedule by manipulating the remainder of the sequence. This also takes care of the scenario that having dispatched a job resulting in a different temporal makespan would have resulted in the same final makespan if another optimal dispatching sequence would

have been chosen. That is to say the data generation takes into consideration when there are multiple optimal solutions to the same problem instance.

7.3 Time dependant dispatching rules

At each dispatch iteration k, a number of preference pairs are created, which is then repeated for all the N_{train} problem instances created. A separate data set is deliberately created for each dispatch iterations, as the initial feeling is that dispatch rules used in the beginning of the schedule building process may not necessarily be the same as in the middle or end of the schedule. As a result there are K linear scheduling rules for solving a $n \times m$ job-shop.

7.3. No longer one model for all steps

7.4 Selecting preference pairs

Defining the size of the preference set as $l = |\Upsilon|$, then **??** gives the size of the feature training set as $|\Phi| = \frac{1}{2}l$. If l is too large, than sampling needs to be done in order for the ordinal regression in Chapter 7 to be computationally feasible.

The strategy approached in Paper I was to follow a *single* optimal job $J_j \in \mathcal{O}^{(k)}$ (chosen at random), thus creating $|\mathcal{O}^{(k)}| \cdot |\mathcal{S}^{(k)}|$ feature pairs at each dispatch k, resulting in a training size of,

$$l = \sum_{i=1}^{N_{\text{train}}} \left(2|\mathcal{O}_i^{(k)}| \cdot |\mathcal{S}_i^{(k)}| \right) \tag{7.3}$$

For the problem spaces considered there, that sort of simple sampling of the state space was sufficient for a favourable outcome. However for a considerably harder problem spaces (see Chapter 4), preliminary experiments were not satisfactory.

A brute force approach was adopted to investigate the feasibility of finding optimal weights $\mathbf{w}$ for Eq. (2.12). By applying CMA-ES (discussed thoroughly in Chapter 8) to directly minimize the mean $C_{\max}$ w.r.t. the weights $\mathbf{w}$, gave a considerably more favourable result in predicting optimal versus suboptimal dispatching paths. So the question put forth is, why was the ordinal regression not able to detect it? The nature of the CMA-ES is to explore suboptimal routes until it converges to an optimal one. Implying that the previous approach of only looking into one optimal route is not sufficient information. Suggesting that the training set should incorporate a more complete knowledge about *all* possible preferences, i.e., make also the distinction between suboptimal and sub-suboptimal features, etc. This would require a Pareto ranking for the job-list, $\mathcal{L}$, which can be used to make the

distinction to which feature sets are equivalent, better or worse, and to what degree (i.e. giving a weight to the preference)? By doing so, the training set becomes much greater, which of course would again need to be sampled in order to be computationally feasible to learn.

For instance Li and Olafsson (2005) used decision trees to 'rediscover' LPT by using the dispatching rule to create its training data. The limitations of using heuristics to label the training data is that the learning algorithm will mimic the original heuristic (both when it works poorly and well on the problem instances) and does not consider the real optimum. In order to learn new heuristics that can outperform existing heuristics then the training data needs to be correctly labelled. This drawback is confronted in (Malik et al., 2008, Olafsson and Li, 2010, Russell et al., 2009) by using an optimal scheduler, computed off-line.

These aspects are the main motivation for the data generation in this dissertation. All problem instances are correctly labelled w.r.t. their optimum makespan, found with analytical means.* In order to create training instances (and subsequently preference pairs) both a features resulting in optimal solutions are gathered (following optimal trajectories) and features that would have been chosen if a dispatching rule had been implemented (following DR trajectories). In the latter case, the trajectories pursued here, will be the SDRs from Section 2.4 as well as randomly dispatching operations.

To summarise, one needs to consider two main aspects of the generation of the training data: *I*) what sort of rankings should be compared during each step? And *II*) which path(s) should be investigated ? *II.i)* Pursuing solely optimal trajectories? *II.ii)* Creating random dispatches? Or *II.iii)* following other means: CMA-ES computed weights, single priority dispatching rules, etc.

7.4.1 Ranking strategies

The following ranking strategies were implemented for adding preference pairs to S defined by Eq. (7.1), they were first reported in Paper V,

> **Basic ranking,** S_b**,** i.e., all optimum rankings r_1 versus all possible suboptimum rankings r_i, $i \in \{2, \ldots, n'\}$, preference pairs are added – same basic set-up introduced in Paper I. Note, $|S_b|$ is defined in Eq. (7.3).
>
> **Full subsequent rankings,** S_f**,** i.e., all possible combinations of r_i and r_{i+1} for $i \in \{1, \ldots, n'\}$, preference pairs are added.

*Optimal solution were found using Gurobi Optimization, Inc. (2014), a commercial software package for solving large-scale linear optimization and a state-of-the-art solver for mixed integer programming.

Partial subsequent rankings, S_p**,** i.e., sufficient set of combinations of r_i and r_{i+1} for $i \in \{1, \ldots, n'\}$, are added to the training set – e.g. in the cases that there are more than one operation with the same ranking, only one of that rank is needed to compared to the subsequent rank. Note that $S_p \subset S_f$.

All rankings, S_a**,** denotes that all possible rankings were explored, i.e., r_i versus r_j for $i, j \in \{1, \ldots, n'\}$ and $i \neq j$, preference pairs are added.

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

7.4.2 Experimental study

To test the validity of different ranking and strategies from Section 7.4, a training set of N_{train} problem instances of 6×5 job-shop and flow-shop summarised in Table 3.2 (omitting the job and machine variations of job-shop). The size of the preference set, S, for different trajectory and ranking strategies are depicted in **??**. Note, for now 10×10 problem spaces will be ignored, due to the extreme computational cost of correctly labelling each trajectory. However, in Section 7.5 an optimum path will be pursued for said dimension.

A linear preference (PREF) model was created for each preference set, S, for every problem space considered. A box-plot with deviation from optimality, ρ, defined by Eq. (2.17), is presented in **??**. From the figures it is apparent there can be a performance edge gained by implementing a particular ranking or trajectory strategy, moreover the behaviour is analogous across different disciplines.

Ranking strategies

There is no statistical difference between S_f and S_p ranking-schemes across all disciplines (cf. **????**), which is expected since S_f is designed to contain the same preference information as S_f. However neither of the Pareto ranking-schemes outperform the original S_b set-up from Paper I. The results hold for the test set as well.

Combining the ranking schemes, S_{all}, improves the individual ranking-schemes across all disciplines, except in the case of $S_b^{opt}\big|_{\mathcal{P}_1}$ and $S_b^{rnd}\big|\mathcal{P}_2$, in which case there were no statistical difference. Now, for the test set, the results hold, however there is no statistical difference between S_b and S_{all} for most trajectories $\{S^{opt}, S^{cma}, S^{rnd}\}\big|\mathcal{P}_1$ and $\{S^{opt}, S^{rnd}\}\big|\mathcal{P}_2$. Now, whereas a smaller preference set is preferred, its opted to use the S^b ranking scheme henceforth.

Moreover, it is noted that the learning algorithm is able to significantly outperform the original heuristics, MWR and CMA-ES (white), used to create the training data S^{mwr} (grey) and S^{cma} (yellow), respectively (cf. **????**). For both $\mathcal{P}_1$ and $\mathcal{P}_2$, linear ordinal

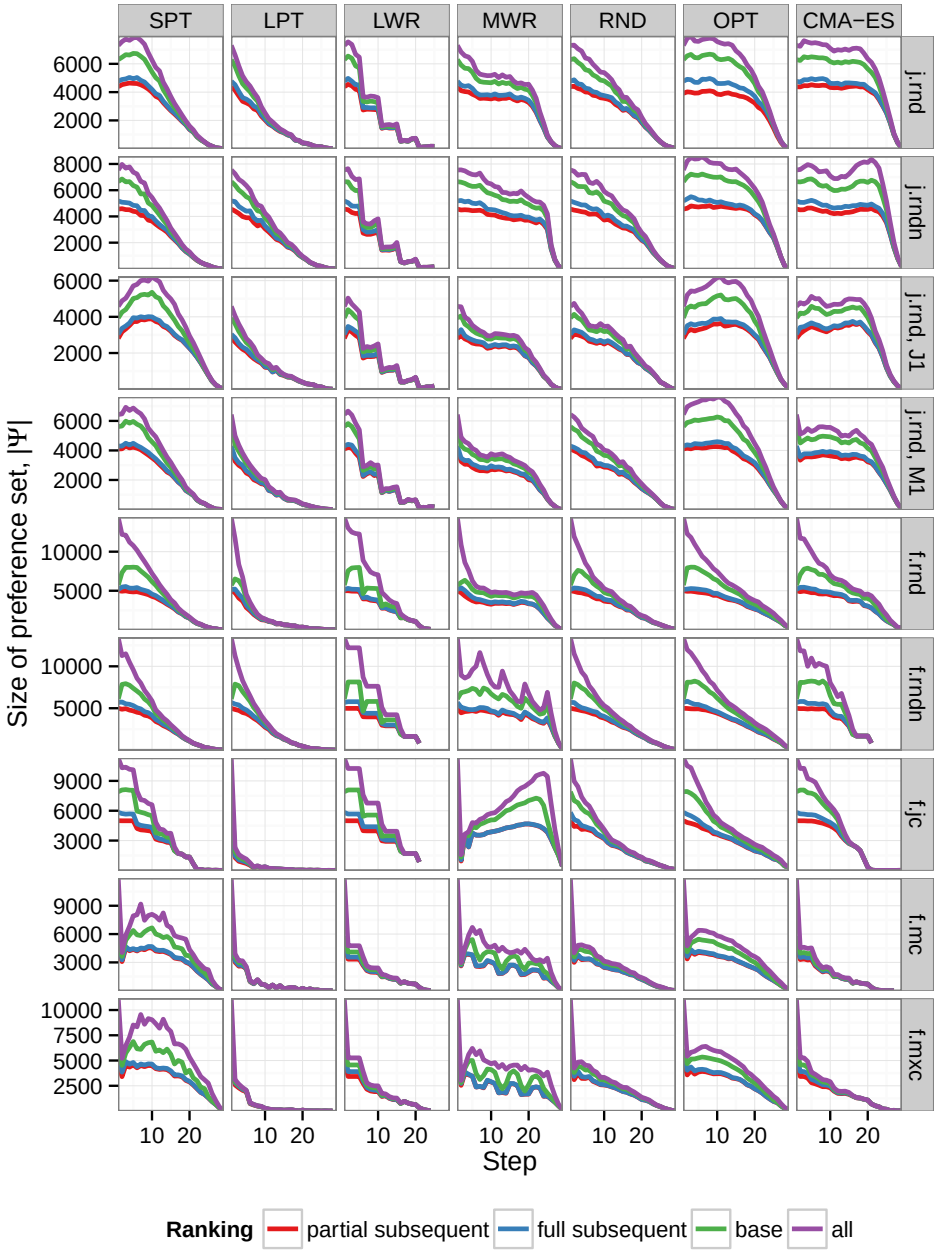


Figure 7.1: Size of preference set, $l = |S|$, for different trajectory strategies and ranking schemes (where $N_{\text{train}} = 500$)

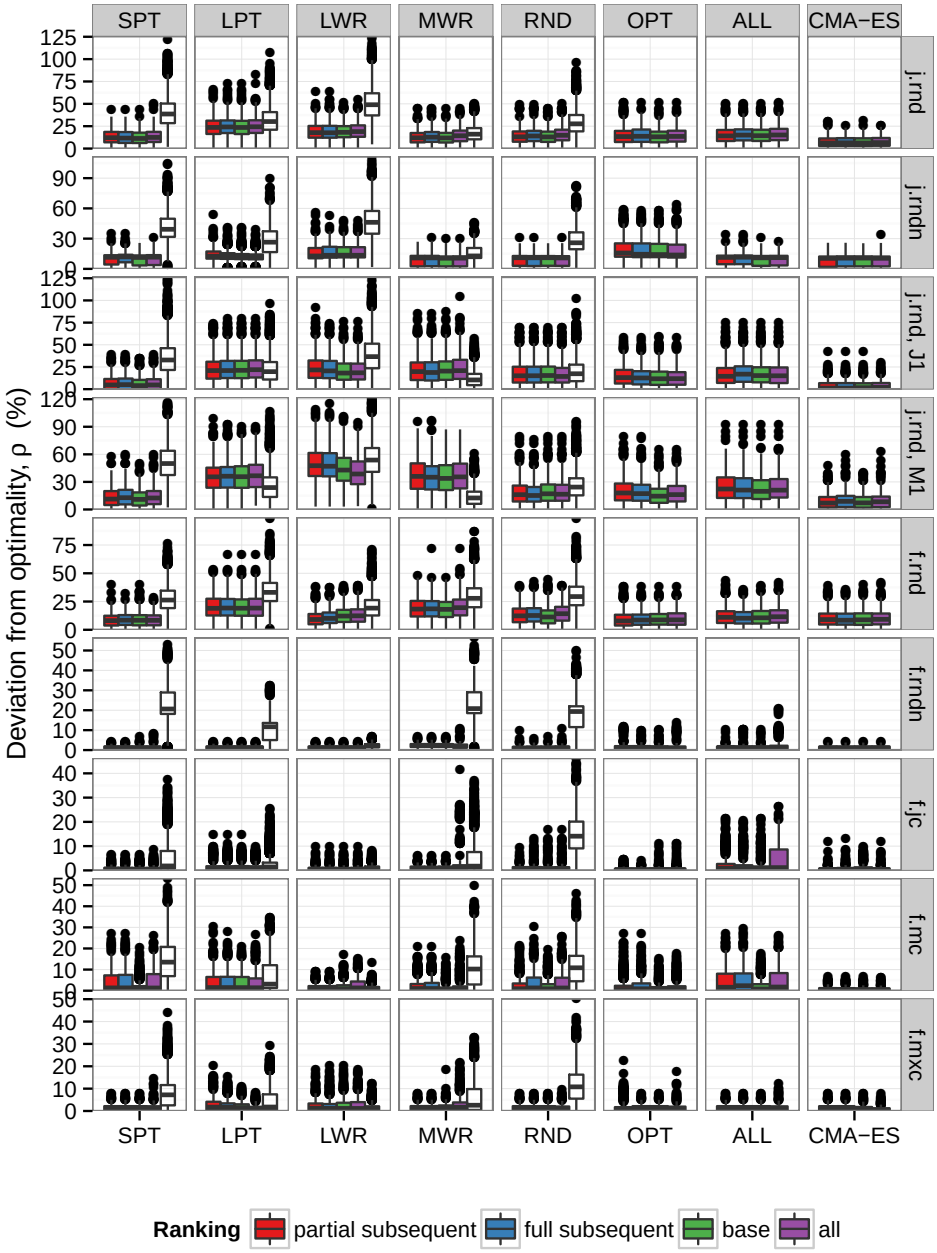


Figure 7.2: Box-plot for various Φ and Ψ set-up. The trajectories the models are based on are depicted in white.

regression models based on S^{mwr} are significantly better than *MWR*, irrespective of the ranking schemes. Whereas the fixed weights found via CMA-ES are only outperformed by linear ordinal regression models based on $\{S_b^{cma}, S_{all}^{cma}\}$. This implies that ranking scheme needs to be selected appropriately. Result hold for the test data.

Trajectory strategies

Learning preference pairs from a good scheduling policies, such as S^{cma} and S^{mwr}, gave considerably more favourable results than tracking optimal paths (cf. **????**). Suboptimal routes are preferred when dealing with problem1 (for all ranking schemes), however when encountering problem2 the choice of ranking schemes can yield the exact opposite.

It is particularly interesting there is no statistical difference between S^{opt} and S^{rnd} for both $\{S_b, S_f\}\big|\mathcal{P}_1$ and $\{S_b, S_f, S_p\}\big|\mathcal{P}_2$ ranking-schemes. That is to say, tracking optimal dispatches gives the same performance as completely random dispatches. This indicates that exploring only optimal trajectories can result in a training set which the learning algorithm is inept to determine good dispatches in the circumstances when newly encountered features have diverged from the learned feature set labelled to optimum solutions.

Finally, S^{all} gave the best combination across all disciplines. Adding suboptimal trajectories with the optimal trajectory gives the learning algorithm a greater variety of preference pairs for getting out of local minima.

Following CMA-ES guided trajectory

The rational for using the S^{cma} strategy was mostly due to the fact a linear classifier is creating the training data (using the weights found via CMA-ES optimisation), hence the training data created should be linearly separable, which in turn should boost the training accuracy for a linear classification learning model. However, this strategy is easily outperformed by the single priority based dispatching rule MWR guiding the training data collection, S^{mwr}.

7.4.3 Summary and conclusion

As the experimental results showed in **??**, the ranking of optimal* and suboptimal features are of paramount importance. The subsequent rankings are not of much value, since they are disregarded anyway. However, the trajectories to create training instances have to be varied.

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

Unlike (Malik et al., 2008, Olafsson and Li, 2010, Russell et al., 2009), learning only on optimal training data was not fruitful. However, inspired by the original work by Li and Olafsson (2005), having DR guide the generation of training data (except correctly labelling with analytic means) gave meaningful preference pairs which the learning algorithm could learn. In conclusion, henceforth, the training data will be generate with S_b^{all} scheme.

7.5 Time independent dispatching rules

As stated in **??**, a separate data set is deliberately created for each dispatch iteration, as it is initially assumed that dispatch rules used in the schedule building might differ in the beginning, the middle or towards the end of the process. As a result there is a local linear model for each dispatch; a total of K linear models for solving $n \times m$ job-shop. Now, if we were to create a global rule, then there would have to be one model for all dispatches iterations. The approach in Paper I was to take the mean weight for all stepwise linear models, i.e., $\bar{w}_i = \frac{1}{K}\sum_{k=1}^{K} w_i^{(k)}$ where $\mathbf{w}^{(k)}$ is the linear weight resulting from learning preference set $S^{(k)}$ at dispatch k.

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 K dispatches. This would amount to a substantial training set, and for S to be computationally feasible to learn, S has to be filtered to size $l_{\max}$.

7.5. No longer the case, one model instead of K stepwise-models

7.6 Discussion and conclusions

There's a large mustard-mine near here. And the moral of that is –
The more there is of mine, the less there is of yours.

The Duchess

8

Evolutionary Learning of CDRs

Genetic algorithms (GA) are one of the most widely used approaches in JSP literature (Pinedo, 2008). However, in that case an extensive number of schedules need to be evaluated, and even for low dimensional JSP, it can quickly become computationally infeasible. GAs can be used directly on schedules Ak and Koc (2012), Cheng et al. (1996, 1999), Qing-dao-er ji and Wang (2012), Tsai et al. (2007). However, then there are many concerns that need to be dealt with. To begin with there are nine encoding schemes for representing the schedules Cheng et al. (1996), in addition, special care must be taken when applying cross-over and mutation operators in order for the schedules, now in the role of 'chromosomes,' to still remain feasible. Moreover, in case of JSP, GAs are not adapt for fine-tuning around optima. Luckily a subsequent local search can mediate the optimisation (Cheng et al., 1999, Meeran and Morshed, 2012).

The most predominant approach in hyper-heuristics, a framework of creating *new* heuristics from a set of predefined heuristics, is genetic programming Burke et al. (2013). Dispatching rules based genetic algorithms (DRGA) Dhingra and Chandna (2010), Nguyen et al. (2013), Vázquez-Rodríguez and Petrovic (2009) are a special case of genetic programming Koza and Poli (2005), where GAs are applied indirectly to JSP via dispatching rules, i.e., where a solution is no longer a *proper* schedule but a *representation* of a schedule via applying certain DRs consecutively.

A prevalent approach to solving JSP is to combine several relatively simple dispatching

rules such that they may benefit each other for a given problem space. Generally, this is done on an ad-hoc basis, requiring expert knowledge from heuristics designer, or extensive exploration of suitable combinations of heuristics. The approach in this Chapter, is to automate that selection, by translating dispatching rules into measurable features and optimising what their contribution should be via evolutionary search. The framework is straight forward and easy to implement and shows promising results. Various data distributions from Chapter 3 are investigated, however only trained on the lower dimension, 6×5, yet, validated on higher dimension, 10×10.

Moreover, **??** shows that the choice of objective function for evolutionary search is worth investigating. Since the optimisation is based on minimising the expected mean of the fitness function over a large set of problem instances, which can vary within. Then normalising the objective function can stabilise the optimisation process away from local minima.

As previously discussed in Chapter 1, there are two main viewpoints on how to approach scheduling problems: *i)* local level by building schedules for one problem instance at a time, and *ii)* global level by building schedules for all problem instances at once. For local level construction a simple construction heuristic is applied. The schedule's features are collected at each dispatch iteration from which a learning model will inspect the feature set to discriminate which operations are preferred to others via ordinal regression. The focus is essentially on creating a meaningful preference set composed of features and their ranks as the learning algorithm is only run once to find suitable operators for the value function. This is the approach taken in Paper I. Expanding on that work, this study will explore a global level construction viewpoint where there is no feature set collected beforehand since the learning model is optimised directly via evolutionary search. This involves numerous costly value function evaluations. In fact it involves an indirect method of evaluation whether one learning model is preferable to another, w.r.t. which one yields a better expected mean.

8.1 Experimental set-up

8.1. Throughout a Kolmogorov-Smirnov test with $\alpha = 0.05$ is applied to determine statistical significance between methodologies.

Inspired by DRGA, the approach taken in this study is to optimise the weights $\mathbf{w}$ in Eq. (2.12) directly via evolutionary search such as covariance matrix adaptation evolution strategy (CMA-ES) Hansen and Ostermeier (2001). This has been proven to be a very efficient numerical optimisation technique.

Using standard set-up of parameters of the CMA-ES optimisation, the runtime was limited to 288 hours on a cluster for each training set given in Table 3.2 and in every case the optimisation reached its maximum walltime.

8.2 Performance measures

Generally, evolutionary search only needs to minimise the expected fitness value. However, the approach in Paper I was to use the known optimum to correctly label which operations' features were optimal when compared to other possible operations. Therefore, it would be of interest to inspect if there is any performance edge gained by incorporating optimal labelling in evolutionary search. Therefore, two objective functions will be considered, namely,

$$ES_{C_{\max}} := \min \mathbb{E}\left[C_{\max}\right] \tag{8.1}$$

for optimising w.r.t. $C_{\max}$ directly, and on the other hand

$$ES_{\rho} := \min \mathbb{E}\left[\rho\right] \tag{8.2}$$

which optimises w.r.t. the resulting $C_{\max}$ scaled to its true optimum, i.e., Eq. (2.17).

Main statistics of the experimental run are given in Table 8.1 and depicted in Fig. 8.2 for both approaches. In addition, evolving decision variables, here weights $\mathbf{w}$ for Eq. (2.12), are depicted in Fig. 8.3.

In order to compare the two objective functions, the best weights reported were used for Eq. (2.12) on the corresponding training data. Its box-plot of percentage relative deviation from optimality, defined by Eq. (2.17), is depicted in Fig. 8.1 and Table 8.2 present its main statistics; mean, median, standard deviation, minimum and maximum values.

In the case of $\mathcal{P}^{6\times5}_{f.rndn}$, Eq. (8.2) gave a considerably worse results, since the optimisation got trapped in a local minima, as the erratic evolution of the weights in Section 8.3 suggest. For other problem spaces, Eq. (8.1) gave slightly better results than Eq. (8.2). However, there was no statistical difference between adopting either objective function. Therefore, minimisation of expectation of ρ, is preferred over simply using the unscaled resulting makespan.

8.3 Problem difficulty

The evolution of fitness per generation from the CMA-ES optimisation of Eq. (8.2) is depicted in Fig. 8.2. Note, all problem spaces reached their allotted computational time without converging. In fact $\mathcal{P}^{6\times5}_{f.rnd}$ and $\mathcal{P}^{6\times5}_{j.rndn}$ needed restarting during the optimisation

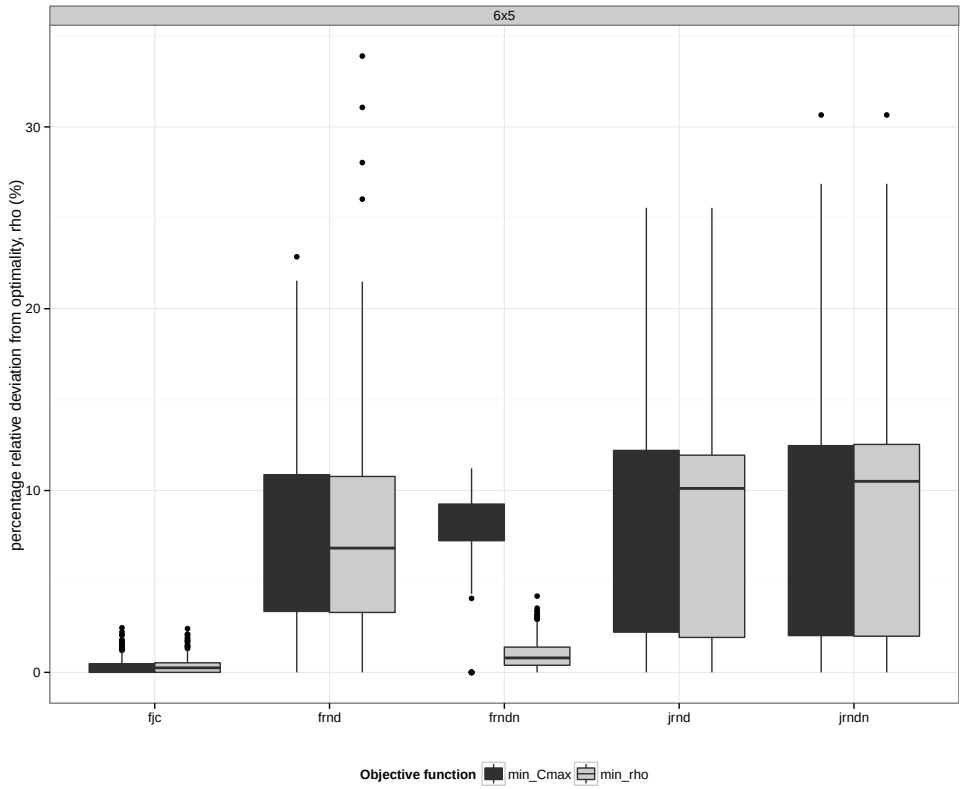


Figure 8.1: Box-plot of training data for percentage relative deviation from optimality, defined by Eq. (2.17), when implementing the final weights obtained from CMA-ES optimisation, using both objective functions from Eqs. (8.1) and (8.2), left and right, respectively.

process. Furthermore, the evolution of the decision variables $\mathbf{w}$ are depicted in Fig. 8.3. As one can see, the relative contribution for each weight clearly differs between problem spaces. Note, that in the case of $\mathcal{P}^{6\times5}_{j.rndn}$ (cf. Section 8.3), CMA-ES restarts around generation 1,000 and quickly converges back to its previous fitness. However, lateral relation of weights has completely changed, implying that there are many optimal combinations of weights to be used. This can be expected due to the fact some features in Table 2.2 are a linear combination of others, e.g. $\varphi_3 = \varphi_1 + \varphi_2$.

8.3.1 Scalability

As a benchmark, the linear ordinal regression model (PREF) from Paper I was created. Using the weights obtained from optimising Eq. (8.2) and applying them on their 6×5

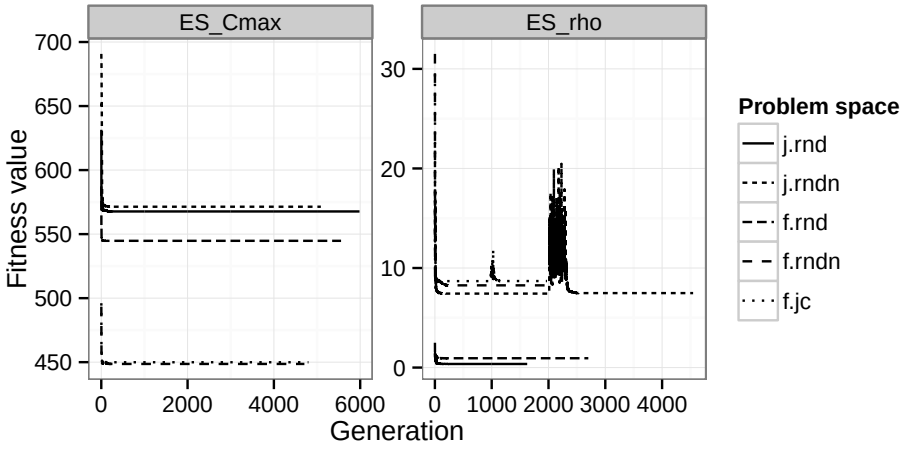


Figure 8.2: Fitness for optimising (w.r.t. Eqs. (8.1) and (8.2) above and below, receptively), per generation of the CMA-ES optimisation.

training data. Their main statistics of Eq. (2.17) are reported in Table 8.2 for all training sets described in Table 3.2. Moreover, the best SDR from which the features in Table 2.2 were inspired by, are also reported for comparison, i.e., most work remaining (MWR) for all JSP problem spaces, and least work remaining (LWR) for all FSP problem spaces.

To explore the scalability of the learning models, a similar comparison to Section 8.3 is made for applying the learning models on their corresponding 10 $\times$ 10 testing data. Results are reported in Table 8.3. Note, that only resulting $C_{\max}$ is reported as the optimum makespan is not known and Eq. (2.17) is not applicable.

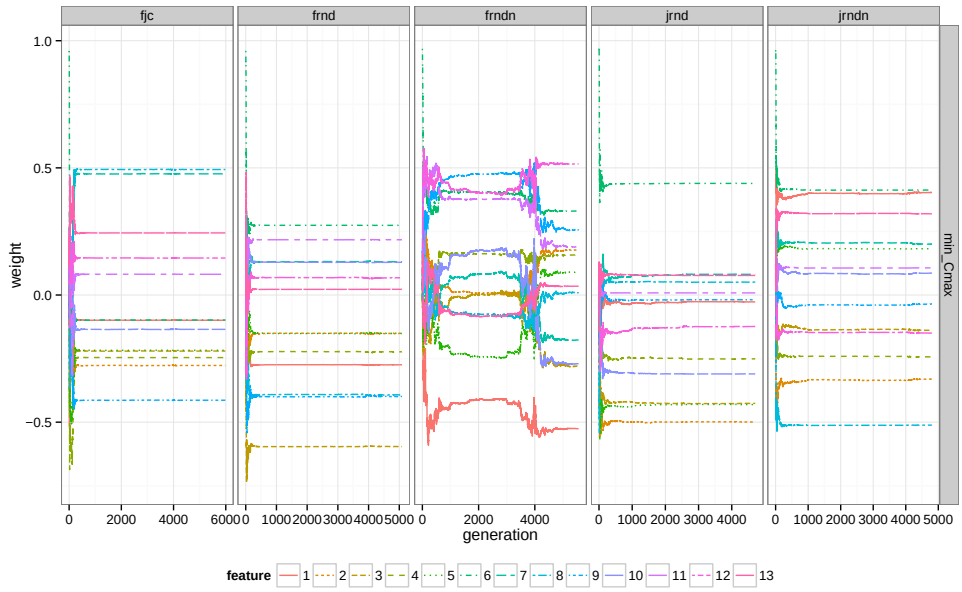
Table 8.1: Final results for CMA-ES optimisation; total number of generations and function evaluations and its resulting fitness value for both performance measures considered.

(a) w.r.t. Eq. (8.1)

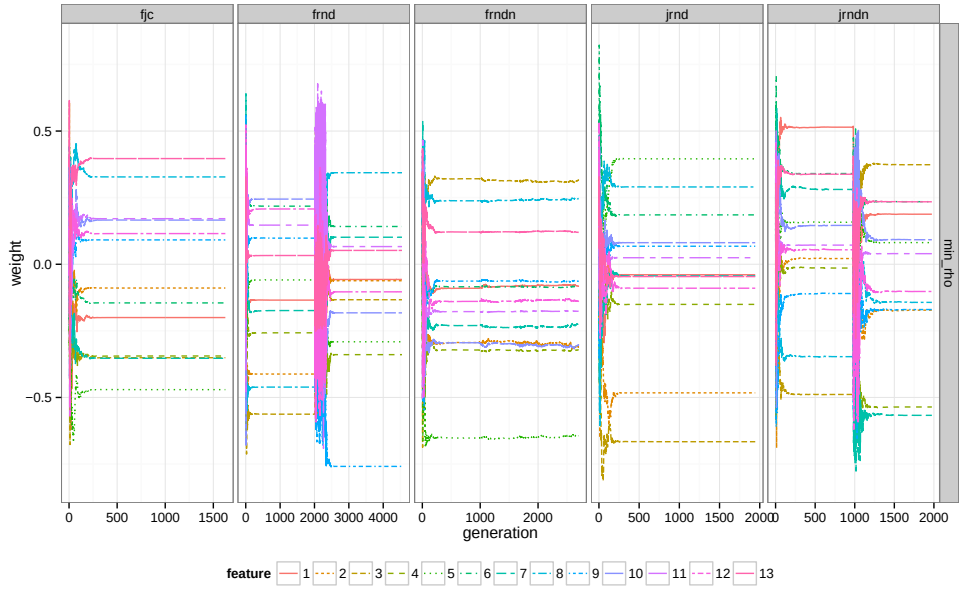
$\mathcal{P}$	#gen	#eval	$ES_{C_{\max}}$
j.rnd	4707	51788	448.612
j.rndn	4802	52833	449.942
f.rnd	5088	55979	571.394
f.rndn	5557	61138	544.764
f.jc	5984	65835	567.688

(b) w.r.t. Eq. (8.2)

$\mathcal{P}$	#gen	#eval	ES_{ρ}
j.rnd	1944	21395	8.258
j.rndn	1974	21725	8.691
f.rnd	4546	50006	7.479
f.rndn	2701	29722	0.938
f.jc	1625	17886	0.361



(a) minimise w.r.t. Eq. (8.1)



(b) minimise w.r.t. Eq. (8.2)

Figure 8.3: Evolution of weights of features (given in Table 2.2) at each generation of the CMA-ES optimisation. Note, weights are normalised such that $\|\mathbf{w}\| = 1$.

Table 8.2: Main statistics of percentage relative deviation from optimality, ρ, defined by Eq. (2.17) for various models, using corresponding 6×5 training data.

(a) $\mathcal{P}^{6\times5}_{j.rnd}$

model	mean	med	sd	min	max
$ES_{C_{max}}$	8.54	10	6	0	26
ES_{ρ}	8.26	10	6	0	26
PREF	10.18	11	7	0	30
MWR	16.48	16	9	0	45

(b) $\mathcal{P}^{6\times5}_{j.rndn}$

model	mean	med	sd	min	max
$ES_{C_{max}}$	8.68	11	6	0	31
ES_{ρ}	8.69	11	6	0	31
PREF	10.00	11	6	0	31
MWR	14.02	13	8	0	37

(c) $\mathcal{P}^{6\times5}_{f.rnd}$

model	mean	med	sd	min	max
$ES_{C_{max}}$	7.44	7	5	0	23
ES_{ρ}	7.48	7	5	0	34
PREF	9.87	9	7	0	38
LWR	20.05	19	10	0	71

(d) $\mathcal{P}^{6\times5}_{f.rndn}$

model	mean	med	sd	min	max
$ES_{C_{max}}$	8.09	8	2	0	11
ES_{ρ}	0.94	1	1	0	4
PREF	2.38	2	1	0	7
LWR	2.25	2	1	0	7

(e) $\mathcal{P}^{6\times5}_{f.jc}$

model	mean	med	sd	min	max
$ES_{C_{max}}$	0.33	0	0	0	2
ES_{ρ}	0.36	0	0	0	2
PREF	1.08	1	1	0	5
LWR	1.13	1	1	0	6

Table 8.3: Main statistics of $C_{\max}$ for various models, using corresponding 10×10 test data.

(a) $\mathcal{P}^{10\times10}_{j.rnd}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	922.51	914	73	741	1173
ES_{ρ}	931.37	931	71	735	1167
PREF	1011.38	1004	82	809	1281
MWR	997.01	992	81	800	1273

(b) $\mathcal{P}^{10\times10}_{j.rndn}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	855.85	857	50	719	1010
ES_{ρ}	855.91	856	51	719	1020
PREF	899.94	898	56	769	1130
MWR	897.39	898	56	765	1088

(c) $\mathcal{P}^{10\times10}_{f.rnd}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	1178.73	1176	80	976	1416
ES_{ρ}	1181.91	1179	80	984	1404
PREF	1215.20	1212	80	1006	1450
LWR	1284.41	1286	85	1042	1495

(d) $\mathcal{P}^{10\times10}_{f.rndn}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	1065.48	1059	32	992	1222
ES_{ρ}	980.11	980	8	957	1006
PREF	987.49	988	9	958	1011
LWR	986.94	987	9	959	1010

(e) $\mathcal{P}^{10\times10}_{f.jc}$

model	mean	med	sd	min	max
$ES_{C_{\max}}$	1135.44	1134	286	582	1681
ES_{ρ}	1135.47	1134	286	582	1681
PREF	1136.02	1135	286	582	1685
LWR	1136.49	1141	287	581	1690

8.4 Discussion and conclusions

Data distributions considered in this study either varied w.r.t. the processing time distributions, continuing the preliminary experiments in Paper I, or w.r.t. the job ordering permutations – i.e., homogeneous machine order for FSP versus heterogeneous machine order for JSP. From the results based on 6×5 training data given in Table 8.2, it's obvious that CMA-ES optimisation substantially outperforms the previous PREF methods from Paper I for all problem spaces considered. Furthermore, the results hold when testing on 10×10 (cf. Table 8.3), suggesting the method is indeed scalable to higher dimensions.

Moreover, the study showed that the choice of objective function for evolutionary search is worth investigating. There was no statistical difference from minimising the fitness function directly and its normalisation w.r.t. true optimum (cf. Eqs. (8.1) and (8.2)), save for $\mathcal{P}^{6\times5}_{f.rndn}$. Implying, even though ES doesn't rely on optimal solutions, there are some problem spaces where it can be of great benefit. This is due to the fact that the problem instances can vary greatly within the same problem space Paper III. Thus normalising the objective function would help the evolutionary search to deviate from giving too much weight for problematic problem instances.

The weights for Eq. (2.12) in Paper I were found using supervised learning, where the training data was created from optimal solutions of randomly generated problem instances. As an alternative, this study showed that minimising the mean makespan directly using a brute force search via CMA-ES actually results in a better CDRs. The nature of CMA-ES is to explore suboptimal routes until it converges to an optimal one. Implying that the previous approach of only looking into one optimal route may not produce a sufficiently rich training set. That is, the training set should incorporate a more complete knowledge on *all* possible preferences, i.e., make also the distinction between suboptimal and sub-suboptimal features, etc. This would require a Pareto ranking of preferences which can be used to make the distinction to which feature sets are equivalent, better or worse – and to what degree, i.e., by giving a weight to the preference. This would result in a very large training set, which of course could be re-sampled in order to make it computationally feasible to learn.

The main drawback of using evolutionary search for learning optimal weights for Eq. (2.12) is how computationally expensive it is to evaluate the mean expected fitness. Even for a low problem dimension 6-job 5-machine JSP, each optimisation run reached their walltime of 288 hours without converging. Now, 6×5 JSP requires 30 sequential operations where at each time step there are up to 6 jobs to choose from – i.e., its complexity is $\mathcal{O}(n^{n \cdot m})$ making it computationally infeasible to apply this framework for higher dimensions as is. However, evolutionary search only requires the rank of the

candidates and therefore it is appropriate to retain a sufficiently accurate surrogate for the value function during evolution in order to reduce the number of costly true value function evaluations, such as the approach in Paper II. This could reduce the computational cost of the evolutionary search considerably, making it feasible to conduct the experiments from Section 8.1 for problems of higher dimensions, e.g., with these adjustments it is possible to train on 10×10 and test on for example 14×14 to verify whether scalability holds for even higher dimensions.

8.4. A prevalent approach to solving job shop scheduling problems is to combine several relatively simple dispatching rules such that they may benefit each other for a given problem space. Generally, this is done in an ad-hoc fashion, requiring expert knowledge from heuristics designers, or extensive exploration of suitable combinations of heuristics. The approach here is to automate that selection by translating dispatching rules into measurable features and optimising what their contribution should be via evolutionary search. The framework is straight forward and easy to implement and shows promising results. Various data distributions are investigated for both job shop and flow shop problems, as is scalability for higher dimensions. Moreover, the study shows that the choice of objective function for evolutionary search is worth investigating. Since the optimisation is based on minimising the expected mean of the fitness function over a large set of problem instances which can vary within the set, then normalising the objective function can stabilise the optimisation process away from local minima.

The adventures first... explanations take such a dreadful time.

The Gryphon

9

Experiments

9.0. Compare CMA-ES to PREF models

There's something to be said for having a good opening line. Morbi commodo, ipsum sed pharetra gravida, orci $x = 1/a$ magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

$$\zeta = \frac{1039}{\pi}$$

9.0. When applying rollout features from **??**, then is sensible to keep track of the best solution found (even though they hadn't been followed), they will referred to as *fortified* solution.

Table 9.1: Results for time dependent models, i.e., each consists of K-models – one for each dispatch step.

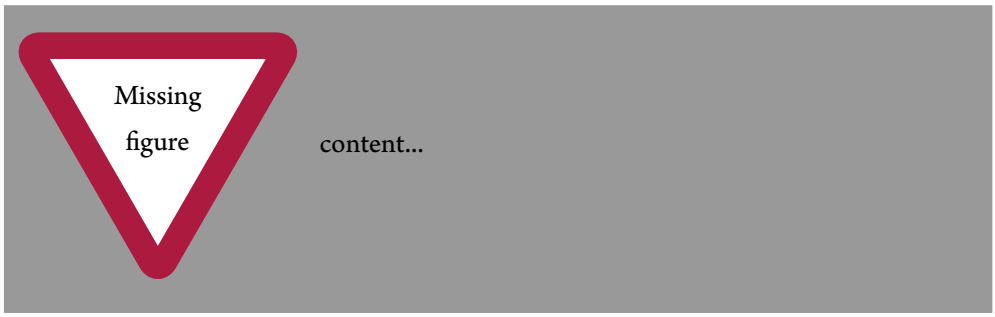
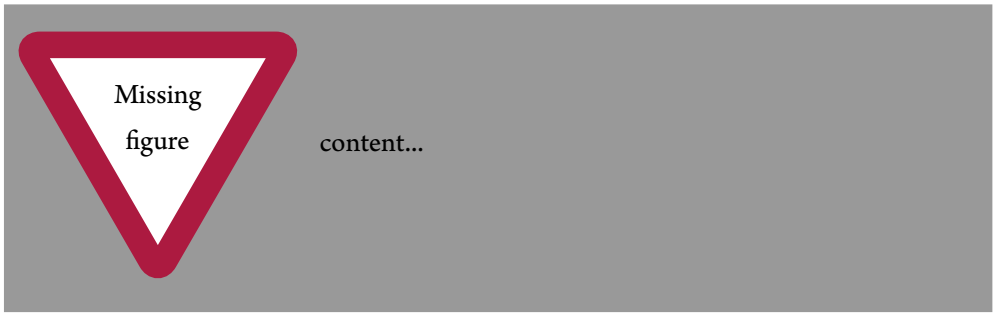


Table 9.2: Results for time independent models



9.1 Time dependent models

9.2 Time independent models

Tut, tut, child! Everything's got a moral, if only you can find it.

The Duchess

10 Conclusions

10.0. Write overall conclusions of dissertation!

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper

The analysis-phase of ALICE is heavily dependent on having an expert policy one wants to mimic, i.e., knowing the *optimal* solutions for the sake of imitation learning.

Understandably, knowing the true optimum is an unreasonable claim in many situations, especially for high dimensional problem instances. Luckily, there seems to be the possibility to circumvent querying the expert altogether, and still have reasonable performance. By applying *locally optimal learning to search* (Chang et al., 2015) it is possible to use imitation learning even when the reference policy is poor. Although it's noted that the quality (w.r.t near-optimality) of reference policy is in accordance to its performance, as is to be expected.

CHAPTER 10. CONCLUSIONS

A cat may look at a king. I've read that in some book, but I don't remember where.

Alice

References

J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3):391–401, 1988.

B. Ak and E. Koc. A Guide for Genetic Algorithm Based on Parallel Machine Scheduling and Flexible Job-Shop Scheduling. *Procedia - Social and Behavioral Sciences*, 62:817–823, Oct. 2012.

M. Ancău. On solving flowshop scheduling problems. *Proceedings of the Romanian Academy. Series A*, 13(1):71–79, 2012.

D. Applegate and W. Cook. A computational study of the job-shop scheduling instance. *ORSA Journal on Computing*, 3:149–156, 1991.

H. Asmuni, E. K. Burke, J. M. Garibaldi, B. McCollum, and A. J. Parkes. An investigation of fuzzy multiple heuristic orderings in the construction of university examination timetables. *Computers and Operations Research*, 36(4):981–1001, 2009.

A. Banharnsakun, B. Sirinaovakul, and T. Achalakul. Job shop scheduling with the best-so-far abc. *Engineering Applications of Artificial Intelligence*, 25(3):583–593, 2012.

J. E. Beasley. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990. URL `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`.

D. A. Berry and B. Fristedt. *Bandit Problems: Sequential Allocation of Experiments*. Springer, October 1985.

D. P. Bertsekas, J. N. Tsitsiklis, and C. Wu. Rollout algorithms for combinatorial optimization. *Journal of Heuristics*, 3(3):245–262, 1997.

N. Biggs, E. K. Lloyd, and R. J. Wilson. *Graph Theory, 1736-1936*. Clarendon Press, New York, NY, USA, 1986.

P. Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183, 1993.

E. Burke, S. Petrovic, and R. Qu. Case-based heuristic selection for timetabling problems. *Journal of Scheduling*, 9:115–132, 2006.

E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64 (12):1695–1724, 2013.

J. Carlier. Ordonnancements a contraintes disjonctives. *Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*, 12(4):333–350, 1978.

L. Carroll. *Alice's Adventures in Wonderland*. Macmillan, 1865.

L. Carroll. *Through the Looking-Glass, and What Alice Found There*. Macmillan, 1871.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

F.-C. R. Chang. A study of due-date assignment rules with constrained tightness in a dynamic job shop. *Computers and Industrial Engineering*, 31(1–2):205–208, 1996.

K. Chang, A. Krishnamurthy, A. Agarwal, H. D. III, and J. Langford. Learning to search better than your teacher. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 2058–2066, 2015.

T. Chen, C. Rajendran, and C.-W. Wu. Advanced dispatching rules for large-scale manufacturing systems. *The International Journal of Advanced Manufacturing Technology*, Feb. 2013.

R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation. *Computers & Industrial Engineering*, 30 (4):983–997, 1996.

R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, 36(2):343–364, Apr. 1999.

D. Corne and A. Reynolds. Optimisation and generalisation: Footprints in instance space. In R. Schaefer, C. Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 22–31. Springer, Berlin, Heidelberg, 2010.

E. Demirkol, S. Mehta, and R. Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.

A. Dhingra and P. Chandna. 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.

I. Drobouchevitch and V. Strusevich. Heuristics for the two-stage job shop scheduling problem with a bottleneck machine. *European Journal of Operational Research*, 123(2): 229 – 240, 2000. ISSN 0377-2217.

R. Dudek, S. Panwalkar, and M. Smith. The lessons of flowshop scheduling research. *Operations Research*, 40(1):7–13, 1992.

C. Duin and S. Voß. The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs. *Networks*, 34:181–191, 1999.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL `http://www.csie.ntu.edu.tw/~cjlin/liblinear`.

H. Fisher and G. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules*, pages 225–251. Prenice-Hall, Englewood Cliffs, N.J., 1963.

Free Software Foundation, Inc. GLPK (gnu linear programming kit) (version 4.55) [software], 2014. URL `http://www.gnu.org/software/glpk/`.

J. Gao, M. Gen, L. Sun, and X. Zhao. A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Comput. Ind. Eng.*, 53(1): 149–162, Aug. 2007. ISSN 0360-8352.

M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

E. Geirsson. Rollout algorithms for job-shop scheduling. Master's thesis, University of Iceland, Reykjavík, Iceland, May 2012.

B. Giffler and G. L. Thompson. Algorithms for solving production-scheduling problems. *Operations Research*, 8(4):487–503, 1960.

C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, Feb. 2001.

A. Guinet and M. Legrand. Reduction of job-shop problems to flow-shop problems with precedence constraints. *European Journal of Operational Research*, 109(1):96–110, 1998.

Gurobi Optimization, Inc. Gurobi optimization (version 6.0.0) [software], 2014. URL `http://www.gurobi.com/`.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, June 2001.

R. Haupt. A survey of priority rule-based scheduling. *OR Spectrum*, 11:3–16, 1989.

T. Helleputte. *LiblineaR: Linear Predictive Models Based on the LIBLINEAR C/C++ Library*, 2015. R package version 1.94-2.

J. Heller. Some numerical experiments for an m× j flow shop and its decision-theoretical aspects. *Operations Research*, 8(2):pp. 178–184, 1960.

R. Herbrich, T. Graepel, and K. Obermayer. *Large Margin Rank Boundaries for Ordinal Regression*, chapter 7. MIT Press, Mar. 2000.

T. Hildebrandt, J. Heger, and B. Scholz-Reiter. 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*, pages 257–264, 2010.

N. B. Ho, J. C. Tay, and E. M. Lai. An effective architecture for learning and evolving flexible job-shop schedules. *European Journal Of Operational Research*, 179:316–333, 2007.

A. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2):390–434, 1999.

M. Jayamohan and C. Rajendran. Development and analysis of cost-based dispatching rules for job shop scheduling. *European Journal of Operational Research*, 157(2):307–321, 2004.

T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

S. Kalyanakrishnan and P. Stone. Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, 84(1-2):205–247, June 2011.

A. Keane. Genetic programming, logic design and case-based reasoning for obstacle avoidance. In C. Dhaenens, L. Jourdan, and M.-E. Marmion, editors, *Learning and Intelligent Optimization*, volume 8994 of *Lecture Notes in Computer Science*, pages 104–118. Springer International Publishing, 2015. ISBN 978-3-319-19083-9.

P. Korytkowski, S. Rymaszewski, and T. Wiśniewski. Ant colony optimization for job shop scheduling using multi-attribute dispatching rules. *The International Journal of Advanced Manufacturing Technology*, Feb. 2013.

J. R. Koza and R. Poli. Genetic programming. In E. Burke and G. Kendal, editors, *Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 5. Springer, 2005.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

S. Lawrence. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

X. Li and S. Olafsson. Discovering dispatching rules using data mining. *Journal of Scheduling*, 8:515–527, 2005.

C.-J. Lin, R. C. Weng, and S. S. Keerthi. Trust region newton method for logistic regression. *J. Mach. Learn. Res.*, 9:627–650, June 2008.

M.-S. Lu and R. Romanowski. Multicontextual dispatching rules for job shops with dynamic job arrival. *The International Journal of Advanced Manufacturing Technology*, Jan. 2013.

A. M. Malik, T. Russell, M. Chase, and P. Beek. Learning heuristics for basic block instruction scheduling. *Journal of Heuristics*, 14(6):549–569, Dec. 2008.

A. S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

S. Meeran and M. Morshed. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. *Journal of intelligent manufacturing*, 23(4):1063–1078, 2012.

L. Mönch, J. W. Fowler, and S. J. Mason. *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*, volume 52 of *Operations Research/Computer Science Interfaces Series*, chapter 4. Springer, New York, 2013.

S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology*, Feb. 2013.

S. Olafsson and X. Li. Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, 128(1):118–126, 2010.

S. S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1): 45–61, 1977.

F. Pezzella, G. Morganti, and G. Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008. Part Special Issue: Search-based Software Engineering.

B. Pfahringer, H. Bensusan, and C. Giraud-carrier. Meta-learning by landmarking various learning algorithms. In *in Proceedings of the 17th International Conference on Machine Learning, ICML'2000*, pages 743–750. Morgan Kaufmann, 2000.

M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3 edition, 2008.

R. Qing-dao-er ji and Y. Wang. A new hybrid genetic algorithm for job shop scheduling problem. *Computers & Operations Research*, 39(10):2291–2299, Oct. 2012.

C. Reeves. A genetic algorithm for flowshop sequencing. *Computer Operations Research*, 22:5–13, 1995.

J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

K. H. Rosen. *Discrete Mathematics and Its Applications*, chapter 9, pages 631–700. McGraw-Hill, Inc., New York, NY, USA, 5 edition, 2003.

B. Roy and B. Sussmann. Les problemes d'ordonnancement avec contraintes disjonctives. *Note D.S.*, 9, 1964.

T. Runarsson. Ordinal regression in evolutionary computation. In T. Runarsson, H.-G. Beyer, E. Burke, J. Merelo-Guervós, L. Whitley, and X. Yao, editors, *Parallel Problem Solving from Nature - PPSN IX*, volume 4193 of *Lecture Notes in Computer Science*, pages 1048–1057. Springer, Berlin, Heidelberg, 2006.

T. Runarsson, M. Schoenauer, and M. Sebag. Pilot, rollout and monte carlo tree search methods for job shop scheduling. In Y. Hamadi and M. Schoenauer, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 160–174. Springer Berlin Heidelberg, 2012.

T. Russell, A. M. Malik, M. Chase, and P. van Beek. Learning heuristics for the superblock instruction scheduling problem. *IEEE Trans. on Knowl. and Data Eng.*, 21(10):1489–1502, Oct. 2009.

I. Sabuncuoglu and M. Bayiz. Job shop scheduling with beam search. *European Journal of Operational Research*, 118(2):390–412, 1999.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

K. Smith-Miles and S. Bowly. Generating new test instances by evolving in instance space. *Computers & Operations Research*, 63:102–113, 2015.

K. Smith-Miles and L. Lopes. Generalising algorithm performance in instance space: A timetabling case study. In C. Coello, editor, *Learning and Intelligent Optimization*, volume 6683 of *Lecture Notes in Computer Science*, pages 524–538. Springer, Berlin, Heidelberg, 2011.

K. Smith-Miles, R. James, J. Giffin, and Y. Tu. A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. In T. Stützle, editor, *Learning and Intelligent Optimization*, volume 5851 of *Lecture Notes in Computer Science*, pages 89–103. Springer, Berlin, Heidelberg, 2009.

R. H. Storer, S. D. Wu, and R. Vaccari. New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10):1495–1509, 1992.

É. D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, pages 1–17, 1993.

J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers and Industrial Engineering*, 54 (3):453–473, 2008.

S. Thiagarajan and C. Rajendran. Scheduling in dynamic assembly job-shops to minimize the sum of weighted earliness, weighted tardiness and weighted flowtime of jobs. *Computers and Industrial Engineering*, 49(4):463–503, 2005.

J.-T. Tsai, T.-K. Liu, W.-H. Ho, and J.-H. Chou. 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, Aug. 2007.

R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by local search. *INFORMS Journal on Computing*, 8(3):302–317, 1996.

J. A. Vázquez-Rodríguez and S. Petrovic. A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, 16(6):771–793, Dec. 2009.

R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 2002.

J.-P. Watson, L. Barbulescu, L. D. Whitley, and A. E. Howe. Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14:98–123, 2002.

REFERENCES

F. Werner and A. Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58(2):191–211, 1995.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

W. Xia and Z. Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48(2):409–425, 2005.

L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown. SATzilla-07: The design and analysis of an algorithm portfolio for SAT. *Principles and Practice of ...*, 2007.

T. Yamada and R. Nakano. A genetic algorithm applicable to large-scale job-shop instances. In B. M. R. Manner, editor, *Parallel Problem Solving from Nature - PPSN II*, pages 281–290. Elsevier, 1992.

J.-M. Yu, H.-H. Doh, J.-S. Kim, Y.-J. Kwon, D.-H. Lee, and S.-H. Nam. Input sequencing and scheduling for a reconfigurable manufacturing system with a limited number of fixtures. *The International Journal of Advanced Manufacturing Technology*, Jan. 2013.

W. Zhang and T. G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th international joint conference on Artificial Intelligence*, volume 2 of *IJCAI'95*, pages 1114–1120, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

What is the use of repeating all that stuff, if you don't explain it as you go on? It's by far the most confusing thing I ever heard!

The Mock Turtle

A Ordinal Regression

Ordinal regression has been previously presented in Runarsson (2006), but given here for completeness. The preference learning task of linear classification presented there is based on the work proposed in (Fan et al., 2008, Lin et al., 2008). The modification relates to how the point pairs are selected and the fact that a $L2$-regularized logistic regression is used.

A.1 Preference set

The ranking problem is specified by a *preference set*,

$$\Psi := \{(\mathbf{x}_i, y_i)\}_{i=1}^{N} \subset X \times Y \tag{A.1}$$

consisting of N (solution, rank)-pairs, where $Y = \{r_1, \ldots, r_N\}$ is the outcome space with ordered ranks $r_1 > r_2, > \ldots > r_N$.

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

$$\mathbf{x}_i \succ \mathbf{x}_j \quad \Leftrightarrow \quad h(\mathbf{x}_i) > h(\mathbf{x}_j) \tag{A.2}$$

where the symbol $\succ$ denotes 'is preferred to.'

In ordinal regression the task is to obtain function h that can for a given pair $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$ distinguish between two different outcomes: $y_i > y_j$ and $y_j > y_i$. The task is, therefore, transformed into the problem of predicting the relative ordering of all possible pairs of examples (Herbrich et al., 2000, Joachims, 2002). However, it is sufficient to consider only all possible pairs of adjacent ranks (see also Shawe-Taylor and Cristianini (2004) for yet an alternative formulation). The preference set, composed of pairs, is then as follows,

$$\Psi = \left\{ (\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}), t_k = \operatorname{sign}(y_k^{(1)} - y_k^{(2)}) \right\}_{k=1}^{N'} \subset X \times Y \tag{A.3}$$

where $(y_k^{(1)} = r_i) \wedge (y_k^{(2)} = r_{i+1})$, and vice versa $(y_k^{(1)} = r_{i+1}) \wedge (y_k^{(2)} = r_i)$, resulting in $N' = 2(N-1)$ possible adjacently ranked preference pairs. The rank difference is denoted by $t_k \in \{-1, 1\}$.

In order to generalize the technique to different solution data types and model spaces an implicit kernel-defined feature space $\Phi \subset \mathbb{R}^d$ of dimension d, with corresponding feature mapping $\boldsymbol{\varphi} : X \mapsto \Phi$ is applied, i.e., the feature vector $\boldsymbol{\varphi}(\mathbf{x}) = [\varphi_1(\mathbf{x}), \ldots, \varphi_d(\mathbf{x})]^T \in \Phi$. Thus the preference set defined by Eq. (A.3) is redefined as follows,

$$\Psi = \left\{ \left(\boldsymbol{\varphi}(\mathbf{x}_k^{(1)}), \boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) \right), t_k = \operatorname{sign}(y_k^{(1)} - y_k^{(2)}) \right\}_{k=1}^{N'} \subset \Phi \times Y. \tag{A.4}$$

A.2 Linear preference

The function used to induce the preference is defined by a linear function in the kernel-defined feature space,

$$h(\mathbf{x}) = \sum_{i=1}^{d} w_i \varphi_i(\mathbf{x}) = \left\langle \mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle \tag{A.5}$$

where $\mathbf{w} = [w_1, \ldots, w_d] \in \mathbb{R}^d$ has weight w_i for feature φ_i.

The aim now is to find a function h that encounters as few training errors as possible on Ψ. Applying the method of large margin rank boundaries of ordinal regression described in Herbrich et al. (2000), the optimal $\mathbf{w}^*$ is determined by solving the following task,

$$\min_{\mathbf{w}} \quad \tfrac{1}{2} \left\langle \mathbf{w} \cdot \mathbf{w} \right\rangle + \tfrac{C}{2} \sum_{k=1}^{N'} \xi_k^2 \tag{A.6}$$

subject to $t_k \left\langle \mathbf{w} \cdot (\boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) - \boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) \right\rangle \geq 1 - \xi_k$ and $\xi_k \geq 0$, $k = 1, \ldots, N'$. The degree of constraint violation is given by the margin slack variable ξ_k and when greater than 1 the

corresponding pair are incorrectly ranked. Note that,

$$h(\mathbf{x}_i) - h(\mathbf{x}_j) = \left\langle \mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x}_i) - \boldsymbol{\varphi}(\mathbf{x}_j) \right\rangle \tag{A.7}$$

and minimising $\left\langle \mathbf{w} \cdot \mathbf{w} \right\rangle$ in Eq. (A.6) maximises the margin between rank boundaries, i.e., the distance between adjacently ranked pair $h(\mathbf{x}^{(1)})$ and $h(\mathbf{x}^{(2)})$.

A.3 Non-linear preference

In the case that the preference set Υ defined by Eq. (A.4) is not linearly separable, a common way of coping with non-linearity is to apply the 'kernel-trick' to transform Υ onto a higher dimension. In which case, the dot product in Eq. (A.5) is replaced by a kernel function κ.

In terms of training data, the optimal $\mathbf{w}^*$ can be expressed as,

$$\mathbf{w}^* = \sum_{k=1}^{N'} a^* t_k \left(\boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) - \boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) \right) \tag{A.8}$$

and the function h may be reconstructed as follows,

$$\begin{aligned} h(\mathbf{x}) = \left\langle \mathbf{w}^* \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle &= \sum_{k=1}^{N'} a^* t_k \left(\left\langle \boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle - \left\langle \boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle \right) \right) \\ &= \sum_{k=1}^{N'} a^* t_k \left(\kappa(\mathbf{x}_k^{(1)}, \mathbf{x}) - \kappa(\mathbf{x}_k^{(2)}, \mathbf{x}) \right) \end{aligned} \tag{A.9}$$

where $\kappa(\mathbf{x}, \mathbf{z}) = \left\langle \boldsymbol{\varphi}(\mathbf{x}) \cdot \boldsymbol{\varphi}(\mathbf{z}) \right\rangle$ is the chosen kernel and a_k^* are the Lagrangian multipliers for the constraints that can be determined by solving the dual quadratic programming problem,

$$\max_a \sum_{k=1}^{N'} a_k - \frac{1}{2} \sum_{i=1}^{N'} \sum_{j=1}^{N'} a_i a_j t_i t_j \left(K(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)}) + \frac{1}{C} \delta_{ij} \right) \tag{A.10}$$

subject to $\sum_{k=1}^{N'} a_k t_k = 0$ and $a_k \geq 0$ for all $k \in \{1, \ldots, N'\}$, and where,

$$K(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)}) = \kappa(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(1)}) - \kappa(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(2)}) - \kappa(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}) + \kappa(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(2)})$$

and δ_{ij} is the Kronecker δ defined to be 1 iff $i = j$ and 0 otherwise.

Kernel functions

There are several choices for a kernel κ, e.g., *polynomial kernel*,

$$\kappa_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) \quad = \quad \left(1 + \left\langle \mathbf{x}_i \cdot \mathbf{x}_j \right\rangle\right)^p \tag{A.11}$$

of order p, or the most commonly used kernel in the literature which implements a Gaussian radial basis function, the *rbf kernel*,

$$\kappa_{\text{rbf}}(\mathbf{x}_i, \mathbf{x}_j) \quad = \quad e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2} \tag{A.12}$$

for $\gamma > 0$.

A.4 Parameter setting and tuning

The regulation parameter C in Eqs. (A.6) and (A.10), controls the balance between model complexity and training errors, and must be chosen appropriately. A high value for C gives greater emphasis on correctly distinguishing between different ranks, whereas a low C value results in maximising the margin between classes.

A.5 Scaling

It some cases it becomes necessary to scale the features $\boldsymbol{\varphi}$ from Section 2.5 first, especially if implementing a kernel method in Eq. (A.5). In the case of JSP, scaling makes the features less sensitive to varying problem instances. Moreover, for surrogate modelling (cf. Paper II), it is important to scale the features $\boldsymbol{\varphi}$ as the evolutionary search zooms in on a particular region of the search space.

A standard method of doing so is by scaling the preference set such that all points are in some range, typically $[-1, 1]$. That is, scaled $\tilde{\boldsymbol{\varphi}}$ is,

$$\tilde{\varphi}_i = 2(\varphi_i - \underline{\varphi}_i)/(\overline{\varphi}_i - \underline{\varphi}_i) - 1 \qquad \forall i \in \{1, \ldots, d\} \tag{A.13}$$

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

$$\underline{\varphi}_i = \min\{\varphi_i \mid \forall \boldsymbol{\varphi} \in \Phi\} \quad \text{and} \quad \overline{\varphi}_i = \max\{\varphi_i \mid \forall \boldsymbol{\varphi} \in \Phi\} \tag{A.14}$$

where $i \in \{1 \ldots d\}$.

A.6 Implementation

To linear ordinal regression, then it's best to use LIBLINEAR: A Library for Large Linear Classification by Fan et al. (2008), which contains implementations in several programming languages. The preferred choice of the author was the R-package `LiblineaR` by Helleputte (2015). However, if more sophisticated kernel methods are sought after, then LIBSVM: A Library for Support Vector Machines by Chang and Lin (2011) is an obvious substitute.

APPENDIX A. ORDINAL REGRESSION

Part II

Papers

But it's no use going back to yesterday, because I was a different person then.

Alice

I

Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

School of Engineering and Natural Sciences, University of Iceland, Iceland

Learning and Intelligent Optimization

Reprinted, with permission, from *Learning and Intelligent Optimization* (2011). Copyright 2011 by Springer Berlin Heidelberg.

I

This page is intentionally left blank.

Take care of the sense, and the sounds will take care of themselves.

The Duchess

II

Sampling Strategies in Ordinal Regression for Surrogate Assisted Evolutionary Optimization

Helga Ingimundardóttir, Tómas Philip Rúnarsson

School of Engineering and Natural Sciences, University of Iceland, Iceland

Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on

Reprinted, with permission, from *Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on* (2011). Copyright 2011 by IEEE.

II

This page is intentionally left blank.

I wonder if I've been changed in the night? Let me think. Was I the same when I got up this morning? I almost think I can remember feeling a little different. But if I'm not the same, the next question is 'Who in the world am I?' Ah, that's the great puzzle!

Alice

III

Determining the Characteristic of Difficult Job Shop Scheduling Instances for a Heuristic Solution Method

Helga Ingimundardóttir, Tómas Philip Rúnarsson

School of Engineering and Natural Sciences, University of Iceland, Iceland

Learning and Intelligent Optimization

Reprinted, with permission, from *Learning and Intelligent Optimization* (2012). Copyright 2012 by Springer Berlin Heidelberg.

III

This page is intentionally left blank.

It would be so nice if something made sense for a change.

Alice

IV

Evolutionary Learning of Weighted Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir

School of Engineering and Natural Sciences, University of Iceland, Iceland

International Conference on Evolutionary Computation Theory and Applications (ECTA) – authorslist

IV.0. Find second publication for Paper IV – Selected papers ISI?

Reprinted, with permission, from *International Conference on Evolutionary Computation Theory and Applications (ECTA)* (2014). Copyright 2014 by SCITEPRESS.

IV

This page is intentionally left blank.

Reeling and Writhing, of course, to begin with, and then the different branches of arithmetic – Ambition, Distraction, Uglification, and Derision.

The Mock Turtle

V

Generating Training Data for Learning Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

School of Engineering and Natural Sciences, University of Iceland, Iceland

Learning and Intelligent Optimization – Nominated for best paper award

Reprinted, with permission, from *Learning and Intelligent Optimization* (2015). Copyright 2015 by Springer International Publishing.

V

This page is intentionally left blank.

If everybody minded their own business, the world would go around a great deal faster than it does.

The Duchess

VI

Supervised Learning Linear Composite Dispatch Rules for Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

School of Engineering and Natural Sciences, University of Iceland, Iceland

NA – Accepted

VI.0. Paper VI not yet submitted

Reprinted, with permission, from *NA* (2015). Copyright 2015 by Not yet submitted.

VI

This page is intentionally left blank.

But then, shall I never get any older than I am now? That'll be a comfort, one way – never to be an old woman – but then – always to have lessons to learn!

Alice

VII

Imitation Learning Linear Composite Dispatch Rules for Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

School of Engineering and Natural Sciences, University of Iceland, Iceland

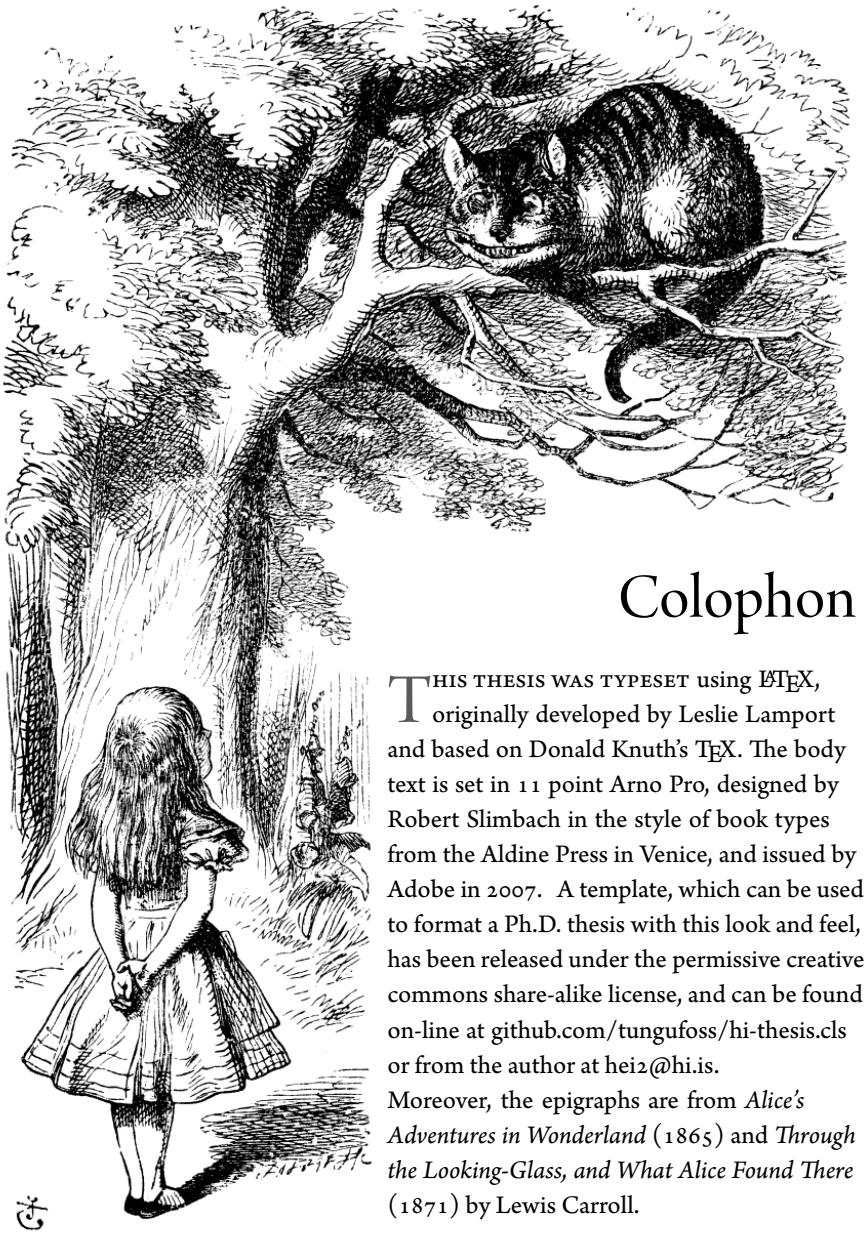
NA – Submitted

VII.0. Paper VII not yet submitted

Reprinted, with permission, from *NA* (2015). Copyright 2015 by Not yet submitted.

VII

This page is intentionally left blank.



Colophon

This thesis was typeset using LaTeX, originally developed by Leslie Lamport and based on Donald Knuth's TeX. The body text is set in 11 point Arno Pro, designed by Robert Slimbach in the style of book types from the Aldine Press in Venice, and issued by Adobe in 2007. A template, which can be used to format a Ph.D. thesis with this look and feel, has been released under the permissive creative commons share-alike license, and can be found on-line at github.com/tungufoss/hi-thesis.cls or from the author at hei2@hi.is.

Moreover, the epigraphs are from *Alice's Adventures in Wonderland* (1865) and *Through the Looking-Glass, and What Alice Found There* (1871) by Lewis Carroll.

If you don't know where you are going, any road will take you there.

Lewis Carroll

Alice's Adventures in Wonderland

What is the use of a book, without pictures or conversations? **Alice**

Oh my ears and whiskers, how late it's getting! **Rabbit**

Curiouser and curiouser! **Alice**

Paper III: ~~*I wonder if I've been changed in the night? Let me think. Was I the same when I got up this morning? I almost think I can remember feeling a little different. But if I'm not the same, the next question is 'Who in the world am I?' Ah, that's the great puzzle!*~~ **Alice**

Speak English! I don't know the meaning of half those long words, and I don't believe you do either! **Eaglet**

I can't explain myself, I'm afraid, Sir, because I'm not myself you see. **Alice**

Paper VI: ~~*If everybody minded their own business, the world would go around a great deal faster than it does.*~~ **The Duchess**

Chapter 3: ~~*If it had grown up, it would have made a dreadfully ugly child; but it makes rather a handsome pig, I think.*~~ **Alice**

We're all mad here. **The Cat**

Why is a raven like a writing desk? **The Hatter**

Twinkle, twinkle, little bat! How I wonder what you're at. **The Hatter**

Off with her head! **The Queen**

Chapter 10: ~~*Tut, tut, child! Everything's got a moral, if only you can find it.*~~ **The Duchess**

Paper II: ~~*Take care of the sense, and the sounds will take care of themselves.*~~ **The Duchess**

We called him Tortoise because he taught us. **The Mock Turtle**

Paper V: ~~*Reeling and Writhing, of course, to begin with, and then the different branches of arithmetic – Ambition, Distraction, Uglification, and Derision.*~~ **The Mock Turtle**

Well, I never heard it before, but it sounds uncommon nonsense. **The Mock Turtle**

Chapter 1: ~~*Begin at the beginning and go on till you come to the end: then stop.*~~ **The King**

Chapter 6: ~~*I don't believe there's an atom of meaning in it.*~~ **Alice**

Chapter 4: ~~*Sentence first – verdict afterwards.*~~ **The Queen**

You're nothing but a pack of cards! **Alice**

Paper VII: ~~*But then, shall I never get any older than I am now? That'll be a comfort, one way – never to be an old woman – but then – always to have lessons to learn!*~~ **Alice**

References: ~~*A cat may look at a king. I've read that in some book, but I don't remember where.*~~ **Alice**

I think I should understand that better, if I had it written down: but I can't quite follow it as you say it. **Alice**

That's nothing to what I could say if I chose. **The Duchess**

Now, I give you fair warning, either you or your head must be off, and that in about half no time! Take your choice! **The Queen**

Paper IV: ~~*It would be so nice if something made sense for a change.*~~ **Alice**

Chapter 2: ~~*Read the directions and directly you will be directed in the right direction.*~~ **Doorknob**

No wonder you're late. Why, this watch is exactly two days slow. **Mad Hatter**

Let me see: four times five is twelve, and four times six is thirteen, and four times seven is – oh dear! I shall never get to twenty at that rate! **Alice**

Chapter 7: ~~*It was much pleasanter at home, when one wasn't always growing larger and smaller, and being ordered about by mice and rabbits.*~~ **Alice**

Chapter 5: ~~*Well! I've often seen a cat without a grin; but a grin without a cat! It's the most curious thing I ever say in my life!*~~ **Alice**

Chapter 8: ~~*There's a large mustard mine near here. And the moral of that is – The more there is of mine, the less there is of yours.*~~ **The Duchess**

Appendix A: ~~*What is the use of repeating all that stuff, if you don't explain it as you go on? It's by far the most confusing thing I ever heard!*~~ **The Mock Turtle**

Ah! Then yours wasn't a really good school. Now at ours they had at the end of the bill. French, music, and washing – extra. **The Mock Turtle**

Oh, how I wish I could shut up like a telescope! I think I could, if I only knew how to begin. **Alice**

Paper I: ~~*But it's no use going back to yesterday, because I was a different person then.*~~ **Alice**

It was much pleasanter at home, when one wasn't always growing larger and smaller, and being ordered about by mice and rabbits. **Alice**

Chapter 9: ~~*The adventures first... explanations take such a dreadful time.*~~ **The Gryphon**

Either the well was very deep, or she fell very slowly, for she had plenty of time as she went down to look about her and to wonder what was going to happen next. **Narrator**

Let me see: four times five is twelve, and four times six is thirteen, and four times seven is - oh dear! I shall never get to twenty at that rate! However, the Multiplication Table doesn't signify: let's try Geography. London is the capital of Paris, and Paris is the capital of Rome, and Rome - no, that's all wrong, I'm certain! I must have been changed for Mabel! **Alice**

I have proved by actual trial that a letter, that takes an hour to write, takes only about 3 minutes to read

Lewis Carroll

Through the Looking-Glass

Quotes from *Through the Looking-Glass, and What Alice Found There* (1871) by Lewis Carroll. *It's a poor sort of memory that only works backward.* **The Queen**

Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that! **The Queen**

Now, Kitty, let's consider who it was that dreamed it all. This is a serious question, my dear, and you should not go on licking your paw like that – as if Dinah hadn't washed you this morning! **Alice**

You see, Kitty, it must have been either me or the Red King. He was part of my dream, of course – but then I was part of his dream, too! Was it the Red King, Kitty? You were his wife, my dear, so you ought to know – oh, Kitty, do help to settle it! I'm sure your paw can wait!" But the provoking kitten only began on the other paw, and pretended it hadn't heard the question. **Alice**

Sometimes I've believed as many as six impossible things before breakfast. **The Queen**

Always speak the truth, think before you speak, and write it down afterwards. **The Queen**