ALICE: Analysis & Learning Iterative Consecutive Executions

Helga Ingimundardóttir

Faculty of Industrial Eng., Mechanical Eng. and Computer Science
University of Iceland
2015

Dissertation for the degree of doctor of philosophy

ALICE: Analysis & Learning Iterative Consecutive Executions

Helga Ingimundardóttir

School of Engineering and Natural Sciences
Faculty of Industrial Eng., Mechanical Eng. and Computer Science
Reykjavík, December 2015

A dissertation presented to the University of Iceland School of Engineering and Natural Sciences in candidacy for the degree of doctor of philosophy.

Doctoral committee
Prof. Tómas Philip Rúnarsson
Faculty of Engineering, University of Iceland

Prof. Gunnar Stefánsson
Faculty of Physical Sciences, University of Iceland

Michèle Sebag
TAO (INRIA Saclay – Île-de-France)

Opponents
Unknown
Undefined

Unknown
Undefined

ALICE: Analysis & Learning Iterative Consecutive Executions
© 2015 Helga Ingimundardóttir
Printed in Iceland by Unkown
ISBN Undefined

Allt fyrir móður mína

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

Abstract

Over the years there have been many approaches to create dispatching rules for scheduling. Recent past efforts have focused on direct search methods (e.g. genetic programming) or training on data (e.g. supervised learning). The dissertation will examine the latter and give a framework called Analysis & Learning Iterative Consecutive Executions (ALICE) on how to do it effectively.

Defining training data as $\{\boldsymbol{\varphi}(\mathbf{x}_i(k)), y_i(k)\}_{k=1}^{K} \in \mathcal{D}$ the dissertation will show: *i)* samples $\boldsymbol{\varphi}(\mathbf{x}_i)$ should represent the induced data distribution $\mathcal{D}$. This done by updating the learned model in an active imitation learning fashion; *ii)* y_i is labelled using an expert policy via a solver; *iii)* data needs to be balanced, as the set is unbalanced w.r.t. dispatching step k, and *iv)* to improve upon localised stepwise features $\boldsymbol{\varphi}$, it's possible to incorporate $(K-k)$ roll-outs where the learned model can be construed as a deterministic pilot heuristic.

When querying an expert policy, there is an abundance of valuable information that can be utilised for learning new models. For instance, it's possible to seek out when the scheduling process is most susceptible to failure. Furthermore, generally stepwise optimality (or classification accuracy) implies good end performance, here minimising the final makespan. However, as the impact of suboptimal moves is not fully understood, then the measure needs to be adjusted for its intended trajectory.

Using these guidelines, it becomes easier to create custom dispatching rules for one's particular application. For this several different distributions of job-shop will be considered. Moreover, the machine learning approach is based on preference learning which determines what feature states are preferable to others. However, that could easily be substituted for other learning methods or applied to other shop-constraints or family of scheduling problems that are based on iteratively applying dispatching rules.

Niður, niður, niður! Ætlaði þetta aldrei að taka enda? Hvað skyldi ég hafa hrapað marga kílómetra?

Lísa

Ágrip

Til eru margar aðferðir við að búa til ákvarðanareglur fyrir áætlanagerð. Undanfarið hefur áherslan í fræðunum verið á beina leit (t.d. gentíska bestun) eða gagnaþjálfun, en ein aðferð við það síðarnefnda er stýrður lærdómur. Í ritgerðinni verður sú aðferð skoðuð nánar og sett fram líkan kallað „Analysis & Learning Iterative Consecutive Executions“ (ALICE) um hvernig megi framkvæma þessa greiningu á skilvirkan máta.

Látum þjálfunargögnin vera $\{\boldsymbol{\varphi}(\mathbf{x}_i(k)), y_i(k)\}_{k=1}^{K} \in \mathcal{D}$ og ritgerðin mun sýna: *i)* úrtök $\boldsymbol{\varphi}(\mathbf{x}_i)$ þurfa að samræmast við gagnadreifinguna $\mathcal{D}$ sem verður unnin úr henni. Þetta er gert með því að uppfæra lærða líkanið með virku námsferli byggðu á eftirlíkingum; *ii)* y_i er merkt með því að nota endurgjöf sérfræðings (gert með bestun); *iii)* gögnin þurfa að vera í jafnvægi, þar sem gagnasettið er í ójafnvægi með tilliti til skrefs k; einnig *iv)* til að betrumbæta lýsingu á núverandi stöðu $\boldsymbol{\varphi}$, er hægt að nota útspilun fyrir næstu $(K - k)$ skref, það er að endalokum ákvarðanaferilsins. Þá má túlka lærða líkanið sem fyrirframákveðna útspilunarreglu.

Þegar sérfræðingur er spurður, verður til mikið af gagnlegum upplýsingum sem hægt er að nýta til að læra ný líkön. Til að mynda er hægt að komast að því hvenær í ákvarðanaferlinu er líklegast að mistök eigi sér stað. Yfirleitt gefa háar líkur á því að besta ákvörðun sé tekin (eða þjálfunarnákvæmni) til kynna góða lokaframmistöðu, þ.e. í þessu samhengi að lágmarka heildartíma fyrir allt ákvarðanaferlið. Þar sem afleiðingar rangra ákvarðana eru ekki alltaf þekktar, þá er betra að uppfæra matið með tilliti til ákvarðanatökunnar sjálfrar.

Með þessari greiningu er einfaldara að búa til sérhæfðar ákvarðanareglur fyrir hverja nýja notkun. Í ritgerðinni verða skoðaðar nokkrar mismunandi tegundir af verkniðurröðun á vélar. Þar að auki verður vélnámið byggt á ákjósanlegri bestun, þar sem gerður er greinarmunur á því hvaða stöður eru betri kostur en aðrar. Ákjósanlegri bestun væri þó hægt að skipta út fyrir aðrar námsaðferðir, hægt væri að bæta við fleiri skorðum á verkefnið eða beita sömu námsaðferð á aðra tegund af verkefnum af svipuðum toga.

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

Contents

Listing of figures **xi**

Listing of tables **xiii**

Listing of algorithms **xv**

Listing of publications **xvi**

Nomenclature **xviii**

I Prologue **1**

1 Introduction **3**
- 1.1 Rice's framework for algorithm selection 4
- 1.2 Previous work 5
- 1.3 Contributions 9
- 1.4 Supplementary material 11
- 1.5 Outline 11

2 Job-shop Scheduling Problem **14**
- 2.1 Mathematical formulation 15
- 2.2 Construction heuristics 16
- 2.3 Example 18
- 2.4 Single priority based dispatching rules 22
- 2.5 Features for job-shop 22
- 2.6 Composite dispatching rules 23
- 2.7 Rice's framework for job-shop 27

3 Problem generators **31**
- 3.1 Job-shop 32

3.2 Flow-shop . . . 33
3.3 Benchmark problem suite . . . 34

4 Problem difficulty **37**
4.1 Distribution difficulty . . . 39
4.2 Defining *easy* versus *hard* schedules . . . 41
4.3 Consistency of problem instances . . . 44
4.4 Conclusion . . . 48

5 Evolutionary Search **49**
5.1 Experimental setting . . . 50
5.2 Performance measures . . . 51
5.3 Experimental study . . . 51
5.4 Conclusions . . . 55

6 Generating Training Data **59**
6.1 Job-shop tree representation . . . 59
6.2 Labelling schedules w.r.t. optimal decisions . . . 61
6.3 Computational growth . . . 61
6.4 Trajectory sampling strategies . . . 62

7 Analysing Solutions **66**
7.1 Making optimal decisions . . . 67
7.2 Making suboptimal decisions . . . 67
7.3 Optimality of extremal features . . . 70
7.4 Simple blended dispatching rule . . . 79
7.5 Feature evolution . . . 82
7.6 Emergence of problem difficulty . . . 82
7.7 Conclusions . . . 89

8 Preference Learning **90**
8.1 Ordinal regression for job-shop . . . 90
8.2 Selecting preference pairs . . . 91
8.3 Scalability of dispatching rules . . . 93
8.4 Ranking strategies . . . 93
8.5 Trajectory strategies . . . 94
8.6 Stepwise sampling bias . . . 100
8.7 Conclusions . . . 103

9 Feature Selection 107
9.1 Validation accuracy . . . 108
9.2 Pareto front . . . 108
9.3 Inspecting weight contribution to end-result . . . 111
9.4 Evolution of validation accuracy . . . 112
9.5 Comparison to other approaches . . . 112
9.6 Conclusions . . . 117

10 Imitation Learning 118
10.1 Passive imitation learning . . . 119
10.1.1 Prediction with expert advice . . . 119
10.1.2 Follow the perturbed leader . . . 120
10.1.3 Experimental study . . . 120
10.2 Active imitation learning . . . 121
10.2.1 DAgger parameters . . . 124
10.2.2 Experimental study . . . 126
10.3 Summary of imitation learning experimental studies . . . 129
10.4 Conclusions . . . 131

11 Pilot Model 133
11.1 Single feature roll-outs . . . 134
11.2 Multi feature roll-outs . . . 134
11.3 Conclusions . . . 138

12 OR-Library Comparison 141
12.1 Experimental study . . . 141
12.2 Conclusions . . . 146

13 Conclusions 147
13.1 Executive summary . . . 148
13.2 Future work . . . 150

References 152

A Ordinal Regression 159
A.1 Preference set . . . 159
A.2 Ordinal Regression . . . 160
A.3 Logistic Regression . . . 161
A.4 Non-Linear Preference . . . 161

A.5 Parameter setting and tuning 162
A.6 Scaling 163
A.7 Implementation 163

II Papers **165**

I Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling **167**

II Sampling Strategies in Ordinal Regression for Surrogate Assisted Evolutionary Optimization **183**

III Determining the Characteristic of Difficult Job Shop Scheduling Instances for a Heuristic Solution Method **191**

IV Evolutionary Learning of Linear Composite Dispatching Rules for Scheduling **197**

V Generating Training Data for Learning Linear Composite Dispatching Rules for Scheduling **213**

VI Discovering Dispatching Rules From Data Using Imitation Learning **227**

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

Alice

Listing of figures

1.1 Flow-chart for Rice's framework for algorithm selection 6
1.2 Various methods for solving JSP 8
1.3 Class diagram for ALICE 12

2.1 The Mad Hatter's Tea Party 19
2.2 Graph representation for JSP 21
2.3 Gantt chart of a partial JSP schedule 29
2.4 Gantt charts of completed JSP using SDRs 30

3.1 Examples of job processing times of different FSP structures 34

4.1 Box-plots of deviation from optimality, ρ, when applying SDRs 40–41

5.1 Log fitness for CMA-ES optimisation 53
5.2 Evolution of weights for features 54
5.3 Box-plot for deviation from optimality, ρ, after CMA-ES optimisation 56

6.1 Partial Game Tree for job-shop 60
6.2 Size of feature set, $|\Phi|$ 65

7.1 Number of unique optimal dispatches 68
7.2 Probability of choosing optimal move 69
7.3 Mean deviation from optimality, ρ, for best and worst case scenario for making one suboptimal move 71
7.4 Probability of extremal feature being optimal move 73–74
7.5 Probability of SDR being optimal move 75–76
7.6 Mean deviation from optimality, ρ, for best and worst case scenario for not following a fixed policy 77–78
7.7 Box plot of $\mathcal{P}_{j.rnd}^{10\times10}$ deviation from optimality, ρ, for BDR where SPT is applied for the first followed by MWR. 80

7.8 Mean stepwise evolution of $\tilde{\boldsymbol{\varphi}}$ 83–84
7.9 Stepwise K-S test for features $\boldsymbol{\varphi}$ segregated w.r.t. easy and hard problems are drawn from the same continuous data distribution. 86
7.10 Stepwise significance of a correlation coefficient for $\mathcal{P}^{6\times5}_{j.rnd}$ features $\boldsymbol{\varphi}$, segregated w.r.t. easy and hard problems, with resulting deviation from optimality, ρ. 87

8.1 Size of preference set, $|\Psi|$ 95
8.2 Box-plot for various Φ and Ψ set-up 96
8.3 Size of preference set, $|\Psi_p|$ 101
8.4 Log probability for stepwise sampling for Ψ_p^{OPT} based on $\mathcal{P}^{10\times10}_{\mathrm{train}}$ 105
8.5 Box-plots for deviation from optimality, ρ, using various stepwise bias sampling strategies 105

9.1 Various methods of reporting validation accuracy for preference learning 109
9.2 Scatter plot for validation accuracy (%) against its corresponding mean expected ρ (%) for all 697 linear models from Eq. (9.1) 109
9.3 Normalised weights for CDR models from Table 9.1 113
9.4 Probability of choosing optimal move for models corresponding to highest mean validation accuracy and lowest mean deviation from optimality, ρ 114
9.5 Box-plot for deviation from optimality, ρ, for the best CDR preference models . . . 115

10.1 Box-plots for deviation from optimality, ρ, following either expert policy or perturbed leader for $\mathcal{P}^{6\times5}_{\mathrm{train}}$ and $\mathcal{P}^{10\times10}_{j.rnd}$ 123
10.2 Box-plot for deviation from optimality, ρ, where preference set is sampled to various sized $|\Psi_p^{\mathrm{DA7}}| = l_{\max}$ using $\mathcal{P}^{10\times10}_{j.rnd}$ 127
10.3 Box-plots for deviation from optimality, ρ, using active imitation learning for $\mathcal{P}^{6\times5}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rnd}$ using equal re-sampling 128
10.4 Box plot for $\mathcal{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using either expert policy, DAgger or following perturbed leader strategies. 129

11.1 Box-plot for single roll-out features 135
11.2 Box-plot for multi roll-out features 137
11.3 Box-plot for multi roll-out featuers based on ES.$C_{\max}$ 140

12.1 Box-plot for deviation from best known solution, ρ, trained on $\mathcal{P}^{10\times10}_{j.rnd}$ 142

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

The Queen

Listing of tables

1 Summary of experimental designs in Part II xvii

2.1 Example of 4×5 JSP 19
2.2 Feature space $\mathcal{F}$ for JSP 24

3.1 JSP and FSP problems spaces used in Part II 31
3.2 Problem space distributions used in experimental studies. 35
3.3 Benchmark problems from OR-Library used in experimental studies. 36

4.1 Threshold for ρ for easy and hard schedules 42
4.2 Percentage of 6×5 instances classified as easy or hard 43
4.3 Percentage of 10×10 instances classified as easy or hard 43
4.4 Percentage of 6×5 instances simultaneously classified as easy 45
4.5 Percentage of 6×5 instances simultaneously classified as hard 46
4.6 Percentage of 10×10 instances simultaneously classified as easy 47
4.7 Percentage of 10×10 instances simultaneously classified as hard 47

5.1 Final results for CMA-ES optimisation 52
5.2 Main statistics for Fig. 5.3a 58

6.1 Total number of features in Φ for all K steps 65

7.1 Main statistics for $\mathcal{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using BDR that changes from SDR at a fixed time step k. 81
7.2 Number of problem instances after segregating $\mathcal{P}^{6\times5}_{j.rnd}$ w.r.t. difficulty and trajectory. 85

8.1 Total number of preferences in Ψ_p for all K steps 98
8.2 Relative ordering w.r.t. mean ρ and size of its preference set, $l = |\Psi_p|$, for trajectories in Section 6.4 98
8.3 Main statistics for deviation from optimality, ρ, using $\mathcal{P}^{10\times10}_{\text{train}}$ based on various trajectories for Ψ_p 99

8.4 Main statistics for deviation from optimality, ρ, based on various stepwise sampling strategies 106

9.1 Mean validation accuracy and mean expected deviation from optimality, ρ, for all CDR models on the Pareto front from Fig. 9.2. 110
9.2 Main statistics for $\mathcal{P}^{10\times10}_{\text{train}}$ deviation from optimality, ρ, using models from Eq. (9.1) corresponding to lowest mean ρ or highest accuracy in Eq. (9.2) 116

10.1 Number of problem instances explored for the collection of training set. 122
10.2 Main statistics for $\mathcal{P}^{6\times5}_{\text{train}}$ and $\mathcal{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, following either expert policy or perturbed leader 122
10.3 Main statistics for $\mathcal{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using either expert policy, imitation learning or following perturbed leader strategies. 130

11.1 Main statistics for top three single extremal values for $\{\varphi_i\}_{i=17}^{24}$ rollout 135
11.2 Main statistics for $\{\varphi_i\}_{i=17}^{24}$ rollout preference models 139

12.1 Comparison results of OR-Library based on $\mathcal{P}^{10\times10}_{j.rnd}$ training data 143
12.2 Frequency of finding best makespan 145

Well, I never heard it before, but it sounds uncommon nonsense.

The Mock Turtle

Listing of algorithms

1 Pseudo code for constructing a JSP sequence using a deterministic scheduling policy . . . 18
2 Perturbed leader . . . 121
3 Imitation learning . . . 125
4 DAgger: Dataset Aggregation for JSP . . . 125
5 Overview of Analysis & Learning Iterative Consecutive Executions framework . 148

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

The Queen

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 Linear Composite Dispatching Rules for Scheduling

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

Paper VI Discovering Dispatching Rules From Data Using Imitation Learning

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, written as a monograph. 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	$\Phi^{\text{OPT}}, \Psi_{b,\text{equal}}$	K
II	$\mathbb{R}$-fun.	CMA-ES	surrogate sampling strategies	1
III	JSP	SDR	MWR	1
IV	JSP, FSP	CMA-ES	$\text{ES}.C_{\max}, \text{ES}.\rho$	1
V	JSP	PREF	$\{\Phi^{\pi} : \pi \in \{\text{OPT}, \text{MWR}, \text{RND}, \text{ES}.\rho, \text{ALL}\}\}$ $\{\Psi_{r,\text{equal}} : r \in \{a, b, f, p\}\}$	K
VI	JSP, FSP	PREF	$\{\Phi^{\pi} : \pi \in \{\text{OPT}, \text{OPT}\varepsilon, \text{DA}i\}\}$ $\{\Psi_{p,b} : b \in \{\text{equal}, \text{adjdbl2nd}\}\}$	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 VI. 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 8, which is a tailored algorithm, and *ii)* evolutionary search in Chapter 5, 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 6 and Paper V using passive imitation learning, whereas Chapter 10 and Paper VI included active imitation learning with greatly improved results. Moreover, the training data contains an abundance of information that can be used to determine 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 7 along with tying together the preliminary work in Paper III. Furthermore, Chapter 12 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. Finally, the dissertation concludes in Chapter 13 with discussion and addresses future work.

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

The Duchess

Nomenclature

Rice's Framework

$\mathcal{P}$	Problem space or instance space
$\mathcal{F}$	Feature space, i.e., measurable properties of the instances in $\mathcal{P}$
$\mathcal{A}$	Algorithm space
$\mathcal{Y}$	Performance space, i.e., the outcome for $\mathcal{P}$ using an algorithm from $\mathcal{A}$
Υ	Mapping for algorithm and feature space onto performance space

Job-shop Scheduling

n	number of jobs in shop
m	number of machines in shop
$\mathcal{J}$	set of jobs, $\{J_1, \ldots, J_j, \ldots, J_n\}$
$\mathcal{M}$	set of machines, $\{M_1, \ldots, M_a, \ldots, M_m\}$
p_{ja}	processing time for job J_j on machine M_a
$\boldsymbol{\sigma}_j$	machine ordering for job J_j
$x_s(j, a)$	starting time for job J_j on machine M_a
$x_f(j, a)$	finishing time for job J_j on machine M_a
$s(a, j)$	slot between current and previous task on machine M_a
$\mathcal{L}$	ready-list of jobs that have unassigned tasks, $\mathcal{L} \subset \mathcal{J}$
$C_{\max}$	makespan, i.e. maximum completion times for all tasks
$\boldsymbol{\chi}$	sequence of dispatches J_j to create (partial) schedule/solution
$\mathcal{U}(u_1, u_2)$	uniform distribution from the interval $I = [u_1, u_2] \subset \mathbb{R}$
ρ	percentage relative deviation from optimality
K	number of dispatches needed for a complete schedule, $K = n \cdot m$

Ordinal Regression

d	number of distinct features, i.e., dimension of $\mathcal{F}$
N	number of problem instances
Φ	training set
Ψ	preference set
l	size of preference set, $l = \lvert S \rvert$
$\boldsymbol{\varphi}(k)$	feature set, i.e., post-decision state, of a (partial) schedule at time k
$\tilde{\boldsymbol{\varphi}}$	scaled feature set, such that $\tilde{\boldsymbol{\varphi}}_i \in [-1, 1]$ for all $i \in \{1, .., d\}$
$\mathcal{O}^{(k)}$	set of optimal dispatches at time k
$\mathcal{S}^{(k)}$	set of suboptimal dispatches at time k
$\mathbf{w}$	linear weights for features $\boldsymbol{\varphi}$
h	linear classification model, $h(\mathbf{x}) = \langle \mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x}) \rangle$

Experimental Settings

Φ^{OPT}	training data is guided by (random) optimum trajectory
$\Phi^{\langle \text{SDR} \rangle}$	training data is guided by a single priority dispatching rule, where SDR $\in \{$SPT, LPT, LWR, MWR, I
$\Phi^{\langle \text{CMA-ES} \rangle}$	training data is guided by trajectory using by CMA-ES obtained weights, either $\Phi^{\text{ES}.C_{\max}}$ or $\Phi^{\text{ES}.\rho}$
Φ^{ALL}	union of all aforementioned trajectories, i.e., $\Phi^{\text{ALL}} = \Phi^{\text{OPT}} \cup \Phi^{\langle \text{SDR} \rangle} \cup \Phi^{\langle \text{CMA-ES} \rangle}$
Ψ_b	preference set added w.r.t. basic ranking
Ψ_f	preference set added w.r.t. full subsequent ranking
Ψ_p	preference set added w.r.t. partial subsequent ranking
Ψ_a	preference set containing all possible combination of rankings
p^{equal}	all preferences sampled equally
p^{opt}	preferences sampled proportional w.r.t. its stepwise optimality
p^{bcs}	preferences sampled reciprocally proportional w.r.t. its stepwise best case scenario of suboptimal dispatches

p^{wcs} preferences sampled reciprocally proportional w.r.t. its stepwise worst case scenario of suboptimal dispatches

Subscripts and Superscripts

j refers to job J_j

a refers to machine M_a

k refers to dispatch/time step k for a schedule, $k \in \{1, .., K\}$

o optimal job J_o

s suboptimal job J_s

Acronyms

ALICE Analysis & Learning Iterative Consecutive Executions

JSP Job-shop scheduling problem

FSP Flow-shop scheduling problem

DR dispatching rule

SDR single priority dispatching rule

CDR composite priority dispatching rule

BDR blended composite priority dispatching rule

SPT Shortest Processing Time rule

LPT Largest Processing Time rule

LWR Least Work Remaining rule

MWR Most Work Remaining rule

RND Random dispatches

We're all mad here.

The Cat

Acknowledgements

This thesis owes its existence to the help, support, and inspiration of many.

0.0. Fill in later...

Reykjavík, December 2015
Helga Ingimundardóttir

Part I

Prologue

This page is intentionally left blank.

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 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 are 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 7. 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 $\boldsymbol{\varphi}(\mathbf{x}) = \left[\varphi_1(\mathbf{x}), \ldots, \varphi_d(\mathbf{x})\right]^T \in \mathcal{F}$ 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 8 will come into play in the framework.

1.2 Previous work

The literature in scheduling mainly focuses on different objectives, e.g., Chang (1996) minimised the due-date tightness and Drobouchevitch and Strusevich (2000), Gao et al. (2007) looked

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

into solving for bottleneck machines, or even multi-objective JSP (Tay and Ho, 2008, Vázquez-Rodríguez and Petrovic, 2009, Xia and Wu, 2005). 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). Formal mathematical model for JSP is given in Chapter 2.

In order to find an optimal (or near optimal) solution for scheduling problems one could either use exact methods or heuristics methods. Exact methods guarantee an optimal solution, however, job-shop scheduling is strongly NP-hard* (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

*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.

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 summarises 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.

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.

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 priority *dispatching rule* (DR), 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* (GP), is promising. However, there is large number of rules to choose from, thus their combinations require expert knowledge or extensive trial-and-error process.

DRs are a very useful approach to dealing with scheduling environments because they are quick to implement (by computers and shop floor operators) and can cope with dynamic changes. Furthermore, DRs are relatively easy to interpret which can be of paramount importance for some end-users. For instance, Keane (2015) used GP to creates features for *Case Based Reasoning* (CBR), 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

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

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).

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 off-line. 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 CBR, 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, which is divided into two main phases: *i)* analysis, and *ii)* machine learning based on the analysis.

Analysis

By dwelling on optimal solutions and trying 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.

Learning

The machine learning approach considered in this dissertation is a supervised one. In particular, preference learning, which is a data driven approach which determines what feature states are preferable to others. Defining the training data as $\{\boldsymbol{\varphi}(\mathbf{x}_i(k)), y_i(k)\}_{k=1}^{K} \in \mathcal{D}$ then: *i)* samples $\mathbf{x}_i$ should represent the induced data distribution $\mathcal{D}$. This can be achieved by updating the learned model in an active imitation learning fashion, similar to the work of Ross and Bagnell (2010), Ross et al. (2011), in particular their DAgger framework; *ii)* y_i is labelled using a solver; *iii)* data needs to be balanced, as the set is unbalanced w.r.t. dispatching step k, and *iv)* to improve upon localised stepwise features $\boldsymbol{\varphi}$, it's possible to incorporate $(K-k)$ roll-outs where the learned model can be construed as a deterministic pilot heuristic.

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

ALICE

It's the believe of the author, that the methodology of going about this can be applied to any kind of optimisation problem which involves sequential decision making. As such, then it's suitable to name the framework: *Analysis & Learning Iterative Consecutive Executions*, or ALICE* for short. For demonstration purposes, this dissertation will solely be focusing on applying ALICE to dispatching rules for job-shop scheduling.

The ALICE framework mainly involves inspecting the stepwise optimality, ξ_π, for a heuristic policy π and it's relation to its end-result (here the makespan), ζ_π, as it defines its *footprint* in instance space (detailed in Chapters 4 and 7). This is done for a set of benchmark algorithms $\pi \in \mathcal{A}$, during the *analysis* phase, which are then used to guide the training for subsequent *learned* policy, $\hat{\pi}$. Finally, $\hat{\pi}$, can be post-processed in the same manner as done in the pre-processing phase, i.e., inspect $\xi_{\hat{\pi}}$ and $\zeta_{\hat{\pi}}$.

1.4 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 proposed 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.** Figure 1.3 displays the code's class diagram in relation to the thesis.

1.5 Outline

The dissertation is oriented around job-shop scheduling, which is explained in detail in Chapter 2. Due to scarcity of real-world data, we let random problem generators suffice. They are described in Chapter 3. 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,

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

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

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

Figure 1.3: Class diagram for ALICE, C# implementation available at github

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.

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 redefines the measure and compares a set of widely used single priority dispatching rules on different problem spaces. The analysis is done in more depth in Chapter 7 in the hopes of extrapolating where and when an algorithm excels in order to aid its failing aspects, which will be beneficial information for the creation of learning models in Chapter 8, as they are dependant on features based on those same dispatching rules under investigation.

An approach based on supervised learning, mostly on optimal schedules will be investigated and its effectiveness illustrated by improving upon well known dispatching rules for job-shop scheduling in Chapters 8 to 11. The method of generating training data and its stepwise sampling bias is critical for the success of the method, as shown in Sections 8.5 and 8.6. Moreover, models should be created in an iterative fashion such that the learned state spaces correspond to ones that the learned policy will eventually encounter, this is done in Chapter 10. Chapters 9 and 11, on the other hand explore how the baseline preference model of 16 features progresses if you drop or add additional features, respectively.

In addition to single priority dispatching rules, more sophisticated models obtained from direct optimisation, namely evolutionary search from Chapter 5, are used to compare the proposed

preference models. A comparison study using the OR-Library benchmark suite is done in Chapter 12.

Finally, the thesis concludes and proposes future work in Chapter 13.

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 one ore more objectives. Job-shop's analogy is from the 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 have 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: 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). Therefore, without loss of generality, this dissertation is structured around JSP.

*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.

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 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.

The 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,

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

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

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

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,

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

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 pre-emption is not allowed, the only applicable slots are whose idle time can process the entire operation, i.e.,

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

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 1 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 Algorithm 1, however, only 97% of instances when choosing the smallest slot.

Algorithm 1 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)
    χ ← ∅                                          ▷ initial current dispatching sequence
    for k ← 1 to K = n · m do                      ▷ at each dispatch iteration
        for all J_j ∈ L^(k) ⊂ J do                 ▷ inspect job-list
            χ^j ← {χ_i}_{i=1}^{k−1} ∪ J_j          ▷ partial temporal schedule
            φ^j ← φ ∘ Υ(χ^j)                       ▷ features for post-descision state
            I_j^π ← π(φ^j)                         ▷ priority for J_j
        end for
        j* ← argmax_{j∈L^(k)} {I_j^π}              ▷ choose highest priority
        χ_k ← J_{j*}                               ▷ dispatch j*
    end for
    return C_max^π ← Υ(χ)                          ▷ 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,

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

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 mean, e.g., asking a riddle or reciting a poem to the group.

*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

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., χ.

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 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, $\{\varphi_i\}_{i=1}^{8}$ and $\{\varphi_i\}_{i=9}^{16}$ 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, $\{\varphi_i\}_{i=1}^{16}$ 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, $\{\varphi_i\}_{i=17}^{20}$, 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 $\{\varphi_i\}_{i=21}^{24}$ 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. For the majority of the dissertation only $\{\varphi_i\}_{i=1}^{16}$ features will be considered, since the calculation of global features $\{\varphi_i\}_{i=17}^{24}$ is somewhat computationally intensive. They will be specifically addressed in Chapter 11.

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

*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	$C_{\max}^{\mathrm{SPT}(\mathbf{x}^k)}$	SPT
φ_{18}	final makespan using LPT	$C_{\max}^{\mathrm{LPT}(\mathbf{x}^k)}$	LPT
φ_{19}	final makespan using LWR	$C_{\max}^{\mathrm{LWR}(\mathbf{x}^k)}$	LWR
φ_{20}	final makespan using MWR	$C_{\max}^{\mathrm{MWR}(\mathbf{x}^k)}$	MWR
φ_{RND}	final makespans using 100 random rollouts	$\{C_{\max}^{\mathrm{RND}(\mathbf{x}^k)}\}_{i=1}^{100}$	
φ_{21}	mean for φ_{RND}	$\mathbb{E}\left\{\varphi_{\mathrm{RND}}\right\}$	RNDmean
φ_{22}	standard deviation for φ_{RND}	$\sqrt{\mathbb{E}\left\{\varphi_{\mathrm{RND}}^2\right\} - \mathbb{E}\left\{\varphi_{\mathrm{RND}}\right\}^2}$	RNDstd
φ_{23}	minimum value for φ_{RND}	$\min\{\varphi_{\mathrm{RND}}\}$	RNDmin
φ_{24}	maximum value for φ_{RND}	$\max\{\varphi_{\mathrm{RND}}\}$	RNDmax

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 $\boldsymbol{\chi}^j$.

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 **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 8 and Chapter 5, 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,

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

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, $\{\varphi_i\}_{i=1}^{16}$, 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,

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

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

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

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

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

Equation (2.17) measures the discrepancy between predicted value and true outcome, and is commonly referred to as a loss function, which we would like to minimise for π.

The mapping $\Upsilon : \mathcal{A} \times \mathcal{F} \mapsto \mathcal{Y}$ is the step-by-step construction heuristic in Algorithm 1.

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.

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.

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 VI are the same as described here.

Although real-world instances are desirable, unfortunately they are scarce. Hence in some experiments (mainly in Chapter 12), problem instances from OR-Library maintained by Beasley (1990) will be used as benchmark problems. They are 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*

*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 $Jm|2sets|C_{\max}$ 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.

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

*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

An example of distribution of processing times are depicted in Fig. 3.1, where machine indices are on the horizontal axis, job indices are shape-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 et al. (1992) introduce a set of JSP problems

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

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.

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 $m = 10$

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.

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 8 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 experimental study in the subsequent sections focuses rather on few simple algorithms, namely the SDRs described in Section 2.4, i.e., we will limit the algorithm space to,

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

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 Fig. 4.1. 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 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 $\mathcal{P}^{n\times m}_{f.jc}$, $\mathcal{P}^{n\times m}_{f.mc}$ and $\mathcal{P}^{n\times m}_{f.mxc}$ 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.,

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

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

$$\mathcal{H}(a) := \left\{\mathbf{x} \ : \ \rho = \Upsilon(a, \mathbf{x}) > \rho^{3\text{rd. Qu.}}\right\} \tag{4.2b}$$

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.,

$$\frac{|\mathcal{E}(a)|}{N_{\text{train}}} \cdot 100\% \quad \text{and} \quad \frac{|\mathcal{H}(a)|}{N_{\text{train}}} \cdot 100\% \tag{4.3}$$

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. (4.3). 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. (4.3). 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.,

$$\frac{|\mathcal{E}(a_i) \cap \mathcal{E}(a_j)|}{N_{\text{train}}} \cdot 100\% \quad \text{or} \quad \frac{|\mathcal{H}(a_i) \cap \mathcal{H}(a_j)|}{N_{\text{train}}} \cdot 100\% \tag{4.4}$$

where $a_i, a_j \in \mathcal{A}$. Note, when $a_i = a_j$ then Eq. (4.4) is equivalent to Eq. (4.3).

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 7.6.

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. (4.4). 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. (4.4). 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. (4.4). 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. (4.4). 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 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 7.6 will attempt to explain the evidence show in 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,

$$\sum_{a\in\mathcal{A}}|\mathcal{E}(a)| \approx N_{\text{train}} \quad \text{and} \quad \sum_{a\in\mathcal{A}}|\mathcal{H}(a)| \approx N_{\text{train}} \tag{4.5}$$

due to the fact Eq. (4.2) 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. (4.5) varies, resulting in an unbalanced sample size when restricted to a single SDR, which is done in Section 7.6. It's for that reason we adjust $A := \langle\text{SDR}\rangle$ for a single SDR inspection, i.e., then the 'easy' and 'hard' problems are each approximately $\frac{1}{4}N_{\text{train}}$, and instances don't necessarily coincide across different SDRs.

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.

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

5
Evolutionary Search

Genetic algorithms (GA) are one of the most widely used approaches in JSP literature (Pinedo, 2008). GA is search heuristic that is inspired by the process of natural selection, and is a subclass of *evolutionary algorithms* (EA), which generate solutions to optimisation problems using techniques based on natural evolution, such as inheritance, mutation, selection, and crossover.

When applying GAs to JSP, 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.

As previously discussed in Chapter 1, there are two main viewpoints on how to approach scheduling problems: *i)* tailored algorithms where schedules are built for one problem instance at a time, and *ii)* general algorithms where schedules are built for all problem instances at once. For tailored algorithms 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 chapter will explore a general algorithms construction viewpoint where there is no feature set collected beforehand since the learning model is optimised directly via evolutionary search. The framework was first reported in Paper IV, and later used to improve the preference models in Paper V.

Evolutionary search 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 a better expected mean. For that reason, it can be mediated with the use of preference learning, as discussed in Paper II, albeit for traditional test functions suite (in particular Rosenbrock's function and Sphere model).

5.1 Experimental setting

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 Paper IV, was to automate that selection similar to DRGA, by translating dispatching rules into measurable features and optimising what their contribution should be via evolutionary search, i.e., optimise the weights $\mathbf{w}$ in Eq. (2.12) directly using *covariance matrix adaptation evolution strategy* (CMA-ES) by Hansen and Ostermeier (2001), which has been proven to be a very efficient numerical optimisation technique. The framework is straight forward and easy to implement and shows promising results.

Moreover, Section 5.2 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.

Preliminary experiments were first reported in Papers IV and V using: *i)* standard set-up of parameters of the CMA-ES optimisation, and *ii)* runtime was limited to 288 hours on a cluster for each $\mathcal{P}^{6\times5}_{\text{train}}$ problem set given in Table 3.2, where in every case the optimisation reached its

maximum walltime. Paper IV had one model for all K time steps, whereas Paper V used a different stepwise model at each dispatch iteration.

Various data distributions from Chapter 3 are investigated. Due to computational cost, only 6×5 instances were initially considered. However, since then the scheduling subroutines have been made more time-efficient, making $\mathcal{P}^{10\times10}_{\text{train}}$ applicable in a reasonable amount of time. CMA-ES models will be mostly trained on the lower dimension, 6×5, and only the general random 10×10 JSP case is explored. Finally, to check robustness, models are validated on benchmark tests sets from the OR-Library (cf. Section 3.3).

5.2 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,

$$\text{ES}.C_{\max} := \min \mathbb{E}\left\{C_{\max}\right\} \tag{5.1a}$$

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

$$\text{ES}.\rho := \min \mathbb{E}\left\{\rho\right\} \tag{5.1b}$$

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

5.3 Experimental study

Main statistics of the experimental run are given in Table 5.1 and depicted in Fig. 5.1 for both approaches. In addition, evolving decision variables, here weights $\mathbf{w}$ for Eq. (2.12), are depicted in Fig. 5.2.

The evolution of fitness per generation from the CMA-ES optimisation of Eq. (5.1) is depicted in Fig. 5.1. Note, most problem spaces reached their allotted maximum function evaluations* without converging. In fact several problem spaces, e.g., $\mathcal{P}^{6\times5}_{f.rnd}$, needed restarting during the optimisation process. Notice $\mathcal{P}^{6\times5}_{j.rnd,J_1}$ especially, then ES.$C_{\max}$ needs more than twice the amount of function evaluations than using ES.ρ as an objective.

*Computational budget was set to 50,000 function evaluations

Table 5.1: Final results for CMA-ES optimisation; total number of generations and function evaluations and its resulting fitness value for both objective functions.

			optimise Eq. (5.1a)			optimise Eq. (5.1b)		
$\mathcal{P}_{\text{train}}$	|Model|*		#gen	#eval	ES.$C_{\max}$	#gen	#eval	ES.ρ
j.rnd	6x5	1	1713	20544	476.34	4168	50004	6.23
j.rnd	6×5	K	2274	50006	467.62	2274	50006	4.38
j.rndn	6×5	1	4168	50004	442.99	4168	50004	8.28
j.rndn	6×5	K	2274	50006	435.87	2274	50006	6.60
$j.rnd, J_1$	6×5	1	4168	50004	666.03	1867	22392	3.26
$j.rnd, J_1$	6×5	K	2274	50006	658.57	2274	50006	2.13
$j.rnd, M_1$	6×5	1	2086	25020	603.46	3037	36432	5.60
$j.rnd, M_1$	6×5	K	2274	50006	592.85	2274	50006	3.66
f.rnd	6×5	1	3683	44184	570.15	4168	50004	7.34
f.rnd	6×5	K	2274	50006	558.37	2274	50006	5.07
f.rndn	6×5	1	1829	21936	508.63	4168	50004	0.92
f.rndn	6×5	K	2274	50006	508.72	2274	50006	0.94
f.jc	6×5	1	4168	50004	567.80	4168	50004	0.34
f.jc	6×5	K	2274	50006	567.74	2274	50006	0.36
f.mc	6×5	1	1796	21540	579.38	1731	20760	0.44
f.mc	6×5	K	2274	50006	578.85	2274	50006	0.34
f.mxc	6×5	1	4168	50004	578.35	4168	50004	1.08
f.mxc	6×5	K	2274	50006	578.09	2274	50006	0.37
j.rnd	10×10	1	966	11592	898.22	2997	35952	10.49

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

Furthermore, the evolution of the decision variables $\mathbf{w}$ are depicted in Fig. 5.2. As one can see, the relative contribution for each weight clearly differs between problem spaces. Note, that in the case of $\mathcal{P}_{j.rnd}^{6\times5}$ (cf. Figs. 5.1 and 5.2a), CMA-ES restarts around generation 1,600 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. $\varphi_3 = \varphi_1 + \varphi_2$. Moreover, from Fig. 5.2b we see that the evolution of weights w.r.t. each step k is quite erratic and Eq. (5.1a) is somewhat dissimilar to its Eq. (5.1b) counterpart, yet their resulting ρ values are not significantly different. Most likely, this can be explained by feature equivalence.

In order to compare the two objective functions in Eq. (5.1), the best weights reported were used for Eq. (2.12) on the corresponding training and test set. Its box-plot of deviation from optimality, ρ, defined by Eq. (2.17), is depicted in Fig. 5.3a and Table 5.2 presents its main

Figure 5.1: Log fitness for optimising w.r.t. Eq. (5.1), per generation of the CMA-ES optimisation.

statistics: mean, median, standard deviation, minimum and maximum values.

In most cases (except for $\mathcal{P}^{6\times5}_{f.rndn}$, $\mathcal{P}^{6\times5}_{f.jc}$) there was a significant difference w.r.t. lower mean ρ, when using a separate model for each time step, as is to be expected as the optimal dispatching rules used in the beginning of the scheduling process may not necessarily be the same as in the middle, or end of the schedule. Alas, stepwise models aren't appropriate when inspecting robustness towards different problem spaces.* Hence, a single model for all iterations is preferred.

Regarding the choice of objective function in Eq. (5.1), then there is no statistical difference between adopting either objective function with respect to training and test set, save for time independent $\mathcal{P}^{6\times5}_{f.mxc}$. Now, when applying the time independent models to the OR-Library benchmark data sets,** depicted in Fig. 5.3b, then we see a clear performance boost when using Eq. (5.1b) in: *i)* $\mathcal{P}^{6\times5}_{f.jc}$, $\mathcal{P}^{6\times5}_{f.mc}$ and $\mathcal{P}^{6\times5}_{f.mxc}$ for JSP, and *ii)* $\mathcal{P}^{6\times5}_{f.jc}$ and $\mathcal{P}^{6\times5}_{f.mxc}$ for FSP. Therefore, minimisation of expectation of ρ, is preferred over simply using the unscaled resulting makespan. Also it's noted that in Paper IV, then $\mathcal{P}^{6\times5}_{f.jc}$ optimised w.r.t. Eq. (5.1a) gave a considerably worse

*Note, time dependant models are inapplicable for OR-Library, since their size $n \times m$ does not match many of the sizes in the benchmark set. However, this is irrelevant for time independent models.

**Best configuration is reported in Table 12.1.

(a) evolution of weights for $\mathcal{P}^{n\times m}_{j.rnd}$ time independent models

(b) stepwise evolultion of final weights for $\mathcal{P}^{6\times 5}_{j.rnd}$ time dependent models

Figure 5.2: Evolution of weights for features (given in Table 2.2). Note, weights are normalised such that $\|\mathbf{w}\| = 1$.

results, since the optimisation got trapped in a local minima.

Furthermore, notice how $\mathcal{P}^{6\times5}_{f.mxc}$ obtains a significantly better mean ρ (from 52.8% down to 24.46%) for the JSP (cf. Fig. 5.3b) then it did for it's corresponding problem space, which was the only setting where Eq. (5.1b) was significantly different than Eq. (5.1a) (worsening by $\Delta\rho \approx +1\%$).

5.4 Conclusions

Data distributions considered in this study incorporated more problem spaces than in its initial reports in Paper IV. Furthermore, both time dependent and independent models were optimised with CMA-ES. The former generally obtained lower mean ρ. However, the latter was often equally good, with the added benefit of being applicable for higher (or lower) dimensionality, which was then tested using the benchmark set from the OR-Library.

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. Eq. (5.1)), save for time independent $\mathcal{P}^{6\times5}_{f.mxc}$, when applying the models to their corresponding training and test set. However, preliminary experiments in Paper IV and application on unseen data sets from the OR-Library, showed great improvement when using Eq. (5.1b) over Eq. (5.1a). Implying, even though CMA-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 (cf. Chapter 4 and Paper III). Thus normalising the objective function would help the evolutionary search to deviate from giving too much weight for problematic problem instances.

The main drawback of using evolutionary search for finding optimal weights for Eq. (2.12) is how computationally expensive it is to evaluate the mean expected fitness. Even for a low problem dimension such as 6-job 5-machine JSP, each optimisation run reached their maximum allotted function evaluations without converging. Now, 6×5 JSP requires 30 sequential operations where at each time step there are up to 6 jobs to choose from, in fact its complexity is $\mathcal{O}(n!^m)$ (Giffler and Thompson, 1960) making it computationally infeasible to apply this framework for higher dimensions as is. Especially, considering that it's preferred to run these experiments several times – e.g. in Paper IV $\mathcal{P}^{6\times5}_{f.jc}$ got stuck in local minima for ES.$C_{\max}$, which could have been avoided by restarting the optimisation. 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 5.3 for problems of higher dimensions, e.g., with these adjustments it is possible to train on 10×10

(a) Models applied to corresponding training and test set

(b) Models applied to ORLIB test set

Figure 5.3: Box-plot for deviation from optimality, ρ, when implementing the final weights obtained from CMA-ES optimisation, using objective functions from Eq. (5.1).

with greater ease, or even considering even higher dimensions.

Table 5.2: Main statistics for Fig. 5.3a

$\mathcal{P}_{\text{train}}$	|Model|		Eq. (5.1)	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
j.rnd	6×5	1	ES.$C_{\max}$	0.00	1.96	5.62	6.65	10.38	30.77
j.rnd	6×5	K	ES.$C_{\max}$	0.00	1.47	4.71	5.92	8.48	30.77
j.rnd	6×5	1	ES.ρ	0.00	2.04	5.59	6.55	9.97	33.10
j.rnd	6×5	K	ES.ρ	0.00	1.39	4.64	5.74	8.72	34.75
j.rndn	6×5	1	ES.$C_{\max}$	0.00	2.25	10.36	8.73	12.08	29.38
j.rndn	6×5	K	ES.$C_{\max}$	0.00	1.93	9.54	7.89	12.02	25.67
j.rndn	6×5	1	ES.ρ	0.00	2.25	10.37	8.62	12.22	25.00
j.rndn	6×5	K	ES.ρ	0.00	1.95	9.71	7.88	12.04	30.85
$j.rnd, J_1$	6×5	1	ES.$C_{\max}$	0.00	0.00	1.03	3.64	6.27	29.74
$j.rnd, J_1$	6×5	K	ES.$C_{\max}$	0.00	0.00	0.51	2.96	4.74	27.39
$j.rnd, J_1$	6×5	1	ES.ρ	0.00	0.00	0.94	3.56	6.12	30.43
$j.rnd, J_1$	6×5	K	ES.ρ	0.00	0.00	0.57	2.99	4.53	30.86
$j.rnd, M_1$	6×5	1	ES.$C_{\max}$	0.00	0.71	4.35	5.78	9.12	26.25
$j.rnd, M_1$	6×5	K	ES.$C_{\max}$	0.00	0.38	3.67	4.84	7.51	29.56
$j.rnd, M_1$	6×5	1	ES.ρ	0.00	0.63	4.42	5.76	9.16	26.25
$j.rnd, M_1$	6×5	K	ES.ρ	0.00	0.35	3.77	4.86	7.77	28.87
f.rnd	6×5	1	ES.$C_{\max}$	0.00	3.24	6.76	7.74	11.23	34.24
f.rnd	6×5	K	ES.$C_{\max}$	0.00	2.31	5.13	6.10	8.62	39.95
f.rnd	6×5	1	ES.ρ	0.00	3.30	6.79	7.90	11.45	32.88
f.rnd	6×5	K	ES.ρ	0.00	2.32	5.15	5.97	8.76	27.68
f.rndn	6×5	1	ES.$C_{\max}$	0.00	0.39	0.80	1.00	1.41	4.66
f.rndn	6×5	K	ES.$C_{\max}$	0.00	0.39	0.81	1.00	1.40	4.86
f.rndn	6×5	1	ES.ρ	0.00	0.39	0.80	1.00	1.41	4.66
f.rndn	6×5	K	ES.ρ	0.00	0.39	0.80	0.99	1.40	4.86
f.jc	6×5	1	ES.$C_{\max}$	0.00	0.00	0.26	0.39	0.55	9.41
f.jc	6×5	K	ES.$C_{\max}$	0.00	0.00	0.26	0.37	0.54	4.25
f.jc	6×5	1	ES.ρ	0.00	0.00	0.25	0.36	0.53	2.95
f.jc	6×5	K	ES.ρ	0.00	0.00	0.26	0.38	0.57	4.25
f.mc	6×5	1	ES.$C_{\max}$	0.00	0.00	0.24	0.46	0.69	4.93
f.mc	6×5	K	ES.$C_{\max}$	0.00	0.00	0.16	0.38	0.57	3.69
f.mc	6×5	1	ES.ρ	0.00	0.00	0.25	0.47	0.69	4.93
f.mc	6×5	K	ES.ρ	0.00	0.00	0.17	0.38	0.56	6.29
f.mxc	6×5	1	ES.$C_{\max}$	0.00	0.00	0.20	0.47	0.74	5.84
f.mxc	6×5	K	ES.$C_{\max}$	0.00	0.00	0.17	0.43	0.65	5.84
f.mxc	6×5	1	ES.ρ	0.00	0.19	0.68	1.14	1.68	12.10
f.mxc	6×5	K	ES.ρ	0.00	0.00	0.17	0.40	0.63	5.84
j.rnd	10×10	1	ES.$C_{\max}$	0.13	7.97	10.57	10.86	13.47	25.35
j.rnd	10×10	1	ES.ρ	0.88	7.99	10.64	10.87	13.40	27.81

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

6 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 Algorithm 1, 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,

$$\Phi := \bigcup_{i=1}^{N_{\text{train}}} \bigcup_{k=1}^{K} \bigcup_{J_j \in \mathcal{L}^{(k)}} \left\{ \boldsymbol{\varphi}^j \ : \ \mathbf{x}_i \in \mathcal{P}_{\text{train}}^{n \times m} \right\} . \tag{6.1}$$

6.1 Job-shop tree representation

Continuing with the example from Section 2.3, Fig. 6.1 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 given this scenario? Assuming J_4 would be dispatched first, the bottom layer depicts all the next possible partial schedules.

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

Figure 6.1: 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.

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.

6.2 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.

6.3 Computational growth

The creation of the game tree for JSP can be done recursively for all possible permutations of dispatches, 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 no sub-trees merge.
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

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

**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.

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.

6.4 Trajectory sampling strategies

For each feature in Eq. (6.1) we need to keep track of the resulting makespan for its dispatched job. As a result, we obtain the meta-data from Fig. 1.1 as follows,

$$\{\Phi^{\pi}, \mathcal{Y}^{\pi}\} := \left\{ \{\boldsymbol{\varphi}^j, C_{\max}^{\pi_\star}(\boldsymbol{\chi}^j)\} \ : \ J_j \in \mathcal{L}^{(k)} \right\}_{k=1}^{K} \in \mathcal{F} \times \mathcal{Y} \tag{6.2}$$

for a single problem instance $\mathbf{x} \in \mathcal{P}_{\text{train}}$, and where $C_{\max}^{\pi_\star}(\boldsymbol{\chi}^j)$ denotes the optimal makespan (i.e. following the expert policy $\pi_\star$) from the resulting post-decision state $\boldsymbol{\chi}^j$.

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 5) will also be used.

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

To clarify, the trajectory sampling strategies for collecting a feature set and its corresponding labelling for Eq. (6.2) are the following:

Optimum trajectory, $\Phi^{\mathbf{OPT}}$ or $\Phi^{\pi_\star}$, 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 trajectories, $\Phi^{\mathbf{ES}.\rho}$ and $\Phi^{\mathbf{ES}.C_{\max}}$, 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 optimisation from Chapter 5.

All trajectories, $\Phi^{\mathbf{ALL}}$, denotes all aforementioned trajectories were explored, i.e.,
$$\Phi^{\text{ALL}} := \left\{\Phi^{A} \;:\; \forall A \in \{\pi_\star,\ \text{SPT, LPT, LWR, MWR, RND, ES}.\rho, \text{ES}.C_{\max}\}\right\}$$

When following optimal trajectory, then due to the nature of the sequence representation (i.e. χ), the earlier stages for $\mathcal{P}_{j.rnd}$ of the dispatching are more or less equivalent and thus irrelevant (cf. Fig. 7.3). 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 $\Phi^{\langle\text{SDR}\rangle}$ and $\Phi^{\langle\text{CMA-ES}\rangle}$ 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 for example LWR is considered more favourable for flowshop rather than MWR (cf. Chapter 4).

The number of features that were collected on a step-by-step basis for $\mathcal{P}_{\text{train}}^{6\times 5}$ in Table 3.2 is illustrated in Fig. 6.2. 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 say minimising $\mathcal{L}$ and from Eq. (6.1) we have $|\Phi(k)| \propto |\mathcal{L}^{(k)}|$. Whereas MWR tries to keep the jobs more balanced, hence more steady $|\mathcal{L}|$, until at $k > (K - n)$ then $|\mathcal{L}| \lneq (K - k)$, which explains the sharp decent near the end for MWR. Table 6.1 gives the total size for $|\Phi|$, indicating the number of optimisations needed for obtaining $\mathcal{Y}$.

Figure 6.2: Size of 6×5 feature set, $|\Phi|$, over different trajectory strategies

Table 6.1: Total number of features in Φ for all K steps. Note '–' denotes not available.

Track	$\mathcal{P}^{6\times5}_{\text{train}}, N_{\text{train}} = 500$									$\mathcal{P}^{10\times10}_{\text{train}}, N_{\text{train}} = 300$		
	j.rnd	*j.rndn*	*j.rnd*, J_1	*j.rnd*, M_1	*f.rnd*	*f.rndn*	*f.jc*	*f.mc*	*f.mxc*	*j.rnd*	*j.rndn*	*f.rnd*
SPT	63197	63074	64560	61320	63287	63123	53678	66995	66216	211351	–	–
LPT	63516	63374	63595	62864	63535	63320	53746	66356	65662	210490	–	–
LWR	52500	52500	52500	52500	52500	52500	52500	52500	5250	165000	–	–
MWR	79230	82500	78327	77934	79288	82500	80546	82498	8245	280739	–	–
RND	71390	71608	71445	71463	71427	71945	71558	71456	7149	252515	–	–
OPT	76592	78176	74109	74069	70037	69180	69716	71602	7102	272858	277717	211763
ES.ρ	78443	81248	78673	78866	68986	55943	60755	53707	74997	277851	–	–
ES.$C_{\max}$	79343	81226	77903	79078	68602	56789	54781	52502	52510	276634	–	–
ALL	564211	573706	561112	558094	537662	515300	497280	517616	537121	1947438	277717	211763

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

Alice

7

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 more meaningful overall outcome.

Essentially this also 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 for the quality of a solution. If results are contradictory, then 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 to 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 7.1 and 7.2 depict the mean over all the training data, which are quite noisy

functions. Thus, for clarity purposes, they are fitted with local polynomial regression, making the boundary points sometimes biased. Paper VI depicts the raw mean as is, albeit only for 10×10 problem spaces, which is also done here for Figs. 7.3 to 7.5 and 7.8.

7.1 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 6.4), and inspect the evolution of its features (defined in Table 2.2) throughout the dispatching process, which is detailed in Chapter 6. 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 7.1 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. 7.2 depicts the probability of choosing an optimal dispatch at each iteration.

To generalise, we could consider the probability of optimality as a sort of stepwise 'training accuracy.' Then for a given policy π, we'd formalise its optimality (yet still maintaining optimal trajectory) as,

$$\xi_{\pi}^{\star} := \mathbb{E}_{\pi_\star}\Big\{\pi_\star = \pi\Big\} \tag{7.1}$$

that is to say the mean likelihood of our policy π being equivalent to the expert policy $\pi_\star$, i.e., $Y^{\pi_\star} = Y^{\pi}$. Note, for $\xi_{\pi}^{\star}$ we only need $\{\Phi^{\pi_\star}, \mathcal{Y}^{\pi_\star}\}$ from Eq. (6.2): *i)* retrace $\pi_\star$ as done in Algorithm 1, and *ii)* inspect if the job J_{j^*} chosen by π yields the same $C_{\max}^{\pi_\star(\boldsymbol{x}^{j^*})}$ as the true optimum, $C_{\max}^{\pi_\star}$.

7.2 Making suboptimal decisions

Looking at Fig. 7.2, $\mathcal{P}_{j.rnd}^{10\times 10}$ 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. 7.3 depicts the best and worst case scenario of deviation from optimality, ρ, once you've fallen off the optimal

(a) 6×5

(b) 10×10

Figure 7.1: Number of unique optimal dispatches (lower bound).

(a) 6×5

(b) 10×10

Figure 7.2: Probability of choosing optimal move (at random)

track, defined as follows,

$$\zeta^{\star}_{\min}(k) \quad := \quad \mathbb{E}_{\pi_\star}\left\{ \min(\rho) \ : \ \forall C^{\chi^j}_{\max} \gneq C^{\pi_\star}_{\max} \wedge J_j \in \mathcal{L}^{(k)} \right\} \tag{7.2a}$$

$$\zeta^{\star}_{\max}(k) \quad := \quad \mathbb{E}_{\pi_\star}\left\{ \max(\rho) \ : \ \forall C^{\chi^j}_{\max} \gneq C^{\pi_\star}_{\max} \wedge J_j \in \mathcal{L}^{(k)} \right\} \tag{7.2b}$$

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. 7.5 shows the probability of optimality when following a fixed SDR (i.e. if Eq. (7.1) is conditioned on π itself instead of $\pi_\star$).

It is interesting that for JSP, then making suboptimal decisions makes more of an impact on the resulting makespan as the dispatching process progresses. 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 the damage has 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.

7.3 Optimality of extremal features

The training accuracy from Eq. (7.1) of the aforementioned features from Table 2.2, or 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, i.e., $\xi^{\star}_{\pm\varphi_i}$, is depicted in Fig. 7.4, 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, $\xi^{\star}_{\text{RND}}$ (cf. Fig. 7.2). 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., $\{\varphi_i\}_{i=17}^{24}$, 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.

*Additional problem spaces for $\xi^{\star}_{\pm\varphi_i}$ can be found in Shiny application: Features > Extremal.

**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 7.3: Mean deviation from optimality, ρ, for best and worst case scenario of making *one* suboptimal dispatch (i.e. $\zeta^{\star}_{\min}$ and $\zeta^{\star}_{\max}$), depicted as lower and upper bound, respectively. Moreover, mean suboptimal move is given as dashed line.

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. 7.4a has $\rho = 1.3\%$ and increases to 8.8% in Fig. 7.4b), implying that 100 random roll-outs for are not sufficient for fully exploring 10×10 state-space, yet highly competitive for 6×5.

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 7.5 depicts the corresponding probabilities from Fig. 7.4 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 Fig. 4.1), then there is a some correspondence between high probability of stepwise optimality and low ρ. Alas, this isn't always the case, for $\mathcal{P}^{10\times 10}_{j.rnd}$ $\xi^{\star}_{\text{SPT}}$ always outperforms $\xi^{\star}_{\text{LPT}}$ in choosing a dispatch which may result in an optimal schedule. However, this does not transcend to SPT having a lower ρ value than LPT. Hence, it's not enough to just learn optimal behaviour, one needs to investigate what happens once we encounter suboptimal state spaces.

Since we know that our SDR heuristics aren't perfect, and they're bound to make mistakes at some point. It's interesting to see how that stepwise optimality evolves for its intended trajectory, thereby updating Eq. (7.1) to

$$\xi_{\pi} := \mathbb{E}_{\pi}\Big\{\pi_{\star} = \pi\Big\} \tag{7.3}$$

Figure 7.5 shows the difference between $\xi^{\star}_{\langle\text{SDR}\rangle}$ and $\xi_{\langle\text{SDR}\rangle}$. Similarly for Eq. (7.2),

$$\zeta^{\pi}_{\min}(k) := \mathbb{E}_{\pi}\Big\{\min_{J_j\in\mathcal{L}^{(k)}}(\rho) \;:\; \forall C^{\pi_\star}_{\max}(\boldsymbol{x}^{j}) \neq C^{\pi_\star}_{\max}(\boldsymbol{x}^{j^*}) \wedge j^* = \operatorname*{argmax}_{J_j\in\mathcal{L}^{(k)}}\{\pi(\boldsymbol{\varphi}^{j})\}\Big\} \tag{7.4a}$$

$$\zeta^{\pi}_{\max}(k) := \mathbb{E}_{\pi}\Big\{\max_{J_j\in\mathcal{L}^{(k)}}(\rho) \;:\; \forall C^{\pi_\star}_{\max}(\boldsymbol{x}^{j}) \neq C^{\pi_\star}_{\max}(\boldsymbol{x}^{j^*}) \wedge j^* = \operatorname*{argmax}_{J_j\in\mathcal{L}^{(k)}}\{\pi(\boldsymbol{\varphi}^{j})\}\Big\} \tag{7.4b}$$

$$\zeta^{\pi}_{\mu}(k) := \mathbb{E}_{\pi}\Big\{\rho \;:\; C^{\pi_\star}_{\max}(\boldsymbol{x}^{j^*}) \wedge j^* = \operatorname*{argmax}_{J_j\in\mathcal{L}^{(k)}}\{\pi(\boldsymbol{\varphi}^{j})\}\Big\} \tag{7.4c}$$

with the additional metric ζ^{π}_{μ}, which gives the mean evolution for deviation from optimality, ρ, when following a fixed policy π. Note, $\zeta^{\pi_\star}_{\min} = \zeta^{\star}_{\min}$, $\zeta^{\pi_\star}_{\max} = \zeta^{\star}_{\max}$ and $\zeta^{\pi_\star}_{\mu} \mapsto 0$. Figure 7.6 depicts Eq. (7.4) for expert policy $\pi_\star$ and SDRs.

(a) $\mathcal{P}^{6\times 5}_{j.rnd}$

Figure 7.4: Probability of extremal feature being optimal move

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

Figure 7.4 (cont.)

(a) 6×5

Figure 7.5: Probability of SDR yielding optimal move. Both optimal (solid: $\xi^\star_{\langle \mathrm{SDR} \rangle}$) and SDR-based (dashed: $\xi_{\langle \mathrm{SDR} \rangle}$) trajectories are inspected.

(b) 10 × 10

Figure 7.5 (cont.): Note, due to computational complexity, only $\mathcal{P}_{j.rnd}^{10\times10}$ has SDR-based trajectories also inspected. Otherwise, only optimal is pursued.

(a) 10 × 10

Figure 7.6: Mean deviation from optimality, ρ, for best and worst case scenario when not following a fixed policy π (i.e. $\zeta^{\pi}_{\min}$ and $\zeta^{\pi}_{\max}$), depicted as lower and upper bound, respectively. Moreover, mean evolution of ρ for π (i.e. ζ^{π}_{μ}) is given as a dashed line.

(b) 6×5

Figure 7.6 (cont.): Note, $\{\zeta^{\pi^\star}_{\min}, \zeta^{\pi^\star}_{\max}\}$ are illustrated jointly for $\mathcal{P}_{\text{train}}$ in Fig. 7.3

A means of interpreting Eq. (7.4), is that given a fixed policy π, then $\zeta^{\pi}_{\min}$ describes the potential improvement (iff $\zeta^{\pi}_{\min} < \zeta^{\pi}_{\mu}$) for changing the policy. Whereas, $\zeta^{\pi}_{\max}$ indicates the disadvantages of changing course. When $\zeta^{\pi}_{\min} > \zeta^{\pi}_{\mu}$, then clearly π is not a good policy for said problem space, e.g., for the final dispatches of ζ^{SPT}_{μ} for $\mathcal{P}^{6\times 5}_{j.rnd,J_1}$ or $\mathcal{P}^{10\times 10}_{j.rnd}$.

Remark: Eqs. (7.3) and (7.4) are based on corresponding meta-data, $\{\Phi^{\pi}, \mathcal{Y}^{\pi}\}$, from Eq. (6.2), whereas Eqs. (7.1) and (7.2) reuse the same expert meta-data, $\{\Phi^{\pi_\star}, \mathcal{Y}^{\pi_\star}\}$.

7.4 Simple blended dispatching rule

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 switching between two SDRs at a predetermined time point.

For instance, MWR and 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. Going back to Fig. 7.5b a presumably good BDR for $\mathcal{P}^{10\times 10}_{j.rnd}$ would be starting with $\xi^{\star}_{\text{SPT}}(k)$ and then switching over to $\xi^{\star}_{\text{MWR}}(k)$ at around time step $k = 40$, where the SDRs change places in outperforming one another. In addition, we can see that even though $\xi_{\text{SPT}}(k)$ is generally more likely to find optimal dispatches in the initial steps, shortly after $k = 15$ then $\xi_{\text{MWR}}(k)$ becomes a contender again. A box-plot of deviation from optimality, ρ, for $\mathcal{P}^{10\times 10}_{\text{train}}$ is depicted in Fig. 7.7 for a switch between SPT to MWR at time steps $k \in \{10, 15, 20, 30, 40\}$. Main statistics are given in Table 7.1.

This little manipulation between SDRs does outperform SPT immensely, yet doesn't manage to gain the performance edge of MWR. 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 at $k = 40$ 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. 7.5 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 7.7 does however show, that by applying MWR instead of SPT in the latter stages, does help the schedule to be more compact w.r.t. SPT. However, the fact remains that the schedules have diverged too far from what MWR would have been able to achieve on its own, i.e., using SPT downgrades the performance of MWR.

Figure 7.7: Box plot of $\mathcal{P}_{j.rnd}^{10\times10}$ deviation from optimality, ρ, for BDR where SPT is applied for the first 10%, 15%, 20%, 30% or 40% of the dispatches, followed by MWR.

Table 7.1: Main statistics for $\mathcal{P}_{j.rnd}^{10\times10}$ deviation from optimality, ρ, using BDR that changes from SDR at a fixed time step k.

SDR #1	SDR #2	k	Set	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
SPT	–	K	train	20.38	41.15	50.70	51.31	59.18	94.20
SPT	–	K	test	22.75	41.39	49.53	50.52	58.60	93.03
MWR	–	K	train	**4.42**	**17.84**	**21.74**	22.13	26.00	47.78
MWR	–	K	test	**3.37**	**17.07**	21.39	21.65	25.98	**41.80**
SPT	MWR	10	train	5.54	17.98	21.75	**21.99**	**25.43**	**44.02**
SPT	MWR	10	test	5.87	17.29	**20.78**	**21.28**	**24.67**	44.47
SPT	MWR	15	train	4.76	18.24	22.04	22.49	26.65	49.86
SPT	MWR	15	test	7.42	17.60	21.38	21.83	25.45	45.98
SPT	MWR	20	train	5.76	18.98	22.46	23.01	26.97	41.59
SPT	MWR	20	test	8.31	18.64	22.92	23.29	27.10	49.93
SPT	MWR	30	train	9.77	20.89	25.60	25.76	30.01	50.94
SPT	MWR	30	test	4.39	21.20	26.08	26.25	30.58	49.88
SPT	MWR	40	train	13.04	23.42	28.12	28.94	33.67	54.98
SPT	MWR	40	test	8.55	24.20	28.16	28.98	33.20	57.21

Changing to MWR at $k \leq 20$ is not statically significant from MWR (boost in mean ρ is at most 0.5%). However, after $k > 20$ then the BDR starts diverging from MWR. But as pointed in Section 7.2, it's not so fatal to make bad moves in the very first dispatches for $\mathcal{P}_{j.rnd}^{10\times10}$, hence little is gained with improved classification accuracy in that region. But this does tell us that ξ_π is a more reliable indicator than $\xi_\pi^\star$ when it comes to choosing appropriate model parameters. Alas, ξ_π requires collecting the meta-data $\{\Phi^\pi, \mathcal{Y}^\pi\}$ from Eq. (6.2) for its policy π, whereas $\xi_\pi^\star$ reuses $\{\Phi^{\pi_\star}, \mathcal{Y}^{\pi_\star}\}$ for each new policy π.

Revisiting Fig. 7.6a, then we see $\zeta_\mu^{\text{SPT}}(40)$ has already surpassed $\zeta_\mu^{\text{MWR}}(K)$ and there are 60 operations left to dispatch. So a switch for BDR at $k = 40$ never had a chance of improvement. However, at $k \leq 15$ then $\zeta_\mu^{\text{SPT}}(k) < \zeta_\mu^{\text{MWR}}(k)$, which were appropriate turning points for BDR (although not statistically significant).

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.

7.5 Feature evolution

In order to put the extremal features from Fig. 7.4 into perspective, it's worth comparing them with how the evolution of the features are over time, depicted in Fig. 7.8 for $\mathcal{P}_{j.rnd}^{6\times5}$ and $\mathcal{P}_{j.rnd}^{10\times10}$.* Note that the optimal trajectory describes how 'good' features should aspire to be like. We can also notice that the relative ranking in $\{\varphi_i\}_{i=17}^{24}$ is proportional their expected mean deviation from optimality, ρ (i.e. $\zeta_\mu^\pi(K)$). 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.

7.6 Emergence of problem difficulty

The main focus now is on knowing *when* during the scheduling process easy and hard problems diverge, this will be done using Φ^{ALL} and $\Phi^{\langle\text{SDR}\rangle}$ (conditioned on the followed trajectory) for $\mathcal{P}_{j.rnd}^{6\times5}$.** Individual $\Phi^{\langle\text{SDR}\rangle}$ are segregated w.r.t. its own quartiles defined in Eq. (4.2), whereas Φ^{ALL} is using the joint quartiles given in Table 4.1a. Note, if a joint quartile is used for $\Phi^{\langle\text{SDR}\rangle}$, then the segregation becomes highly unbalanced, e.g., for MWR the bulk of the problem instances are considered 'easy' and there is only a single 'hard' problem instance, as a result there can be no comparison between the two. The number of segregated problem instances for given in Table 7.2.

*Additional problem spaces can be found in Shiny application: Features > Evolution.

**Additional problem spaces can be found in Shiny application: Footprints > Stepwise.

(a) $\mathcal{P}^{6\times 5}_{j.rnd}$

Figure 7.8: Mean stepwise evolution of $\tilde{\phi}_i$, which is scaled according to Eq. (A.16)

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

Figure 7.8 (cont.)

Table 7.2: Number of problem instances after segregating $\mathcal{P}_{j.rnd}^{6\times5}$ w.r.t. difficulty and trajectory.

(a) Used in Fig. 7.9

Track	#Easy	#Hard	#Significant
LPT	126	125	111
LWR	125	126	73
MWR	125	125	150
SPT	127	126	93
Σ	503	502	427

(b) Used in Fig. 7.10

	#Significant	
Track	Easy	Hard
LPT	81	63
LWR	121	48
MWR	44	69
SPT	79	77
ALL	380	228
Σ	705	485

Rather than visualising 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 and use a t-test for testing the population correlation coefficient.

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 correction is used to counteract the multiple comparisons, i.e., each stepwise comparison has the significant level

$$\alpha_k = \frac{\alpha}{K} \tag{7.5}$$

thus maintaining the $\sum_{k=1}^{K} \alpha_k = \alpha$ significance level. However, with our limited sample size both α and α_k significance levels are reported, where entries with Bonferroni correction are especially highlighted.

Figure 7.9 indicates the timesteps when easy and hard feature distributions differ. Number of problem instances of segregated sets are given in Table 7.2a. In the initial stages, the features are more or less the same. However, there is a clear time of divergence (many of which rejected with Bonferroni correction) towards the end of the scheduling process: *i)* around the half way mark for MWR; *ii)* $k = 20$ for LPT, and *iii)* around $k = 25$ for SPT and LWR. Knowing this time of divergence, we could inspect the features from that time step onwards and check if they

Figure 7.9: Stepwise K-S test for features $\boldsymbol{\varphi}$ segregated w.r.t. easy and hard problems are drawn from the same continuous data distribution.

belong to our set of pre-classified 'easy' features. If not then it could be appropriate to choose jobs that do not correspond that particular SDR, that is to say if Fig. 7.6b indicates that on average $\{\zeta_{\min}^{\langle \mathrm{SDR} \rangle}(k), \zeta_{\max}^{\langle \mathrm{SDR} \rangle}(k)\}$ are performing better than if we'd continue with our intended trajectory (i.e. $\zeta_{\mu}^{\langle \mathrm{SDR} \rangle}(k)$). This applies especially for SPT and LPT.

Unfortunately, this information comes a little to late too be of much use for all SDR save for MWR, as $\zeta_{\mu}^{\langle \mathrm{SDR} \rangle}(k)$ is quite high for $\langle \mathrm{SDR} \rangle \in \{\mathrm{SPT}, \mathrm{LPT}\}$, and for LWR nothing can be done as there is no option of dispatching any other job than the single one remaining (cf. e.g. Fig. 7.8a).

Furthermore, Fig. 7.10 shows when easy or hard features are significantly correlated to the deviation from optimality, ρ. There we can see an apparent difference in correlation between individual features with the resulting schedule depending in what stage it is in the scheduling process, implying that their influence varies over the dispatching sequencing.

Figure 7.10: Stepwise significance of a correlation coefficient for $\mathcal{P}^{6\times 5}_{j.rnd}$ features $\boldsymbol{\varphi}$, segregated w.r.t. easy and hard problems, with resulting deviation from optimality, ρ.

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 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' roll-out features were introduced in Table 2.2.

What is surprising is that when looking at Fig. 7.10 then the active feature for the used SDR (i.e. φ_1 for SPT & LPT and φ_7 for LWR & MWR) then they are hardly ever significantly correlated to final ρ. However, their usage effects other features that are a key indicator throughout the dispatching process. Therefore we would need to take into account the joint interaction of features, and not look at them on their own as we do now.

Take for instance φ_7, which we know is a good indicator (i.e. MWR) and use φ_3 as well, then it's possible to obtain a model (cf. Chapter 9) that boosts MWR in performance by $\Delta\rho \approx -10\%$.

Note, even though some feature are hardly ever correlated w.r.t. ρ, then that does not necessarily imply that they're a bad attribute. Take for instance φ_8, which is a discrete simplification of φ_7, whose purpose was to complete advanced jobs (i.e. LWR) or balance progress for all jobs (i.e. MWR). In the case of $\mathcal{P}^{6\times 5}_{j.rnd}$ then $-\varphi_8$ yields $\rho = 18\%$, whereas its more sophisticated counterpart, $+\varphi_7$ has a slightly lower $\rho = 16\%$ (cf. Fig. 7.4a).

It is possible for a JSP schedule to have more than one sequential dispatching representation. It is especially true during 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 difference between the difficulties for the initial steps of the scheduling process (cf. Fig. 7.9). As we can see from Table 7.2, the number of problem instances used for statistical testing is quite limited when restricting to a single algorithm. Using the non-uniqueness of $\boldsymbol{\chi}$ to our advantage, where there are many jobs that have non-conflicting machines, thereby making subsequent dispatches equivalent to the previous one, i.e., $\boldsymbol{\chi}^{(k)} \approx \boldsymbol{\chi}^{(k\pm 1)}$. Therefore it's reasonable, when labelled optimal data is scarce, to inspect the stepwise statistical testing based on sliding window of the preceding and subsequent step, i.e., test at time k is based on:

$$\boldsymbol{\varphi}_i(k) := \left\{\varphi_i(k') \ : \ \forall \varphi_i \in \Phi\right\}_{k'=k-1}^{k+1} \tag{7.6}$$

for all individual local features $\varphi_i \in \{1, \ldots, 16\}$ from Table 2.2.

7.7 Conclusions

The main objective of this chapter was to illustrate the interaction of a specific algorithm on a given problem structure and its properties. This can be considered as finding the footprint from single priority dispatching rules introduced in Section 2.4 on those problem spaces. Presumably, we could use that information to infer the complexity of our synthesised problem spaces summarised in Table 3.2. From Fig. 7.4 we noticed that high stepwise optimality, $\xi^{\star}_{\pi}$, generally implies low deviation from optimality, ρ. However, that is by no means a guarantee, as there is clearly an important factor *when* suboptimal moves are made, as Fig. 7.3 showed for $\{\zeta^{\star}_{\min}, \zeta^{\star}_{\max}\}$. However, that is based on the expert policy $\pi^{\star}$, which is quite optimistic. Since our deterministic policy isn't perfect, then it's non-trivial to anticipate that effect on our end-result. As $\zeta^{\star}_{\langle\cdot\rangle}$ is based on making only *one* sub-optimal move, and as there will undoubtedly be many more, then the measures $\xi^{\star}$ and $\zeta^{\star}_{\langle\cdot\rangle}$ were adjusted to their intended policy, i.e., $\zeta^{\pi}_{\langle\cdot\rangle}$ and $\zeta^{\pi}_{\langle\cdot\rangle}$. The pros being that gives a better picture of the policy's performance evolution. But the con is that this requires an intensive labelling process for each proposed policy. Whereas, $\xi^{\star}$ and $\zeta^{\star}_{\langle\cdot\rangle}$ only were dependent on meta-data from expert policy which could be used over and over again for measuring any policy π.

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. Section 7.6 showed that the emergence of a problem instances difficulty w.r.t. its algorithm was not noticeable until the very end of the dispatching process. Preferably we'd like to know this information sooner, in order to steer the algorithm towards a more promising state space where the features are 'known' to have better performance.

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. 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.

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

Rabbit

8 Preference Learning

Learning models considered in this dissertation are based on ordinal regression in which the supervised 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 Rúnarsson (2006), and given in Appendix A for completeness.

8.1 Ordinal regression for job-shop

Using the training set $\{\Phi^{\pi}, \mathcal{Y}^{\pi}\}$, given in Eq. (6.2) by following some policy π, let $\boldsymbol{\varphi}^{o} \in \Phi^{\pi}$ denote the post-decision state when dispatching job J_o corresponds to an optimal schedule being built. All post-decisions states corresponding to suboptimal dispatches, J_s, are denoted by $\boldsymbol{\varphi}^{s} \in \Phi^{\pi}$.

Let's label feature sets which 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. The preference learning problem is specified by a set of preference pairs,

$$\Psi := \bigcup_{\{\mathbf{x}_i\}_{i=1}^{N_{\text{train}}}} \left\{ \{\mathbf{z}^{o}, +1\}, \{\mathbf{z}^{s}, -1\} \ : \ \forall (J_o, J_s) \in \mathcal{O}^{(k)} \times \mathcal{S}^{(k)} \right\}_{k=1}^{K} \subset \Phi \times Y \tag{8.1}$$

where: *i)* $\Phi \subset \mathcal{F}$ is the training set of $d = 16$ features (cf. the local features from Table 2.2); *ii)* $Y = \{-1, +1\}$ is the outcome space; *iii)* at each dispatch $k \in \{1, \ldots, K\}$, and *iv)*

$J_o \in \mathcal{O}^{(k)}$, $J_s \in \mathcal{S}^{(k)}$ are optimal and suboptimal dispatches, respectively.

A negative example is only created as long as J_s actually results in a worse makespan, i.e., $C_{\max}^{\pi_\star}(\boldsymbol{\chi}^s) \gneq C_{\max}^{\pi_\star}(\boldsymbol{\chi}^o)$, since there can exist situations in which more than one operation can be considered optimal. Hence, $\mathcal{O}^{(k)} \cup \mathcal{S}^{(k)} = \mathcal{L}^{(k)}$, and $\mathcal{O}^{(k)} \cap \mathcal{S}^{(k)} = \emptyset$. If the makespan would be unaltered, the pair is omitted from Υ, 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, i.e., $C_{\max}^{\pi_\star}(\boldsymbol{\chi}^j)$ for all $J_j \in \mathcal{L}^{(k)}$. 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.

Since $Y = \{+1, -1\}$, we can use logistic regression, which 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 Eq. (A.9). 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 from Eq. (2.12) where $\pi(\cdot)$ is the classification model obtained by the preference set, Υ, defined by Eq. (8.1).

8.2 Selecting preference pairs

Defining the size of the preference set as $l = |\Upsilon|$, then Eq. (8.1) gives the size of the feature training set as $|\Phi| = \frac{1}{2}l$, which is given in Fig. 6.2 and Table 6.1. If l is too large, than sampling needs to be done in order for the ordinal regression 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 preference set of size,

$$l = \sum_{i=1}^{N_{\text{train}}} \left(2|\mathcal{O}_i^{(k)}| \cdot |\mathcal{S}_i^{(k)}| \right) \tag{8.2}$$

For the problem spaces considered in Paper I, that sort of simple sampling of the state space was sufficient for a favourable outcome. However, for a considerably harder problem spaces (cf. Chapter 4) and not to mention increased number of jobs and machines, 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 5) 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 preference 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 preference 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.

All problem instances are correctly labelled w.r.t. their optimum makespan, found with analytical means.* The main motivation for the data generation of Ψ that will be used in preference learning, will now need to consider the following main aspects:

PREF.1 Which path(s) π should be investigated to collect training instances, i.e., Φ^{π}. Should they be features gathered resulting in : *i)* optimal solutions (querying expert policy $\pi_{\star}$)? *ii)* suboptimal solutions when a DR is implemented (following a fixed policy π), or *iii)* combination of both?

PREF.2 What sort of rankings should be compared during each step?

PREF.3 What sort of stepwise sampling strategy is needed for a good *single* time independent model?

The collection of the training set Φ in PREF.1 (which is described in Chapter 6) is of paramount of importance, as the subsequent preference pairs in Ψ are highly dependent on the quality of Φ. Since the labelling of Φ is quite computationally intensive, its collection should be done

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

parsimoniously in order to not waste valuable time and resources. On the other hand, PREF.2 and PREF.3 are easy to inspect, once Φ has been chosen. The following sections will try to address these research questions.

8.3 Scalability of dispatching rules

In Paper I a separate data set was 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. 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 $\Psi(k)$ at dispatch k.

A more sophisticated way, would be to create a *new* linear model, where the preference set, Ψ, is the aggregation of all preference pairs across the K dispatches. This would amount to a substantial training set, and for Ψ to be computationally feasible to learn, Ψ has to be filtered to size $l_{\max}$. The default set-up will be,

$$l_{\max} := \begin{cases} 5 \cdot 10^5 & \text{for} \quad 10 \times 10 \quad \text{JSP} \\ 10^5 & \text{for} \quad 6 \times 5 \quad \text{JSP} \end{cases} \tag{8.3}$$

which is roughly 60%-70% amount of preferences encountered from one pass of sampling a K-stepped trajectory using a fixed policy $\hat{\pi}$ for the default N_{train} (cf. Table 8.1). Sampling is done randomly, with equal probability.

8.4 Ranking strategies

First let's address PREF.2. The various ranking strategies for adding preference pairs to Ψ defined by Eq. (8.1) were first reported in Paper V, and are the following,

Basic ranking, Ψ_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, $|\Psi_b|$ is defined in Eq. (8.2).

Full subsequent rankings, Ψ_f, i.e., all possible combinations of r_i and r_{i+1} for $i \in \{1, \ldots, n'\}$, preference pairs are added.

Partial subsequent rankings, Ψ_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 $\Psi_p \subset \Psi_f$.

All rankings, Ψ_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 the rankings of the job-list, $\mathcal{L}^{(k)}$, at time step k, is as follows,

$$r_1 > r_2 > \ldots > r_{n'} \quad (n' \leq n) \tag{8.4}$$

By definition the following property holds:

$$\Psi_p \subset \Psi_f \subset \Psi_b \subset \Psi_a \tag{8.5}$$

To test the validity of different ranking strategies for PREF.2, a training set of $N_{\text{train}} = 500$ problem instances of $\mathcal{P}^{6\times5}_{j.rnd}$ and $\mathcal{P}^{6\times5}_{f.rnd}$ is collected for all trajectories described in Section 6.4. The size of the preference set, $|\Psi|$, is depicted in Fig. 8.1 for each iteration k. From which, a linear preference model is created for each preference set, Ψ. A box-plot for deviation from optimality, ρ, defined by Eq. (2.17), is presented in Fig. 8.2. From the figure it is apparent there can be a performance edge gained by implementing a particular trajectory strategy, yet ranking scheme seems to be irrelevant. Moreover, the behaviour is analogous across all other $\mathcal{P}^{6\times5}_{\text{train}}$ in Table 3.2.

First let's restrict the models to $\mathcal{P}^{6\times5}_{\text{train}}$. There is no statistical difference between Ψ_f and Ψ_p ranking-schemes across all disciplines, which is expected since Ψ_f is designed to contain the same preference information as Ψ_f (cf. Eq. (8.5)). However, neither of the Pareto ranking-schemes outperform the original Ψ_b set-up from Paper I. The results hold for the test set as well. Any statistical difference between ranking schemes were for Ψ_a, where it was considered slightly lacking than some of its counterparts. Since a smaller preference set is preferred, its opted to use the Ψ_p ranking scheme henceforth as the default set-up for PREF.2.

Moving on to higher dimension, results for $\mathcal{P}^{10\times10}_{j.rnd}$ were similar to $\mathcal{P}^{6\times5}_{\text{train}}$. Only exception begin that ranking schemes showed difference in performance when using Φ^{OPT}, where Ψ^{OPT}_p come on top. Strengthening our previous choice of Ψ_p as standard ranking scheme.

8.5 Trajectory strategies

We'd like to inspect which trajectory is the best to use for Ψ. Paper V only considered $\mathcal{P}^{6\times5}_{j.rnd}$ and $\mathcal{P}^{6\times5}_{j.rndn}$, however, results for $\mathcal{P}^{6\times5}_{\text{train}}$ and $\mathcal{P}^{10\times10}_{j.rnd}$ are currently available.* Models from Fig. 8.2 are limited to the ones corresponding to Ψ_p. Moreover, main statistics for $\mathcal{P}^{10\times10}$ are given in

*Additional problem spaces can be found in Shiny application: Preference Models > Trajectories & ranks.

Figure 8.1: Size of $\mathcal{P}^{6\times 5}_{\text{train}}$ preference set, $l = |\Psi|$, for different trajectory strategies and ranking schemes (where $N_{\text{train}} = 500$)

Figure 8.2: Box-plot for various Φ and Ψ set-up using $\mathcal{P}^{6\times5}_{\text{train}}$. The trajectories the models are based on are depicted in white on the far right.

Table 8.3. Figure 8.3 jointly illustrates the size of the preference set used, i.e., $|\Psi_p|$ from Fig. 8.1. Table 8.1 reports the total amount of preferences for all K dispatches.

Table 8.2 reports the relative ordering of trajectories, ordered w.r.t. their mean deviation from optimality, ρ, and their size of preference set, i.e., $|\Psi_p|$. Models that are statistically better are denoted by '$\succ$' otherwise considered equivalent.

For most problem spaces Ψ_p^{LPT} was the worst trajectory to pursue. Looking back at Fig. 6.2, then even though Φ^{LPT} was not the trajectory with the least features, the amount of equivalent features w.r.t. $C_{\max}$ are far too many to make a meaningful preference set out of it. It's only for $\mathcal{P}_{j.rndn}^{6\times5}$ that there is another trajectory with fewer preferences, namely Ψ_p^{LWR} (cf. Fig. 8.1), and in that case LWR is the worst model. Model that come on top, are those that have a varied Ψ. However, aggregating features from all trajectories is not a good idea, as the preference set then becomes too varied for a satisfactory result.

Learning preference pairs from a good scheduling policies, such as $\Phi^{\text{ES}.C_{\max}}$, $\Phi^{\text{ES}.\rho}$ and Φ^{MWR}, gave considerably more favourable results than tracking optimal paths, save for $\mathcal{P}_{f.jc}^{6\times5}$ where the ordering is reversed. Generally, suboptimal routes are preferred. However, even though LWR is a better policy than MWR for FSP, then Φ^{LWR} is a worse candidate than e.g. Φ^{MWR}, but as discussed before, it's due to the lack of varied dispatches for the trajectory.

It is particularly interesting there is statistical difference between Φ^{OPT} and Φ^{RND}, where the latter had improved performance for all JSP problem spaces. In those cases, tracking optimal dispatches gives worse performance as pursuing completely random dispatches. This indicates that exploring only expert policy 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.

Generally, adding suboptimal trajectories with the expert policy, i.e., Φ^{ALL}, gives the learning algorithm a greater variety of preference pairs for getting out of local minima. However, for some problem spaces, e.g., $\mathcal{P}_{f.rnd}^{6\times5}$ and $\mathcal{P}_{f.mc}^{6\times5}$ then additional suboptimal solutions that are too diverse yield a worse outcome than Φ^{OPT} would achieve on its own.

Table 8.1: Total number of preferences in $l = |\Psi_p|$ for all K steps. Note '–' denotes not available.

Track	$\mathcal{P}^{6\times5}_{\text{train}}, N_{\text{train}} = 500$									$\mathcal{P}^{10\times10}_{\text{train}}, N_{\text{train}} = 300$		
	j.rnd	*j.rndn*	*j.rnd*, J_1	*j.rnd*, M_1	*f.rnd*	*f.rndn*	*f.jc*	*f.mc*	*f.mxc*	*j.rnd*	*j.rndn*	*f.rnd*
SPT	73926	68410	74416	65150	79388	70808	68956	89788	92036	285912	–	–
LPT	43456	58540	28498	34136	36162	54684	11548	23260	17308	151444	–	–
LWR	46580	46306	32326	41554	64226	68628	69124	40150	40110	163546	–	–
MWR	83756	102092	53246	62056	87376	111708	106226	65882	64692	370104	–	–
RND	72824	80358	52210	61670	77148	77080	64550	55288	55398	313346	–	–
OPT	100910	111736	79404	90948	95388	93036	81306	79836	78440	453662	470522	299952
ES.ρ	93006	111068	64050	89504	77142	63120	45404	36608	74556	427032	–	–
ES.$C_{\max}$	108390	111346	73168	95920	83058	61992	47412	35484	36052	432650	–	–
ALL	622848	689856	457318	540938	599888	601056	494526	426296	458592	2595758	470522	299952

Table 8.2: Relative ordering w.r.t. mean ρ and size of its preference set, $l = |\Psi_p|$, for trajectories in Section 6.4

Problem		Ordering of trajectories
$\mathcal{P}^{6\times5}_{j.rnd}$	ρ	ES.$C_{\max}$ $\succ$ ES.ρ $\succ$ ALL $\equiv$ SPT $\equiv$ MWR $\equiv$ RND $\succ$ OPT $\succ$ LWR $\succ$ LPT
	l	ALL $\gg$ ES.$C_{\max}$ $>$ OPT $>$ ES.ρ $>$ MWR $>$ SPT $>$ RND $>$ LWR $>$ LPT
$\mathcal{P}^{6\times5}_{j.rndn}$	ρ	ES.$C_{\max}$ $\equiv$ ES.ρ $\equiv$ MWR $\equiv$ RND $\equiv$ ALL $\equiv$ SPT $\succ$ LPT $\succ$ LWR $\succ$ OPT
	l	ALL $\gg$ OPT $>$ ES.$C_{\max}$ $>$ ES.ρ $>$ MWR $>$ RND $>$ SPT $>$ LPT $>$ LWR
$\mathcal{P}^{6\times5}_{j.rnd,J_1}$	ρ	ES.$C_{\max}$ $\succ$ ES.ρ $\succ$ SPT $\succ$ ALL $\equiv$ RND $\succ$ OPT $\equiv$ LWR $\equiv$ MWR $\succ$ LPT
	l	ALL $\gg$ OPT $>$ SPT $>$ ES.$C_{\max}$ $>$ ES.ρ $>$ MWR $>$ RND $>$ LWR $>$ LPT
$\mathcal{P}^{6\times5}_{j.rnd,M_1}$	ρ	ES.$C_{\max}$ $\equiv$ ES.ρ $\succ$ SPT $\equiv$ ALL $\equiv$ RND $\succ$ LWR $\succ$ OPT $\succ$ MWR $\succ$ LPT
	l	ALL $\gg$ ES.$C_{\max}$ $>$ OPT $>$ ES.ρ $>$ SPT $>$ MWR $>$ RND $>$ LWR $>$ LPT
$\mathcal{P}^{6\times5}_{f.rnd}$	ρ	SPT $\equiv$ ES.$C_{\max}$ $\equiv$ OPT $\equiv$ ES.ρ $\succ$ ALL $\succ$ LWR $\succ$ MWR $\equiv$ RND $\equiv$ LPT
	l	ALL $\gg$ OPT $>$ MWR $>$ ES.$C_{\max}$ $>$ SPT $>$ RND $>$ ES.ρ $>$ LWR $>$ LPT
$\mathcal{P}^{6\times5}_{f.rndn}$	ρ	ES.ρ $\equiv$ ES.$C_{\max}$ $\equiv$ RND $\equiv$ LWR $\equiv$ ALL $\succ$ SPT $\equiv$ LPT $\equiv$ OPT $\succ$ MWR
	l	ALL $\gg$ MWR $>$ OPT $>$ RND $>$ SPT $>$ LWR $>$ ES.ρ $>$ ES.$C_{\max}$ $>$ LPT
$\mathcal{P}^{6\times5}_{f.jc}$	ρ	OPT $\succ$ SPT $\equiv$ LWR $\succ$ RND $\equiv$ MWR $\succ$ LPT $\equiv$ ES.$C_{\max}$ $\equiv$ ES.ρ $\succ$ ALL
	l	ALL $\gg$ MWR $>$ OPT $>$ LWR $>$ SPT $>$ RND $>$ ES.$C_{\max}$ $>$ ES.ρ $>$ LPT
$\mathcal{P}^{6\times5}_{f.mc}$	ρ	ES.ρ $\succ$ LWR $\succ$ ES.$C_{\max}$ $\equiv$ MWR $\succ$ OPT $\equiv$ LPT $\equiv$ RND $\equiv$ SPT $\succ$ ALL
	l	ALL $\gg$ SPT $>$ OPT $>$ MWR $>$ RND $>$ LWR $>$ ES.ρ $>$ ES.$C_{\max}$ $>$ LPT
$\mathcal{P}^{6\times5}_{f.mxc}$	ρ	RND $\equiv$ OPT $\equiv$ ES.ρ $\equiv$ SPT $\equiv$ ALL $\equiv$ MWR $\succ$ LWR $\succ$ ES.$C_{\max}$ $\succ$ LPT
	l	ALL $\gg$ SPT $>$ OPT $>$ ES.ρ $>$ MWR $>$ RND $>$ LWR $>$ ES.$C_{\max}$ $>$ LPT
$\mathcal{P}^{10\times10}_{j.rnd}$	ρ	ES.$C_{\max}$ $\equiv$ ES.ρ $\succ$ MWR $\succ$ RND $\succ$ SPT $\equiv$ OPT $\succ$ ALL $\succ$ LPT $\equiv$ LWR
	l	ALL $\gg$ OPT $>$ ES.$C_{\max}$ $>$ ES.ρ $>$ MWR $>$ RND $>$ SPT $>$ LWR $>$ LPT

Table 8.3: Main statistics for deviation from optimality, ρ, using $\mathcal{P}^{10\times10}_{\text{train}}$ based on various trajectories for Ψ_p

Problem	Track	Set	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
$\mathcal{P}^{10\times10}_{j.rnd}$	SPT	train	10.07	23.68	28.76	29.22	34.21	57.58
	SPT	test	14.59	24.25	29.18	30.06	35.10	64.72
	LPT	train	14.94	29.14	36.33	37.13	43.42	75.56
	LPT	test	13.01	29.80	35.72	36.63	43.46	77.17
	LWR	train	17.05	29.33	37.34	37.23	43.18	68.81
	LWR	test	15.68	31.01	37.06	37.63	42.64	63.69
	MWR	train	8.32	17.65	21.70	22.80	27.12	45.60
	MWR	test	**2.02**	16.83	22.42	22.61	26.77	54.37
	RND	train	10.14	21.19	26.01	27.22	32.33	50.69
	RND	test	8.60	22.03	26.67	27.70	32.11	56.56
	OPT	train	7.87	23.34	29.30	30.73	36.47	61.45
	OPT	test	8.31	24.05	31.85	32.31	39.74	66.42
	ES.ρ	train	**2.22**	12.07	15.57	16.43	20.54	42.82
	ES.ρ	test	2.72	12.20	**15.37**	**16.80**	**20.16**	**39.16**
	ES.$C_{\max}$	train	4.20	**11.58**	**15.56**	**16.07**	**19.64**	**38.24**
	ES.$C_{\max}$	test	5.58	**12.19**	15.74	16.90	20.30	47.52
	ALL	train	8.89	25.67	33.56	34.19	40.67	71.89
	ALL	test	11.39	26.50	34.52	33.65	40.18	65.10
$\mathcal{P}^{10\times10}_{j.rndn}$	OPT	train	18.02	33.89	40.53	41.51	48.15	75.30
	OPT	test	15.31	33.38	40.67	40.58	47.46	73.12
$\mathcal{P}^{10\times10}_{f.rnd}$	OPT	train	3.55	17.85	22.36	22.56	27.24	43.60
	OPT	test	3.33	17.15	21.55	22.34	27.08	43.36

Comparing Ψ^{π}_{p} to its corresponding policy π used to guide its collection, then usually the preference model outperformed the π it was trying to mimic. The exceptions being: *i)* MWR for $\mathcal{P}^{6\times5}_{j.rnd,J_1}$ and $\mathcal{P}^{6\times5}_{j.rnd,M_1}$ (and $\mathcal{P}^{10\times10}_{j.rnd}$ was statistically insignificant); *ii)* LWR for $\mathcal{P}^{6\times5}_{f.mc}$ and $\mathcal{P}^{6\times5}_{f.mxc}$; *iii)* LPT was statistically insignificant for $\mathcal{P}^{6\times5}_{j.rnd,M_1}$, and *iv)* ES.$C_{\max}$ and ES.ρ for all problem spaces, save for $\mathcal{P}^{6\times5}_{j.rndn}$ which was statistically insignificant. Revisiting Fig. 7.6, then when Ψ^{π}_{p} succeeds its original policy π, it implies the learning model was able to steer the learned policy towards $\zeta^{\pi}_{\min}$. In fact, its improvement is proportional to its spread* from ζ^{π}_{μ} to $\zeta^{\pi}_{\min}$ or $\zeta^{\pi}_{\max}$. Therefore, a good preference set based on Φ^{π} not only has to have a low ζ^{π}_{μ} to mimic, but also the policy π needs to be sufficiently different from $\zeta^{\pi}_{\min}$ and $\zeta^{\pi}_{\max}$ for adequate learning. That is why $\Phi^{\langle\text{CMA-ES}\rangle}$ strategies were not good enough for preference learning, as their $\zeta^{\pi}_{\langle\cdot\rangle}$ spread was the lowest compared to the other fixed DRs.

The rational for using the $\Phi^{\langle\text{CMA-ES}\rangle}$ strategies was mostly due to the fact a linear classifier is creating the training data (using the weights found via CMA-ES optimisation in Eq. (5.1)), hence

*Consult Shiny application: Optimality > Best and worst case scenario.

the training data created should be linearly separable, which in turn should boost the training accuracy for a linear classification learning model. However, these strategies is not outperforming the original DR used in guiding the training data collection.

8.6 Stepwise sampling bias

Experiments in Section 8.5 clearly showed that following the expert policy is not without its faults. There are many obstacles to consider to improve the model. For instance, it was chosen to sample $l_{\max}$ in Eq. (8.3) with equal probability. But inspecting the effects of making suboptimal choices varies as a function of time (cf. Chapter 7), perhaps its stepwise bias should rather be done proportional to the mean cumulative loss to a particular time step? Following strategies for stepwise bias for PREF.3 will now be proportional to:

Bias.1 **(equal)** equal probability.

Bias.2 **(opt)** inverse optimality for random dispatches, i.e., $1 - \xi^{\star}_{\text{RND}}$.

Bias.3 **(bcs)** best case scenario for mean ρ, i.e., $\zeta^{\star}_{\min}$.

Bias.4 **(wcs)** worst case scenario for mean ρ, i.e., $\zeta^{\star}_{\max}$.

Bias.5 **(featsize)** inversely proportional to $|\Phi^{\text{OPT}}|$, defined as

$$\max_{k'}\{|\Phi^{\text{OPT}}(k')|\} - |\Phi^{\text{OPT}}(k)| + \min_{k}\{|\Phi^{\text{OPT}}(k')|\}$$

Bias.6 **(prefsize)** inversely proportional to $|\Psi_p^{\text{OPT}}|$, defined as

$$\max_{k'}\{|\Psi_p^{\text{OPT}}(k')|\} - |\Psi_p^{\text{OPT}}(k)| + \min_{k'}\{|\Psi_p^{\text{OPT}}(k')|\}$$

Bias.7 **(dbl1st)** twice as much weight on the first half of the dispatches.

Bias.8 **(dbl2nd)** twice as much weight on the second half of the dispatches.

Moreover, all strategies are also adjusted to the preference set size, i.e., $1/|\Psi_p^{\text{OPT}}|$. The sampling strategy for Ψ_p in Papers IV and V was Bias.1 and serves as a baseline. Motivation for Bias.2 is that way samples from dispatches that are less likely to be optimal than simply at random (cf. Fig. 7.2) are emphasised. Whereas, Bias.3 and Bias.4 are more focused on sampling w.r.t. the final measure, where the mean ρ is given *one* suboptimal move, otherwise it's assumed expert policy is followed (cf. Fig. 7.3). The adjustment of preference set tries to give equal emphasis on stepwise features, as they substantially decrease over time (cf. Figs. 6.2 and 8.3), which proved favourable

Figure 8.3: Size of $\mathcal{P}^{6\times 5}_{\text{train}}$ preference set, $l = |\Psi_p|$, for different trajectory strategies

in preliminary experiments, then Bias.5 and Bias.6 motivation is to boost that adjustment even further. Lastly, Bias.7 and Bias.8 are very simplified versions of the aforementioned strategies. Figure 8.4 jointly illustrates the stepwise bias strategies for $\mathcal{P}^{10\times10}_{\text{train}}$.

It's possible to circumvent the choice of stepwise sampling strategy by creating a preference model for each time step k, for a grand total of K models. By doing so it's possible to capture local changes in the schedule, as we've already seen the evolution of features varies. Moreover, for CMA-ES optimisation then a stepwise *new* model was generally better than a *single* global one (cf. Fig. 5.3). However, in that case it's not possible to test those models against other dimensions, e.g., test benchmarks suite from OR-Library (cf. Table 3.3).

Figure 8.5 depicts box-plots for deviation from optimality, ρ, using the various sampling strategies for $\mathcal{P}^{10\times10}$. Main statistics for $\mathcal{P}^{10\times10}_{\text{train}}$ are reported in Table 8.4. In addition to the stepwise bias strategies (both adjusted and not) a stepwise model (one for each step k) is given for reference.

First off, counter-intuitively the stepwise model is not the best configuration. By applying one of these aforementioned sampling strategies it's possible to achieve better results than applying a local model for each time step. In fact, for $\mathcal{P}^{10\times10}_{j.rndn}$ a stepwise model was the worst approach (with up to 12% increased error). This could possibly be explained by the fact that there are quite a few non-conflicting operations. As a result there is this vague change in 'time' for consecutive steps. Therefore, using a complete data set which aggregates all time steps (or arguably over a few steps) is more beneficial for learning, it is dealt with on a more sustainable grounds.

Adjusting Ψ^{OPT}_p to its stepwise size generally improved the sampling strategy (up to mean $\Delta\rho \approx -7\%$), where Bias.2 and Bias.1 were equivalent for $\mathcal{P}^{10\times10}_{j.rnd}$. Whereas, $\mathcal{P}^{10\times10}_{j.rndn}$ Bias.4 was significantly worsened by mean $\Delta\rho \approx +3\%$ if adjusted. Other $\mathcal{P}^{10\times10}_{j.rndn}$ strategies were equivalent w.r.t. adjustment. Reverting back to Fig. 7.9, then we saw that near the end of the dispatching process then for all SDRs there was a clear segregation of features w.r.t. its difficulty. This implies great predictability of features in that time region. However, since those data points are scarce they get overrun by the superabundant preference pairs from the preceding dispatches, unless they are appropriately superimposed to be relevant for classification.

For all problem spaces, there was no significant difference between stepwise models to either Bias.5 or Bias.6. In the case of $\mathcal{P}^{10\times10}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{f.rnd}$ when a stepwise model is promising, then superimposing the adjustment of preference set gives the best overall outcome. Whereas, in the case of $\mathcal{P}^{10\times10}_{j.rndn}$ then a single stepwise model is not as adequate as its single counterparts, then over emphasising w.r.t. the set works poorly. All other bias strategies for $\mathcal{P}^{10\times10}_{j.rndn}$ came out similar, with the exception of Bias.3 being slightly lacking, yet better than Bias.5 and Bias.6.

Furthermore, Bias.2 work just as well as its simplified version Bias.7 in $\mathcal{P}^{10\times10}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rndn}$, but for $\mathcal{P}^{10\times10}_{f.rnd}$ the simpler version was slightly better. Similarly, Bias.8 was equivalent to Bias.4. Note in $\mathcal{P}^{10\times10}_{f.rnd}$ then Bias.8 serves as just as well as its best strategies Bias.6.

8.7. CONCLUSIONS

To summarise, adjusting the preference set to give each step equal probability is a good first step. Moreover, when time dependent model are good then further exaggeration of the adjustment to the preference set (such as Bias.5 and Bias.6) is best. However, a severely simplified version can be just as good. With these configurations it was possible to improve mean deviation from optimality, ρ, by: *i)* 14% using adjusted Bias.6 for $\mathcal{P}^{10\times10}_{j.rnd}$; *ii)* 1% using adjusted Bias.7 (not significant improvement) for $\mathcal{P}^{10\times10}_{j.rndn}$, and *iii)* 8% using adjusted Bias.6 for $\mathcal{P}^{10\times10}_{f.rnd}$.

Using the simplified version of the best configuration for Ψ_p^{OPT}, i.e., adjusted Bias.8, for the best $\mathcal{P}^{10\times10}_{j.rnd}$ trajectory from Section 8.5, namely $\Psi_p^{\mathrm{ES}.C_{\max}}$, then it's possible to get a 4.5% mean boost in performance, i.e., 11.39% and 11.73% for training and test set, respectively. Note, optimisation w.r.t. Eq. (5.1a) achieved 10.57% and 11.33% mean ρ for training and test set, respectively, which is statistically insignificant from the adjusted preference model.

8.7 Conclusions

Since the preference set is ideally aggregated and possibly re-sampled to adjust for lacking $|\Psi(k)|$ count, then Ψ needs to be sampled to size $l_{\max}$ such that it contains maximum information, yet with minimal amount of preference pairs. By use of partial subsequent Pareto ranking to address PREF.2, denoted Ψ_p, from Section 8.4, it's possible to reduce $|\Psi|$ significantly, without loss in performance.

Experimental results in Section 8.5 for PREF.1 illustrated that unlike Malik et al. (2008), Olafsson and Li (2010), Russell et al. (2009) learning only on optimal training data was not fruitful. However, Section 8.6 showed if stepwise sampling for PREF.3 is done appropriately then it's possible to boost performance significantly, even outperforming a single model for each time step. First and foremost the stepwise bias in sampling needs to counter-act the disproportionate amount of features towards the end. Moreover, additional emphasis to the latter stages of the dispatches is beneficial as that's when JSP is more susceptible to failure. Furthermore, since the problem spaces showed difference boost in performance depending on the various complexities of its best sampling strategy, its simpler version is recommended, namely configuration denoted Bias.8.

Inspired by the original work by Li and Olafsson (2005), having fixed DRs guide the generation of training data (except correctly labelling with analytic means) gives meaningful preference pairs which the learning algorithm could learn. The best strategy was by using the weights from CMA-ES optimisation, obtained by optimising Eq. (5.1) directly. Its preference model was able to be statistically insignificant to its guiding policy (cf. Section 8.6). However, we have yet been able to outperform direct optimisation.

Generally aggregating trajectories, from optimal and suboptimal policies, boosts performance. However, they need to be chosen carefully, since with increased aggregation it can become

counter-productive as the features are too dissimilar. A more sophisticated approach in combing the two strategies is needed.

Figure 8.4: Log probability for stepwise sampling for Ψ_p^{OPT} based on $\mathcal{P}_{\mathrm{train}}^{10\times 10}$

Figure 8.5: Box-plots for deviation from optimality, ρ, where Ψ_p^{OPT} is sampled with various stepwise bias strategies. Moreover, time dependent models shown on far left as reference.

Table 8.4: Main statistics for deviation from optimality, ρ, using $\mathcal{P}^{10\times10}_{\text{train}}$ based on various stepwise sampling strategies for Ψ_p^{OPT}. Models are ordered w.r.t. mean ρ

	|Model|*	Bias	Adj.	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
	1	prefsize	T	5.12	12.26	15.77	16.48	19.77	39.14
	1	prefsize	F	5.85	16.82	21.24	21.75	25.52	56.08
	K	equal	F	8.32	17.33	21.42	21.87	25.56	42.93
	1	featsize	T	5.56	16.84	22.08	22.22	26.89	54.47
	1	dbl2nd	T	6.95	18.35	23.44	24.26	28.64	55.81
	1	bcs	T	6.05	18.57	23.85	24.52	29.18	47.83
	1	wcs	T	8.08	20.66	26.38	27.99	34.64	60.84
	1	dbl2nd	F	8.58	21.79	27.04	28.91	34.36	68.22
$\mathcal{P}^{10\times10}_{j.rnd}$	1	equal	T	9.97	21.76	27.46	29.13	35.17	60.84
	1	wcs	F	10.99	23.33	28.41	29.73	34.97	60.84
	1	featsize	F	6.04	22.34	28.97	29.82	35.77	68.22
	1	bcs	F	10.18	22.04	28.24	29.89	35.16	68.22
(default)	1	equal	F	7.87	23.34	29.30	30.73	36.47	61.45
	1	opt	T	7.87	23.38	30.52	31.55	38.46	63.85
	1	dbl1st	T	10.99	24.47	31.07	31.93	38.82	80.11
	1	opt	F	10.18	24.57	31.50	32.76	39.42	70.47
	1	dbl1st	F	8.89	26.38	33.25	34.26	41.32	69.51
	1	dbl1st	T	15.02	33.23	40.31	40.66	46.78	73.57
	1	opt	T	13.93	33.23	40.26	40.75	48.30	77.34
	1	equal	T	18.86	34.34	40.79	41.41	48.39	75.35
	1	dbl2nd	F	14.59	34.25	40.49	41.46	47.06	72.82
	1	opt	F	16.64	34.14	40.79	41.49	47.38	72.95
(default)	1	equal	F	18.02	33.89	40.53	41.51	48.15	75.30
	1	wcs	F	18.94	33.59	41.47	41.81	48.97	75.35
	1	bcs	F	19.14	34.60	41.62	42.41	48.78	76.30
$\mathcal{P}^{10\times10}_{j.rndn}$	1	wcs	T	19.38	35.36	42.17	42.48	48.90	73.78
	1	dbl1st	F	20.46	35.15	41.80	42.73	48.54	74.61
	1	dbl2nd	T	20.28	35.42	42.02	43.02	49.00	81.62
	1	bcs	T	19.27	37.70	44.98	45.58	52.05	81.37
	1	featsize	F	25.72	41.99	48.32	49.22	56.00	79.55
	1	prefsize	F	24.88	41.60	49.93	49.79	57.79	78.22
	1	featsize	T	18.70	42.84	49.74	50.56	58.39	79.71
	1	prefsize	T	18.70	42.99	50.54	50.95	58.85	79.71
	K	equal	F	25.91	45.09	52.23	52.26	59.52	87.15
	1	prefsize	T	2.54	11.08	14.56	14.78	18.24	30.25
	1	featsize	T	3.26	11.38	14.80	15.06	18.32	29.20
	1	dbl2nd	T	3.35	11.93	15.41	16.12	20.25	34.93
	1	bcs	T	3.35	12.21	16.15	16.38	20.04	34.93
	1	equal	T	4.66	12.98	16.96	17.21	20.74	43.03
	1	prefsize	F	3.41	13.17	17.47	17.65	21.86	34.70
	1	wcs	T	1.04	13.73	17.49	18.07	22.22	43.03
	1	featsize	F	3.43	14.01	18.09	18.65	22.95	43.03
$\mathcal{P}^{10\times10}_{f.rnd}$	1	bcs	F	3.43	14.24	18.29	18.74	23.02	43.03
	K	equal	F	2.68	14.85	18.74	19.45	23.24	38.55
	1	dbl2nd	F	3.43	15.41	19.39	19.52	23.94	43.03
	1	dbl1st	T	3.14	14.96	18.74	19.67	23.68	48.13
	1	wcs	F	3.55	17.28	21.19	21.65	25.99	44.98
	1	opt	T	3.55	16.59	21.11	21.67	26.38	46.22
(default)	1	equal	F	3.55	17.85	22.36	22.56	27.24	43.60
	1	dbl1st	F	3.55	17.70	22.41	22.58	27.26	44.98
	1	opt	F	3.45	19.31	23.86	24.16	28.65	47.31

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

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

Alice

9 Feature Selection

From Chapter 5 there exists linear weights $\mathbf{w}$ for Eq. (2.12) found with evolutionary optimisation that achieve a lower deviation from optimality, ρ, than any of the preference models from Chapter 8 has been able to outperform.* This goes to show that the $d = 16$ features are 'enough' – meaning there is not a need for defining new ones just yet. However, the optimal weights for Eq. (5.1) were quite erratic (cf. Fig. 5.2). Perhaps the features from $\Phi^{\langle \text{CMA-ES} \rangle}$ are contradictory, and therefore not suitable for preference learning.

Furthermore, the SDRs we've inspected so-far are based on two job-attributes from Table 2.2, namely: *i)* φ_1 for SPT and LPT, and *ii)* φ_7 for LWR and MWR, by choosing the lowest value for SPT and LWR, and highest value for LPT and MWR, i.e., the extremal values for those attributes. These SDRs achieve a remarkably low ρ, suggesting maybe not that many additional features are needed to achieve a competitive result.

For this study we will consider all combinations of feature attributes using either one, two, three or all $d = 16$ of them, for a grand total of:

$$\binom{d}{1} + \binom{d}{2} + \binom{d}{3} + \binom{d}{d} = 697 \tag{9.1}$$

The reason for such a limiting number of active features, are due to the fact we want to keep the

*Although $\Psi_p^{\text{ES}\cdot C_{\max}}$ can be statistically insignificant to its original trajectory iff stepwise sampling is adjusted w.r.t. its preference set (cf. Section 8.6).

models simple enough for improved model interpretability. Furthermore, we will continue to use our baseline preference set from Papers I and IV to VI, namely Ψ_p^{OPT}.

For each feature combination, a linear preference model is created, where Ψ_p is limited to the predetermined feature combination. This was done for all $\mathcal{P}_{\text{train}}^{10\times 10}$ in Table 3.2 (save for job and machine variation), each consisting of $N_{\text{train}} = 300$ problem instances. Moreover, in order to report the validation accuracy, 20% (i.e. $N_{\text{val}} = 60$) of the training set was set aside for reporting the accuracy.

9.1 Validation accuracy

As the preference set Ψ_p has both preference pairs belonging to optimal ranking, and subsequent rankings, it is not of primary importance to classify *all* rankings correctly, just the optimal ones. Therefore, instead of reporting the validation accuracy based on the classification problem of the correctly labelling the entire problem set Ψ_p, it's opted that the validation accuracy is obtained using Eq. (7.1), namely the probability of choosing an optimal decision given the resulting linear weights.* However, in this context, the mean throughout the dispatching process is reported, i.e.,

$$\overline{\xi_\pi^\star} = \frac{1}{K}\sum_{k=1}^{K} \xi_\pi^\star(k) \tag{9.2}$$

Figure 9.1 shows the difference between the two measures of reporting validation accuracy.

Validation accuracy based on Eq. (9.2) only takes into consideration the likelihood of choosing the optimal move at each time step. However, the classification accuracy is also trying to correctly distinguish all subsequent rankings in addition of choosing the optimal move, as expected that measure is considerably lower.

9.2 Pareto front

When training the learning model one wants to keep the validation accuracy high, as that would imply a higher likelihood of making optimal decisions, which would in turn translate into a low final makespan. To test the validity of this assumptions, each of the 697 models is run on the preference set, and its mean ρ is reported against its corresponding validation accuracy Eq. (9.2) in Fig. 9.2. The models are colour-coded w.r.t. the number of active features, and a line is drawn through its Pareto front. Those solutions are labelled with their corresponding model ID. Moreover, the Pareto front over all 697 models, irrespective of active feature count, is denoted with triangles. Their values are reported in Table 9.1, where the best objective is given in boldface.

*Due to superabundant number of models then calculating the *preferable* ξ_π from Eq. (7.3) is not viable.

Figure 9.1: Various methods of reporting validation accuracy for preference learning

Figure 9.2: Scatter plot for validation accuracy (%) against its corresponding mean expected ρ (%) for all 697 linear models from Eq. (9.1). Pareto fronts are for each active feature count, and labelled with their model ID. Moreover, actual Pareto front over all models is marked with triangles.

Table 9.1: Mean validation accuracy and mean expected deviation from optimality, ρ, for all CDR models on the Pareto front from Fig. 9.2.

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

9.3 Inspecting weight contribution to end-result

Figure 9.3 depicts $\mathbf{w}$ for all of the learned CDR models reported in Table 9.1. The weights have been normalised for clarity purposes, such that it is scaled to $\|\mathbf{w}\| = 1$, thereby giving each feature their proportional contribution to the preference I_j^π defined by Eq. (2.11). These weights will now be explored further, along with testing whether models are statistically significant to one another, using a Kolmogorov-Smirnov test with $\alpha = 0.05$.

For $\mathcal{P}_{j.rnd}^{10\times10}$ there is no statistical difference between models (2.69, 3.355, 3.358, 3.524), w.r.t. ρ and the latter three w.r.t. accuracy. These models are therefore equivalently 'best' for the problem space. As Fig. 9.3 shows, φ_3, φ_7 and φ_{11} are similar in value, and in the case of 3.358, then φ_9 has similar contribution as φ_3 for the other models. Which, as standalone models are 1.6 and 1.3, respectively, and yield equivalent mean ρ and accuracy. As these features often coincide in job-shop it is justifiable to use only either one, as they contains the same information as its counterpart.* Most likely, the equivalence of these features is indicating that the schedules are rarely able to dispatch in earlier slots, i.e., $\varphi_3 \approx \varphi_9$.

In addition, (2.111, 3.300) and (16.1, 3.355) are statistically insignificant w.r.t. validation accuracy for $\mathcal{P}_{j.rnd}^{10\times10}$. However, they have considerable performance difference w.r.t. ρ ($\Delta\rho \approx 18\%$). So even looking at mean stepwise optimality from Eq. (9.2) by itself is very fickle, because slight variations can be quite dramatic to the end result.

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

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

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

*Note, $\varphi_3 \leq \varphi_9$, where $\varphi_3 = \varphi_9$ when J_j is the latest job on M_a, otherwise J_j is placed in a previously created slot on M_a.

For both $\mathcal{P}^{10\times10}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rndn}$, model 1.13 is on the Pareto front. The model corresponds to feature φ_7, and in both cases has a weight strictly greater than zero (cf. Fig. 9.3). Revisiting Eq. (2.14), we observe that this implies the learning model was able to discover MWR as one of the Pareto solutions, and as is expected, there is no statistical difference to between 1.13 and MWR.

As one can see from Fig. 9.2, adding additional features to express the linear model boosts performance in both validation accuracy and expected mean for ρ, i.e., the Pareto fronts are cascading towards more desirable outcome with higher number of active features. However, there is a cut-off point for such improvement, as using all features is generally considerably worse off due to overfitting of classifying the preference set.

9.4 Evolution of validation accuracy

Let's inspect the models corresponding to the minimum mean ρ and highest mean validation accuracy, highlighted in Table 9.1 and inspect the evolution of $\xi^{\star}_{\pi}(k)$ for those models in Fig. 9.4, again using probability of randomly guessing an optimal move (i.e. $\xi^{\star}_{\text{RND}}$ from Fig. 7.2) as a benchmark. As one can see for both $\mathcal{P}^{10\times10}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rndn}$, despite having a higher mean validation accuracy overall, the probabilities vary significantly. A lower mean ρ is obtained when the validation accuracy is gradually increasing over time, and especially during the last phase of the scheduling.* Revisiting Fig. 7.3b, this trend indicates that it's likelier for the resulting makespan to be considerably worse off if suboptimal moves are made at later stages, than at earlier stages. Therefore, it's imperative to make the 'best' decision at the 'right' moment, not just look at the overall mean performance. Hence, the measure of validation accuracy as discussed in Section 9.1 should take into consideration the impact a suboptimal move yields on a step-by-step basis, e.g., Eq. (7.1) should be weighted w.r.t. a curve such as Eq. (7.2).

9.5 Comparison to other approaches

Main statistics for the best models from Eq. (9.1) are reported in Table 9.2. Going back to the original SDRs discussed in Section 2.4 along with the CMA-ES obtained weights for Eq. (5.1a) and compare then against the best CDR models, a box-plot for ρ is depicted in Fig. 9.5. Firstly, there is a statistical difference between all models, and the CDR model corresponding to minimum mean ρ value, is the clear winner for $\mathcal{P}^{10\times10}_{j.rndn}$ and $\mathcal{P}^{10\times10}_{f.rnd}$. On the other hand, for $\mathcal{P}^{10\times10}_{j.rnd}$ it loses by $\Delta\rho \approx +2.6\%$ to ES.$C_{\max}$ optimisation. In all cases the there is substantial perfor-

*It's almost illegible to notice this shift directly from Fig. 9.4, as the difference between the two best models is oscillating up to only 3% at any given step. In fact $\mathcal{P}^{10\times10}_{j.rndn}$ has the most clear difference w.r.t. classification accuracy of indicating when a minimum ρ model excels at choosing the preferred move.

Figure 9.3: Normalised weights for CDR models from Table 9.1, models are grouped w.r.t. its dimensionality, d. Note, a triangle indicates a solution on the Pareto front. Furthermore, bottom annotation indicates their objectives $\{\overline{\rho}, \overline{\xi^{\star}_{\text{CDR}}}\}$.

Figure 9.4: Probability of choosing optimal move for models corresponding to highest mean validation accuracy and lowest mean deviation from optimality, ρ, compared to the baseline of probability of choosing an optimal move at random (dashed)

Figure 9.5: Box-plot for deviation from optimality, ρ, for the best CDR preference models (cf. Table 9.1) and compared against their best reference model.

mance boost w.r.t. SDRs: *i)* -8.2% from $\mathcal{P}^{10\times10}_{j.rnd}$'s MWR; *ii)* -5.9% from $\mathcal{P}^{10\times10}_{j.rndn}$'s MWR, and *iii)* -6.5% from $\mathcal{P}^{10\times10}_{f.rnd}$'s LWR). However, the best model w.r.t. maximum validation accuracy, then the CDR model shows a lacklustre performance. In some cases it's better off, e.g., compared to $\mathcal{P}^{10\times10}_{f.rnd}$'s LWR ($\Delta\rho \approx -1\%$), yet for job-shop it doesn't surpass the performance of MWR. This implies, the learning model is over-fitting the training data. Results hold for the test set.

Table 9.2: Main statistics for $\mathcal{P}^{10\times10}_{\text{train}}$ deviation from optimality, ρ, using models from Eq. (9.1) corresponding to lowest mean ρ or highest accuracy in Eq. (9.2)

	NrFeat.	Model	Best	Set*	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
$\mathcal{P}^{10\times10}_{j.rnd}$		3.524	ρ	train	2.29	9.70	13.20	13.27	15.90	27.70
		3.524	ρ	test	1.50	10.20	13.47	13.73	17.24	35.33
		16.1	$\overline{\xi^\star}$	train	7.87	23.34	29.30	30.73	36.47	61.45
		16.1	$\overline{\xi^\star}$	test	8.31	23.88	30.32	31.46	37.70	67.24
$\mathcal{P}^{10\times10}_{j.rndn}$		3.281	ρ	train	0.56	10.33	12.82	12.96	15.78	25.52
		3.281	ρ	test	2.87	11.02	13.40	13.68	16.87	26.05
		16.1	$\overline{\xi^\star}$	train	18.02	33.89	40.53	41.51	48.15	75.30
		16.1	$\overline{\xi^\star}$	test	14.06	32.85	39.73	40.34	47.46	76.69
$\mathcal{P}^{10\times10}_{f.rnd}$		3.539	ρ	train	1.53	13.51	16.14	16.70	20.29	35.24
		3.539	ρ	test	3.92	13.38	16.93	17.42	21.02	32.46
		16.1	$\overline{\xi^\star}$	train	3.55	17.85	22.36	22.56	27.24	43.60
		16.1	$\overline{\xi^\star}$	test	3.33	17.37	21.62	22.46	27.32	43.70

*Here the full training set is used, $N_{\text{train}} = 300$, hence the slight change in mean ρ compared to Table 9.1 which was only based on the validation set (the latter $N_{\text{val}} = 60$)

9.6 Conclusions

When training the learning model, it's not sufficient to only optimise w.r.t. highest mean validation accuracy defined in Eq. (9.2). As Section 9.4 showed, there is a trade-off between making the over-all best decisions versus making the right decision on crucial time points in the scheduling process, as Fig. 7.3 clearly illustrated. It is for this reason, traditional feature selection such as *add1* and *drop1* were unsuccessful in preliminary experiments, and thus resorting to exhaustively searching all feature combinations. This also opens of the question of how should validation accuracy be measured? Since the model is based on learning preferences, both based on optimal versus suboptimal, and then varying degrees of sub-optimality. As we are only looking at the ranks in a 'black and white' fashion, such that the makespans need to be strictly greater to belong to a higher rank, then it can be argued that some ranks should be grouped together if their makespans are sufficiently close. This would simplify the training set, making it (presumably) less of contradictions and more appropriate for preference learning. Or simply the validation accuracy in Eq. (9.2) could be weighted w.r.t. the difference in makespan. During the dispatching process, there are some pivotal times which need to be especially taken care off. Figure 7.3 showed how making suboptimal decisions were more of a factor during the later stages, whereas for flow-shop the case was exact opposite.

Note, from Section 8.6 it's possible to sample w.r.t. stepwise bias such that it gives preference pairs that are more relevant to its end-performance. In other words, weighing the measure from Eq. (9.2) via the sampling strategy. Presumably, if such adjusted bias were applied to this study, then greater performance boost could be achieved.

10

Imitation Learning

Despite the abundance of information gathered by following expert policy, the knowledge obtained is not enough by itself. Since the learning model is not perfect, it is bound to make a mistake eventually. When it does, the model is in uncharted territory as there is no certainty that the samples already collected are able to explain the current situation. For this we propose investigating features from suboptimal trajectories as well, since the future observations depend on previous predictions. A straight forward approach would be to inspect the trajectories of promising SDRs or CDRs, this was done in Section 8.5 with good results. The reasoning behind it was that they would be beneficial for learning, as they might help the model to escape from local minima once off the coveted optimal path. By simply adding training data obtained by following the trajectories of well-known SDRs, their aggregated training set yielded better models with lower deviation from optimality, ρ. However, this was done in a fairly ad-hoc manner, which we'd like to automate even further. Therefore, it would be worth while to try out active imitation learning by Ross and Bagnell (2010), Ross et al. (2011), such that the learned policy following an optimal trajectory is used to collect training data, and the learned model is updated. This can be done over several iterations, the benefit being that the states which are likely to occur in practice are investigated, and as such used to dissuade the model from making poor choices. Alas, this comes at great computational cost due to the substantial amounts of states that need to be optimised for their correct labelling. Making it only practical for job-shop of relatively low dimension, or only a few iterations.

The preference model presented in Chapters 8 and 9 are comprised of collecting snap-shots of the state space by following a fixed policy, and verifying the resulting optimal makespan from each possible state. This can be looked at as *imitation learning* (IL), since we're trying to imitate the fixed policy via preference learning.

Up until now, the training data from Chapters 8 and 9 has been created from optimal *or* suboptimal solutions of randomly generated problem instances, i.e., traditional *passive* imitation learning. As JSP is a sequential decision making process, errors are bound to emerge. Due to compound effect of making suboptimal dispatches, the model leads the schedule astray from learned state-spaces, resulting in the new input being foreign to the learned model.

Inspired by the work of Ross and Bagnell (2010), Ross et al. (2011), the methodology of generating training data will now be such that it will iteratively improve upon the model, such that the state-spaces learned will be representative of the state-spaces the eventual model would likely encounter, known as DAgger for imitation learning. Thereby, eliminating the ad-hoc nature of choosing trajectories to learn, by rather letting the model lead its own way in a self-perpetuating manner until it converges.

Performance boost

In order to boost training accuracy even further, two strategies will be explored:

Boost.1 increasing number of preferences used in training (i.e. varying $l_{\max} \leq |\Psi|$),

Boost.2 introducing more problem instances (denoted EXT in experimental setting).

Note, Boost.1 will be addressed in Section 10.2. However, Boost.2 strategy will be explored in Sections 10.1 and 10.2. Summary of N_{train} is given in Table 10.1.

10.1 Passive imitation learning

Using the terms from game-theory used in Cesa-Bianchi and Lugosi (2006), then our problem is a basic version of the sequential prediction problem where the predictor (or forecaster), π, observes each element of a sequence $\boldsymbol{\chi}$ of jobs, where at each time step $k \in \{1, \ldots, K\}$, before the k-th job of the sequence is revealed, the predictor guesses its value χ_k on the basis of the previous $k-1$ observations.

10.1.1 Prediction with expert advice

Before going further, let's formalise following an expert policy, Φ^{OPT}, from Section 8.5. Let's assume we know the expert policy $\pi^{\star}$, which we can query what is the optimal choice of $\chi_k = j^*$

at any given time step k. Now we can use Eq. (2.13) to back-propagate the relationship between post-decision states and $\hat{\pi}$ with preference learning via our collected feature set, denoted Φ^{OPT}, i.e., we collect the features set corresponding following optimal tasks J_{j^*} from $\pi^\star$ in Algorithm 1. This baseline trajectory sampling for adding features to the feature set is a pure strategy where at each dispatch, an optimal task was chosen. This was originally introduced in Paper I and explored further in Paper VI.

By querying the expert policy, $\pi_\star$, the ranking of the job-list, $\mathcal{L}$, is determined by Eq. (8.4) such that r_i is preferable to r_{i+1}. In our study, we know the rank is proportional to its optimum makespan, hence the optimal job-list is the following,

$$\mathcal{O}^{(k)} = \left\{ r_i \ : \ r_i \propto \min_{J_j \in \mathcal{L}^{(k)}} C_{\max}^{\pi_\star}(\boldsymbol{x}^j) \right\} \tag{10.1}$$

found by solving the current partial schedule to optimality. When $|\mathcal{O}^{(k)}| > 1$, there can be several trajectories worth exploring. However, only one is chosen at random. This is deemed sufficient as the number of problem instances, N_{train}, is relatively large.

10.1.2 Follow the perturbed leader

By allowing a predictor to randomise it's possible to achieve improved performance (Cesa-Bianchi and Lugosi, 2006, Hannan, 1957), which is the inspiration for our new strategy, where we follow the *Perturbed Leader*, denoted OPTε. Its pseudo code is given in Algorithm 2 and describes how the expert policy (i.e. optimal trajectory) from Section 10.1.1 is subtly 'perturbed' with $\varepsilon = 10\%$ likelihood, by choosing a job corresponding to the second best $C_{\max}$ instead of a optimal one with some small probability.

10.1.3 Experimental study

To address Boost.2 for the conventional Φ^{OPT} trajectory the extended training set was simply obtained by iterating over more examples, given in Table 10.1.

Figure 10.1 depicts a box-plot for deviation from optimality, ρ, using $\mathcal{P}_{\text{train}}^{6\times5}$ and $\mathcal{P}_{j.rnd}^{10\times10}$. Main statistics are reported in Table 10.2. Results show that following the perturbed leader significantly improved following the expert policy for $\mathcal{P}_{j.rndn}^{6\times5}$, $\mathcal{P}_{j.rnd,J_1}^{6\times5}$, $\mathcal{P}_{j.rnd,M_1}^{6\times5}$, $\mathcal{P}_{f.rndn}^{6\times5}$ and $\mathcal{P}_{f.mc}^{6\times5}$. Other $\mathcal{P}_{\text{train}}^{6\times5}$ problem spaces and $\mathcal{P}_{j.rnd}^{10\times10}$ had insignificant performance boost.

Results showed that the expert policy is a promising starting point. However, since job-shop is a sequential prediction problem, all future observations are dependent on previous operations. Therefore, learning sampled states that correspond only to optimal or near-optimal schedules isn't of much use when the preference model has diverged too far. This is due to the learner's

Algorithm 2 Pseudo code for choosing job J_{j^*} following a perturbed leader.

Require: Ranking $r_1 \succ r_2 \succ \cdots > r_{n'}$ $(n' \leq n)$ of the job-list, $\mathcal{L}$ ▷ query $\pi_\star$

```
procedure PerturbedLeader(L, π_⋆)
    ε ← 0.1                                          ▷ likelihood factor
    p ← U(0,1) ∈ [0,1]                               ▷ uniform probability
    O ← {j ∈ L : r_j = r_1}                          ▷ optimal job-list
    S ← {j ∈ L : r_j > r_1}                          ▷ suboptimal job-list
    if p < ε and n' > 1 then
        return j* ∈ {j ∈ S : r_j = r_2}              ▷ any second best job
    else
        return j* ∈ O                                ▷ any optimal job
    end if
end procedure
```

predictions affects future input observations during its execution, which violates the crucial i.i.d. assumptions of the learning approach, and ignoring this interaction leads to poor performance. In fact, Ross and Bagnell (2010) proves, that assuming the model has a training error of ε, then the total compound error (for all K dispatches) the classifier induces itself grows quadratically, $\mathcal{O}(\varepsilon K^2)$, for the entire schedule, rather than having linear loss, $\mathcal{O}(\varepsilon K)$, if it were i.i.d.

10.2 Active imitation learning

To amend performance from Φ^{OPT}-based models, suboptimal state-spaces were explored in Paper V by inspecting the features from successful SDRs, $\Phi^{\langle\text{SDR}\rangle}$, by passively observing a full execution of following the task chosen by the corresponding SDR. This required some trial-and-error as the experiments showed that features obtained by SDR trajectories were not equally useful for learning.

To automate this process, inspiration from *active* imitation learning presented in Ross et al. (2011) is sought, called *Dataset Aggregation* (DAgger) method, which addresses a no-regret algorithm in an on-line learning setting. The novel meta-algorithm for IL learns a deterministic policy guaranteed to perform well under its induced distribution of states. The method is closely related to Follow-the-leader (cf. Section 10.1.2), however, with a more sophisticated leverage to the expert policy. In short, it entails the model π_i that queries an expert policy (same as in Section 10.1.1), $\pi_\star$, its trying to mimic, but also ensuring the learned model updates itself in an iterative fashion, until it converges. The benefit of this approach is that the states that are likely to occur in practice are also investigated and as such used to dissuade the model from making poor choices. In fact, the method queries the expert about the desired action at individual post-decision states which are both based on past queries, and the learner's interaction with the *current*

Table 10.1: Number of problem instances, $\mathcal{P} = \{\mathbf{p}_i, \boldsymbol{\sigma}_i\}_{i=1}^{N}$, explored for the collection of training set, Φ, in experimental setting.

$n \times m$	$N_{\text{train}}^{\langle\text{default}\rangle}$	$N_{\text{train,EXT}}^{\text{OPT}}$	$N_{\text{train,EXT}}^{\text{DA}i}$
6×5	500	5000	$500(i+1)$
10×10	300	1000	$300(i+1)$

Table 10.2: Main statistics for $\mathcal{P}_{\text{train}}^{6\times5}$ and $\mathcal{P}_{j.rnd}^{10\times10}$ deviation from optimality, ρ, following either expert policy or perturbed leader

	Set	Track	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
$\mathcal{P}_{j.rnd}^{6\times5}$	train	OPT	0.00	8.89	15.38	16.46	22.62	51.23
	train	OPT, EXT	0.00	9.57	15.37	16.32	21.95	57.23
	train	OPTε	0.00	7.34	12.92	**14.12**	20.16	55.78
$\mathcal{P}_{j.rndn}^{6\times5}$	train	OPT	0.00	12.58	20.80	20.84	27.46	58.61
	train	OPTε	0.00	3.60	11.34	**10.43**	13.48	29.01
$\mathcal{P}_{j.rnd,J_1}^{6\times5}$	train	OPT	0.00	6.38	13.06	14.91	21.74	58.61
	train	OPTε	0.00	3.89	8.97	**11.51**	16.88	62.62
$\mathcal{P}_{j.rnd,M_1}^{6\times5}$	train	OPT	0.00	9.95	17.95	20.09	28.23	79.19
	train	OPTε	0.00	4.08	11.40	**12.85**	19.09	50.19
$\mathcal{P}_{f.rnd}^{6\times5}$	train	OPT	0.00	4.46	8.64	**9.68**	13.81	38.53
	train	OPTε	0.00	4.58	9.07	10.14	14.59	31.94
$\mathcal{P}_{f.rndn}^{6\times5}$	train	OPT	0.00	0.78	1.39	1.54	2.13	9.72
	train	OPTε	0.00	0.59	1.19	**1.38**	1.81	10.32
$\mathcal{P}_{f.jc}^{6\times5}$	train	OPT	0.00	0.14	0.37	**0.53**	0.74	4.55
	train	OPTε	0.00	0.14	0.46	0.69	0.94	11.81
$\mathcal{P}_{f.mc}^{6\times5}$	train	OPT	0.00	0.25	1.07	3.24	5.42	27.18
	train	OPTε	0.00	0.22	0.82	**2.12**	2.26	27.18
$\mathcal{P}_{f.mxc}^{6\times5}$	train	OPT	0.00	0.28	0.99	**1.43**	2.17	7.77
	train	OPTε	0.00	0.28	0.98	**1.43**	2.17	7.77
$\mathcal{P}_{j.rnd}^{10\times10}$	train	OPT	7.87	23.34	29.30	30.73	36.47	61.45
	test	OPT	8.31	23.88	30.32	31.46	37.70	67.24
	train	OPT, EXT	6.61	31.82	39.88	40.93	48.69	93.40
	test	OPT, EXT	9.50	31.94	39.77	40.84	48.79	90.05
	train	OPTε	9.50	23.01	29.00	**29.94**	35.92	58.65
	test	OPTε	8.53	24.02	29.52	**30.03**	34.91	62.29

Figure 10.1: Box-plots for deviation from optimality, ρ, following either expert policy or perturbed leader for $\mathcal{P}^{6\times5}_{\text{train}}$ and $\mathcal{P}^{10\times10}_{j.rnd}$

environment.

DAgger has been proven successful on a variety of benchmarks, such as: the video games Super Tux Kart and Super Mario Bros. or handwriting recognition – in all cases greatly improving traditional supervised imitation learning approaches (Ross et al., 2011), and real-world applications, e.g. autonomous navigation for large unmanned aerial vehicles (Ross et al., 2013). To illustrate the effectiveness of DAgger, the Super Mario Bros. experiment gives a very simple and informative understanding of the benefits of the algorithm. In short, Super Mario Bros. is a platform game where the protagonist, Mario, must move across the stage without being hit by enemies or falling through gaps within a certain time limit. One of the reasons the supervised approaches failed, were due to Mario getting stuck up against an obstacle, instead of jumping over it. However, the expert would always jump over them at a greater distance beforehand, and therefore the learned controller would not know of these scenarios. With iterative methods, Mario would encounter these problematic situations and eventually learn how to get himself unstuck.

The policy of imitation learning at iteration $i > 0$ is a mixed strategy given as follows,

$$\pi_i = \beta_i \pi_\star + (1 - \beta_i)\hat{\pi}_{i-1} \tag{10.2}$$

where $\pi_\star$ is the expert policy and $\hat{\pi}_{i-1}$ is the learned model from the previous iteration. Note, for the initial iteration, $i = 0$, a pure strategy of $\pi_\star$ is followed. Hence, $\hat{\pi}_0$ corresponds to the preference model from Section 10.1.1 (i.e. $\Phi^{\mathrm{IL}0} = \Phi^{\mathrm{OPT}}$).

Equation (10.2) shows that β controls the probability distribution of querying the expert policy $\pi_\star$ instead of the previous imitation model, $\hat{\pi}_{i-1}$. The only requirement for $\{\beta_i\}_i^\infty$ according to Ross et al. (2011) is that $\lim_{T\to\infty} \frac{1}{T}\sum_{i=0}^{T} \beta_i = 0$ to guarantee finding a policy $\hat{\pi}_i$ that achieves ε surrogate loss under its own state distribution limit.

Algorithm 3 explains the pseudo code for how to collect partial training set, $\Phi^{\mathrm{IL}i}$ for i-th iteration of imitation learning. Subsequently, the resulting preference model, $\hat{\pi}_i$, learns on the aggregated datasets from all previous iterations, namely,

$$\Phi^{\mathrm{DA}i} = \bigcup_{i'=0}^{i} \Phi^{\mathrm{IL}i'} \tag{10.3}$$

and its update procedure is detailed in Algorithm 4.

10.2.1 DAgger parameters

Due to time constraints, then maximum number of iterations is $T = 7$ are inspected for $\mathcal{P}_{j.rnd}^{6\times5}$ and $\mathcal{P}_{j.rnd}^{10\times10}$ using Bias.1. In addition, there will be three mixed strategies for $\{\beta_i\}_{i=0}^{T}$ in Eq. (10.2)

Algorithm 3 Pseudo code for choosing job J_{j^*} using imitation learning (dependent on iteration i) to collect training set $\Phi^{\mathrm{IL}i}$; either by following optimal trajectory, $\pi_\star$, or preference model from previous iterations, $\hat{\pi}_{i-1}$.

Require: $i \geq 0$ ▷ imitation learning is *passive* if $i = 0$ and *active* otherwise
Require: Ranking $r_1 \succ r_2 \succ \cdots > r_{n'}$ $(n' \leq n)$ of the job-list, $\mathcal{L}$ ▷ query $\pi_\star$

```
procedure IL(i, π̂_{i−1}, π_⋆)
    p ← U(0, 1) ∈ [0, 1]                                   ▷ uniform probability
    if i > 0 then
        if unsupervised then
            β_i ← 0                                        ▷ always apply imitiation
        else if decreasing supervision then
            β_i ← 0.5^i                                    ▷ liklier to choose imitation with each iteration
        else (fixed supervision)
            β_i ← 0.5                                      ▷ equally likely to choose optimal vs. imitation
        end if
    else (fixed supervision)
        β_i ← 1                                            ▷ always follow expert policy (i.e. optimal)
    end if
    if p > β_i then
        return j* ← argmax_{j∈L}{I_j^{π̂_{i−1}}}           ▷ best job based on π̂_{i−1}, cf. Algorithm 1
    else
        O ← {j ∈ L : r_j = r_1}                            ▷ optimal job-list
        return j* ∈ O                                      ▷ any optimal job
    end if
end procedure
```

Algorithm 4 DAgger: Dataset Aggregation for JSP

Require: $T \geq 1$

```
procedure DAgger(π_⋆, Φ^{IL0}, T)
    π̂_0 ← Train(Φ^{IL0})                                   ▷ initial model, Section 10.1.1 iff Φ^{IL0} = Φ^{OPT}
    for i ← 1 to T do                                      ▷ at each imitation learning iteration
        Let π_i = β_i π_⋆ + (1 − β_i) π̂_{i−1}              ▷ Eq. (10.2)
        Sample a K-solution using π_i                      ▷ cf. Algorithm 3: IL(i, π̂_{i−1}, π_⋆)
        Φ^{ILi} = {(s, π_⋆(s))}                            ▷ visited states by π_i and actions given by expert
        Φ^{DAi} ← Φ^{DAi−1} ∪ Φ^{ILi}                      ▷ aggregate datasets, cf. Eq. (10.3)
        π̂_{i+1} ← Train(Φ^{DAi})                           ▷ preference model from Chapter 8
    end for
    return best π̂_i on validation                          ▷ best preference model
end procedure
```

considered:

DAβ.1 fixed supervision with $\beta_i = 0.5$ save for $\beta_0 = 1$,

DAβ.2 decreasing supervision with $\beta_i = 0.5^i$,

DAβ.3 unsupervised with $\beta_i = I(i = 0)$, where I is the indicator function.*

Note, DAβ.2 starts as DAβ.1 and decays exponentially towards DAβ.3. Moreover, DAβ.3 is a simple parameter-free version of the DAgger algorithm and often performs best in practice (Ross et al., 2011).

10.2.2 Experimental study

To address Boost.1, then Ψ_p^{DA7} for $\mathcal{P}_{j.rnd}^{10\times10}$ was trained with varying size $l_{\max}$, from 50,000 to its full size 3,626,260 with 50,000 interval. The default value for $l_{\max}$ given in Eq. (8.3) is denoted in boldface. There was no statistical significance in boost of performance, hence $l_{\max}$ is kept unchanged.

Regarding Boost.2 for DAgger trajectories the extended set consisted of each iteration encountering N_{train} *new* problem instances. For a grand total of

$$N_{\text{train, EXT}}^{\text{DA}i} = N_{\text{train}} \cdot (i + 1) \tag{10.4}$$

problem instances explored for the aggregated extended training set used for the learning model at iteration i. This way, we use the extended training data sparingly, as labelling for each problem instances is computationally intensive. As a result, the computational budget for DAgger is the same regardless whether there are new problem instances used or not, i.e., $|\Phi^{\text{DA}i}| \approx |\Phi_{\text{EXT}}^{\text{DA}i}|$.

A box-plot of deviation from optimality, ρ, is given in Fig. 10.3. Notice that if DAgger continually uses the same problem instances, then not much is gained after the first iteration, as performance stagnates quickly thereafter. This is due to the fact that there is not enough variance between $\Phi^{\text{IL}i}$, hence the aggregated feature set $\Phi^{\text{DA}i}$ and $\Phi^{\text{DA}(i-1)}$ is only slightly perturbed with each iterations. Which from Section 10.1.3 we saw extended aggregation was not a successful modification for the expert policy. Although, it's noted that by introducing sub-optimal state spaces the preference model is not as drastically bad as the extended optimal policy, even though $|\Phi_{\text{EXT}}^{\text{DA}i}| \gg |\Phi_{\text{EXT}}^{\text{OPT}}|$ for $\mathcal{P}_{j.rnd}^{10\times10}$ after $i > 2$. This goes to show that 'too much' data is no longer a bad influence. But rather, when using new problem instances at each iterations, the feature set becomes varied enough that situations arise that can be learned to achieve a better represented classification problem which yields a lower mean deviation from optimality, ρ.

*$\beta_0 = 1$ and $\beta_i = 0, \forall i > 0$.

Figure 10.2: Box-plot for deviation from optimality, ρ, where preference set is sampled to various sized $|\Psi_p^{\text{DA}_7}| = l_{\max}$ using $\mathcal{P}_{j.rnd}^{10\times10}$

Regarding different strategies for β_i values in Eq. (10.2), then all strategies improved the first iteration (i.e. DA1 improves OPT). After that then the choice of β_i starts to matter. For DAβ.1 there was no further improvement over all $T = 7$ iterations using $\mathcal{P}_{j.rnd}^{6\times5}$ (therefore not considered for $\mathcal{P}_{j.rnd}^{10\times10}$). For DA$\beta$.2 then significant improvement was achieved at $i = 5$ using $\mathcal{P}_{j.rnd}^{6\times5}$. However, for $\mathcal{P}_{j.rnd}^{10\times10}$ then no significant improvement was after $i > 1$. On the other hand, DAβ.3 had some unexpected performance for $\mathcal{P}_{j.rnd}^{6\times5}$ as it deteriorated for $i \in \{2, 3\}$, but at $i = 4$ it got back on track before stagnating. Of all the suggested β_i strategies then DAβ.3 managed to get the best overall performance, and therefore Boost.2 was also applied to that approach.

For other $\mathcal{P}_{\text{train}}^{6\times5}$ problem spaces* then there was not much difference in DAβ.2 and DAβ.3, although the latter was slightly better for $\mathcal{P}_{f.rnd}^{6\times5}$. Note, for flow-shop problem spaces then DAgger was not fruitful, as either the iterations were statistically insignificant from the model obtained from the expert policy, or performance slightly downgraded with each iteration. Although, it's noted that those experiments were done with reusing the same N_{train} problem spaces over and over again (i.e. not applying Boost.2).

The best $\{\beta_i\}_{i=1}^{T}$ configuration for $\mathcal{P}_{\text{train}}^{6\times5}$ and $\mathcal{P}_{j.rnd}^{10\times10}$ was DAβ.3 using an extended data set

*Consult Shiny application: Preference models > Imitation Learning

Figure 10.3: Box-plots for deviation from optimality, ρ, using active imitation learning for $\mathcal{P}^{6\times5}_{j.rnd}$ and $\mathcal{P}^{10\times10}_{j.rnd}$ using equal re-sampling (i.e. Bias.1)

Figure 10.4: Box plot for $\mathcal{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using either expert policy, DAgger or following perturbed leader strategies.

(i.e. Boost.2). Therefore, for $\mathcal{P}^{10\times10}_{j.rnd}$ that configuration will also be tried for adjusted Bias.8, however, iterations were stopped as soon as performance downgrades. Box-plot for deviation from optimality, ρ, is depicted in Fig. 10.4, and main statistics are given in Table 10.3. Both adjusted Bias.8 and the corresponding Bias.1 from Fig. 10.3 are shown together. We notice that with each iteration DAgger improves: *i)* for Bias.1 with Boost.2 then $i = 1$ starts with increasing $\Delta\rho \approx +1.39\%$. However, after that first iteration there is a performance boost of $\Delta\rho \approx -15.11\%$ after $i = 2$ and after that $\Delta\rho \approx -1.31\%$ until the final iteration $T = 7$, and on the other hand*ii)* when using adjusted Bias.8, only one iteration is needed, as $\Delta\rho \approx -11.68$ for $i = 1$, and after that it stagnates with $\Delta\rho \approx +0.55\%$ for $i = 2$ (therefore $i = 3$ was not run).

10.3 Summary of imitation learning experimental studies

A summary of $\mathcal{P}^{10\times10}_{j.rnd}$ best passive and active imitation learning models w.r.t. deviation from optimality, ρ, from Sections 10.1.3 and 10.2.2, respectively, are illustrated in Fig. 10.4, and main statistics are given in Table 10.3. To summarise, the following trajectories are used: *i)* expert policy, trained on Φ^{OPT}; *ii)* perturbed leader, trained on $\Phi^{\text{OPT}\varepsilon}$, and *iii)*

Table 10.3: Main statistics for $\mathcal{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using either expert policy, imitation learning or following perturbed leader strategies.

π^*	i^{**}	Bias	Set	N_{train}	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
OPT	0	equal	train	300	7.87	23.34	29.30	30.73	36.47	61.45
OPT	0	equal	test	300	8.31	23.88	30.32	31.46	37.70	67.24
DA1	1	equal	train	600	9.47	24.92	31.51	32.12	37.96	66.29
DA1	1	equal	test	300	4.77	23.77	30.34	31.40	37.81	73.73
DA2	2	equal	train	900	2.36	12.82	16.65	17.01	21.06	39.25
DA2	2	equal	test	300	1.72	12.57	16.38	16.89	20.66	42.44
DA3	3	equal	train	1200	0.98	12.50	16.28	16.82	20.67	37.93
DA3	3	equal	test	300	0.26	12.32	16.01	16.52	20.22	41.62
DA4	4	equal	train	1500	3.04	11.83	15.29	15.92	19.66	40.70
DA4	4	equal	test	300	0.26	11.70	15.20	15.69	19.14	37.99
DA5	5	equal	train	1800	2.18	11.89	15.38	15.90	19.59	40.60
DA5	5	equal	test	300	0.26	11.78	15.20	15.75	19.24	40.73
DA6	6	equal	train	2100	2.28	11.90	15.30	15.89	19.62	40.70
DA6	6	equal	test	300	1.53	11.82	15.21	15.72	19.17	38.16
DA7	7	equal	train	2400	1.56	11.84	15.34	15.70	19.37	35.45
DA7	7	equal	test	300	1.41	11.72	15.20	15.72	19.11	39.86
OPT	0	adjdbl2nd	train	300	6.05	18.60	23.85	24.50	29.04	55.81
OPT	0	adjdbl2nd	test	300	5.56	19.16	24.24	25.19	30.42	55.52
DA1	1	adjdbl2nd	train	600	2.08	**9.44**	**12.30**	**12.82**	**15.67**	**29.63**
DA1	1	adjdbl2nd	test	300	**0.00**	**9.22**	**12.39**	**12.73**	**15.85**	35.17
DA2	2	adjdbl2nd	train	900	**0.93**	10.01	12.91	13.37	16.40	31.19
DA2	2	adjdbl2nd	test	300	0.39	9.84	13.13	13.44	16.62	**34.57**
OPTε	0	equal	train	300	4.52	21.31	27.63	28.04	33.69	63.74
OPTε	0	equal	test	300	8.54	22.03	27.26	27.94	33.02	60.38
OPTε	0	adjdbl2nd	train	300	4.64	13.63	17.56	18.07	21.66	36.25
OPTε	0	adjdbl2nd	test	700	1.91	13.18	16.48	16.89	20.28	35.60

*For DAgger $i = 0$ is the conventional expert policy (i.e. DA0 = OPT).
**If $i = 0$ then *passive* imitation learning. Otherwise, for $i > 0$ it is considered *active* imitation learning.

imitation learning, trained on $\Phi^{\text{DA}i}_{\text{EXT}}$ ($i \leq 7$ for Bias.1 and $i \leq 2$ for adjusted Bias.8). As a reference, ES.$C_{\max}$ model from optimising Eq. (5.1a) and MWR are shown on the far right of Fig. 10.4.

At first we see that the perturbed leader ever so-slightly improves the mean for ρ, rather than using the baseline expert policy. However, active imitation learning is by far the best improvement. With each iteration of DAgger, the models improve upon the previous one with each iteration.

In both cases, DAgger outperforms MWR: *i)* after $i = 7$ iterations by $\Delta\rho \approx -6.12\%$ for Bias.1 with Boost.2, and *ii)* after $i = 1$ iteration by $\Delta\rho \approx -9.31\%$ for adjusted Bias.8. Note, for Bias.1 without Boost.2, then DAgger was unsuccessful, and the aggregated data set downgrades the performance of the previous iterations, making it best to learn solely on the initial expert

policy for that model configuration.

When compared to ES.$C_{\max}$, then neither extended imitation learning approaches outperformed the direct optimisation. After $T = 7$ iterations for Bias.1 there was still $\Delta\rho \approx +5.13\%$ difference, and for $i = 2$ for adjusted Bias.8 that difference was almost halved, or $\Delta\rho \approx +2.8\%$ compared to optimising Eq. (5.1a).

10.4 Conclusions

This study showed, that when accumulating training data for supervised learning using DAgger, it's possible to automate its generation in such a way that the resulting model will be an accurate representative of the instances it will later come across. Or to phrase it in words of the Nobel-Prize-winning Irish playwright:

> "Imitation is not just the sincerest form of flattery
> – it's the sincerest form of learning."
>
> George Bernard Shaw

The experimental study in Section 10.2.2 showed that DAgger for job-shop is sensitive to choice of β_i in Eq. (10.2). The best configuration was an unsupervised approach (i.e. DAβ.3), which concurs to the findings of Ross et al. (2011).

Regarding using an extended data set (i.e. Boost.2), then it's not successful for the expert policy, as ρ increased approximately 10%. This could most likely be counter-acted by increasing $l_{\max}$ to reflect the additional examples. What is interesting though, is that Boost.2 is well suited for active imitation learning, using the same $l_{\max}$ as before. Note, the amount of problems used for $N^{\text{OPT}}_{\text{train, EXT}}$ is equivalent to $i = 9$ or $i = 2\frac{1}{3}$ iterations of extended DAgger for $\mathcal{P}^{6\times5}_{\text{train}}$ and $\mathcal{P}^{10\times10}_{j.rnd}$, respectively. The *new* varied data gives the aggregated feature set more information of what is important to learn in subsequent iterations, as those new states are more likely to be encountered 'in practice' rather than 'in theory.' Not only does the active imitation learning converge faster, it also consistently improves with each iterations if new instances are used.

The number of iterations needed depend on the quality of the model configurations. When using the baseline Bias.1 the imitation model was iterated for $T = 7$ iterations. Slowly improving with each iteration. However, when the preferred adjusted Bias.8 stepwise bias then after only two iterations a better performance was achieved. Alas, after the third iteration the model had already stagnated with slightly, yet insignificant, worse mean deviation from optimality, ρ.

Maximum Mean Discrepancy (MMD) imitation learning by Kim and Pineau (2013) is an iterative algorithm similar to DAgger. However, the expert policy is only queried when needed in order to reduce computational cost. This occurs when a metric of a new state is sufficiently large enough from a previously queried states (to ensure diversity of learned optimal states).

Moreover, in DAgger all data samples are equally important, irrespective of its iteration, which can require great number of iterations to learn how to recover from the mistakes of earlier policies. To address the naïvety of the data aggregation, MMD suggests only aggregating a new data point if it is sufficiently different to previously gathered states, *and* if the current policy has made a mistake. Additionally, there are multiple policies, each specialising in a particular region of the state space where previous policies made mistakes. Although MMD has better empirical performance (based on robot applications), it requires defining metrics, which in the case of job-shop is non-trivial (cf. Paper III and Chapter 4), and fine-tuning thresholds etc., whereas DAgger can be straightforwardly implemented, parameter-free and obtains competitive results, although with some computational overhead due to excess expert queries.

Main drawback of DAgger is that it quite aggressively quires the expert, making it impractical for some problems, especially if it involves human experts. To confront that, Judah et al. (2012) introduced *Reduction-based Active Imitation Learning* (RAIL), which involves a dynamic approach similar to DAgger, but more emphasis is used to minimise the expert's labelling effort. In fact, it's possible to circumvent querying the expert altogether and still have reasonable performance. If *Locally Optimal Learning to Search* (LOLS) by Chang et al. (2015) is applied, then it is possible to use imitation learning (similar to DAgger framework) when the reference policy is poor (i.e. $\pi_\star$ in Eq. (10.2) is suboptimal), 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.

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

11 Pilot Model

Roll-out algorithms, also known as Pilot Method (Bertsekas et al., 1997, 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 Rúnarsson et al. (2012). Continuing with that work, Geirsson (2012) compares several pilot heuristics, e.g., *Randomly Chosen Dispatch Rules* which is similar to $\{\varphi_i\}_{i=17}^{20}$ (but here one roll-out per fixed SDR). The motivation being, that a SDR-based roll-outs 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 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 $\{\varphi_i\}_{i=21}^{24}$, the weights $\mathbf{w}$ yield the deterministic pilot heuristic. Although in Geirsson's work, other statistics were used for guidance, e.g., quartile and octile.

*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*.

Remark: the roll-outs considered in Table 2.2, are with a relatively frugal budget, only 100 roll-outs per lever is considered – all evenly distributed between levers. However, using the multi-armed bandit paradigm, it's possible to allocate roll-outs originating from the job-list with bias towards more promising levers.

Note, in the case of random roll-outs (namely $\{\varphi_i\}_{i=21}^{24}$) then the final makespan resulting in the pursued trajectory might not necessarily be the best final makespan found during the dispatching process, this is reported as its 'fortified' result, denoted $\rho_{\text{fort.}}$.

11.1 Single feature roll-outs

A model based on each of the extremal (i.e. minimum or maximum value) values for $\{\varphi_i\}_{i=17}^{24}$ was created. The three best models for each problem space is reported, namely minimum values for φ_{17}, φ_{21} and φ_{23}. Box-plot for deviation from optimality, ρ, is depicted in Fig. 11.1, and its main statistics are given in Table 11.1. In all cases, the fortified makespan was significantly better than the final makespan of the pursued trajectory, save for $\mathcal{P}_{f.rnd}^{6\times5}$ using minimum φ_{23}, which was statistically insignificant.

Revisiting Fig. 4.1, then SPT is never the best SDR for any of the problem spaces (cf. Fig. 4.1). However, if we choose the minimum SPT from every possible operation onwards gives the best result of $\{\varphi_i\}_{i=17}^{20}$. This twist in SPT application boots performance by: *i)* $\Delta\rho \approx -39\%$ for $\mathcal{P}_{j.rnd}^{6\times5}$; *ii)* $\Delta\rho \approx -26\%$ for $\mathcal{P}_{f.rnd}^{6\times5}$, and *iii)* $\Delta\rho \approx -43\%$ for $\mathcal{P}_{j.rnd}^{10\times10}$. Bearing Fig. 7.3b in mind, then notice how $\xi^{\star}_{-\varphi_{17}}$ differs from $\xi^{\star}_{-\varphi_1}$. This implies that it's better to *not* choose the job SPT would 'normally' pick, in the initial stages. As we saw from Fig. 7.7 then pursuing SPT drastically derailed ρ performance if it ventured off the optimal trajectory. However, SPT $\xi^{\star}_{-\varphi_1}$ shows that on average SPT is policy that is likely to be optimal. Therefore making SPT roll-outs with φ_{17}, namely, repeatedly applying a $(K-k)$-lookahead for $-\varphi_1$. Then φ_{17} manages to overcome the shortcomings of pursuing $-\varphi_1$ on its own for only a 1-step lookahead.

Regarding random roll-outs, then the greedy φ_{23} came on top for $\mathcal{P}_{\text{train}}^{6\times5}$, whereas for $\mathcal{P}_{j.rnd}^{10\times10}$ then a better fortified mean result was achieved by following φ_{21}.

11.2 Multi feature roll-outs

When using random roll-outs there are many strategies to choose which job is the most promising for future roll-outs. For this reason, let's consider the preference models from Chapter 8 with additional features $\{\varphi_i\}_{i=17}^{24}$ as the weights can now be considered as its deterministic pilot heuristic.

Figure 11.1: Box-plot for deviation from optimality, ρ, for top three single extremal values for $\{\varphi_i\}_{i=17}^{24}$ roll-out using $\mathcal{P}_{\text{train}}$. SPT shown for reference on the far right.

Table 11.1: Main statistics for top three single extremal values for $\{\varphi_i\}_{i=17}^{24}$ rollout using $\mathcal{P}_{\text{train}}$

	$\boldsymbol{\varphi}$.Ext.	NrFeat		Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
$\mathcal{P}^{6\times5}_{j.rnd}$	φ_{17}.min	1	ρ	0.00	2.15	4.86	5.94	8.98	27.90
	φ_{21}.min	1	ρ	0.00	1.43	4.00	4.83	7.16	23.47
	φ_{21}.min	1	$\rho_{\text{fort.}}$	0.00	0.00	0.00	0.90	1.28	10.69
	φ_{23}.min	1	ρ	0.00	0.00	0.00	1.14	1.76	10.31
	φ_{23}.min	1	$\rho_{\text{fort.}}$	0.00	0.00	0.00	**0.78**	0.94	8.46
$\mathcal{P}^{6\times5}_{f.rnd}$	φ_{17}.min	1	ρ	0.00	3.15	7.15	7.71	11.19	30.24
	φ_{21}.min	1	ρ	0.00	1.38	3.42	4.06	6.32	15.12
	φ_{21}.min	1	$\rho_{\text{fort.}}$	0.00	0.00	0.37	**1.02**	1.57	7.53
	φ_{23}.min	1	ρ	0.00	0.00	0.66	1.51	2.25	11.26
	φ_{23}.min	1	$\rho_{\text{fort.}}$	0.00	0.00	0.39	1.23	1.87	11.26
$\mathcal{P}^{10\times10}_{j.rnd}$	φ_{17}.min	1	ρ	1.85	8.99	11.67	11.82	14.12	26.09
	φ_{21}.min	1	ρ	2.13	7.33	9.49	9.73	11.76	19.53
	φ_{21}.min	1	$\rho_{\text{fort.}}$	0.00	4.46	6.01	**5.97**	7.46	13.32
	φ_{23}.min	1	ρ	0.25	6.37	8.72	8.90	10.98	20.10
	φ_{23}.min	1	$\rho_{\text{fort.}}$	0.00	5.47	7.33	7.24	9.07	16.38

If we only use $\{\varphi_i\}_{i=17}^{20}$ it requires 4 deterministic $(K-k)$ step roll-outs at each time step. Whereas, introducing $\{\varphi_i\}_{i=21}^{24}$ costs an additional 100 random roll-outs for each time step. Therefore we'll consider both using only the first 20 features as it's relatively low computational budget, and also the computationally intensive full model of 24 features.

The experimental set-up will consider the stepwise sampling biases from Section 8.6: *i)* Bias.1 (i.e. equal probability), and *ii)* adjusted Bias.8 (i.e. double emphasis on second half). Furthermore from training data will be using either the expert policy or following the weights obtained from minimising w.r.t. ES.$C_{\max}$ defined in Eq. (5.1a). Both trajectories were detailed in Section 8.5. Box-plot for deviation from optimality, ρ, is depicted in Fig. 11.2, and its main statistics are given in Table 11.2

First off, there was no statistical difference between stepwise sampling strategy. Exceptions for $\mathcal{P}_{j.rnd}^{6\times5}$ and $\mathcal{P}_{j.rnd}^{10\times10}$ being 20.1.OPT and 24.1.OPT, favouring adjusted Bias.8, same as Section 8.6 previously showed for 16.1 models. However, w.r.t. its fortified result there was no significant difference any more.

Furthermore, the choice of trajectory starts to become irrelevant when using roll-out features. Most configuration had no significant difference. However, ES.$C_{\max}$ was preferred over expert policy for: *i)* $\mathcal{P}_{j.rnd}^{6\times5}$ w.r.t. Bias.1 using 20 features, and *ii)* $\mathcal{P}_{j.rnd}^{10\times10}$ for all configurations. This agrees with the results from Chapter 8 for 16.1 models.

As expected, when more computational budget for $\boldsymbol{\varphi}$ is allocated, the quality of the preference model increases, namely (median based on all configurations): *i)* for $\mathcal{P}_{j.rnd}^{6\times5}$ improved $\Delta\rho \approx -4.3\%$ from 16.1 to 20.1, and $\Delta\rho \approx -5.9\%$ from 20.1 to 24.1; *ii)* for $\mathcal{P}_{f.rnd}^{6\times5}$ improved $\Delta\rho \approx -3.3\%$ from 16.1 to 20.1, and $\Delta\rho \approx -3.8\%$ from 20.1 to 24.1, and *iii)* for $\mathcal{P}_{j.rnd}^{10\times10}$ improved $\Delta\rho \approx -5.7\%$ from 16.1 to 20.1, and $\Delta\rho \approx -5.6\%$ from 20.1 to 24.1.

The best configuration, namely following ES.$C_{\max}$ with adjusted stepwise bias Bias.8, is depicted with the CMA-ES obtained weights in Fig. 11.3. The local 16.1.ES.Cmax model was statistically insignificant from the baseline CMA-ES obtained weights. From the figure we can see how the models significantly improves with increased number of roll-outs.

By using preference models to create a deterministic pilot heuristic it's possible to improve the mean deviation from optimality, ρ. Especially if we consider using $\{\varphi_i\}_{i=17}^{20}$ compared to the best single based roll-out, namely minimum φ_{17}, then the improvement (all were significant) for adjusted Bias.8 following ES.$C_{\max}$ was: *i)* 5.2% compared to 5.9% for $\mathcal{P}_{j.rnd}^{6\times5}$; *ii)* 6.4% compared to 7.7% for $\mathcal{P}_{f.rnd}^{6\times5}$, and *iii)* 10.1% compared to 11.8% for $\mathcal{P}_{j.rnd}^{10\times10}$. When $\{\varphi_i\}_{i=21}^{24}$ were added, then the results for mean $\{\rho, \rho_{\text{fort.}}\}$ were as follows: *i)* $\{1.3\%, 0.8\%\}$ compared to $\{1.1\%, 0.8\%\}$ using φ_{23} for $\mathcal{P}_{j.rnd}^{6\times5}$; *ii)* $\{1.4\%, 1.0\%\}$ compared to $\{1.5\%, 1.2\%\}$ using φ_{23} for $\mathcal{P}_{f.rnd}^{6\times5}$, and *iii)* $\{6.4\%, 4.7\%\}$ compared to $\{9.7\%, 6.0\}\%$ using φ_{21} for $\mathcal{P}_{j.rnd}^{10\times10}$. Regarding $\mathcal{P}_{\text{train}}^{6\times5}$ there was no significant difference to minimum φ_{23}. Whereas, for $\mathcal{P}_{j.rnd}^{10\times10}$ the preference model paid off,

Figure 11.2: Box-plot for deviation from optimality, ρ, using roll-out features. Corresponding models only using $\{\varphi_i\}_{i=1}^{16}$ features shown for reference on the far left.

with an improvement of $\Delta\rho \approx -3.3\%$ and $\Delta\rho_{\text{fort.}} \approx -1.3\%$. This is also the case where the greedy approach of following minimum φ_{23} was unsuccessful. Note, both dimensions had the same computational budget of 100 random roll-outs for each dispatch. In the case for $\mathcal{P}_{\text{train}}^{6\times5}$ then 100 roll-outs is quite enough. Notice in Fig. 11.2 that the minimum is very often found (the 3rd quartile is often 0), and mean $\rho_{\text{fort.}} \leq 1.5\%$. However, as the possibilities for operations grows exponentially with increased dimensionality, then for $\mathcal{P}_{j.rnd}^{10\times10}$ we notice that we need to be more mindful how we allocate our $(K-k)$ roll-outs to achieve a good performance.

11.3 Conclusions

Revisiting Fig. 7.4, then $(K-k)$-step lookahead (i.e. $\{\varphi_i\}_{i=17}^{24}$) gave consistently the best (single) indicators for finding good solutions. This makes sense, as they're designed to measure end-performance, which is something that the initial 1-step attributes (i.e. $\{\varphi_i\}_{i=1}^{16}$) are struggling to measure.

By incorporating roll-outs then Eq. (2.12) can be considered as a deterministic pilot heuristic, which we could learn via preference models from Chapter 8. However, currently they're not feasible for direct optimisation as was done in Chapter 5 as that would require too many costly function evaluations.

Although, for low dimension job-shop (i.e. $\mathcal{P}_{\text{train}}^{6\times 5}$) the learned deterministic policy did not statistically improve performance, as it was equally adequate to pursue minimum φ_{23} on its own. However, going to higher dimension, as was done for $\mathcal{P}_{j.rnd}^{10\times 10}$, then finally we're able to reap the fruit of one's labour.

Unfortunately, $\{\varphi_i\}_{i=21}^{24}$ are not practical features for high dimensional data due to excessive computational cost. Nevertheless, bearing Fig. 7.3b 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 $\mathcal{P}_{f.rnd}^{10\times 10}$ 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 $\mathcal{P}_{j.rnd}^{10\times 10}$, 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.

Table 11.2: Main statistics for $\{\varphi_i\}_{i=17}^{24}$ rollout preference models using $\mathcal{P}_{\text{train}}$

	Bias	Track	NrFeat		Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
$\mathcal{P}^{6\times5}_{j.rnd}$	equal	OPT	20	ρ	0.00	2.60	5.87	6.53	9.39	21.34
	equal	OPT	20	$\rho_{\text{fort.}}$	0.00	0.95	3.43	4.08	6.43	16.22
	equal	OPT	24	ρ	0.00	0.00	0.64	1.35	1.94	10.10
	equal	OPT	24	$\rho_{\text{fort.}}$	0.00	0.00	0.00	0.76	0.93	8.03
	adjdbl2nd	OPT	20	ρ	0.00	1.95	4.62	5.65	8.39	27.22
	adjdbl2nd	OPT	20	$\rho_{\text{fort.}}$	0.00	0.83	3.64	4.29	6.92	18.50
	adjdbl2nd	OPT	24	ρ	0.00	0.00	0.00	1.26	1.64	14.18
	adjdbl2nd	OPT	24	$\rho_{\text{fort.}}$	0.00	0.00	0.00	**0.71**	0.88	11.16
	equal	ES.$C_{\max}$	20	ρ	0.00	1.55	4.35	5.07	7.80	27.22
	equal	ES.$C_{\max}$	20	$\rho_{\text{fort.}}$	0.00	0.46	3.03	3.71	5.81	18.50
	equal	ES.$C_{\max}$	24	ρ	0.00	0.00	0.55	1.29	1.92	12.42
	equal	ES.$C_{\max}$	24	$\rho_{\text{fort.}}$	0.00	0.00	0.00	0.80	0.98	9.89
	adjdbl2nd	ES.$C_{\max}$	20	ρ	0.00	1.57	4.50	5.22	7.86	27.22
	adjdbl2nd	ES.$C_{\max}$	20	$\rho_{\text{fort.}}$	0.00	0.82	3.52	4.12	6.39	17.50
	adjdbl2nd	ES.$C_{\max}$	24	ρ	0.00	0.00	0.25	1.31	2.04	9.11
	adjdbl2nd	ES.$C_{\max}$	24	$\rho_{\text{fort.}}$	0.00	0.00	0.00	0.82	1.14	8.03
$\mathcal{P}^{6\times5}_{f.rnd}$	equal	OPT	20	ρ	0.00	2.21	5.12	5.94	8.71	29.12
	equal	OPT	20	$\rho_{\text{fort.}}$	0.00	0.85	3.09	3.71	5.83	16.29
	equal	OPT	24	ρ	0.00	0.00	0.76	1.39	2.19	9.28
	equal	OPT	24	$\rho_{\text{fort.}}$	0.00	0.00	0.00	**0.84**	1.26	8.06
	adjdbl2nd	OPT	20	ρ	0.00	2.55	5.07	5.88	8.42	29.12
	adjdbl2nd	OPT	20	$\rho_{\text{fort.}}$	0.00	1.19	3.23	3.91	5.94	16.29
	adjdbl2nd	OPT	24	ρ	0.00	0.00	0.71	1.39	2.13	10.04
	adjdbl2nd	OPT	24	$\rho_{\text{fort.}}$	0.00	0.00	0.19	1.04	1.55	8.67
	equal	ES.$C_{\max}$	20	ρ	0.00	2.42	5.33	5.97	8.32	28.38
	equal	ES.$C_{\max}$	20	$\rho_{\text{fort.}}$	0.00	0.82	2.99	3.56	5.63	16.29
	equal	ES.$C_{\max}$	24	ρ	0.00	0.00	0.73	1.46	2.27	9.59
	equal	ES.$C_{\max}$	24	$\rho_{\text{fort.}}$	0.00	0.00	0.00	0.96	1.45	8.46
	adjdbl2nd	ES.$C_{\max}$	20	ρ	0.00	2.77	5.74	6.44	9.20	29.12
	adjdbl2nd	ES.$C_{\max}$	20	$\rho_{\text{fort.}}$	0.00	1.17	3.29	3.93	5.93	16.29
	adjdbl2nd	ES.$C_{\max}$	24	ρ	0.00	0.00	0.63	1.42	2.45	10.52
	adjdbl2nd	ES.$C_{\max}$	24	$\rho_{\text{fort.}}$	0.00	0.00	0.17	1.03	1.63	9.75
$\mathcal{P}^{10\times10}_{j.rnd}$	equal	OPT	20	ρ	1.89	14.99	19.41	20.06	24.64	45.14
	equal	OPT	20	$\rho_{\text{fort.}}$	1.89	8.92	11.36	11.36	13.94	21.29
	equal	OPT	24	ρ	0.87	7.30	9.72	10.15	12.92	23.35
	equal	OPT	24	$\rho_{\text{fort.}}$	0.00	4.21	5.66	5.79	7.20	13.49
	adjdbl2nd	OPT	20	ρ	5.73	12.39	15.96	16.23	19.54	37.12
	adjdbl2nd	OPT	20	$\rho_{\text{fort.}}$	3.93	8.96	11.24	11.40	13.57	22.30
	adjdbl2nd	OPT	24	ρ	0.34	6.18	8.19	8.59	10.43	22.14
	adjdbl2nd	OPT	24	$\rho_{\text{fort.}}$	0.00	4.08	5.79	5.79	7.15	12.79
	equal	ES.$C_{\max}$	20	ρ	2.76	9.13	12.16	12.26	15.14	34.60
	equal	ES.$C_{\max}$	20	$\rho_{\text{fort.}}$	1.89	6.85	9.12	9.28	11.23	20.07
	equal	ES.$C_{\max}$	24	ρ	0.62	5.24	7.09	7.33	8.90	19.43
	equal	ES.$C_{\max}$	24	$\rho_{\text{fort.}}$	0.24	3.54	4.91	4.93	6.31	12.38
	adjdbl2nd	ES.$C_{\max}$	24	ρ	0.63	4.44	6.44	6.58	8.39	21.36
	adjdbl2nd	ES.$C_{\max}$	24	$\rho_{\text{fort.}}$	0.57	3.31	5.06	**4.92**	6.35	10.93

Figure 11.3: Box-plot for deviation from optimality, ρ, using roll-out features for ES.$C_{\max}$ trajectory. Directly optimised CMA-ES model shown for reference on the far right.

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

The Gryphon

12 OR-Library Comparison

Until now only evolutionary search models from Chapter 5 have been checked for robustness using the OR-Library described in Section 3.3. This was done to show the impact of choosing objective function defined in Eq. (5.1). However, there was no reference made to other models, except for demonstrating how far off the result was from the best known solutions (BKS). Now we will also consider the best configurations for preference models in Chapters 8 to 11.

12.1 Experimental study

There were a grand-total thirteen linear composite priority dispatching rules applied to the OR-Library. Although, all models use the same training data: $\mathcal{P}_{j.rnd}^{10\times10}$. A total of two models from Chapter 5 (i.e. optimised w.r.t. either ES.$C_{\max}$ or ES.ρ) and eleven preference models from Chapters 8 to 11. Their configurations are as follows: *i)* 3.524 from Chapter 9 with Bias.1 sampling; *ii)* 16.1 following expert policy and minimum ES.$C_{\max}$ weights from Chapter 8, both using either Bias.1 or adjusted Bias.8; *iii)* 16.1 following the perturbed leader from Section 10.1.2 using either Bias.1 or adjusted Bias.8; *iv)* 16.1 following second iteration of unsupervised DAgger with extended data from Section 10.2 with stepwise adjusted Bias.8; *v)* 20.1 following expert policy and minimum ES.$C_{\max}$ weights from Section 11.2, both using adjusted Bias.8, and *vi)* 24.1 following minimum ES.$C_{\max}$ weights from Section 11.2, using adjusted Bias.8. Note, *i)* only uses three local features, *ii)* to *iv)* use all 16 local features, *v)*

Figure 12.1: Box-plot for deviation from best known solution, ρ, trained on $\mathcal{P}^{10\times10}_{j.rnd}$

use additional $\{\varphi_i\}_{i=17}^{20}$ roll-out features, and finally *vi)* includes 100 random roll-outs as well. Only a single configuration of $\{\varphi_i\}_{i=21}^{24}$ model was considered, as it is quite computationally expensive for OR-Library as some of the problem dimensions, K, is too great for 100 random roll-outs for each possible encountered operation. However, using only four fixed roll-outs is reasonable. Moreover, when applying $(K-k)$-step lookahead, then it is sensible to keep track of the best solution found (even though they had not been specifically followed further). This was referred to as its *fortified* solution, where $\rho_{\text{fort.}} \leq \rho$. To keep notation short, only $\rho_{\text{fort.}}$ is reported for models that incorporate any $\{\varphi_i\}_{i=17}^{24}$ features. Table 12.1 gives the deviation from best known solution. Note, not all problem instances from Table 3.3 are reported as not all best known solutions were found. Instead 71 out of 82 and 18 out of 32 instances are reported for JSP and FSP, respectively. Note, only best configuration of similar parameter settings was reported in Table 12.1. However, Table 12.2 gives the frequency (as percentage) of how often each configuration managed achieved the best known solution. In the case of FSP's `car1`, only the model using $\{\varphi_i\}_{i=21}^{24}$ features found that makespan. However, for JSP's `la12` even the local-based feature models found the same makespan as well. Table 12.2 also summarises the division of model configuration that found the lowest makespan (which for `la12` and `car1` coincides with BKS). In addition the 1%, 5% and 10% deviation from lowest found ρ is reported. This can also be seen in the box-plot given in Fig. 12.1.

Table 12.1: Comparison results of OR-Library based on $\mathcal{P}_{j.rnd}^{10\times10}$ training data

				CMA-ES		Preference models						
				16.1		16.1			3.524	20.1		24.1
	ID	$n\times m$	BKS	Eq. (5.1)	ρ	π	Bias	ρ	ρ	π	$\rho_{\text{fort.}}$	$\rho_{\text{fort.}}$
$\mathcal{P}_{abz}$	5	10×10	1234	ES.ρ	11.91	ES.$C_{\max}$	adjdbl2nd	12.56	11.91	ES.$C_{\max}$	8.67	**2.27**
	6	10×10	943	ES.ρ	12.51	DA2	adjdbl2nd	17.82	13.89	ES.$C_{\max}$	10.82	**2.44**
	7	20×15	656	ES.ρ	11.59	ES.$C_{\max}$	adjdbl2nd	13.26	17.53	ES.$C_{\max}$	16.46	**10.52**
	8	20×15	665	ES.$C_{\max}$	20.30	DA2	adjdbl2nd	19.55	21.50	ES.$C_{\max}$	**13.68**	15.19
	9	20×15	678	ES.ρ	22.12	ES.$C_{\max}$	adjdbl2nd	20.80	16.67	ES.$C_{\max}$	11.95	**11.21**
$\mathcal{P}_{ft}$	6	6×6	55	ES.ρ	7.27	DA2	adjdbl2nd	5.45	12.73	ES.$C_{\max}$	5.45	**1.82**
	10	10×10	930	ES.ρ	28.92	OPTε	adjdbl2nd	26.02	30.00	OPT	25.27	**14.73**
	20	20×5	1165	ES.ρ	18.37	ES.$C_{\max}$	equal	13.91	21.29	ES.$C_{\max}$	9.10	**5.41**
$\mathcal{P}_{la}$	1	10×5	666	ES.ρ	2.55	OPTε	adjdbl2nd	3.75	10.96	ES.$C_{\max}$	2.25	**1.05**
	3	10×5	597	ES.ρ	23.95	OPTε	adjdbl2nd	19.43	22.78	ES.$C_{\max}$	15.91	**6.37**
	4	10×5	590	ES.ρ	**3.56**	ES.$C_{\max}$	adjdbl2nd	**3.56**	**3.56**	ES.$C_{\max}$	**3.56**	**3.56**
	6	15×5	926	ES.ρ	10.37	DA2	adjdbl2nd	**10.15**	14.15	ES.$C_{\max}$	**10.15**	**10.15**
	7	15×5	890	ES.$C_{\max}$	**1.57**	ES.$C_{\max}$	adjdbl2nd	**1.57**	3.15	ES.$C_{\max}$	**1.57**	**1.57**
	8	15×5	863	ES.ρ	25.03	ES.$C_{\max}$	equal	14.25	27.00	ES.$C_{\max}$	**9.04**	10.31
	10	15×5	958	ES.ρ	**0.10**	ES.$C_{\max}$	adjdbl2nd	2.51	11.38	ES.$C_{\max}$	**0.10**	0.52
	12	20×5	1039	ES.ρ	**0.00**	DA2	adjdbl2nd	**0.00**	7.12	ES.$C_{\max}$	**0.00**	**0.00**
	13	20×5	1150	ES.ρ	**7.13**	OPTε	adjdbl2nd	**7.13**	8.26	ES.$C_{\max}$	7.57	**7.13**
	16	10×10	945	ES.ρ	7.09	OPTε	equal	8.68	5.71	ES.$C_{\max}$	4.76	**2.12**
	17	10×10	784	ES.ρ	8.80	DA2	adjdbl2nd	9.06	15.31	ES.$C_{\max}$	1.91	**1.02**
	18	10×10	848	ES.ρ	10.02	OPTε	adjdbl2nd	7.67	7.67	ES.$C_{\max}$	14.50	**5.54**
	19	10×10	842	ES.ρ	13.54	ES.$C_{\max}$	equal	12.59	14.85	ES.$C_{\max}$	10.45	**7.24**
	21	15×10	1046	ES.ρ	28.01	DA2	adjdbl2nd	29.35	33.94	ES.$C_{\max}$	23.52	**13.96**
	22	15×10	927	ES.ρ	15.97	ES.$C_{\max}$	adjdbl2nd	18.12	17.15	ES.$C_{\max}$	17.48	**13.16**
	23	15×10	1032	ES.$C_{\max}$	10.08	ES.$C_{\max}$	adjdbl2nd	18.80	20.25	ES.$C_{\max}$	12.98	**9.11**
	24	15×10	935	ES.ρ	14.55	ES.$C_{\max}$	adjdbl2nd	16.26	24.28	ES.$C_{\max}$	16.15	**10.27**
	25	15×10	977	ES.$C_{\max}$	15.66	ES.$C_{\max}$	adjdbl2nd	19.14	19.45	ES.$C_{\max}$	14.94	**10.95**
	26	20×10	1218	ES.ρ	16.42	DA2	adjdbl2nd	17.57	16.50	ES.$C_{\max}$	**14.86**	**14.86**
	27	20×10	1235	ES.ρ	22.43	ES.$C_{\max}$	adjdbl2nd	21.46	27.69	ES.$C_{\max}$	19.68	**15.38**
	28	20×10	1216	ES.ρ	**4.28**	ES.$C_{\max}$	adjdbl2nd	7.65	7.07	ES.$C_{\max}$	8.72	6.17
	29	20×10	1152	ES.ρ	21.61	ES.$C_{\max}$	equal	24.39	23.44	ES.$C_{\max}$	22.57	**13.63**
	30	20×10	1355	ES.$C_{\max}$	2.14	ES.$C_{\max}$	adjdbl2nd	8.71	8.56	ES.$C_{\max}$	**2.07**	3.47
	32	30×10	1850	ES.$C_{\max}$	18.32	ES.$C_{\max}$	adjdbl2nd	14.92	20.22	OPT	14.65	**6.92**
	33	30×10	1719	ES.$C_{\max}$	**7.74**	ES.$C_{\max}$	adjdbl2nd	10.35	8.14	ES.$C_{\max}$	**7.74**	8.03
	34	30×10	1721	ES.ρ	5.81	DA2	adjdbl2nd	10.28	12.96	ES.$C_{\max}$	7.44	**4.18**
	35	30×10	1888	ES.ρ	9.27	OPTε	adjdbl2nd	8.69	8.79	ES.$C_{\max}$	4.18	**2.60**
	36	15×15	1268	ES.ρ	7.89	DA2	adjdbl2nd	7.33	13.01	ES.$C_{\max}$	3.86	**2.13**
	37	15×15	1397	ES.$C_{\max}$	7.59	DA2	adjdbl2nd	7.30	11.02	ES.$C_{\max}$	9.23	**0.86**
	38	15×15	1196	ES.ρ	15.72	DA2	adjdbl2nd	15.89	17.14	ES.$C_{\max}$	15.38	**8.70**
	39	15×15	1233	ES.$C_{\max}$	8.27	ES.$C_{\max}$	adjdbl2nd	8.84	14.44	ES.$C_{\max}$	9.08	**6.73**
	40	15×15	1222	ES.ρ	17.10	DA2	adjdbl2nd	16.94	18.49	ES.$C_{\max}$	10.80	**9.33**
$\mathcal{P}_{orb}$	1	10×10	1059	ES.ρ	20.30	ES.$C_{\max}$	adjdbl2nd	13.03	22.38	OPT	19.36	**5.29**
	2	10×10	888	ES.ρ	19.03	OPTε	adjdbl2nd	12.73	15.20	ES.$C_{\max}$	6.19	**5.07**
	3	10×10	1005	ES.ρ	**12.24**	ES.$C_{\max}$	adjdbl2nd	**12.24**	18.01	OPT	15.32	15.52
	4	10×10	1005	ES.ρ	20.00	OPTε	equal	19.60	19.30	OPT	12.24	**9.95**
	5	10×10	887	ES.ρ	24.13	ES.$C_{\max}$	equal	18.38	28.18	ES.$C_{\max}$	24.13	**12.06**

Continued on next page

Table 12.1 – *Continued from previous page*

				CMA-ES		Preference models						
				16.1		16.1			3.524	20.1		24.1
	ID	$n \times m$	BKS	Eq. (5.1)	ρ	π	Bias	ρ	ρ	π	$\rho_{\text{fort.}}$	$\rho_{\text{fort.}}$
	6	10×10	1010	ES.ρ	25.15	DA2	adjdbl2nd	25.84	25.15	OPT	15.45	**12.77**
	7	10×10	397	ES.$C_{\max}$	17.13	ES.$C_{\max}$	equal	14.86	16.62	ES.$C_{\max}$	10.33	**10.08**
	8	10×10	899	ES.ρ	21.91	ES.$C_{\max}$	equal	21.91	24.58	OPT	12.68	**5.78**
	9	10×10	934	ES.ρ	18.84	DA2	adjdbl2nd	17.88	14.03	ES.$C_{\max}$	11.03	**8.14**
	10	10×10	944	ES.ρ	20.76	ES.$C_{\max}$	adjdbl2nd	20.44	27.65	ES.$C_{\max}$	19.60	**8.58**
$\mathcal{P}_{swv}$	1	20×10	1407	ES.$C_{\max}$	28.29	ES.$C_{\max}$	adjdbl2nd	30.56	33.40	ES.$C_{\max}$	27.29	**14.93**
	2	20×10	1475	ES.$C_{\max}$	27.59	ES.$C_{\max}$	adjdbl2nd	24.00	33.42	ES.$C_{\max}$	17.42	**13.63**
	3	20×10	1398	ES.ρ	25.04	ES.$C_{\max}$	adjdbl2nd	25.75	30.76	ES.$C_{\max}$	18.96	**15.52**
	4	20×10	1470	ES.ρ	29.52	ES.$C_{\max}$	adjdbl2nd	30.41	34.56	ES.$C_{\max}$	23.67	**18.57**
	5	20×10	1424	ES.ρ	23.17	ES.$C_{\max}$	adjdbl2nd	29.00	24.79	ES.$C_{\max}$	21.70	**15.31**
	6	20×15	1675	ES.$C_{\max}$	31.34	ES.$C_{\max}$	adjdbl2nd	25.31	38.39	ES.$C_{\max}$	25.97	**19.28**
	7	20×15	1594	ES.$C_{\max}$	23.34	ES.$C_{\max}$	adjdbl2nd	29.92	31.93	ES.$C_{\max}$	23.34	**17.25**
	8	20×15	1755	ES.ρ	29.63	ES.$C_{\max}$	adjdbl2nd	**23.82**	34.47	ES.$C_{\max}$	25.53	25.75
	9	20×15	1656	ES.$C_{\max}$	28.08	ES.$C_{\max}$	equal	26.75	38.41	ES.$C_{\max}$	28.02	**15.94**
	10	20×15	1743	ES.ρ	40.10	OPTε	adjdbl2nd	40.45	42.74	ES.$C_{\max}$	31.73	**21.23**
	11	50×10	2983	ES.ρ	50.55	ES.$C_{\max}$	equal	34.70	50.89	ES.$C_{\max}$	32.45	**17.5**
	12	50×10	2979	ES.$C_{\max}$	39.48	ES.$C_{\max}$	adjdbl2nd	27.32	37.97	ES.$C_{\max}$	28.33	**18.83**
	13	50×10	3104	ES.$C_{\max}$	34.41	ES.$C_{\max}$	equal	30.09	38.21	ES.$C_{\max}$	27.29	**19.23**
	14	50×10	2968	ES.ρ	32.78	ES.$C_{\max}$	adjdbl2nd	22.78	39.12	ES.$C_{\max}$	20.75	**16.24**
	15	50×10	2886	ES.$C_{\max}$	42.13	ES.$C_{\max}$	adjdbl2nd	33.75	40.89	OPT	26.09	**18.99**
	18	50×10	2852	ES.ρ	3.82	DA2	adjdbl2nd	**3.51**	4.77	ES.$C_{\max}$	3.65	**3.51**
	19	50×10	2843	ES.ρ	5.70	DA2	adjdbl2nd	7.91	11.50	ES.$C_{\max}$	**5.56**	**5.56**
$\mathcal{P}_{yn}$	1	20×20	884	ES.ρ	**10.52**	ES.$C_{\max}$	adjdbl2nd	14.82	14.59	ES.$C_{\max}$	15.84	13.01
	2	20×20	904	ES.$C_{\max}$	16.26	ES.$C_{\max}$	adjdbl2nd	12.94	17.04	ES.$C_{\max}$	18.14	**11.62**
	3	20×20	892	ES.ρ	18.95	ES.$C_{\max}$	equal	20.85	20.29	ES.$C_{\max}$	22.98	**16.03**
	4	20×20	968	ES.ρ	30.17	ES.$C_{\max}$	equal	29.34	32.64	ES.$C_{\max}$	23.14	**13.84**
$\mathcal{P}_{car}$	1	11×5	7038	ES.ρ	11.71	ES.$C_{\max}$	equal	10.19	17.01	OPT	7.47	**0.00**
	2	13×4	7166	ES.ρ	18.84	ES.$C_{\max}$	equal	14.16	23.22	ES.$C_{\max}$	4.10	**3.34**
	3	12×5	7312	ES.ρ	15.78	ES.$C_{\max}$	equal	9.38	**6.40**	ES.$C_{\max}$	7.89	7.84
	4	14×4	8003	ES.ρ	7.67	ES.$C_{\max}$	adjdbl2nd	12.61	13.83	ES.$C_{\max}$	6.10	**5.57**
	6	8×9	8505	ES.ρ	15.29	ES.$C_{\max}$	equal	6.65	11.38	ES.$C_{\max}$	6.51	**1.86**
	7	7×7	6590	ES.ρ	11.79	ES.$C_{\max}$	equal	9.77	9.77	ES.$C_{\max}$	2.58	**1.78**
	8	8×8	8366	ES.ρ	8.39	ES.$C_{\max}$	adjdbl2nd	11.00	11.59	ES.$C_{\max}$	7.42	**4.21**
$\mathcal{P}_{hel}$	2	20×10	136	ES.$C_{\max}$	15.44	ES.$C_{\max}$	adjdbl2nd	14.71	12.50	ES.$C_{\max}$	**6.62**	7.35
$\mathcal{P}_{reC}$	7	20×10	1566	ES.ρ	14.56	ES.$C_{\max}$	adjdbl2nd	14.75	16.35	ES.$C_{\max}$	12.45	**6.32**
	9	20×10	1537	ES.$C_{\max}$	12.88	DA2	adjdbl2nd	12.88	20.17	ES.$C_{\max}$	12.30	**7.48**
	11	20×10	1431	ES.$C_{\max}$	12.44	ES.$C_{\max}$	adjdbl2nd	14.40	25.44	ES.$C_{\max}$	**12.30**	13.49
	13	20×15	1930	ES.ρ	13.52	ES.$C_{\max}$	adjdbl2nd	13.32	14.09	ES.$C_{\max}$	20.26	**8.29**
	15	20×15	1950	ES.ρ	9.23	ES.$C_{\max}$	equal	11.49	10.15	ES.$C_{\max}$	13.49	**8.82**
	17	20×15	1902	ES.$C_{\max}$	19.72	ES.$C_{\max}$	adjdbl2nd	20.24	26.13	ES.$C_{\max}$	20.98	**10.73**
	25	30×15	2513	ES.ρ	20.45	ES.$C_{\max}$	adjdbl2nd	19.78	24.39	ES.$C_{\max}$	21.69	**12.06**
	29	30×15	2287	ES.ρ	16.92	ES.$C_{\max}$	equal	22.43	22.56	ES.$C_{\max}$	21.38	**13.29**
	31	50×10	3045	ES.ρ	19.77	DA2	adjdbl2nd	20.76	24.11	ES.$C_{\max}$	19.61	**15.01**
	33	50×10	3114	ES.ρ	17.66	ES.$C_{\max}$	adjdbl2nd	20.94	22.00	ES.$C_{\max}$	15.48	**10.24**

Table 12.2: Frequency (%) a model configuration found the best makespan

						w.r.t. lowest found $C_{\max}$			
	Type	CDR	Configuration	N	BKS	$= 0$	≤ 1	≤ 5	≤ 10
JSP	PREF	3.524	OPT.equal	71	0.00	1.41	2.82	18.31	49.30
	PREF	16.1	OPT.equal	71	0.00	0.00	0.00	0.00	1.41
	PREF	16.1	OPT.adjdbl2nd	71	0.00	0.00	0.00	0.00	1.41
	PREF	16.1	OPTε.equal	71	0.00	0.00	0.00	0.00	4.23
	PREF	16.1	OPTε.adjdbl2nd	71	1.41	7.04	7.04	12.68	32.39
	PREF	16.1	ES.$C_{\max}$.equal	71	0.00	0.00	0.00	2.82	21.13
	PREF	16.1	ES.$C_{\max}$.adjdbl2nd	71	0.00	5.63	8.45	23.94	61.97
	PREF	16.1	DA2.adjdbl2nd	71	1.41	5.63	7.04	12.68	42.25
	PREF	20.1	OPT.adjdbl2nd	71	0.00	2.82	12.68	30.99	76.06
	PREF	20.1	ES.$C_{\max}$.adjdbl2nd	71	1.41	15.49	22.54	52.11	84.51
	PREF	24.1	ES.$C_{\max}$.adjdbl2nd	71	1.41	87.32	90.14	100.00	100.00
	CMA-ES	16.1	ES.$C_{\max}$	71	1.41	8.45	15.49	32.39	59.15
	CMA-ES	16.1	ES.ρ	71	1.41	9.86	15.49	33.80	59.15
FSP	PREF	3.524	OPT.equal	18	0.00	5.56	5.56	11.11	55.56
	PREF	16.1	OPT.equal	18	0.00	0.00	0.00	0.00	0.00
	PREF	16.1	OPT.adjdbl2nd	18	0.00	0.00	0.00	0.00	0.00
	PREF	16.1	ES.$C_{\max}$.equal	18	0.00	0.00	0.00	16.67	38.89
	PREF	16.1	ES.$C_{\max}$.adjdbl2nd	18	0.00	0.00	0.00	5.56	55.56
	PREF	16.1	OPTε.equal	18	0.00	0.00	0.00	0.00	0.00
	PREF	16.1	OPTε.adjdbl2nd	18	0.00	0.00	0.00	0.00	11.11
	PREF	16.1	DA2.adjdbl2nd	18	0.00	0.00	0.00	0.00	38.89
	PREF	20.1	OPT.adjdbl2nd	18	0.00	0.00	11.11	16.67	72.22
	PREF	20.1	ES.$C_{\max}$.adjdbl2nd	18	0.00	11.11	27.78	61.11	83.33
	PREF	24.1	ES.$C_{\max}$.adjdbl2nd	18	5.56	83.33	88.89	100.00	100.00
	CMA-ES	16.1	ES.$C_{\max}$	18	0.00	0.00	5.56	22.22	66.67
	CMA-ES	16.1	ES.ρ	18	0.00	0.00	11.11	33.33	72.22

12.2 Conclusions

Of the thirteen linear models considered, then the one using random roll-outs (i.e. 24.1) was the consistently best one, which is to be expected as it has the most computational effort in inspecting possible solutions for each problem instance. Although it did not always find the lowest makespan of the models considered (it did for 86.52% instances), it was at least within 5% error. The second best configuration was using $\{\varphi_i\}_{i=17}^{20}$ features. This is also to be expected as it uses four fixed roll-outs. Notice in Table 12.1, when 24.1 is not the best model then the 20.1 model is best configuration (with very few exceptions). Of its two configurations then following minimum ES.$C_{\max}$ proved better than the one based on following expert policy. This concurs with the findings for the preference models using only local features, $\{\varphi_i\}_{i=1}^{16}$.

Furthermore, the model using only three local features has a remarkably low ρ compared to the other more sophisticated $\{\varphi_i\}_{i=1}^{16}$ preference models. This is probably due to the fact that during training the model has the $\mathcal{P}_{j.rnd}^{10\times 10}$ problem space specifically in mind. So when we test the trained models on completely different problem spaces then incorporating so many features may not necessarily be representing for those new instances. However, for a three-feature model, its robustness capabilities are somewhat more as it's focusing its efforts on representing the 'essence' of job-shop instead of paying particular attention to the problem space's individuality.

From the robustness experiments using the OR-Library benchmark suite, we see that the results from the training data to the corresponding test data hold when tested on completely different problem spaces, both with respect to data distribution and dimensionality.

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

The Duchess

13 Conclusions

Current literature still hold single priority dispatching rules in high regard, as they are simple to implement and quite efficient. However, they are generally taken for granted as there is clear lack of investigation of *how* these dispatching rules actually work, and what makes them so successful (or in some cases unsuccessful)?

For instance, of the four SDRs from Section 2.4 this dissertation focused on, why does MWR outperform so significantly for job-shop yet completely fail for flow-shop? MWR seems to be able to adapt to varying distributions of processing times, however, manipulating the machine ordering causes MWR to break down. By inspecting optimal schedules, and meticulously researching what's going on, every step of the way of the dispatching sequence as was done in Chapter 7, some light is shed where these SDRs vary w.r.t. the problem space at hand. Once these simple rules are understood, then it's feasible to extrapolate the knowledge gained and create new composite priority dispatching rules that are likely to be successful. An example of which was a blended dispatching rule in Section 7.4, where we start with the SPT heuristic and switch over to MWR at predetermined time points. Those pivotal time steps were chosen by inspecting where SPT succeeds MWR (and vice versa). By achieving a higher classification accuracy using the new BDR model, it substantially improve its inherited rule SPT. Although, it doesn't transcend to a significantly lower deviation from optimality, ρ, when compared to its other inherited rule MWR. Special care must be taken not to let SPT downgrade MWR's performance. This can be avoided by inspecting how ρ is evolving as a function of time for its intended policy (cf. Fig. 7.6) and only

Algorithm 5 Analysis & Learning Iterative Consecutive Executions (ALICE) framework, given a problem space $\mathcal{P}$, an expert policy $\pi_\star$, and set of benchmark algorithms $\mathcal{A}$.

```
procedure ALICE(P, π_⋆, A)
    Y^{π_⋆} ← {Y(π_⋆, φ(x)) : x ∈ P}                ▷ collect optimal solutions
    Φ^{π_⋆} ← {φ(π^⋆(x)) : x ∈ P}                   ▷ collect optimal meta-data
    for all π ∈ A do                                ▷ for each algorithm
        Y^π ← {Y(π, φ(x)) : x ∈ P}                  ▷ collect solutions
        Φ^π ← {φ(π(x)) : x ∈ P}                     ▷ collect meta-data
        ξ_π ← E{π_⋆ = π : π_⋆}                      ▷ optimality of π (i.e. when Y^{π_⋆} = Y^π)
        ζ_π ← Analyse(ξ_π ↦ Y^π)                    ▷ relation between optimality and end-result
        Φ^π ← Sample(Φ_π, ζ_π)                      ▷ adjust set w.r.t. analysis
    end for
    Φ ← {Φ^π : π ∈ {A ∪ π_⋆}}                       ▷ training set
    π̂ ←Train(Φ)                                     ▷ apply learning algorithm
    return π̂                                        ▷ learned policy
end procedure
```

consider swapping trajectories before they intersect and subsequently diverge in performance. Moreover, the improved classification accuracy is proportional to its difference in performance spread (i.e. $|\zeta^{\text{MWR}}_{\langle\cdot\rangle} - \zeta^{\text{SPT}}_{\langle\cdot\rangle}|$) in that region.

13.1 Executive summary

The framework proposed in the dissertation, called Analysis & Learning Iterative Consecutive Executions, or ALICE, is roughly described in Algorithm 5. The dissertation focused on *analysing* single priority dispatching rules, starting with Chapter 4 by defining 'difficulty' of problem instances. Then Chapter 7 continued exploring the dispatching rules even further, on step-by-step basis in order of trying to explain the performance of a dispatching rule by investigating the strength and weaknesses from empirical evidence.

The *learning* phase of ALICE is focusing on linear preference based imitation learning models. The models classify as a tailored algorithm, and they're compared to a general algorithm called CMA-ES from Chapter 5. It's noted that CMA-ES has the 'unfair' advantage of optimising the end-result directly, whereas preference models are more focusing on predictability (via classification accuracy). Therefore, when training data is contradictory it's non-trivial to achieve exceptional performance. To address that drawback, the latter half of the dissertation introduced various strategies to improve model configuration for improved learning, most notably:

i) stepwise sampling bias for balancing time-dependent data sets (cf. Section 8.6),

ii) exhaustive feature selection and bearing in mind the polysemy of reporting training accuracy for preference learning, i.e., should it be the traditional classification accuracy (that deals with classifying all ranks correctly) or just focusing on the stepwise optimality (classifying the optimal rank). Moreover, by incorporating *i)* it's possible to weigh the training accuracy w.r.t. time-step (cf. Chapter 9),

iii) allowing the predictor to randomise boosts performance as following the expert policy by itself is too difficult to learn on its own (cf. Section 10.1.2),

iv) following sub-optimal deterministic policies, yet labelling with an optimal solver, generally improves the guiding policy (cf. Section 8.5),

v) active update procedure using DAgger ensures sample states the learned model is likely to encounter is integrated to the preference set (cf. Section 10.2),

vi) keeping track of fortified solutions using roll-out features (cf. Chapter 11) .

Moreover, several problem distributions and dimensionality from Chapter 3 were considered with sometimes contradictory results. Fortunately, the performance seemed to hold when going to higher dimension (i.e. from $\mathcal{P}^{6\times5}$ to $\mathcal{P}^{10\times10}$). Thereby justifying only considering 'easy' JSP in terms of computational effort before investing valuable time for higher dimensional experiments. However, problem distributions is a key component, and the learned model should try to represent its intended (test) dataset as close as possible. Furthermore, Chapter 12 showed that results from $\mathcal{P}_{\text{train}}$ to corresponding $\mathcal{P}_{\text{test}}$ holds for completely different test application, e.g., OR-Library benchmark suite.

Creating new dispatching rules is by no means trivial. For job-shop there is the hidden interaction between processing times and machine ordering that's hard to measure. Due to this artefact, feature selection is of paramount importance, and then it becomes the case of not having too many features, as they are likely to hinder generalisation due to over-fitting in training the preference model, as was seen in Section 8.5 for several proposed policies. However, the features need to be explanatory enough to maintain predictive ability. For this reason Eq. (2.12) was limited to up to three active features in Chapter 9, as the full feature set was clearly suboptimal w.r.t. its CMA-ES benchmark from Chapter 5. By using features based on the SDRs, along with some additional local features describing the current schedule, it was possible to 'discover' the SDRs when given only one active feature. Although there is not much to be gained by these models, they at least serve as a sanity check for the learning models are on the right track.

Furthermore, by adding additional features, a boost in performance was gained, resulting in a composite priority dispatching rule that outperformed all of the SDR benchmarks. Although, the best preference model of 3 active features was still not better than the CMA-ES model for

$\mathcal{P}^{10\times 10}_{j.rnd}$ using 16 features. However, it's starting to close in on the gap, as previously $\Delta\rho \approx -6\%$ (using $\Psi_p^{\text{ES.}C_{\max}}$ from Section 8.5) and now $\Delta\rho \approx -2\%$ (cf. CDR #3.524 in Table 9.2) in favour of evolutionary search (cf. Table 5.2).

13.2 Future work

The DAgger updating framework from Section 10.2 proposed starting with the model based on the expert policy, Algorithm 4 only relies on *some* initial learned model. So in theory it should be possible to improve the $\Psi_p^{\text{ES.}C_{\max}}$ set-up even further, by applying DAgger afterwards to that learned model. Or in general substituting the learned Ψ_p^{OPT} to some other *good* initial model. Perhaps even starting with the perturbed leader which has very similar motivation as following the expert policy, yet with substantially better performance straight off the bat.

We saw in Section 8.5 that preference models using training data from following SDR policy (i.e. $\Phi^{\langle\text{SDR}\rangle}$) are good for improving its original heuristic. However, this did not transcend for $\Phi^{\langle\text{CMA-ES}\rangle}$, which was statistically insignificant with the right stepwise sampling bias. The nature of CMA-ES is to explore suboptimal routes until it converges to an optimal one. So perhaps, if $\Phi^{\langle\text{CMA-ES}\rangle}$ wasn't based on following the CMA-ES trajectory, but rather using the actual features encountered during its optimisation. Alas, CMA-ES used a computational budget of 50,000 function evaluations, each consisting of the expectation of N_{train} problem instances. So even though Fig. 5.1 becomes relatively stable after a few generations, it would still yield a gigantic feature set that needs to be filtered before going through the optimisation phase of correctly labelling them.

From Chapters 11 and 12 we saw that pilot models achieved the lowest deviation from optimality, ρ, of all other proposed models from the dissertation. Generally the more roll-outs that are made, then lower fortified makespan is found. However, in some cases using just four fixed roll-outs were better (cf. Table 12.1). In particular, we saw in Section 11.1 how pursuing $-\varphi_{17}$ (which is basically repeatedly applying w.r.t. $-\varphi_1$ for a $(K-k)$-lookahead) significantly increased performance for ρ. Now, instead of doing fixed roll-outs based on SDRs, such as $\{\varphi_i\}_{i=17}^{20}$, then it could be worth investigating a single roll-out of learned policy, $\hat{\pi}$. Usually, the learned policy surpasses SPT (i.e. $-\varphi_1$) w.r.t. stepwise optimality, i.e., $\xi^\star_{\hat{\pi}} \geq \xi^\star_{-\varphi_1}$. So presumably even better performance could be achieved, without resorting intensive computational budget of a hundred (or more) random roll-outs.

The analysis-phase of ALICE is heavily dependent on having an expert policy it's trying 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 by 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.

> So a prudent man must always follow in the footsteps of great men and imitate those who have been outstanding. If his own prowess fails to compare with theirs, at least it has an air of greatness about it.
>
> Niccolò di Bernardo dei Machiavelli (1513)

Just as this quote applied to *new principalities acquired with one's own arms and prowess* centuries ago, it equally applies when setting up novel supervised learning algorithms. Namely, when it comes to designing algorithms there needs to be emphasis on where to innovate and imitate when visiting state-spaces.

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.

REFERENCES

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

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

N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*, chapter 4. Cambridge University Press, New York, NY, USA, 2006.

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.

N. di Bernardo dei Machiavelli. *The Prince*, chapter 6. Penguin Books, 1513. Trans. George Bull, 1961.

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.

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.

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/`.

J. Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 3: 97–139, 1957.

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.

REFERENCES

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.

K. Judah, A. Fern, and T. G. Dietterich. Active imitation learning via reduction to I.I.D. active learning. *CoRR*, abs/1210.4876, 2012.

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.

B. Kim and J. Pineau. Maximum mean discrepancy imitation learning. In *Robotics: Science and Systems*, 2013.

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 *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.

S. Ross and D. Bagnell. Efficient reductions for imitation learning. In Y. W. Teh and D. M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10)*, volume 9, pages 661–668, 2010.

S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In G. J. Gordon and D. B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 627–635. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

REFERENCES

S. Ross, N. Melik-Barkhudarov, K. Shankar, A. Wendel, D. Dey, J. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 1765–1772, May 2013.

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

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.

T. P. Rúnarsson. Ordinal regression in evolutionary computation. In T. P. Rúnarsson, 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. P. Rúnarsson, 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.

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.

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.

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 Rúnarsson (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,

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

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,

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

A.2 Ordinal Regression

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

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

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,

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

subject to $t_k \langle \mathbf{w} \cdot (\boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) - \boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) \rangle \geq 1 - \xi_k$ 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 Logistic Regression

Let $\mathbf{z}$ denote either $\boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) - \boldsymbol{\varphi}(\mathbf{x}_k^{(2)})$ with $t_k = +1$ or $\boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) - \boldsymbol{\varphi}(\mathbf{x}_k^{(1)})$ with $t_k = -1$, positive or negative example respectively.

Logistic regression learns the optimal parameters $\mathbf{w} \in \mathbb{R}^d$ determined by solving the following task,

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\left\langle \mathbf{w} \cdot \mathbf{w} \right\rangle + C \sum_{i=1}^{N'} \log\left(1 + e^{-y_i \left\langle \mathbf{w} \cdot \mathbf{z}_i \right\rangle}\right) \tag{A.8}$$

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

$$\mathcal{P}\big(y = \pm 1 | \mathbf{z}, \mathbf{w}\big) = \frac{1}{1 + e^{-y \left\langle \mathbf{w} \cdot \mathbf{z}_i \right\rangle}}. \tag{A.9}$$

The logistic regression defined in Eq. (A.8) is solved iteratively, in particular using Trust Region Newton method (cf. Lin et al., 2008), which generates a sequence $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ converging to the optimal solution $\mathbf{w}^*$ of Eq. (A.8).

A.4 Non-Linear Preference

In the case that the preference set Ψ 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 Ψ 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,

$$\mathbf{w}^* = \sum_{k=1}^{N'} a^* t_k \left(\boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) - \boldsymbol{\varphi}(\mathbf{x}_k^{(2)})\right) \tag{A.10}$$

and the function $h(\cdot)$ from Eq. (A.7) may be reconstructed as follows,

$$\begin{aligned} h(\mathbf{x}) = \left\langle \mathbf{w}^* \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle &= \sum_{k=1}^{N'} a^* t_k \left(\left\langle \boldsymbol{\varphi}(\mathbf{x}_k^{(1)}) \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle - \left\langle \boldsymbol{\varphi}(\mathbf{x}_k^{(2)}) \cdot \boldsymbol{\varphi}(\mathbf{x}) \right\rangle \right) \right) \\ &= \sum_{k=1}^{N'} a^* t_k \left(\kappa(\mathbf{x}_k^{(1)}, \mathbf{x}) - \kappa(\mathbf{x}_k^{(2)}, \mathbf{x}) \right) \end{aligned} \tag{A.11}$$

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,

$$\max_a \sum_{k=1}^{N'} a_k - \frac{1}{2} \sum_{i=1}^{N'} \sum_{j=1}^{N'} a_i a_j t_i t_j \left(K\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)}\right) + \frac{1}{C} \delta_{ij} \right) \tag{A.12}$$

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,

$$\begin{aligned} K\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}, \mathbf{x}_j^{(2)}\right) &= \kappa\left(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(1)}\right) - \kappa\left(\mathbf{x}_i^{(1)}, \mathbf{x}_j^{(2)}\right) \\ &\quad - \kappa\left(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(1)}\right) + \kappa\left(\mathbf{x}_i^{(2)}, \mathbf{x}_j^{(2)}\right) \end{aligned} \tag{A.13}$$

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,*

$$\kappa_{\text{poly}}(\mathbf{x}_i, \mathbf{x}_j) = \left(1 + \left\langle \mathbf{x}_i \cdot \mathbf{x}_j \right\rangle\right)^p \tag{A.14}$$

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

$$\kappa_{\text{rbf}}(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2} \tag{A.15}$$

for $\gamma > 0$.

A.5 Parameter setting and tuning

The regulation parameter C in Eqs. (A.6), (A.8) and (A.12), 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.6 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,

$$\tilde{\varphi}_i = 2(\varphi_i - \underline{\varphi}_i)/(\overline{\varphi}_i - \underline{\varphi}_i) - 1 \qquad \forall i \in \{1, \ldots, d\} \tag{A.16}$$

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

$$\underline{\varphi}_i = \min\{\varphi_i \mid \forall \boldsymbol{\varphi} \in \Phi\} \quad \text{and} \quad \overline{\varphi}_i = \max\{\varphi_i \mid \forall \boldsymbol{\varphi} \in \Phi\} \tag{A.17}$$

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

A.7 Implementation

To use 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

This page is intentionally left blank.

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

doi: 10.1007/978-3-642-25566-3_20

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

Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling

Helga Ingimundardottir and Thomas Philip Runarsson

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

Abstract. This paper introduces a framework in which dispatching rules for job-shop scheduling problems are discovered by analysing the characteristics of optimal solutions. Training data is created via randomly generated job-shop problem instances and their corresponding optimal solution. Linear classification is applied in order to identify good choices from worse ones, at each dispatching time step, in a supervised learning fashion. The method is purely data-driven, thus less problem specific insights are needed from the human heuristic algorithm designer. Experimental studies show that the learned linear priority dispatching rules outperforms common single priority dispatching rules, with respect to minimum makespan.

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 problems class, such as job-shop scheduling, it is possible to construct problem instances where one heuristic would outperform another. Given the ad-hoc nature of the heuristic design process there is clearly room for improving the process. Recently a number of attempt have been made to automate the heuristic design process. Here we focus on the job-shop problem. Various learning approaches have been applied to this task such as, reinforcement learning [1], evolutionary learning [2], and supervised learning [3,4]. The approach taken here is a supervised learning classifier approach.

In order to find an optimal (or near optimal) solution for job-shop scheduling problem (JSSP) one could either use exact methods or heuristics methods. Exact methods guarantee an optimal solution, however, JSSP is NP-hard [5]. 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. A common way of finding a good feasible solution for the JSSP is by applying heuristic dispatching rules, e.g., choosing a task corresponding to longest/shortest operation time; most/least successors; or ranked positional weight, i.e., sum of operation times of its predecessors. Ties are broken in an arbitrary fashion or by another heuristic rule. Recently it has been shown that

C.A. Coello Coello (Ed.): LION 5, LNCS 6683, pp. 263–277, 2011.
© Springer-Verlag Berlin Heidelberg 2011

combining dispatching rules is promising [2], however, there is large number of rules to choose from and so combinations requires expert knowledge or extensive trial-and-error. A summary of over 100 classical dispatching rules can be found in [6].

The alternative to hand-crafting heuristics for the JSSP, 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 [3], where a LPT-heuristic is applied. Then 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 [4,7] by using an optimal scheduler, computed off-line. The optimal solutions are used as training data and a decision tree learning algorithm applied as before. Preferring simple to complex models, the resulting dispatching rules gave significantly more optimal 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 [8] using case based reasoning. Training data is guided by the two best heuristics for timetable scheduling. The authors point out that in order for their framework to be successful, problem features need to be sufficiently explanatory and training data need to be selected carefully so they can suggest the appropriate solution for a specific range of new cases.

In this work we investigate an approach based on supervised learning on optimal schedules and illustrate its effectiveness by improving upon well known dispatch rules for job-shop scheduling. The approach differs from previous studies, as it uses a simple linear combination of features found using a linear classifier. The method of generating training data is also shown to be critical for the success of the method. In section 2 priority dispatch rules for the JSSP problem are discussed, followed by a description of the linear classifier in section 3. An experimental study is then presented in section 4. The paper concludes with a summary of main findings.

2 Priority Dispatch Rules for Job-Shop Scheduling

The job-shop scheduling task considered here is where n jobs are scheduled on a set of m machines, subject to the constraint that each job must follow a predefined machine order and that a machine can handle at most one job at a time. The objective is to schedule the jobs so as to minimize the maximum completion times, also known as the makespan.

Each job j has an indivisible operation time on machine a, $p(j,a)$, which is assumed to be integral, where $j \in \{1,..,n\}$ and $a \in \{1,..,m\}$. Starting time of job j on machine a is denoted $x_s(a,j)$ and its completion time is denoted x_f and

$$x_f(a,j) = x_s(a,j) + p(j,a) \tag{1}$$

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

$$x_s(\sigma(j,a),j) \geq x_f(\sigma(j,a-1),j) \quad j \in \{1,..,n\},\ a \in \{2,..,m\} \tag{2}$$

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

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

for all $i,j \in \{1,..,n\}$ and $a \in \{1,..,m\}$. The time in which machine a is idle between jobs j and $j-1$ is called slack time,

$$s(a,j) = x_s(a,j) - x_f(a,j-1). \tag{4}$$

The makespan is the maximum completion time

$$z = \max\{x_f(j,m) \mid j = 1,..,n\}. \tag{5}$$

Dispatching rules are of a construction heuristics, where one starts with an empty schedule and adds on one job at a time. When a machine is free the dispatching rule inspects the waiting jobs and selects the job with the highest priority. The priority may depend on which job has the most work remaining (MWKR); least work remaining (LWKR); shortest immediate processing time (SPT); and longest immediate processing time (LPT). These are the most effective dispatching rules. However there are many more available, e.g. randomly selecting an operation with equal possibility (RND); minimum slack time (MST); smallest slack per operation (S/OP); and using the aforementioned dispatching rules with predetermined weights. A survey of more than 100 of such rules was given in 1977 by [6]. It has recently been shown that a careful combination of basic dispatching rules can perform significantly better [9].

In order to apply a dispatching rule a number of features of the schedule being built must be computed. The features of particular interest were obtained from inspecting the aforementioned single priority-based dispatching rules. Some features are directly observed from the partial schedule. The temporal scheduling features applied in this paper for a job j to be dispatched on machine a are: 1) processing time for job j on its next machine a; 2) work remaining for job j; 3) start-time of job j; 4) end-time of j; 5) when machine a is next free; 6) current makespan for all jobs; 7) slack time for machine a; 8) slack time for all machines; and 9) slack time weighted w.r.t number of number of jobs already dispatched. Fig. 1 shows an example of a temporal partial schedule for a six job and six machine job-shop problem. The numbers in the boxes represent the job identification j. The width of the box illustrates the processing times for a given job for a particular machine M_i (on the vertical axis). The dashed boxes represent the resulting partial schedule for when a particular job is scheduled next. As one can see, there are 17 jobs already scheduled, and 6 potential jobs to be dispatched next. If the job with the shortest processing time were to be scheduled next then job 4 would be dispatched. A dispatch rule may need to perform a one-step look-ahead and observes features of the partial schedule to make a decision, for example by observing the resulting temporal makespan.

Fig. 1. A schedule being built, the dashed boxes represent six different possible jobs that could be scheduled next using a dispatch rule

These resulting observed features are sometimes referred to as an *after-state* or *post-decision state*. Other dispatch rules use features not directly observable from the current partial schedule, for example by assigning jobs with most total processing time remaining.

Problem instances are generated stochastically by fixing the number of jobs and machines and sampling a discrete processing time from the uniform distribution $U(R, 100)$. The machine order is a random permutation. Two different processing times were explored, namely $U(50, 100)$ and $U(1, 100)$ for all machines. For each processing time distribution 500 instances were generated for a six job and six machine job-shop problem. Their optimal solution were then found using the GNU linear programming kit [10]. The optimal solutions are used to determine which job should be dispatched in order to create an optimal schedule and which ones are not. When a job is dispatched the features of the partial schedule change. The aim of the linear learning algorithm, discussed in the following section, is to determine which features are better than others. That is, features created when a job is scheduled in order to build the known optimal solution as opposed to features generated by dispatching jobs that will result in a sub-optimal schedule.

3 Logistic Regression

The preference learning task of linear classification presented here is based on the work presented in [11,12]. The modification relates to how the point pairs are selected and the fact that a $L2$-regularized logistic regression is used.

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

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

$$S = \left\{ \left\{ \phi^{(o)} - \phi_j^{(s)}, +1) \right\}_{k=1}^{\ell}, \left\{ \phi_j^{(s)} - \phi^{(o)}, -1) \right\}_{k=1}^{\ell} \mid \forall j \in J^{(k)} \right\} \subset \Phi \times Y \quad (6)$$

where $\Phi \subset \mathbb{R}^d$ is the training set of d features, $Y = \{-1, +1\}$ is the outcome space, $\ell = n \times m$ is the total number of dispatches and $j \in J^{(k)}$ are the possible suboptimal dispatches at dispatch (k). In this study, there are $d = 9$ features, and the training set is created from known optimal sequences of dispatch.

Now consider the model space $h \in \mathcal{H}$ of mappings from points to preferences. Each such function h induces an ordering $\succ$ on the points by the following rule:

$$\phi^{(o)} \succ \phi^{(s)} \quad \Leftrightarrow \quad h(\phi^{(o)}) > h(\phi^{(s)}) \quad (7)$$

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

$$h(\phi) = \sum_{i=1}^{d} w_i \phi_i. \quad (8)$$

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

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + C \sum_{i=1}^{l} \log \left(1 + e^{-y_i \langle \mathbf{w} \cdot \mathbf{z}_i \rangle} \right) \quad (9)$$

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

$$P(y = \pm 1 | \mathbf{z}, \mathbf{w}) = \frac{1}{1 + e^{-y \langle \mathbf{w} \cdot \mathbf{z} \rangle}}. \quad (10)$$

The logistic regression defined in (9) is solved iteratively, in particular using Trust Region Newton method [12], which generates a sequence $\{\mathbf{w}^{(k)}\}_{k=1}^{\infty}$ converging to the optimal solution $\mathbf{w}^*$ of (9).

The regulation parameter C in (9), controls the balance between model complexity and training errors, and must be chosen appropriately. It is also important

to scale the features ϕ first. A standard method of doing so is by scaling the training set such that all points are in some range, typically $[-1, 1]$. That is, scaled $\tilde{\phi}$ is

$$\tilde{\phi}_i = 2(\phi_i - \underline{\phi}_i)/(\overline{\phi}_i - \underline{\phi}_i) - 1 \quad i = 1, \ldots, d \tag{11}$$

where $\underline{\phi}_i$, $\overline{\phi}_i$ are the maximum and minimum i-th component of all the feature variables in set Φ. Scaling makes the features less sensitive to process times.

Logistic regression makes optimal decisions regarding optimal dispatches and at the same time efficiently estimates a posteriori probabilities. The optimal $\mathbf{w}^*$ obtained from the training set, can be used on any new data point, ϕ, and their inner product is proportional to probability estimate (10). Hence, for each feasible job j that may be dispatched, ϕ_j denotes the corresponding post-decision state. The job chosen to be dispatched, j^*, is the one corresponding to the highest preference estimate, i.e

$$j^* = \operatorname*{argmax}_j h(\phi_j) \tag{12}$$

where $h(\cdot)$ is the linear classification model (lin) obtained by the training data.

4 Experimental Study

In the experimental study we investigate the performance of the linear dispatching rules trained on problem instance generated using production times according to distributions $U(1, 100)$ and $U(50, 100)$. The resulting linear models is referred to as $lin_{U(1,100)}$ and $lin_{U(50,100)}$, respectively. These rules are compared with the single priority dispatching rules mentioned previously. The goal is to minimize the makespan, here the optimum makespan is denoted μ_{opt}, and the makespan obtained from a dispatching rule by μ_{DR}. Since the optimal makespan varies between problem instances the following performance measure is used:

$$\rho = \frac{\mu_{\text{DR}}}{\mu_{\text{opt}}} \tag{13}$$

which is always greater or equal to 1.

There were 500 problem instances generated using six machines and six jobs, for both $U(1, 100)$ and $U(50, 100)$ processing times distributions. Throughout the experimental study, a Kolmogorov-Smirnov goodness-of-fit hypothesis test with a significance level 0.05 is used to check if there is a statistical difference between the models in question.

4.1 Data Generation

An optimal sequence of job dispatches is known for each problem instance. The sequence indicates in which order the jobs should be dispatched. A job is placed at the earliest available time slot for its next machine, whilst still fulfilling constraints (2) and (3). Unfinished jobs are dispatched one at a time according to the optimal sequence. After each dispatch the schedule's current

features are updated based on the half-finished schedule. This sequence of job assignments is by no means unique. Take for instance Fig. 1, let's say job #1 would be dispatched next, and in the next iteration job #2. Now this sequence would yield the same schedule as if job #2 would have been dispatched first and then job #1 in the next iteration. In this particular instance one could not infer that choosing job #1 is optimal and #2 is suboptimal (or vice versa) since they can both yield the same optimal solution, however the state of the schedule has changed and thus its features. Care must be taken in this case that neither resulting features are labeled as undesirable. Only the resulting features from a dispatch resulting in a suboptimal solution should be labeled undesirable. This is the approach taken here. Nevertheless, there may still be a chance that having dispatched a job resulting in a different makespan would have resulted in the same makespan if another optimal scheduling path were to have been chosen. That is, there are multiple optimal solutions to the same problem instance. We will ignore this for the current study, but note that our data may be slightly corrupted for this reason. In conclusion, at each time step a number of feature pair are created, they consist of the features resulting from optimal dispatch versus features resulting from suboptimal dispatches.

When building a complete schedule $n \times m$ dispatches must be made sequentially. At each dispatch iteration a number of data pairs are created which can then be multiplied by the number of problem instance created. We deliberately create a separate data set for each dispatch iterations, as our 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 we will have $n \times m$ linear scheduling rules for solving a $n \times m$ JSSP.

4.2 Training Size and Accuracy

Of the 500 schedule instances, 20% were devoted solely to validation, in order to optimize the parameters of the learning algorithm. Fig. 2 shows the ratio from optimum makespan, ρ in (13), of the validation set as a function of training size for both processing time distributions considered. As one might expect, a larger training set yields a better result. However, a training size of only 200 is deemed sufficient for both distributions, and will be used here on after, yielding the remaining unused 200 instances as its test set. The training accuracy reported by the *lin*-model during training with respect to choosing the optimal job at each time step is depicted in Fig. 3 for both data distribution considered. The models obtained from using the training set corresponding to $U(1, 100)$ and $U(50, 100)$ data distributions are referred to as $lin_{U(1,100)}$ and $lin_{U(50,100)}$, respectively. The training accuracy, that is the ability to dispatch jobs according to an optimal solution, increases as more jobs are dispatched. This seems reasonable since the features initially have little meaning and hence are contradictory. It becomes easier to predict good dispatches towards the end of the schedule. This illustrates the care needed in selecting training data for learning scheduling rules.

Fig. 2. Ratio from optimum makespan, ρ, for the validation set as a function of size of training set. Solid line represents model $lin_{U(1,100)}$ and dashed line represents model $lin_{U(50,100)}$

Fig. 3. Training accuracy as a function of sequence of dispatching decisions. Solid line represents model $lin_{U(1,100)}$ and dashed line represents data distributions $lin_{U(50,100)}$

Table 1. Mean value, standard deviation, median value, minimum and maximum values of the ratio from optimum makespan, ρ, using the test sets $U(1,100)$ (top) and $U(50,100)$ (bottom)

$U(1,100)$	mean	std	med	min	max
$lin_{U(1,100)}$	1.0842	0.0536	1.0785	1.0000	1.2722
SPT	1.6707	0.2160	1.6365	1.1654	2.2500
$MWRM$	1.2595	0.1307	1.2350	1.0000	1.7288
$LWRM$	1.8589	0.2292	1.8368	1.2907	2.6906

$U(50,100)$	mean	std	med	min	max
$lin_{U(50,100)}$	1.0724	0.0446	1.0713	1.0000	1.2159
SPT	1.7689	0.2514	1.7526	1.2047	2.5367
$MWRM$	1.1835	0.0994	1.1699	1.0217	1.5561
$LWRM$	1.9422	0.2465	1.9210	1.3916	2.6642

4.3 Comparison with Single Priority Dispatching Rules

The performance of the two learned linear priority dispatch rules, ($lin_{U(1,100)}$, $lin_{U(50,100)}$), are now compared with the three most common single priority-based dispatching rules from the literature, which dispatch according to: operation with shortest processing time (SPT), most work remaining ($MWRM$), and least work remaining ($LWRM$). Their ratio from optimum, (13), is depicted in Fig. 4, and corresponding statistical findings are presented in Table 1. Clearly model $lin_{U(R,100)}$ outperforms all conventional single priority-based dispatching rules, but of them $MWRM$ is the most successful. It is interesting to note that for both data distributions, the worst-case scenario (right tail of the distributions) for model $lin_{U(R,100)}$ is noticeably better than the mean obtained using dispatching rules SPT and $LWRM$, so the choice of an appropriate single dispatching rule is of paramount importance.

4.4 Robustness towards Data Distributions

All features are scaled according to (11), which may enable the dispatch rules to be less sensitive to the different processing time distributions. To examine this the dispatch rules $lin_{U(1,100)}$ and $lin_{U(50,100)}$ are tested on both $U(1,100)$ and $U(50,100)$ test sets. The statistics for ρ are presented in Table 2. There is no statistical difference between series #1 and #4, implying that when the dispatch rules are tested on their corresponding test set, they perform equally well. It is also noted that there is no statistical difference between series #2 and #4, implying that rule $lin_{U(50,100)}$ performed equally well on both test sets in question. However, when observing at the test sets, then in both cases there is a statistical difference between applying model $lin_{U(1,100)}$ or $lin_{U(50,100)}$, where the latter yielded a better results. This implies that the rules are actually not robust towards different data distributions in some cases. This is as one may have expected.

Table 2. Mean value, standard deviation, median value, minimum and maximum values of the ratio from optimum makespan, ρ, for the test sets $U(1,100)$ and $U(50,100)$, on both models $lin_{U(1,100)}$ and $lin_{U(50,100)}$

	model	test set	mean	std	med	min	max
#1	$lin_{U(1,100)}$	$U(1,100)$	1.0844	0.0535	1.0786	1.0000	1.2722
#2	$lin_{U(50,100)}$	$U(1,100)$	1.0709	0.0497	1.0626	1.0000	1.2503
#3	$lin_{U(1,100)}$	$U(50,100)$	1.1429	0.1115	1.1158	1.0000	1.5963
#4	$lin_{U(50,100)}$	$U(50,100)$	1.0724	0.0446	1.0713	1.0000	1.2159

Table 3. Feature description and mean weights for models $lin_{U(1,100)}$ and $lin_{U(50,100)}$

Weight	$lin_{U(1,100)}$	$lin_{U(50,100)}$	Feature description
$\bar{w}(1)$	-0.6712	-0.2220	processing time for job on machine
$\bar{w}(2)$	-0.9785	-0.9195	work remaining
$\bar{w}(3)$	-1.0549	-0.9059	start-time
$\bar{w}(4)$	-0.7128	-0.6274	end-time
$\bar{w}(5)$	-0.3268	0.0103	when machine is next free
$\bar{w}(6)$	1.8678	1.3710	current makespan
$\bar{w}(7)$	-1.5607	-1.6290	slack time for this particular machine
$\bar{w}(8)$	-0.7511	-0.7607	slack time for all machines
$\bar{w}(9)$	-0.2664	-0.3639	slack time weighted w.r.t. number of operations already assigned

Table 4. Mean value, standard deviation, median value, minimum and maximum values of the ratio from optimum makespan, ρ, on models $lin_{U(1,100)}$, $lin_{U(50,100)}$, $lin_{U(1,100),\text{fixed } w}$ and $lin_{U(50,100),\text{fixed } w}$ for corresponding test sets

	model	test set	mean	std	med	min	max
#1	$lin_{U(1,100)}$	$U(1,100)$	1.0844	0.0535	1.0786	1.0000	1.2722
#2	$lin_{U(1,100),\text{fixed } w}$	$U(1,100)$	1.0862	0.0580	1.0785	1.0000	1.2722
#3	$lin_{U(50,100)}$	$U(50,100)$	1.0724	0.0446	1.0713	1.0000	1.2159
#4	$lin_{U(50,100),\text{fixed } w}$	$U(50,100)$	1.0695	0.0459	1.0658	1.0000	1.2201

4.5 Fixed Weights

Here we are interested in examining the sensitivity of the weights found for our linear dispatching rules. The weights found for each feature at each sequential dispatching step for models $lin_{U(1,100)}$ and $lin_{U(50,100)}$ are depicted in Fig. 5. These weights are averaged and listed along side their corresponding features in Table 3. The sign and size of these weights are similar for both distributions, but with the exception of features 5 and 1. The average weights are now used throughout the sequence of dispatches, these models are called $lin_{U(1,100),\text{fixed } w}$ or $lin_{U(50,100),\text{fixed } w}$, respectively.

Fig. 4. Histogram of ratio ρ for the dispatching rules $lin_{U(R,100)}$, SPT, $MWRM$ and $LWRM$ for models $lin_{U(1,100)}$ (top) and $lin_{U(50,100)}$ (bottom)

Fig. 5. Weights of features as a function of sequence of dispatching decisions, for test data $U(1, 100)$ (top) and $U(50, 100)$ (bottom)

Table 5. Mean value, standard deviation, median value, minimum and maximum values of the ratio from optimum makespan, ρ, for the test sets $U(1,100)$ and $U(50,100)$, on both fixed weight models $lin_{U(1,100),\text{fixed } w}$ and $lin_{U(50,100),\text{fixed } w}$

	model	test set	mean	std	med	min	max
#1	$lin_{U(1,100),\text{fixed } w}$	$U(1,100)$	1.0862	0.0580	1.0785	1.0000	1.2722
#2	$lin_{U(50,100),\text{fixed } w}$	$U(1,100)$	1.0706	0.0493	1.0597	1.0000	1.2204
#3	$lin_{U(1,100),\text{fixed } w}$	$U(50,100)$	1.1356	0.0791	1.1296	1.0000	1.5284
#4	$lin_{U(50,100),\text{fixed } w}$	$U(50,100)$	1.0695	0.0459	1.0658	1.0000	1.2201

Experimental results in Table 4 indicate that the weights could be held constant since there is no statistical difference between series #1 and #2 and series #3 and #4, i.e. no statistical difference between using varied or fixed weights for both data distributions. Hence, a simpler model using fixed weights should be preferred to the one of varied weights. The experiment described in section 4.4 is also repeated for fixed weights, and its results are listed in Table 5. As for varied weights (cf., Table 2), there is no statistical difference between models #2 and #4. However, unlike using varied weights, there exists a statistical difference between series #1 and #4. Again, looking at the test sets, in both cases there is statistical difference between applying model $lin_{U(1,100),\text{fixed } w}$ or $lin_{U(50,100),\text{fixed } w}$, where the latter yielded again the better result.

5 Summary and Conclusion

In this paper, a supervised learning linear priority dispatch rules (lin) is investigated to find optimal schedules for JSSP w.r.t. minimum makespan. The lin-model uses a heuristic strategy such that jobs are dispatched corresponding to the feature set that yielded the highest proportional probability output (12). The linear priority dispatch rules showed clear superiority towards single priority-based dispatch rules. The method of generating training data is critical for the framework's robustness.

The framework is not as robust with respect to different data distribution in some cases, and thus cannot be used interchangeably for training and testing and still maintain satisfactory results. Most features were of similar weight between the two data distributions (cf., Table 3), however, there are some slight discrepancies between the two distributions, e.g. $\bar{w}(5)$, which could explain the difference in performance between $lin_{U(1,00)}$ and $lin_{U(50,100)}$.

There is no statistical difference between using the linear model with varied or fixed weights when using a corresponding test set, so it is sufficient to apply only the mean varied weight, no optimization of the weight parameters is needed. It is noted that some of the robustness between data distribution is lost by using fixed weights. Hence, when dealing with a test set of known data distributions, it is sufficient to use the simpler fixed model $lin_{U(R,100),\text{fixed } w}$, however when

the data distribution is not known beforehand, it is best to use the slightly more complex varied weights model, and inferring from the experimental data rather use $lin_{U(50,100)}$ to $lin_{U(1,100)}$.

It is possible for a JSSP problem to have more than one optimal solution. However for the purpose of this study, only one optimal solution used for generating training data is sufficient. But clearly the training data set is still corrupted because of multiple ways of representing the same or different (yet equally optimal w.r.t minimum makespan) optimal schedule. One way of overcoming this obstacle is applying mixed integer programming for each possible suboptimal choice, with the current schedule as its initial value to make it absolutely certain that the choice is indeed suboptimal or not.

The proposed approach of discovering learned linear priority dispatching rules introduced in this study, are only compared with three common single priority-based dispatching rules from the literature. Although they provide evidence of improved accuracy, other comparisons of learning approaches, e.g. genetic programming, regression trees and reinforcement learning, need to be looked further into.

Another possible direction of future research is to extend the obtained results to different types of scheduling problems, along with relevant features. The efficiency of this problem solver will ultimately depend on the skills of plausible reasoning and how effectively the features extrapolate patterns yielding rules concerning optimal solutions, if they exist.

The main drawback of this approach is in order for the framework to be applicable one needs to know optimal schedules and their corresponding features in order to learn the preference, which may be difficult if not impossible to compute beforehand for some instances of JSSP using exact methods.

References

1. Zhang, W., Dietterich, T.G.: A Reinforcement Learning Approach to Job-shop Scheduling. In: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1114–1120. Morgan Kaufmann, San Francisco (1995)
2. Tay, J., Ho, N.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering 54(3), 453–473 (2008)
3. Li, X., Olafsson, S.: Discovering Dispatching Rules Using Data Mining. Journal of Scheduling 8(6), 515–527 (2005)
4. Malik, A.M., Russell, T., Chase, M., Beek, P.: Learning heuristics for basic block instruction scheduling. Journal of Heuristics 14(6), 549–569 (2007)
5. Garey, M., Johnson, D., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1(2), 117–129 (1976)
6. Panwalkar, S., Iskander, W.: A Survey of Scheduling Rules. Operations Research 25(1), 45–61 (1977)
7. Russell, T., Malik, A.M., Chase, M., van Beek, P.: Learning Heuristics for the Superblock Instruction Scheduling Problem. IEEE Transactions on Knowledge and Data Engineering 21(10), 1489–1502 (2009)

8. Burke, E., Petrovic, S., Qu, R.: Case-based heuristic selection for timetabling problems. Journal of Scheduling 9(2), 115–132 (2006)
9. Jayamohan, M.: Development and analysis of cost-based dispatching rules for job shop scheduling. European Journal of Operational Research 157(2), 307–321 (2004)
10. Makhorin, A.: GNU linear programming kit. Moscow Aviation Institute, Moscow, Russia, 38 (May 2009), Software available at `http://www.gnu.org/software/glpk/glpk.html`
11. Fan, R.e., Wang, X.r., Lin, C.j.: LIBLINEAR: A Library for Large Linear Classification. Corpus 9, 1871–1874 (2008), Software available at `http://www.csie.ntu.edu.tw/~cjlin/liblinear`
12. Lin, C.j., Weng, R.C.: Trust Region Newton Method for Large-Scale Logistic Regression. Journal of Machine Learning Research 9, 627–650 (2008)

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

doi: 10.1109/ISDA.2011.6121815

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

II

Sampling Strategies in Ordinal Regression for Surrogate Assisted Evolutionary Optimization

Helga Ingimundardottir and Thomas Philip Runarsson
School of Engineering and Natural Sciences
University of Iceland, Reykjavik, Iceland
{hei2, tpr}@hi.is

***Abstract*—In evolutionary optimization surrogate models are commonly used when the evaluation of a fitness function is computationally expensive. Here the fitness of individuals are indirectly estimated by modeling their rank with respect to the current population by use of ordinal regression. This paper focuses on how to validate the goodness of fit for surrogate models during search and introduces a novel validation/updating policy for surrogate models, and is illustrated on classical numerical optimization functions for evolutionary computation. The study shows that for validation accuracy it is sufficient for the approximate ranking and true ranking of the training set to be sufficiently concordant or that only the potential parent individuals should be ranked consistently. Moreover, the new validation approach reduces the number of fitness evaluation needed, without a loss in performance.**

***Keywords*-surrogate models; ordinal regression; sampling; evolutionary optimization**

I. Introduction

Evolutionary optimization is a stochastic and direct search method where a population of individuals are searched in parallel. Typically only the full or partial ordering of these parallel search individuals is needed. For this reason an ordinal regression offers sufficiently detailed surrogates for evolutionary computation [1]. In this case there is no explicit fitness function defined, but rather an indirect method of evaluating whether one individual is preferable to another.

The current approach in fitness approximation for evolutionary computation involves building surrogate fitness models directly using regression. For a recent review of the state-of-the-art surrogate models see [2]–[5]. The fitness model is based on a set of evaluated solutions called the training set. The surrogate model is used to predict the fitness of candidate search individuals. Commonly a fraction of individuals are selected and evaluated within each generation (or over some number of generations [6]), added to the training set, and used for updating the surrogate. The goal is to reduce the number of costly true fitness evaluations while retaining a sufficiently accurate surrogate during evolution. When using ordinal regression a candidate search individual $\mathbf{x}_i$ is said to be preferred over $\mathbf{x}_j$ if $\mathbf{x}_i$ has a higher fitness than $\mathbf{x}_j$. The training set for the surrogate model is therefore composed of pairs of individuals $(\mathbf{x}_i, \mathbf{x}_j)_k$ and a corresponding label $t_k \in [1, -1]$, taking the value $+1$ (or -1) when $\mathbf{x}_i$ has a higher fitness than $\mathbf{x}_j$ (or vice versa). The direct fitness approximation approach does not make full use of the flexibility inherent in the ordering requirement. The technique used here for ordinal regression is kernel based and is described in section II and was first presented in [1]. The use of surrogate models and approximate ranking has made some headway, e.g. [7], however still remains relatively unexplored field of study.

The critical issue in generating surrogate models, for evolutionary strategy (ES) search [8], is the manner in which the training set is constructed. For example, in optimization it is not critical to model accurately regions of the search space with low fitness. It is, however, key to model accurately new search regions deemed potentially lucrative by the evolutionary search method. Furthermore, since the search itself is stochastic, perhaps the ranking need not to be that accurate. Indeed the best μ candidate individuals are commonly selected and the rest disregarded irrespective of their exact ranking.

In the literature new individuals are added to the training set from the new generation of unevaluated search individuals. This seems sensible since this is the population of individuals which need to be ranked. However, perhaps sampling a representative individual, for example the mean of the unevaluated search individuals, may also be useful in surrogate ranking. Typically, the unevaluated individuals are ranked using the current surrogate model and then the best of these are evaluated using the true expensive fitness function and added to the training set. Again, this seems sensible since we are not interesting in low fitness regions of the search space. Nevertheless, it remains unclear whether this is actually the case. Finally, there is the question of knowing when to stop, when is our surrogate sufficiently accurate? Is it necessary to add new search individuals to our training set at every search generation? What do we mean by sufficiently accurate? This paper describes some preliminary experiments with the aim of investigating some of these issues further.

In section III sampling methods, stopping criteria and model accuracy are discussed. Moreover, a strategy for updating the surrogate during search is presented and its effectiveness illustrated using CMA-ES on some numerical optimization functions in section IV. The paper concludes with discussion and summary in section V.

II. Ordinal Regression

Ordinal regression in evolutionary optimization has been previously presented in [1], but is given here for completeness. The ranking problem is specified by a set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{\ell} \subset$

978-1-4577-1676-8/11/$26.00 ©2011 IEEE

$X \times Y$ of ℓ (solution, rank)-pairs, where $Y = \{r_1, \ldots, r_\ell\}$ is the outcome space with ordered ranks $r_1 > r_2, > \ldots > r_\ell$. 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:

$$\mathbf{x}_i \succ \mathbf{x}_j \quad \Leftrightarrow \quad h(\mathbf{x}_i) > h(\mathbf{x}_j) \tag{1}$$

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 [9], [10]. However, it is sufficient to consider only all possible pairs of adjacent ranks, see also [11] for yet an alternative formulation. The training set, composed of pairs, is then as follows:

$$S' = \left\{(\mathbf{x}_k^{(1)}, \mathbf{x}_k^{(2)}), t_k = \text{sign}(y_k^{(1)} - y_k^{(2)})\right\}_{k=1}^{\ell'} \tag{2}$$

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 $\ell' = 2(\ell - 1)$ possible adjacently ranked training 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 with corresponding feature mapping ϕ is applied. Consider the feature vector $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \ldots, \phi_m(\mathbf{x})]^T \in \mathbb{R}^m$ where m is the number of features. Then the surrogate considered may be defined by a linear function in the kernel-defined feature space:

$$h(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) = \langle \mathbf{w} \cdot \phi(\mathbf{x}) \rangle. \tag{3}$$

where $\mathbf{w} = [w_1, ..., w_m] \in \mathbb{R}^m$ has weight w_i corresponding to feature ϕ_i.

The aim now is to find a function h that encounters as few training errors as possible on S'. Applying the method of large margin rank boundaries of ordinal regression described in [9], the optimal $\mathbf{w}^*$ is determined by solving the following task:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\langle \mathbf{w} \cdot \mathbf{w} \rangle + \tfrac{C}{2} \sum_{k=1}^{\ell'} \xi_k^2 \tag{4}$$

subject to $t_k \langle \mathbf{w} \cdot (\phi(\mathbf{x}_k^{(1)}) - \phi(\mathbf{x}_k^{(2)})) \rangle \geq 1 - \xi_k$ and $\xi_k \geq 0$, $k = 1, \ldots, \ell'$. 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

$$h(\mathbf{x}_i) - h(\mathbf{x}_j) = \langle \mathbf{w} \cdot (\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)) \rangle \tag{5}$$

and that minimizing $\langle \mathbf{w} \cdot \mathbf{w} \rangle$ maximizes the margin between rank boundaries, in our case the distance between adjacently ranked pair $h(\mathbf{x}^{(1)})$ and $h(\mathbf{x}^{(2)})$.

Furthermore, it is important to scale the features ϕ first as the evolutionary search zooms in on a particular region of the search space. A standard method of doing so is by scaling the training set such that all solutions are in some range, typically $[-1, 1]$. That is, scaled $\tilde{\phi}$ is

$$\tilde{\phi}_i = 2(\phi_i - \underline{\phi}_i)/(\overline{\phi}_i - \underline{\phi}_i) - 1 \quad i = 1, \ldots, m \tag{6}$$

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

III. Sampling Methods and Improvements

In surrogate modeling, a small sample of training individuals of known fitness are needed to generate an initial surrogate. There after sampling is needed to be conducted for validating and updating the surrogate. Bearing in mind that there is generally a predefined maximum number of expensive function evaluations that can be made, the sampling of test individuals used for validating/updating the surrogate needs to be fruitful.

During evolution different regions of the space are sampled and as a consequence the surrogate ranking model may be insufficiently accurate for new regions of the search space, hence if the surrogate is not updated to reflect the original fitness function it is very probable that the ES converges to a false optimum. It is, therefore, of paramount importance to validate the surrogate during evolution. In the literature this is referred to as model management or evolution control [4].

The accuracy can be validated by generating test individuals in the new region, namely from the new candidate individuals generated at every generation of the ES by reproduction, recombination and mutation. The validation control can either be generation based, i.e. when the surrogate is converging, or individual-based, where at each generation some of the new candidate individuals are evaluated with the exact model and others are evaluated with the surrogate, see [4].

The selection of individuals to be evaluated exactly can be done randomly, however, in [12] it is reported that validating the accuracy of the ranking of potential parent individuals during evolution is most beneficial as they are critical for success. In particular, Kriging surrogate model has two main components: a drift function representing its global expected value of the true fitness function; and a covariance function representing a local influence for each data point on the model, see [13]. For Kriging models an "infill sampling criteria" is implemented by sampling the individuals which the surrogate believes to be in the vicinity of global optima, however in some cases individuals in uncertain areas are also explored, this is referred to as generalized expected improvement [14]. A performance indicator to which strategy should be focused on, i.e. following the global optima vs. getting rid of uncertainties, [15] suggests the distance between approximated optima and its real fitness value, however no obvious correlation between the two ranks could be concluded. Moreover, [13] compares 6 various sampling procedures for updating the training set using the Kriging model. Two main strategies are explored, mainly evaluating the entire candidate population or only a subset. Latter yielding a significantly fewer exact function evaluations and obtain similar goodness of fit. The former strategy mostly focuses on whether all, partial or none of

the training set should be replaced, and whether the outgoing training individuals should be the worst ranking ones (elitist) or chosen at random (universal), where the elitist perspective was considered more favorable. However, reevaluating a subset of the best ranked individuals w.r.t. the surrogate model with the exact fitness function yielded the greatest performance edge of the strategies explored.

When the training accuracy is 100% one way of evaluating the accuracy of the surrogate is through cross validation. The quality of the surrogate is measured as the rank correlation between the surrogate ranking and the true ranking on training data. Here Kendall's τ is used for this purpose [16]. Kendall's τ is computed using the relative ordering of the ranks of all $\ell(\ell-1)/2$ possible pairs. A pair is said to be concordant if the relative ranks of $h(\mathbf{x}_i)$ and $h(\mathbf{x}_j)$ are the same for $f(\mathbf{x}_i)$ and $f(\mathbf{x}_j)$, otherwise they are discordant. Kendall's τ is the normalized difference in the number of concordant and discordant pairs, defined as follows,

$$\tau = \frac{C - D}{\sqrt{C + D + T(h)}\sqrt{C + D + T(f)}} \tag{7}$$

where C and D denote the number of concordant and discordant pairs, respectively, and T denotes number of ties. Two rankings are the same when $\tau = 1$, completely reversed if $\tau = -1$, and uncorrelated for $\tau \approx 0$.

The surrogate ranking validation and improvement strategy using ordinal regression is tested using a covariance matrix adaptation evolution strategy (CMA-ES) [17]. CMA-ES is a very efficient numerical optimization technique, however we still expect to reduce the number of function evaluations needed for search. In [1] the validation policy had to successfully rank all of the candidate individuals, i.e. until $\tau = 1$. If there is no limit to training size then updating the surrogate becomes too computationally expensive, hence the training size needs to be pruned to size to $\overline{\ell}$. In [1] the set was pruned to a size $\overline{\ell} = \lambda$ by omitting the oldest individuals first. These are quite stringent restrictions which can be improved upon. The pruning only considers the age of the individuals, however older individuals might still be of more interest than newer ones if their fitness ranks higher. Thus a more sophisticated way of pruning would be omitting the lowest ranking individuals first. Moreover, candidate individuals are generated randomly using a normal distribution, thus a pseudo individual representing their mean could be of interest as an indicator for the entire population, e.g. by validating this pseudo individual first could give information if the surrogate is outdated w.r.t. the current search space. Furthermore, the validation is only done on the candidate individuals for the current generation in ES where only the μ best ranked individuals will survive to become parents. In evolutionary computing one is interested in the accurate ranking of individuals generated in the neighborhood of parent individuals, hence for sufficient validation of the surrogate, only the μ best ranked individuals should be considered and evaluated, since all other individuals of lower rank will be disregarded in the next iteration of ES. Lastly, one should also investigate the frequency by which the model is validated, e.g. at each generation or every $K > 1$ generations or even have the need for validating adapt with time.

Fig. 1. Anova plot for different validation strategies: 1) prune old individuals, 2) prune bad individuals, 3) adding a pseudo mean candidate individual 4) correctly rank μ best ranked candidate individuals 5) update on every other generation for Rosenbrock's function for dimension $n = 2$

Preliminary tests were conducted on which validation method deemed fruitful, by implementing Rosenbrock's function of dimension $n = 2$, for 1) the setup presented in [1] and comparing it with the aforementioned validation improvements, which were added one at a time. Namely; 2) omitting the worst individuals during the pruning process, instead of the oldest ones; 3) initialize the validation process by using a pseudo individual that represents the mean of the new candidate individuals; 4) requiring that only the μ best candidate individuals are correctly ranked; and 5) validating on every other generation. Experimental results focusing on the number of function evaluations are shown in Fig. 1. There is no statistical difference between omitting oldest or worst ranked individuals from the training set, but this was expected, since both are believed to be representatives of a region of the search space which is no longer of interest. Adding the pseudo mean candidate individual didn't increase the performance edge. When the surrogate was updated on every other generation, it quickly became outdated and more than double function evaluations were needed to achieve the same rate of convergence. However, requiring the correct ranking for only the μ best ranked candidate individuals showed a significant performance edge.

If the training accuracy is not 100% then clearly $\tau < 1$. In this case additional training individuals would be forced for evaluation. However, enforcing a completely concordant ranking, i.e. $\tau = 1$, was deemed to be too strict due to the fact the search is stochastic. Thus the surrogate is said to be sufficiently accurate if $\tau > 0.999$.

Based on these preliminary tests, a pseudo code for the proposed model validation and improvement strategy is described in Fig. 2 where it is implemented at the end of each generation of CMA-ES. The algorithm essentially only evaluates the expensive true fitness function when the surrogate is believed

Initialization: Let $\mathcal{Y}$ denote current training set and its corresponding surrogate by h. Let $\mathcal{X}$ denote population of λ individuals of unknown fitness under inspection.
for $t := 1$ to λ **do** *(validate a test individual)*
  Estimate ranking of $\mathcal{X}$ using h; denoted by $\bar{R}_0$.
  $\mathbf{x}_B \leftarrow \max_{\mathbf{x}\in\mathcal{X}\setminus\mathcal{Y}} \{\bar{R}_0\}$ *(test individual)*.
  Rank $\mathbf{x}_B$ w.r.t. individuals in $\mathcal{Y}$ using h; denoted by $\bar{R}$.
  Evaluate $\mathbf{x}_B$ using true fitness function and evaluate its true rank among individuals in $\mathcal{Y}$; denoted by R.
  $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{x}_B\}$ *(add to training set)*.
  Compare the rankings $\bar{R}$ and R by computing the rank correlation τ.
  if $\tau > 0.999$ **then**
    break (*model is sufficiently accurate*)
  fi
  Update the surrogate h using the new training set $\mathcal{Y}$.
  if μ best individuals of $\bar{R}_0$ have been evaluated **then**
    break (*model is sufficiently accurate*).
  fi
od

Fig. 2. Sampling strategy to validate and improve surrogate models.

Fig. 3. Schema for the sampling strategy.

to have diverged. During each iteration of the validation process there are two sets of individuals, $\mathcal{Y}$ and $\mathcal{X}$, which are the training individuals which have been evaluated with the expensive model, and the candidate individuals (of unknown fitness) for the next iteration of CMA-ES, respectively. The test individuals of interest are those who are believed to become parent individuals in the next generation of CMA-ES, i.e. the μ best ranked candidate individuals according to the surrogate h. The method uses only a simple cross-validation on a single test individual, the one which the surrogate ranks the highest and has not yet been added to the training set. Creating more test individuals would be too costly, but plausible. Once a test individual has been evaluated it is added to the training set and the surrogate h is updated w.r.t. $\mathcal{Y}$, cf. Fig. 3. This is repeated until the surrogate is said to be sufficiently accurate, which occurs if either:

- Kendall's τ statistic between the ranking of the training set using the surrogate, $\bar{R}$, and its true ranking, R, is higher than 0.999, or
- μ best ranked candidate individuals w.r.t. the current surrogate have been added to the training set.

Note that during each update of the surrogate of the ranking of the μ best candidate individuals can change. Thus it is possible to evaluate more then μ test individuals during each validation.

Once the validation algorithm has completed, the training set is pruned to a size $\bar{\ell} = \lambda$ by omitting the lowest ranking individuals .

IV. Experimental Study

In the experimental study CMA-ES is run for several test functions, namely sphere model and Rosenbrock's function, of various dimensions $n = 2, 5, 10$ and 20. The average fitness for 100 independent runs versus the number of function evaluations is reported using the original validation procedure presented in [1] and compared with its new and improved validation procedure presented in Fig. 2, the procedures will be referred to as using "all" or only the "μ best" candidate individuals during the validation, respectively. The parameter setting for the (μ, λ) CMA-ES is as recommended in [17] with population size $\lambda = 4 + \lfloor 3\ln(n) \rfloor$ and the number of parents selected $\mu = \lambda/4$. The stopping criteria used are $1000n$ function evaluation or a fitness less than 10^{-10}. The initial mean search individual is generated from a uniform distribution between 0 and 1. It is also noted that the training set is only pruned to size $\bar{\ell} = \lambda$ subsequent to the validation and improvement procedure introduced in Fig. 2.

A. Sphere model

The first experimental results are presented for the unimodal sphere model of dimension n,

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2. \tag{8}$$

The average fitness versus the number of function evaluations is presented in Fig. 4. A performance edge is achieved by restricting the validation strategy to only having the surrogate correctly rank the μ highest ranking individuals, and thereby saving the algorithm of evaluating individuals that would have been disregarded in the next iteration. Fig. 5 shows the mean intermediate function evaluations that are calculated during the validation process. As one expects, requiring the method to evaluate no more than the μ best ranked candidate individuals results in a lower intermediate function evaluations, generally saving the method one function evaluation per generation, it also achieves a better mean fitness, as shown in Table I.

B. Rosenbrock's function

The first experiment is now repeated for Rosenbrock's function,

$$f(\mathbf{x}) = \sum_{i=2}^{n} 100(x_i - x_{i-1}^2)^2 + (1 - x_{i-1})^2. \tag{9}$$

The average fitness versus the number of function evaluations is presented in Fig. 6 and Fig. 7 shows the mean intermediate function evaluations that are calculated during the

	n	Function eval. mean	Function eval. median	Function eval. sd	Generations mean	Generations median	Generations sd	Fitness mean	Fitness median	Fitness sd
all	2	130.59	132	18.33	49.02	49	6.51	2.35e-09	2.82e-10	1.15e-08
μ	2	81.53	81	9.53	48.11	48	5.02	7.01e-10	2.26e-10	1.35e-09
all	5	702.02	702	67.57	145.15	145	14.96	2.77e-10	1.82e-10	3.64e-10
μ	5	545.25	547	54.27	132.60	132	11.03	1.83e-10	1.46e-10	1.09e-10
all	10	1563.58	1553	117.09	241.83	240	18.47	1.52e-10	1.37e-10	5.03e-11
μ	10	1161.03	1158	79.98	226.60	224	13.86	1.34e-10	1.22e-10	3.80e-11
all	20	3383.83	3377	135.52	423.14	424	20.42	1.27e-10	1.21e-10	2.51e-11
μ	20	2795.28	2804	132.77	372.86	372	16.56	1.17e-10	1.12e-10	1.72e-11

TABLE I
MAIN STATISTICS OF EXPERIMENTAL RESULTS FOR UPDATING SURROGATE WITH ALL OR μ BEST INDIVIDUALS ON SPHERE MODEL.

	n	Function eval. mean	Function eval. median	Function eval. sd	Generations mean	Generations median	Generations sd	Fitness mean	Fitness median	Fitness sd
all	2	389.85	386	63.85	132.31	130	31.25	6.24e-10	3.20e-10	1.05e-09
μ	2	344.91	336	78.58	172.16	170	49.95	7.53e-10	1.66e-10	3.64e-09
all	5	2464.22	2280	748.55	514.59	492	105.77	2.75e-01	1.74e-10	1.01e+00
μ	5	1724.89	1729	295.60	520.66	520	82.79	1.83e-10	1.53e-10	1.05e-10
all	10	6800.50	6495	1258.68	1079.82	1052	177.76	2.79e-01	1.32e-10	1.02e+00
μ	10	6138.48	6143	1398.15	1177.71	1103	310.11	1.99e-01	1.24e-10	8.73e-01
all	20	19968.80	20004	234.66	2494.00	2500	49.60	4.54e-01	2.88e-02	1.08e+00
μ	20	19645.90	20002	1086.37	2687.25	2748	230.50	3.10e-01	3.12e-07	9.97e-01

TABLE II
MAIN STATISTICS OF EXPERIMENTAL RESULTS FOR UPDATING SURROGATE WITH ALL OR μ BEST INDIVIDUALS ON ROSENBROCK'S FUNCTION.

validation process. Despite requiring more generations, the over all function evaluations are significantly lower and yield a better fitness when updating the surrogate on only the μ best individuals as shown in Table II. If all of the candidate individuals have to be ranked correctly, the method will get stuck in local minima for this problem in around 6 out of 100 experiments, however this is not a problem if only the μ best candidate individuals are ranked consistently, except at high dimensions, and even then the μ best individuals policy significantly outperforms evaluating all of the candidate individuals. Clearly the choice of validation policy will influence search performance.

Fig. 4. Mean fitness values versus number of function evaluation by updating surrogate using all (dotted) or μ best (solid) individuals for sphere model.

Fig. 5. Mean intermediate function evaluations versus generation by updating surrogate using all (dotted) or μ best (solid) individuals for sphere model.

V. DISCUSSION AND CONCLUSION

The technique presented in this paper to control the number of true fitness evaluations is based on a single test individual chosen from a set of candidate individuals which the surrogate ranks the highest. The approximate ranking of this test individual is compared with its true ranking in order to determine the quality of the surrogate. This is a simple form of cross-validation. An alternative approach could be to rank all candidate individuals along with the training individuals using the surrogate model. This is followed by the re-ranking of training and candidate individuals using the updated surrogate and comparing it with the previous estimate by computing Kendall's τ. Its aim is to observe a change in ranking between successive updates of the surrogate. This study has shown that during the validation process it is sufficient for τ to be close to 1 or that only the potential parent individuals should be ranked consistently. Moreover, the new validation approach reduces the number of fitness evaluation needed, without a loss in performance although it might take a few more iterations in CMA-ES. The studies presented are exploratory in nature and clearly the approach must be evaluated on a greater range of test functions. These investigations are currently underway for combinatorial optimization problem, e.g. job shop scheduling

Fig. 6. Mean fitness values versus number of function evaluation by updating surrogate using all (dotted) or μ best (solid) individuals for Rosenbrock's function.

Fig. 7. Mean intermediate function evaluations versus generation by updating surrogate using all (dotted) or μ best (solid) individuals for Rosenbrock's function.

problem.

When it comes to modeling surrogates based on training data, the general rule of thumb is the bigger the training set, the more accurate a model. However, there are computational time limits thus pruning of the training set is necessary. Previous studies [13], [18] have reported that replacing random training individuals is not optimal. This study has shown that there is no statistical difference in omitting oldest or lowest-ranking individuals from the training set. Hence, for future work, further investigation on the fitness landscape is needed to determine effectively which search area is no longer of interest and thus unnecessary for the surrogate to approximate correctly. For instance it could be of interest to disregard training individuals with the largest euclidean distance away from the current candidate individuals rather than simply omitting the oldest/lowest-ranking training individuals.

When building surrogates in evolutionary computation one is interested in the quality of ranking of individuals only. For this reason the training accuracy and cross validation is a more meaningful measure of quality for the surrogate model. This is in contrast to regression, where the fitness function is modeled directly and the quality estimated in terms of measures such a least square error. This study has shown that the sampling used for validating the accuracy of the surrogate can stop once the μ best ranked candidate individuals have been evaluated, since they are the only candidate individuals who will survive to become parents in the next generation. Although in some cases the sampling could stop sooner, when the surrogate ranking and true ranking are sufficiently concordant, i.e. τ was close to 1. This slight slack in for τ is allowed due to the fact the ES search is stochastic, however the allowable range in slack for τ needs to be investigated more fully since allowing only $\tau \in [0.999, 1]$ might be too narrow an interval, resulting in an excess of expensive function evaluations needed.

However, in the context of surrogate-assisted optimization the discrepancy between the exact model and its surrogate can be translated as noise, which could be an indicator of the necessary sampling size for validation/updating the surrogate, instead of only focusing on consistently ranking the μ best candidate individuals. Therefore, one can take inspiration from a varying random walk population model suggested by [19] to approximate the population sizing to overcome unnecessary fitness evaluations.

References

[1] T. P. Runarsson, "Ordinal regression in evolutionary computation," in *Parallel Problem Solving from Nature IX (PPSN-2006)*, ser. LNCS, vol. 4193. Springer Verlag, 2006, pp. 1048–1057.

[2] Y. Ong, P. Nair, A. Keane, and K. W. Wong, *Surrogate-Assisted Evolutionary Optimization Frameworks for High-Fidelity Engineering Design Problems*, ser. Studies in Fuzziness and Soft Computing Series. Springer, 2004, ch. 15, pp. 333–358.

[3] A. Sobester, S. Leary, and A. Keane, "On the design of optimization strategies based on global response surface approximation models," *Journal of Global Optimization*, vol. 33, no. 1, pp. 31–59, 2005.

[4] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 1, pp. 3–12, January 2005.

[5] D. Lim, Y.-S. Ong, Y. Jin, and B. Sendhoff, "A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation." New York, New York, USA: ACM Press, 2007, pp. 1288–1295.

[6] Y. Jin, M. Olhofer, and B. Sendhoff, "A framework for evolutionary optimization with approximate fitness functions," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, October 2002.

[7] I. Loshchilov, M. Schoenauer, and M. Sebag, "Comparison-based optimizers need comparison-based surrogates," in *Parallel Problem Solving from Nature XI (PPSN-2010)*, ser. LNCS, vol. 6239. Springer Verlag, 2010, pp. 364–373.

[8] H.-P. Schwefel, *Evolution and Optimum Seeking*. New-York: Wiley, 1995.

[9] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," *Advances in Large Margin Classifiers*, pp. 115–132, 2000.

[10] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining (KDD)*. ACM, 2002.

[11] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[12] T. P. Runarsson, "Constrained evolutionary optimization by approximate ranking and surrogate models," in *Parallel Problem Solving from Nature VII (PPSN-2004)*, ser. LNCS, vol. 3242. Springer Verlag, 2004, pp. 401–410.

[13] A. Ratle, "Optimal sampling strategies for learning a fitness model," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, 1999, pp. 2078–2085.

[14] M. Sasena, P. Papalambros, and P. Goovaerts, "Exploration of metamodeling sampling criteria for constrained global optimization," *Engineering Optimization*, vol. 34, no. 3, pp. 263–278, 2002.

[15] W. Ponweiser, T. Wagner, and M. Vincze, "Clustered multiple generalized expected improvement: A novel infill sampling criterion for surrogate models," *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3515–3522, June 2008.

[16] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938.

[17] N. Hansen and A. Ostermeier, "Completely derandomized selfadaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[18] Y. Jin and J. Branke, "Evolutionary Optimization in Uncertain EnvironmentsA Survey," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, June 2005.

[19] B. Miller, "Noise, sampling, and efficient genetic algorithms," Ph.D., University of Illinois at Urbana-Champaign, Urbana, IL, 1997.

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

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

doi: 10.1007/978-3-642-34413-8_36

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

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

Helga Ingimundardottir and Thomas Philip Runarsson

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

Abstract. Many heuristic methods have been proposed for the job-shop scheduling problem. Different solution methodologies outperform other depending on the particular problem instance under consideration. Therefore, one is interested in knowing how the instances differ in structure and determine when a particular heuristic solution is likely to fail and explore in further detail the causes. In order to achieve this, we seek to characterise features for different difficulties. Preliminary experiments show there are different significant features that distinguish between easy and hard JSSP problem, and that they vary throughout the scheduling process. The insight attained by investigating the relationship between problem structure and heuristic performance can undoubtedly lead to better heuristic design that is tailored to the data distribution under consideration.

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 problems class, such as job-shop scheduling, it is possible to construct problem instances where one heuristic would outperform another. Depending on the underlying data distribution, different heuristics perform differently, commonly known as the *no free lunch* theorem [1]. The success of a heuristic is how it manages to deal with and manipulate the characteristics of its given problem instance. So 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?

In investigating the relationship between problem structure and heuristic effectiveness one can research what [2] calls *footprints* in instance space, which is an indicator how an algorithm generalises over the instance space. This sort of investigation has also been referred to as *landmarking* [3]. It is evident from experiments performed in [2] that one-algorithm-for-all problem instances is not ideal. An algorithm may be favoured for its best overall performance, however

Y. Hamadi and M. Schoenauer (Eds.): LION 6, LNCS 7219, pp. 408–412, 2012.
© Springer-Verlag Berlin Heidelberg 2012

it was rarely the best algorithm available over various subspaces of the instance space. Thus when comparing different algorithms one needs to explore how they perform w.r.t. the instance space, i.e. their footprint.

In this study, the same problem generator is used to create 1,500 problem instances, however the experimental study in section 3 shows that MWRM works well/poorly on a subset of the instances. Since the problem instances are only defined by processing times and its permutation, the interaction between the two is important, because it introduces hidden properties in the data structure making it easy or hard to schedule with for the given algorithm. These underlying characteristics or features define its data structure. So a sophisticated way of discretising the instance space is grouping together problem instances that show the same kind of feature behaviour, in order to infer what is the feature behaviour between *good* and *bad* schedules.

It is interesting to know if the difference in the structure of the schedule is time dependent, 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 completely contradictory outcomes, i.e. one poor and one good schedule? Or will they more or less follow the same path? This essentially answers the question of whether is is in fact feasible to discriminate between *good* and *bad* schedules using the currently selected features as a measure. If results are contradictory, it is an indicator the features selected are not robust enough to capture the essence of the data structure. Additionally, there is also the question of how can one define 'similar' schedules, what measures should be used? This paper describes some preliminary experiments with the aim of investigating the feasibility of finding distinguishing features corresponding to *good* and *bad* schedules in JSSP.

Instead of searching through a large set of algorithms (creating an algorithm portfolio) and determining which algorithm is the most suitable for a given subset of the instance space, as is generally the focus in the current literature [4,5,2], our focus is rather on a single algorithm and understanding *how* it works on the instance space – in the hopes of being able to extrapolate where it excels in order to aid its failing aspects.

The outline of the paper is as follows, in section 2 priority dispatch rules for the JSSP problem are discussed, what features are of interest and how data is generated. A preliminary experimental study is presented in section 3. The paper concludes with a summary of main findings and points to future work.

2 Job-Shop scheduling

The job-shop scheduling task considered here is where n jobs are scheduled on a set of m machines, subject to the constraint that each job must follow a predefined machine order and that a machine can handle at most one job at a time. The objective is to schedule the jobs so as to minimize the maximum completion times, also known as the makespan. For a mathematical formulation of JSSP the reader is recommended [6].

Table 1. Feature space $\mathcal{F}$ for JSSP. Features 1–13 can vary throughout the scheduling process w.r.t. tasks that can be dispatched next, however features 14–16 are static

ϕ	Feature description
ϕ_1	processing time for job on machine
ϕ_2	start-time
ϕ_3	end-time
ϕ_4	when machine is next free
ϕ_5	current makespan
ϕ_6	work remaining
ϕ_7	most work remaining
ϕ_8	slack time for this particular machine
ϕ_9	slack time for all machines
ϕ_{10}	slack time weighted w.r.t. number of operations already assigned
ϕ_{11}	time job had to wait
ϕ_{12}	size of slot created by assignment
ϕ_{13}	total processing time for job
ϕ_{14}	total processing time for all jobs
ϕ_{15}	mean processing time for all jobs
ϕ_{16}	range of processing times over all jobs

2.1 Single-Priority Dispatching Heuristic

Dispatching rules are of a construction heuristics, where one starts with an empty schedule and adds on one job at a time. When a machine is free the dispatching rule inspects the waiting jobs and selects the job with the highest priority. A survey of more than 100 of such priority rules was given in 1977 by [7]. In this paper however, only most work remaining (MWRM) dispatching rule will be investigated.

In order to apply a dispatching rule a number of features of the schedule being built must be computed. The features of particular interest were obtained from inspecting the aforementioned single priority-based dispatching rules. The temporal scheduling features applied in this paper are given in Table 1. These are not the only possible set of features, they are however built on the work published in [6,4] and deemed successful in capturing the essence of a JSSP data structure.

2.2 Data Generation

Problem instances were generated stochastically by fixing the number of jobs and machines and sampling a discrete processing time from the uniform distribution $U(1, 200)$. The machine order is a random permutation of $\{1, ..., m\}$. A total of 1,500 instances were generated for a six job and six machine job-shop problem.

In the experimental study the performance of the MWRM, μ_{MWRM}, and compared with its optimal makespan, μ_{opt}. Since the optimal makespan varies between problem instances the following performance measure is used:

$$\rho = \frac{\mu_{\text{MWRM}}}{\mu_{\text{opt}}}. \tag{1}$$

3 Experimental Study

In order to differentiate between problems, a threshold of a $\rho < 1.1$ and $\rho > 1.3$ was used to classify *easy* and *hard* problems. Of the 1500 instances created, 271 and 161 problems were classified *easy* and *hard* , respectively.

Table 2. Features for *easy* and *hard* problems are drawn from the same data distribution (denoted by ·)

ϕ	dispatch 1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33	35
ϕ_1	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ϕ_2	•	•		•														
ϕ_3	•	•	•	•			•											
ϕ_4	•			•														
ϕ_5	•																	
ϕ_6	•		•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ϕ_7								•		•	•	•	•	•	•		•	•
ϕ_8	•	•	•	•														
ϕ_9	•	•	•	•	•	•	•											
ϕ_{10}	•	•	•	•	•	•	•											
ϕ_{11}	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•			
ϕ_{12}	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
ϕ_{13}		•	•	•			•	•	•	•	•	•	•	•	•	•	•	•
ϕ_{14}																		
ϕ_{15}																		
ϕ_{16}	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

Table 3. Significant correlation (denoted by ·) for *easy* (left) and *hard* (right) problems and resulting ratio from optimality, ρ defined by (1). Commonly significant features across the tables are denoted by •.

Easy ϕ_i	dispatch 1	5	10	15	20	30	35
1							
2			• •	·	· ·	· · · · · · · ·	•
3			· • ·		· ·	• · · · • · ·	• •
4		·	• • ·		· ·	• · · • • • ·	·
5		· · · · ·	· • • · · ·	· ·	· · • · · • • • • • • • • •	•	
6							
7							
8		· · ·		· · ·	• • • • · · · · · · ·	• • • · • · ·	•
9		· ·		·	· · · · •	• · · · · ·	· ·
10		· ·		·	· · · ·	• · · · ·	· ·
11	·		·	·		·	
12		·				·	·
13					·		
14							
15							
16							

Hard ϕ_i	dispatch 1	5	10	15	20	30	35
1	·			·			
2		·	• • ·		·	• •	•
3		·	•			• ·	• •
4			• · •		·	• ·	• • •
5			• •		·	• • • • • • • • •	•
6							·
7							
8					• • • •	• • • •	•
9			·		•	· •	
10			·		·	· •	·
11			·				
12				·		·	
13							·
14							
15							
16							

Table 2 reports where data distributions are the same (denoted by ·). From the table one can see that distribution for ϕ_1, ϕ_6, ϕ_{12} and ϕ_{16}) are (more or less) the same throughout the scheduling process. However there is a clear time of divergence for distribution of slacks; step 6 for ϕ_8 and step 12 for ϕ_9 and ϕ_{10}.

In order to find defining characteristics for *easy* and *hard* problems, a (linear) correlation was computed between features (on a step-by-step basis) to the resulting ratio from optimality. Significant features are reported in Table 3 for *easy* and *hard* problems, (denoted by ·). As one can see from the tables, the significant features for the different difficulties are varying. Some are commonly significant features across the tables (denoted by •).

4 Discussion and Conclusion

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 throughout the scheduling process. And features constant throughout the scheduling process are not correlated with

the end-result. There are some common features for both difficulties considered which define JSSP on a whole. However the significant features are quite different across the two difficulties, implying there is 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.

The feature attributes need to be based on statistical or theoretical grounds. Thus scrutiny in understanding the nature of problem instances is of paramount importance in feature engineering for learning. Which yields feedback into what features are important to devout more attention to, i.e. features that result in a failing algorithm. In general, this sort of investigation 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.

Although this methodology was only implemented on a simple single-priority dispatching rule heuristic, the methodology is easily adaptable for more complex algorithms. The main objective of this work is to illustrate the interaction of a specific algorithm on a given problem structure and its properties.

References

1. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)
2. Corne, D.W., Reynolds, A.P.: Optimisation and Generalisation: Footprints in Instance Space. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI, Part I. LNCS, vol. 6238, pp. 22–31. Springer, Heidelberg (2010)
3. Pfahringer, B., Bensusan, H.: Meta-learning by landmarking various learning algorithms. In: Machine Learning (2000)
4. Smith-Miles, K.A., James, R.J.W., Giffin, J.W., Tu, Y.: A Knowledge Discovery Approach to Understanding Relationships between Scheduling Problem Structure and Heuristic Performance. In: Stützle, T. (ed.) LION 3. LNCS, vol. 5851, pp. 89–103. Springer, Heidelberg (2009)
5. Smith-Miles, K., Lopes, L.: Generalising Algorithm Performance in Instance Space: A Timetabling Case Study. In: Coello Coello, C.A. (ed.) LION 5. LNCS, vol. 6683, pp. 524–538. Springer, Heidelberg (2011)
6. Ingimundardottir, H., Runarsson, T.P.: Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling. In: Coello Coello, C.A. (ed.) LION 5. LNCS, vol. 6683, pp. 263–277. Springer, Heidelberg (2011)
7. Panwalkar, S., Iskander, W.: A Survey of Scheduling Rules. Operations Research 25(1), 45–61 (1977)

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

Alice

IV

IV

Evolutionary Learning of Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

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

Computational Intelligence

doi: 10.1007/978-3-319-26393-9_4

Reprinted, with permission, from *Computational Intelligence* (2016). Copyright 2016 by Springer.

Evolutionary Learning of Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardottir and Thomas Philip Runarsson

IV

Abstract 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.

Keywords Job shop scheduling · Composite dispatching rules · Evolutionary search

1 Job Shop Scheduling

The job-shop scheduling problem (JSP) deals with the allocation of tasks of competing resources where the goal is to optimise a single or multiple objectives—in particular minimising a schedule's maximum completion time, i.e., the makespan, denoted $C_{\max}$. Due to difficulty in solving this problem, heuristics are generally applied. Perhaps the simplest approach to generating good feasible solutions for JSP is by

H. Ingimundardottir · T.P. Runarsson (✉)
Industrial Engineering, Mechanical Engineering and Computer Science,
University of Iceland, Reykjavik, Iceland
e-mail: tpr@hi.is

H. Ingimundardottir
e-mail: hei2@hi.is

© Springer International Publishing Switzerland 2016

J.J. Merelo et al. (eds.), *Computational Intelligence*,
Studies in Computational Intelligence 620, DOI 10.1007/978-3-319-26393-9_4

applying dispatching rules (DR), e.g., choosing a task corresponding to longest or shortest processing time, most or least successors, or ranked positional weight, i.e., sum of processing times of its predecessors. Ties are broken in an arbitrary fashion or by another heuristic rule. Combining dispatching rules for JSP is promising, however, there is a large number of rules to choose from, thus its combinations rely on expert knowledge or extensive trial-and-error process to choose a suitable DR [21]. Hence given the diversity within the JSP paradigm, there is no "one-rule-fits-all" for all problem instances (or shop constraints), however single priority dispatching rules (SDR) based on job processing attributes have proven to be effective [8]. The classical dispatching rules are continually used in research; a summary of over 100 classical DRs for JSP can be found in [16]. However, careful combinations of such simple rules, i.e., composite dispatching rules (CDRs) can perform significantly better [12]. As a consequence, a linear composite of dispatching rules for JSP was presented in [10]. There the goal was to learn a set of weights, $\mathbf{w}$ via ordinal regression such that

$$h(\mathbf{x}_j) = \left\langle \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_j) \right\rangle, \tag{1}$$

yields the preference estimate for dispatching job j that corresponds to post-decision state $\mathbf{x}_j$, where $\boldsymbol{\phi}(\mathbf{x}_j)$ denotes the feature mapping (cf. Sect. 4). In short, Eq. 1 is a simple linear combination of features found using a classifier which is trained by giving more weight to instances that are preferred w.r.t. optimality in a supervised learning fashion. As a result, the job dispatched is the following,

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

A more popular approach in recent JSP literature is applying genetic algorithms (GAs) [17]. 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 [1, 3, 4, 13, 22], however, then there are many concerns that need to be dealt with. To begin with there are nine encoding schemes for representing the schedules [3], in addition, special care must be taken when applying cross-over and mutation operators in order for schedules 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 [4].

The most predominant approach in hyper-heuristics, a framework of creating *new* heuristics from a set of predefined heuristics, is genetic programming [2]. Dispatching rules based genetic algorithms (DRGA) [5, 15, 23] are a special case of genetic programming [14], 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.

There are two main viewpoints on how to approach scheduling problems, (*a*) local level by building schedules for one problem instance at a time; and (*b*) 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 [10]. 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.

IV

2 Outline

In order to formulate the relationship between problem structure and heuristic efficiency, one can utilise Rice's framework for algorithm selection [18]. The framework consists of four fundamental components, namely,

Problem Space or Instance Space $\mathscr{P}$,
set of problem instances;
Feature Space $\mathscr{F}$,
measurable properties of the instances in $\mathscr{P}$;
Algorithm Space $\mathscr{A}$,
set of all algorithms under inspection;
Performance Space $\mathscr{Y}$,
the outcome for $\mathscr{P}$ using an algorithm from $\mathscr{A}$.

For a given problem instance $\mathbf{x} \in \mathscr{P}$ with k features $\boldsymbol{\phi}(\mathbf{x}) = \{\phi_1(\mathbf{x}), \ldots, \phi_k(\mathbf{x})\} \in \mathscr{F}$ and using algorithm $a \in \mathscr{A}$ the performance is $y = Y(a, \boldsymbol{\phi}(\mathbf{x})) \in \mathscr{Y}$, where $Y : \mathscr{A} \times \mathscr{F} \mapsto \mathscr{Y}$ is the mapping for algorithm and feature space onto the performance space. [11, 19, 20] formulate JSP in the following manner: (*a*) problem space $\mathscr{P}$ is defined as the union of N problem instances consisting of processing time and ordering matrices given in Sect. 3; (*b*) feature space $\mathscr{F}$, which is outlined in Sect. 4. Note, these are not the only possible set of features, however, they are built on the work by [10, 19] and deemed successful in capturing the essence of a JSP data structure; (*c*) algorithm space $\mathscr{A}$ is simply the scheduling policies under consideration and discussed in Sect. 5; (*d*) performance space is based on the resulting $C_{\max}$. Different fitness measures are investigated in Sect. 5.1; and (*e*) mapping Y is the step-by-step scheduling process.

In the context of Rice's framework, and returning to the aforementioned approaches to scheduling problems, then the objective is to maximise its expected heuristic performance, i.e.,

(*a*) Local level

$$\max_{\mathscr{P}' \subset \mathscr{P}} \mathbb{E}\left[Y\left(a, \boldsymbol{\phi}(\mathbf{x})\right)\right] \tag{3}$$

where $\mathbf{x} \in \mathscr{P}'$ and algorithm a is obtained via ordinal regression based on the feature space $\mathscr{F}$, i.e., $\mathscr{F}|_{\mathscr{P}'} \mapsto \mathscr{A}$, such as the approach taken in [10], and will be used as a benchmark for the following,

(*b*) Global level

$$\max_{a \in \mathscr{A}} \mathbb{E}\left[Y\left(a, \boldsymbol{\phi}(\mathbf{x})\right)\right] \tag{4}$$

where training data $\mathbf{x} \in \mathscr{P}$ is guided by its algorithm a, i.e., $\mathscr{A} \mapsto \mathscr{P}$. This will be the focus of this study.

IV

Note that the mappings $\boldsymbol{\phi} : \mathscr{P} \mapsto \mathscr{F}$ and $Y : \mathscr{A} \mapsto \mathscr{Y}$ are the same for both paradigms.

The paper concludes in Sect. 6 with discussion and conclusions.

3 Problem Space

For this study synthetic JSP and its subclass, permutation flow shop problem (PFSP), the scheduling task considered here is where n jobs are scheduled on a set of m machines, i.e., problem size $n \times m$, subject to the constraint that each job must follow a predefined machine order and that a machine can handle at most one job at a time. The pair (j, a) refers to the operation of dispatching job j on machine a. As a result, a total of $\ell = n \cdot m$ sequential operations need to be made for a complete schedule.

The objective is to schedule the jobs so as to minimize the maximum completion times, $C_{\max}$, also known as the makespan. For a mathematical formulation of JSP the reader is recommended [10].

There are two fundamental types of problem classes: non-structured versus structured. Firstly there are the "conventional" structured problem classes, where problem instances are generated stochastically by fixing the number of jobs and machines, as well as processing times are i.i.d. and sampled from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e., $p \sim \mathscr{U}(u_1, u_2)$. Two different processing time distributions are explored, namely $\mathscr{P}_{j.rnd}$ where $I = [1, 99]$ and $\mathscr{P}_{j.rndn}$ where $I = [45, 55]$, referred to as random and random-narrow, respectively. The machine order is a random permutation of all of the machines in the job-shop.

Analogous to $\mathscr{P}_{j.rnd}$ and $\mathscr{P}_{j.rndn}$ the problem classes $\mathscr{P}_{f.rnd}$ and $\mathscr{P}_{f.rndn}$, respectively, correspond to the structured PFSP problem classes, however with a homogeneous machine order permutation. Secondly, there are structured problem classes of PFSP which are modelled after real-world flow-shop manufacturing namely job-correlated $\mathscr{P}_{f.jc}$ where job processing times are dependent on job index and

Table 1 Problem space distributions used in Sect. 5

Name	Size	N_{train}	N_{test}	Note
Permutation flow shop problem (PFSP)				
$\mathscr{P}^{6\times5}_{f.rnd}$	6×5	500	–	Random
$\mathscr{P}^{6\times5}_{f.rndn}$	6×5	500	–	Random-narrow
$\mathscr{P}^{6\times5}_{f.jc}$	6×5	500	–	Job-correlated
$\mathscr{P}^{10\times10}_{f.rnd}$	10×10	–	500	Random
$\mathscr{P}^{10\times10}_{f.rndn}$	10×10	–	500	Random-narrow
$\mathscr{P}^{10\times10}_{f.jc}$	10×10	–	500	Job-correlated
Job shop problem (JSP)				
$\mathscr{P}^{6\times5}_{j.rnd}$	6×5	500	–	Random
$\mathscr{P}^{6\times5}_{j.rndn}$	6×5	500	–	Random-narrow
$\mathscr{P}^{10\times10}_{j.rnd}$	10×10	–	500	Random
$\mathscr{P}^{10\times10}_{j.rndn}$	10×10	–	500	Random-narrow

Note Problem instances are synthetic and each problem space is i.i.d. and '–' denotes not available

IV

independent of machine index. Problem instances for PFSP are generated using [24] problem generator.[1]

For each JSP and PFSP class N_{train} and N_{test} instances were generated for training and testing, respectively. Values for N are given in Table 1. Note, difficult problem instances are not filtered out beforehand, such as the approach in [24].

4 Feature Space

When building a complete JSP schedule, a job is placed at the earliest available time slot for its next machine while still fulfilling constraints that each machine can handle at most one job at a time, and jobs need to have finished their previous machines according to its machine order. Unfinished jobs are dispatched one at a time according to some heuristic. After each dispatch the schedule's current features are updated. Features are used to grasp the essence of the current state of the schedule. As seen in Table 2, temporal scheduling features applied in this study are given for each possible post-decision state. An example of a schedule being built is given in Fig. 1, where there are a total of five possible jobs that could be chosen to be dispatched by some dispatching rule. These features would serve as the input for Eq. 1.

It's noted that some of the features directly correspond to a SDR commonly used in practice. For example, if the weights $\mathbf{w}$ in Eq. 1 were all zero, save for $w_6 = 1$, then Eq. 2 yields the job with the highest ϕ_6 value, i.e., equivalent to dispatching rule most work remaining (MWR).

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

Table 2 Feature space $\mathscr{F}$ for $\mathscr{P}$ given the resulting temporal schedule after dispatching an operation (j, a)

$\boldsymbol{\phi}$	Feature description
ϕ_1	Job j processing time
ϕ_2	Job j start-time
ϕ_3	Job j end-time
ϕ_4	When machine a is next free
ϕ_5	Current makespan
ϕ_6	Total work remaining for job j
ϕ_7	Most work remaining for all jobs
ϕ_8	Total idle time for machine a
ϕ_9	Total idle time for all machines
ϕ_{10}	ϕ_9 weighted w.r.t. number of assigned tasks
ϕ_{11}	Time job j had to wait
ϕ_{12}	Idle time created
ϕ_{13}	Total processing time for job j

Fig. 1 Gantt chart of a partial JSP schedule after 15 operations: *Solid boxes* represent previously dispatched jobs, and *dashed boxes* represent the jobs that could be scheduled next. Current $C_{\max}$ denoted as *dotted line*

5 Experimental Study

The optimum makespan[2] is denoted $C_{\max}^{\text{opt}}$, and the makespan obtained from the heuristic model by $C_{\max}^{\text{model}}$. Since the optimal makespan varies between problem instances the performance measure is the following,

$$\rho := \frac{C_{\max}^{\text{model}} - C_{\max}^{\text{opt}}}{C_{\max}^{\text{opt}}} \cdot 100\,\% \tag{5}$$

IV

which indicates the percentage relative deviation from optimality. 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. 1 directly via evolutionary search such as covariance matrix adaptation evolution strategy (CMA-ES) [7]. 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 h on a cluster for each training set given in Sect. 3 and in every case the optimisation reached its maximum walltime.

5.1 Performance Measures

Generally, evolutionary search only needs to minimise the expected fitness value. However, the approach in [10] was to use the known optimum to correctly label which operations' features were optimal when compared to other possible operations (Fig. 2). 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,

$$ES_{C_{\max}} := \min \mathbb{E}[C_{\max}] \tag{6}$$

$$ES_{\rho} := \min \mathbb{E}[\rho] \tag{7}$$

Main statistics of the experimental run are given in Table 3 and depicted in Fig. 3 for both approaches. In addition, evolving decision variables, here weights $\mathbf{w}$ for Eq. 1, are depicted in Fig. 4.

In order to compare the two objective functions, the best weights reported were used for Eq. 1 on the corresponding training data. Its box-plot of percentage relative deviation from optimality, defined by Eq. 5, is depicted in Fig. 2 and Table 4 present its main statistics; mean, median, standard deviation, minimum and maximum values.

[2]Optimum values are obtained by using a commercial software package [6].

Fig. 2 Box-plot of training data for percentage relative deviation from optimality, defined by Eq. (5), when implementing the final weights obtained from CMA-ES optimisation, using both objective functions from Eqs. (6) and (7), *left* and *right*, respectively

IV

Table 3 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. 6			
$\mathscr{P}$	#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. 7			
$\mathscr{P}$	#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

In the case of $\mathscr{P}_{f.rndn}$, Eq. 6 gave a considerably worse results, since the optimisation got trapped in a local minima, as the erratic evolution of the weights in Fig. 4a suggest. For other problem spaces, Eq. 6 gave slightly better results than Eq. 7. 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.

Fig. 3 Fitness for optimising (w.r.t. Eqs. (6) and (7) *above* and *below*, receptively), per generation of the CMA-ES optimisation

5.2 Problem Difficulty

The evolution of fitness per generation from the CMA-ES optimisation of Eq. 7 is depicted in Fig. 3. Note, all problem spaces reached their allotted computational time without converging. In fact $\mathscr{P}_{f.rnd}$ and $\mathscr{P}_{j.rndn}$ needed restarting during the optimisation process. Furthermore, the evolution of the decision variables $\mathbf{w}$ are depicted in Fig. 4. As one can see, the relative contribution for each weight clearly differs between problem spaces. Note, that in the case of $\mathscr{P}_{j.rndn}$ (cf. Fig. 4b), 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 are a linear combination of others, e.g. $\phi_3 = \phi_1 + \phi_2$.

5.3 Scalability

As a benchmark, the linear ordinal regression model (PREF) from [10] was created. Using the weights obtained from optimising Eq. 7 and applying them on their 6×5 training data. Their main statistics of Eq. 5 are reported in Table 4 for all training sets described in Table 1. Moreover, the best SDR from which the features in Table 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 PFSP problem spaces.

To explore the scalability of the learning models, a similar comparison to Sect. 5.2 is made for applying the learning models on their corresponding 10×10 testing data. Results are reported in Table 5. Note, that only resulting $C_{\max}$ is reported as the optimum makespan is not known and Eq. 5 is not applicable.

Fig. 4 Evolution of weights of features (given in Table 2) at each generation of the CMA-ES optimisation. Note, weights are normalised such that $||\mathbf{w}|| = 1$. **a** Minimise w.r.t. Eq. 6. **b** Minimise w.r.t. Eq. 7

Table 4 Main statistics of percentage relative deviation from optimality, ρ, defined by Eq. 5 for various models, using corresponding 6×5 training data

(a) $\mathscr{P}^{6\times5}_{j.rnd}$					
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) $\mathscr{P}^{6\times5}_{j.rndn}$					
$ES_{C_{\max}}$	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) $\mathscr{P}^{6\times5}_{f.rnd}$					
$ES_{C_{\max}}$	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) $\mathscr{P}^{6\times5}_{f.rndn}$					
$ES_{C_{\max}}$	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) $\mathscr{P}^{6\times5}_{f.jc}$					
$ES_{C_{\max}}$	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

6 Discussion and Conclusions

Data distributions considered in this study either varied w.r.t. the processing time distributions, continuing the preliminary experiments in [10] , or w.r.t. the job ordering permutations—i.e., homogeneous machine order for PFSP versus heterogeneous machine order for JSP. From the results based on 6×5 training data given in Table 4, it's obvious that CMA-ES optimisation substantially outperforms the previous PREF methods from [10] for all problem spaces considered. Furthermore, the results hold when testing on 10×10 (cf. Table 5), 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

Table 5 Main statistics of $C_{\max}$ for various models, using corresponding 10×10 test data

Model	Mean	Med	sd	Min	Max
(a) $\mathscr{P}_{j.rnd}^{10\times10}$					
$ES_{C_{\max}}$	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) $\mathscr{P}_{j.rndn}^{10\times10}$					
$ES_{C_{\max}}$	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) $\mathscr{P}_{f.rnd}^{10\times10}$					
$ES_{C_{\max}}$	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) $\mathscr{P}_{f.rndn}^{10\times10}$					
$ES_{C_{\max}}$	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) $\mathscr{P}_{f.jc}^{10\times10}$					
$ES_{C_{\max}}$	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

IV

fitness function directly and its normalisation w.r.t. true optimum (cf. Eqs. (6) and (7)), save for $\mathscr{P}_{f.rndn}$. 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 [11]. Thus normalising the objective function would help the evolutionary search to deviate from giving too much weight for problematic problem instances.

The main drawback of using evolutionary search for learning optimal weights for Eq. 1 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 h 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 $\mathscr{O}(n^{n \cdot m})$ 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 [9]. This could reduce the computational cost of the evolutionary search considerably, making it feasible to conduct the experiments from Sect. 5 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.

IV

References

1. Ak, B., Koc, E.: A guide for genetic algorithm based on parallel machine scheduling and flexible job-shop scheduling. Procedia—Soc. Behav. Sci. **62**, 817–823 (2012)
2. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)
3. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms-I. Representation. Comput. Ind. Eng. **30**(4), 983–997 (1996)
4. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Comput. Ind. Eng. **36**(2), 343–364 (1999)
5. Dhingra, A., Chandna, P.: A bi-criteria M-machine SDST flow shop scheduling using modified heuristic genetic algorithm. Int. J. Eng., Sci. Technol. **2**(5), 216–225 (2010)
6. Gurobi Optimization, Inc.: Gurobi optimization (version 5.6.2) [software]. http://www.gurobi.com/ (2013)
7. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
8. Haupt, R.: A survey of priority rule-based scheduling. OR Spectr. **11**, 3–16 (1989)
9. Ingimundardottir, H., Runarsson, T.P.: Sampling strategies in ordinal regression for surrogate assisted evolutionary optimization. In: 2011 11th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 1158–1163 (2011)
10. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: Coello, C. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6683, pp. 263–277. Springer, Heidelberg (2011)
11. Ingimundardottir, H., Runarsson, T.P.: Determining the characteristic of difficult job shop scheduling instances for a heuristic solution method. In: Hamadi, Y., Schoenauer, M. (eds.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, pp. 408–412. Springer, Heidelberg (2012)
12. Jayamohan, M., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. Eur. J. Oper. Res. **157**(2), 307–321 (2004)
13. Qing-dao-er ji, R., Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. Comput. Oper. Res. **39**(10), 2291–2299 (2012)
14. Koza, J.R., Poli, R.: Genetic programming. In: Burke, E., Kendal, G. (eds.) Introductory Tutorials in Optimization and Decision Support Techniques, chap. 5. Springer (2005)
15. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Learning iterative dispatching rules for job shop scheduling with genetic programming. Int. J. Adv. Manuf. Technol. (2013)
16. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Oper. Res. **25**(1), 45–61 (1977)
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3rd edn. Springer Publishing Company, Incorporated (2008)
18. Rice, J.R.: The algorithm selection problem. Adv. Comput. **15**, 65–118 (1976)
19. Smith-Miles, K., James, R., Giffin, J., Tu, Y.: A knowledge discovery approach to understanding relationships between scheduling problem structure and heuristic performance. In: Sttzle,

T. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 5851, pp. 89–103. Springer, Heidelberg (2009)
20. Smith-Miles, K., Lopes, L.: Generalising algorithm performance in instance space: a timetabling case study. In: Coello, C. (ed.) Learning and Intelligent Optimization. Lecture Notes in Computer Science, vol. 6683, pp. 524–538. Springer, Heidelberg (2011)
21. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput. Ind. Eng. **54**(3), 453–473 (2008)
22. Tsai, J.T., Liu, T.K., Ho, W.H., Chou, J.H.: An improved genetic algorithm for job-shop scheduling problems using Taguchi-based crossover. Int. J. Adv. Manuf. Technol. **38**(9–10), 987–994 (2007)
23. Vázquez-Rodríguez, J.A., Petrovic, S.: A new dispatching rule based genetic algorithm for the multi-objective job shop problem. J. Heuristics **16**(6), 771–793 (2009)
24. Watson, J.P., Barbulescu, L., Whitley, L.D., Howe, A.E.: Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. INFORMS J. Comput. **14**, 98–123 (2002)

IV

This page is intentionally left blank.

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

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

doi: 10.1007/978-3-319-19084-6_22

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

Generating Training Data for Learning Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir(✉) and Thomas Philip Rúnarsson

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

V

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

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

JSP deals with the allocation of tasks of competing resources where the goal is to minimise a schedule's maximum completion time, i.e., the makespan denoted $C_{\max}$. In order to find good solutions, heuristics are commonly applied in research, such as the simple priority based dispatching rules (SDR) from [11]. Composites of such simple rules can perform significantly better [6]. As a consequence, a linear composite of dispatching rules (LCDR) was presented in [3]. The goal there was to learn a set of weights, $\mathbf{w}$, via logistic regression such that

$$h(\mathbf{x}_j) = \left\langle \mathbf{w} \cdot \phi(\mathbf{x}_j) \right\rangle, \tag{1}$$

yields the preference estimate for dispatching job J_j that corresponds to post-decision state $\mathbf{x}_j$, where $\phi(\mathbf{x}_j)$ denotes its feature mapping. The job dispatched is the following,

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

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

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

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

Alternatively, training data could be generated using suboptimal solution trajectories. For instance [7] used decision trees to 'rediscover' largest processing time (LPT, a single priority based dispatching rule) by using LPT 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 heuristics that can outperform existing heuristics, then the training data needs to be correctly labelled. This drawback is confronted in [8,10,15] by using an optimal scheduler, computed off-line. In this study, we will both follow optimal and suboptimal solution trajectories, but for each partial solution the preference pair will be labelled correctly by solving the partial solution to optimality using a commercial software package [1]. For this study most work remaining (MWR), a promising SDR for the given data distributions [4], and the CMA-ES optimised LCDRs from [5] will be deemed worthwhile for generating suboptimal trajectories.

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

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

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

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

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

ϕ	Feature description
ϕ_1	Job processing time
ϕ_2	Job start-time
ϕ_3	Job end-time
ϕ_4	When machine is next free
ϕ_5	Current makespan
ϕ_6	Total work remaining for job
ϕ_7	Most work remaining for all jobs
ϕ_8	Total idle time for machine
ϕ_9	Total idle time for all machines
ϕ_{10}	ϕ_9 weighted w.r.t. number of assigned tasks
ϕ_{11}	Time job had to wait
ϕ_{12}	Idle time created
ϕ_{13}	Total processing time for job

V

1 Problem Space

In this study synthetic JSP data instances are considered with the problem size $n \times m$, where n and m denotes number of jobs and machines, respectively. Problem instances are generated stochastically. By fixing the number of jobs and machines while processing time are i.i.d. samples from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e., $p \sim \mathcal{U}(u_1, u_2)$. Two different processing time distributions are explored, namely $\mathcal{P}_{j.rnd}$ where $I = [1, 99]$ and $\mathcal{P}_{j.rndn}$ where $I = [45, 55]$ are referred to as random and random-narrow, respectively. The machine order is a random permutation of all of the machines in the job-shop.

For each data distribution N_{train} and N_{test} problem instances were generated for training and testing, respectively. Values for N are given in Table 1. Note, that difficult problem instances are not filtered out beforehand, such as the approach in [16].

2 JSP Tree Representation

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

machine, whilst still fulfilling constraints that each machine can handle, which is at most one job at each time, and jobs need to have finished their previous machines according to its machine order. Unfinished jobs, referred to as the job-list denoted $\mathcal{L}$, are dispatched one at a time according to a heuristic. After each dispatch, the schedule's current features are updated based on its resulting partial schedule. For each possible post-decision state the temporal features, $\mathcal{F}$, applied in this study are given in Table 2. These features are based on SDRs which are widespread in practice. For example if $\mathbf{w}$ is zero, save for $w_6 = 1$, then Eq. (1) gives $h(\mathbf{x}_j) > h(\mathbf{x}_i)$, $\forall i$ which are jobs with less work remaining than job J_j, namely Eq. (2) yields the job with the highest ϕ_6 value, i.e., equivalent to dispatching rule most work remaining (MWR).

Figure 1 illustrates how the first two dispatches could be executed for a 6×5 JSP with the machines $a \in \{M_1, ..., M_5\}$ on the vertical axis and the horizontal axis yields the current makespan, $C_{\max}$. The next possible dispatches are denoted as dashed boxes with the job index j within and its length corresponding to processing time p_{ja}. 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 (depicted solid) and one can see the resulting schedule (i.e., what are the next possible dispatches given this new scenario?). Finally, the bottom layer depicts all outcomes if job J_3 on machine M_3 would be dispatched. This sort of tree representation is similar to *game trees* [9] where the root node denotes the initial (i.e., empty) schedule and the leaf nodes denote the complete schedule. 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.

However, one can easily see that this sequence of task assignments is by no means unique. Inspecting a partial schedule further along in the dispatching process such as in Fig. 1 (top layer), then let's say J_1 would be dispatched next, and in the next iteration J_2. This sequence would yield the same schedule as if J_2 would have been dispatched first and then J_1 in the next iteration (since these are non-conflicting jobs). This indicates that some of the nodes in the tree can merge despite states of the partial schedules being different in previous layers. In this particular instance one can not infer that choosing J_1 is better and J_2 is worse (or vice versa) since they can both yield the same solution.

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, but varying permutations on the dispatching sequence (given the same partial initial sequence) can result in very different complete schedules with the same makespan, and thus same deviation from optimality, ρ defined by Eq. (4), which is the measure under consideration. Care must be taken in this case that neither resulting features are labelled as undesirable or suboptimal. Only the resulting features from a dispatch resulting in a suboptimal solution should be labelled undesirable.

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

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

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

The optimum makespan is known for each problem instance. At each time step (i.e., layer of the tree) a number of feature pairs are created. The feature pairs consist of the features ϕ_o resulting from optimal dispatches $o \in \mathcal{O}^{(k)}$, versus features ϕ_s resulting from suboptimal dispatches $s \in \mathcal{S}^{(k)}$ at time k. Note, $\mathcal{O}^{(k)} \cup \mathcal{S}^{(k)} = \mathcal{L}^{(k)}$ and $\mathcal{O}^{(k)} \cap \mathcal{S}^{(k)} = \emptyset$. In particular, each job is compared against another job from 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 in this study is to verify analytically, at each time step, whether it can indeed *somehow* yield an optimal schedule by manipulating the remainder of the sequence, while maintaining the current temporal schedule fixed as its initial state. 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 even 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.

V

3 Selecting Preference Pairs

At each dispatch iteration k, a number of preference pairs are created, which is then iterated over all N_{train} instances available. A separate data set is deliberately created for each dispatch iteration, as the initial feeling is that DRs 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 ℓ linear scheduling rules for solving a $n \times m$ job-shop specified by a set of preference pairs for each step,

$$S = \left\{ \{\phi_o - \phi_s, +1\}, \{\phi_s - \phi_o, -1\} \right\} \subset \Phi \times Y \tag{3}$$

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

3.1 Trajectory Sampling Strategies

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

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

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

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

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

V

3.2 Ranking Strategies

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

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

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

4 Experimental Study

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

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

which indicates the percentage relative deviation from optimality.

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

A linear ordinal regression model (PREF) was created for each preference set, S, for problem spaces $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$. A box-plot with the results of percentage relative deviation from optimality, ρ, defined by Eq. (4), is presented in Fig. 3. The box-plots are grouped w.r.t. trajectory strategies and colour-coded w.r.t. ranking schemes. Moreover, the simple priority dispatching rule MWR and the weights obtained by the CMA-ES optimisation used to obtain the training sets Φ^{mwr} and Φ^{cma} respectively are shown in black in the far left of the group for comparison. From Fig. 3 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. Main statistics are reported in Table 3a and b for $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$, respectively. Models are sorted w.r.t. mean relative error.

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

4.1 Ranking Strategies

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

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

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

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

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

4.2 Trajectory Sampling Strategies

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

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

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

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

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

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

V

It is particularly interesting there is no statistical difference between PREF^{opt} and PREF^{rnd} for both $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$ ranking-models. 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 where 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, PREF^{all} and PREF^{opt} gave the best combination for $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$. However, in the latter case PREF^{rnd} had the best mean relative error although not statistically different from PREF^{all} and PREF^{opt}.

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

4.3 Following CMA-ES Guided Trajectory

The rational for using the Φ^{cma} strategy was mostly due to the fact that a linear classifier created 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 is not the case since PREF^{cma} does not improve the

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

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

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

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

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

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

5 Summary and Conclusion

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

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

[2] 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.

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

References

1. Gurobi Optimization Inc: Gurobi optimization (version 5.6.2) [software] (2013). http://www.gurobi.com/
2. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
3. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: Coello, C.A.C. (ed.) LION 2011. LNCS, vol. 6683, pp. 263–277. Springer, Heidelberg (2011)
4. Ingimundardottir, H., Runarsson, T.P.: Determining the characteristic of difficult job shop scheduling instances for a heuristic solution method. In: Hamadi, Y., Schoenauer, M. (eds.) LION 2012. LNCS, vol. 7219, pp. 408–412. Springer, Heidelberg (2012)
5. Ingimundardottir, H., Runarsson, T.P.: Evolutionary learning of weighted linear composite dispatching rules for scheduling. In: International Conference on Evolutionary Computation Theory and Applications (ECTA) (2014)
6. Jayamohan, M., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. Eur. J. Oper. Res. **157**(2), 307–321 (2004)
7. Li, X., Olafsson, S.: Discovering dispatching rules using data mining. J. Sched. **8**, 515–527 (2005)
8. Malik, A.M., Russell, T., Chase, M., Beek, P.: Learning heuristics for basic block instruction scheduling. J. Heuristics **14**(6), 549–569 (2008)
9. von Neumann, J., Morgenstern, O.: Theory of Games and Economic Behavior (Commemorative Edition). Princeton University Press, Princeton Classic Editions, Princeton (2007)
10. Olafsson, S., Li, X.: Learning effective new single machine dispatching rules from optimal scheduling data. Int. J. Prod. Econ. **128**(1), 118–126 (2010)
11. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Oper. Res. **25**(1), 45–61 (1977)
12. Rosen, K.H.: Discrete Mathematics and its Applications, Chap. 9, 5th edn. McGraw-Hill Inc, New York (2003)
13. Ross, S., Bagnell, D.: Efficient reductions for imitation learning. In: Teh, Y.W., Titterington, D.M. (eds.) Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010), vol. 9, pp. 661–668 (2010). http://www.jmlr.org/proceedings/papers/v9/ross10a/ross10a.pdf
14. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Gordon, G.J., Dunson, D.B. (eds.) Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTAT 2011), vol. 15, pp. 627–635, Journal of Machine Learning Research - Workshop and Conference Proceedings (2011). http://www.jmlr.org/proceedings/papers/v15/ross11a/ross11a.pdf
15. Russell, T., Malik, A.M., Chase, M., van Beek, P.: Learning heuristics for the superblock instruction scheduling problem. IEEE Trans. Knowl. Data Eng. **21**(10), 1489–1502 (2009)
16. Watson, J.P., Barbulescu, L., Whitley, L.D., Howe, A.E.: Contrasting structured and random permutation flow-shop scheduling problems: search-space topology and algorithm performance. INFORMS J. Comput. **14**, 98–123 (2002)

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

VI

VI

Discovering Dispatching Rules From Data Using Imitation Learning

Helga Ingimundardóttir, Tómas Philip Rúnarsson

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

Journal of Scheduling – Submitted

Manuscript:
https://github.com/ALICE-InRu/Thesis/blob/master/papers/InRu15b.pdf

Submitted to *Journal of Scheduling* (2015).

Discovering dispatching rules from data using imitation learning

Case study for the job-shop problem

Helga Ingimundardottir · **Thomas Philip Runarsson**

Submitted: November 6, 2015

VI

Abstract Recent efforts to creating dispatching rules have focused on direct search methods and learning from scheduling data. This paper will examine the latter approach and present a systematic approach for doing so effectively. The key to learning an effective dispatching rule is through the careful construction of the training data, $\{\mathbf{x}_i(k), y_i(k)\}_{k=1}^{K} \in \mathscr{D}$, where: *i)* features of partially constructed schedules $\mathbf{x}_i$ should necessarily reflect the induced data distribution $\mathscr{D}$ for when the rule is applied. This is achieved by updating the learned model in an active imitation learning fashion; *ii)* y_i is labelled optimally using a MIP solver, and *iii)* data needs to be balanced, as the set is unbalanced w.r.t. dispatching step k.

When querying an optimal policy, there is an abundance of valuable information that can be utilised for learning new dispatching rules. For instance, it's possible to seek out when the scheduling process is most susceptible to failure. Generally stepwise optimality (or training accuracy) will imply good end performance, here minimising the makespan. However, as the impact of suboptimal moves is not fully understood, the labelling must be adjusted for its intended trajectory.

Using the guidelines set by the framework the design of custom dispatching rules, for one's particular scheduling application, will be more effective. In the study presented three different distributions of the job-shop will be considered. The machine learning approach considered is based on preference learning, which learns what post decision state is preferable to another. However, alternative learning methods may be applied to the training data generated.

Keywords Scheduling · Composite dispatching rules · Performance Analysis · Imitation Learning · DAgger · Preference Learning

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

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

1 Introduction

Hand crafting heuristics for scheduling is an ad-hoc approach to finding approximate solutions to problems. The practice is time-consuming and its performance can even vary dramatically between different problem instances. The aim of this work is to increase our understanding of this process. In particular the learning of new problem specific priority dispatching rules (DR) will be addressed for a subclass of scheduling problems known as the job-shop scheduling problem (JSP).

A recent editorial of the state-of-the-art approaches [6] in advanced dispatching rules for large-scale manufacturing systems reminds us that: "... most traditional dispatching rules are based on historical data. With the emergence of data mining and on-line analytic processing, dispatching rules can now take predictive information into account." The importance of automated discovery of dispatching rules was also emphasised by [24]. Data for learning can also be generated using a known heuristic on a set of problem instances. Such an approach is taken in [20] for single-machine where a decision tree is learned from the data to have similar logic to the guiding dispatching rule. However, the learned method cannot outperform the original dispatching rule used for the data generation. This drawback is confronted in

[22,35,25] by using an optimal scheduler or policy, computed off-line. The resulting dispatching rules, as decision trees, gave significantly better schedules than using popular heuristics in that field, and a lower worst-case factor from optimality. Although, using optimal policies for creating training data gives vital information on how to learn good scheduling rules an experimental study will show that this is not sufficient. Once these rules make a suboptimal dispatch then they are in uncharted territory and its effects are relatively unknown. This work will illustrate the sensitivity of learned dispatching rule's performance on the way the training data is sampled. For this purpose, JSP is used as a case study to illustrate a methodology for generating meaningful training data, which can be successfully learned using preference-based imitation learning (IL).

The competing alternative to learning dispatching rules from data would be to search the dispatching rule space directly. The prevalent approach in this case would be using an evolutionary algorithm, such as genetic programming (GP). The predominant approach in hyper-heuristics is a framework of creating new heuristics from a set of predefined heuristics via genetic algorithm optimisation [3]. Adopting a two-stage hyper-heuristic approach to generate a set of machine-specific DRs for dynamic job-shop, [27] used genetic programming (GP) to evolve CDRs from basic features, along with evolutionary algorithm to assign a CDR to a specific machine. The problem space consists of job-shops in semiconductor manufacturing, with additional shop constraints, as machines are grouped to similar work centres, which can have different set-up time, workload, etc. In fact, the GP emphasised on efficiently dispatching on the work centres with set-up requirements and batching capabilities, which are rules that are non-trivial to determine manually.

With meta heuristics one can use existing DRs, and use for example portfolio-based algorithm selection [29,9,37], either based on a single instance or class of instances to determine which DR to choose from. Implementing ant colony optimisation to select the best DR from a selection of nine DRs for JSP, experiments from [19] showed that the choice of DR do affect the results and that for all performance measures considered. They showed that it was better to have a all the DRs to choose from rather than just a single DR at a time. A simpler and more straightforward way to automate selection of composite priority dispatching rules (CDR), [13], translated dispatching rules into measurable features which describe the partial schedule and optimise directly what their contribution should be via evolutionary search.

Using case based reasoning for timetable scheduling, training data in [2] is guided by the two best heuristics in the literature. They point out that in order for their framework to be successful, problem features need to be sufficiently explanatory and training data needs to be selected carefully so they can suggest the appropriate solution for a specific range of new cases. When learning new dispatching rules there are several important factors to consider. First and foremost the context in which the training data is constructed will influence the quality of the learned dispatching rule [2]. Since the training data consists of collection of features, the quality of training data is interchangeable to the predictability of features. The training data is necessarily also problem instance specific. In addition to addressing these aspects, the paper will show that during the scheduling process, it will vary when it is most critical to make the 'right' dispatch. Furthermore, depending on the distribution of problem instances these critical moments can vary greatly. Moreover, a supervised learning algorithm will optimize classification accuracy, while it is the actual end-performance of the dispatching rule learned that will determine the success of the learning method.

The outline of the paper is the following, Section 2 gives the mathematical formalities of the scheduling problem, and Section 3 describes the main featues for job-shop, and illustrates how schedules are created with dispatching rules. Section 4 sets up the framework for learning from optimal schedules. In particular, the probability of choosing optimal decisions and the effects of making a suboptimal decision. Furthermore, the optimality of common single priority dispatching rules is investigated. With these guidelines, Section 5 goes into detail on how to create meaningful composite priority dispatching rules using preference learning, focusing on how to compare operations and collect training data with the importance of the sampling strategy applied. Sections 6 and 7 explain the trajectories for sampling meaningful schedules used in preference learning, either using passive or active imitation learning. Experimental results are jointly presented in Section 8 with comparison for a randomly generated problem space. Furthermore, some general adjustments for performance boost is also considered. The paper finally concludes in Section 9 with discussion and conclusions.

2 Job-shop Scheduling

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

This study will consider the $n \times m$ JSP, where n jobs, $\mathscr{J} = \{J_j\}_{j=1}^{n}$, are scheduled on a finite set, $\mathscr{M} = \{M_a\}_{a=1}^{m}$, of m machines. The index j refers to a job $J_j \in \mathscr{J}$ while the index a refers to a machine $M_a \in \mathscr{M}$. Each job requires a number of processing steps or operations, the pair (j,a) refers to the operation, i.e., processing the task of job J_j on machine M_a.

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

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

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

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

for all $J_j \in \mathscr{J}$ and $a \in \{2,..,m\}$. Note, that each job can have its own distinctive flow pattern through the machines, which is independent of the other jobs. However, in the case that all jobs share the same fixed permutation route, it is referred to as flow-shop (FSP). A commonly used subclass of FSP in the literature is permutation flow-shop, which has the added constraint that the processing order of the jobs on the machines must be identical as well, i.e., no passing of jobs allowed [36].

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

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

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

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

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

This family of scheduling problems is denoted by $J||C_{\max}$ [28]. Additional constraints commonly considered are job release-dates and due-dates or sequence dependent set-up times, however, these will not be considered here.

In order to find an optimal (or near optimal) solution for scheduling problems one could either use exact methods or heuristics methods. Exact methods guarantee an optimal solution. However, job-shop scheduling is strongly NP-hard [8]. Any exact algorithm generally suffers from the curse of dimensionality, which impedes the application in finding the global optimum in a reasonable amount of time. Using state-of-the-art software for solving scheduling problems, such as LiSA (A Library of Scheduling Algorithms) [1], which includes a specialised version of branch and bound that manages to find optimums for job-shop problems of up to 14×14 [34]. However, problems that are of greater size, become intractable. Heuristics are generally more time efficient but do not necessarily attain the global optimum. Therefore, job-shop has the reputation of being notoriously difficult to solve. As a result, it's been widely studied in deterministic scheduling theory and its class of problems has been tested on a plethora of different solution methodologies from various research fields [23], all from simple and straight forward dispatching rules to highly sophisticated frameworks.

3 Priority Dispatching Rules

Priority dispatching rules determine, from a list of incomplete jobs, $\mathscr{L}$, which job should be dispatched next. This process, where an example of a temporal partial schedule of six-jobs scheduled on five-machines, is illustrated in Fig. 1. The numbers in the boxes represent the job identification j. The width of the box illustrates the processing times for a given job for a particular machine M_a (on the vertical axis). The dashed boxes represent the resulting partial schedule for when a particular job is scheduled next. Moreover, the current $C_{\max}$ is denoted by a dotted vertical line. The object is to keep this value as small as possible once all operations are complete. As shown in the example there are 15 operations already scheduled. The *sequence* of dispatches used to create this partial schedule is:

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

This refers to the sequential ordering of job dispatches to machines, i.e., (j,a); the collective set of allocated jobs to machines is interpreted by its sequence which is referred to as a schedule. A scheduling policy will pertain to the manner in which the sequence is determined from the available jobs to be scheduled. In our example, the available jobs are given by the job-list $\mathscr{L}^{(k)} = \{J_1,J_2,J_4,J_5,J_6\}$ with the five potential jobs to be dispatched at step $k = 16$ (note that J_3 is completed).

However, deciding which job to dispatch is not sufficient as one must also know where to place it. In order to build tight schedules it is sensible to place a job as soon as it becomes available and such that the machine idle time is minimal, i.e., schedules are non-delay. There may also be a number of different options for such a placement. In Fig. 1 one observes that J_2, to be scheduled on M_3, could be placed immediately in a slot between J_3 and J_4, or after J_4 on this machine. If J_6 had been placed earlier, a slot would have been created between it and J_4, thus creating a third alternative, namely scheduling J_2 after J_6. The time in which machine M_a is idle between consecutive jobs J_j and $J_{j'}$ is called idle time or slack:

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

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

Construction heuristics are designed in such a way that it limits the search space in a logical manner respecting not to exclude the optimum. Here, the construction heuristic, Υ, is to schedule the dispatches as closely together as possible,

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

i.e., minimise the schedule's idle time. More specifically, once an operation (j,a) has been chosen from the job-list $\mathscr{L}$ by some dispatching rule, it can then be placed immediately after (but not prior) to $x_e(j,\boldsymbol{\sigma}_j(a-1))$ on machine M_a due to constraint Eq. (2). However, to guarantee that constraint Eq. (3) is not violated, idle times M_a are inspected as they create a slot which J_j can occupy. Bearing in mind that J_j release time is $x_e(j,\boldsymbol{\sigma}_j(a-1))$ one cannot implement Eq. (6) directly, instead it has to be updated as follows:

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

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

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

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

Priority dispatching rules will use features of operations, such as processing time, in order to determine the job with the highest priority. Consider again Fig. 1, if the job with the shortest processing time (SPT) were to be scheduled next, then J_2 would be dispatched. Similarly, for the longest processing time (LPT) heuristic, J_5 would have the highest priority. Dispatching can also be based on features related to the partial schedule. Examples of these are dispatching the job with the most work remaining (MWR) or alternatively the least work remaining (LWR). A survey of more than 100 of such rules are presented in [26]. However, the reader is referred to an in-depth survey for simple or *single priority dispatching rule* (SDR) by [12]. The SDRs assign an index to each job in the job-list and is generally only based on a few features and simple mathematical operations.

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

$\boldsymbol{\phi}$	Feature description	Mathematical formulation
	job related	
ϕ_1	job processing time	p_{ja}
ϕ_2	job start-time	$x_s(j,a)$
ϕ_3	job end-time	$x_e(j,a)$
ϕ_4	job arrival time	$x_e(j,a-1)$
ϕ_5	time job had to wait	$x_s(j,a) - x_e(j,a-1)$
ϕ_6	total processing time for job	$\sum_{a \in \mathscr{M}} p_{ja}$
ϕ_7	total work remaining for job	$\sum_{a' \in \mathscr{M} \setminus \mathscr{M}_j} p_{ja'}$
ϕ_8	number of assigned operations for job	$\lvert \mathscr{M}_j \rvert$
	machine related	
ϕ_9	when machine is next free	$\max_{j' \in \mathscr{J}_a} \{x_e(j',a)\}$
ϕ_{10}	total processing time for machine	$\sum_{j \in \mathscr{J}} p_{ja}$
ϕ_{11}	total work remaining for machine	$\sum_{j' \in \mathscr{J} \setminus \mathscr{J}_a} p_{j'a}$
ϕ_{12}	number of assigned operations for machine	$\lvert \mathscr{J}_a \rvert$
ϕ_{13}	change in idle time by assignment	$\Delta s(a,j)$
ϕ_{14}	total idle time for machine	$\sum_{j' \in \mathscr{J}_a} s(a,j')$
ϕ_{15}	total idle time for all machines	$\sum_{a' \in \mathscr{M}} \sum_{j' \in \mathscr{J}_{a'}} s(a',j')$
ϕ_{16}	current makespan	$\max_{(j',a') \in \mathscr{J} \times \mathscr{M}_{j'}} \{x_f(j',a')\}$

Designing priority dispatching rules requires recognising the important features of the partial schedules needed to create a reasonable scheduling rule. These features attempt to grasp key attributes of the schedule being constructed. Which features are most important will necessarily depend on the objectives of the scheduling problem. Features used in this study applied for each possible operation encountered are given in Table 1, where the set of machines already dispatched for J_j is $\mathscr{M}_j \subset \mathscr{M}$, and similarly, M_a has already had the jobs $\mathscr{J}_a \subset \mathscr{J}$ previously dispatched. The features of particular interest were obtained by inspecting the aforementioned SDRs. Features ϕ_1-ϕ_8 and ϕ_9-ϕ_{16} are job-related and machine-related, respectively. In fact, [27] note that in the current literature, there is a lack of global perspective in the feature space, as omitting them won't address the possible negative impact an operation (j,a) might have on other machines at a later time, it is for that reason features such as ϕ_{13}-ϕ_{15} are considered, since they are slack related and are a means of indicating the current quality of the schedule. All of the features, $\boldsymbol{\phi}$, vary throughout the scheduling process, w.r.t. operation belonging to the same time step k, with the

VI

exception of ϕ_6 and ϕ_{10} which are static for a given problem instance but varying for each J_j and M_a, respectively.

Priority dispatching rules are attractive since they are relatively easy to implement, perform fast, and find reasonable schedules. In addition, they are relatively easy to interpret, which makes them desirable for the end-user. However, they can also fail unpredictably. A careful combination of dispatching rules has been shown to perform significantly better [16]. These are referred to as *composite priority dispatching rules* (CDR), where the priority ranking is an expression of several dispatching rules. CDRs deal with a greater number of more complicated functions and are constructed from the schedules features. In short, a CDR is a combination of several DRs. For instance let π be a CDR comprised of d DRs, then the index I for $J_j \in \mathscr{L}^{(k)}$ using π is:

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

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

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

While investigating 11 SDRs for JSP, [21] a pool of 33 CDRs was created. This pool strongly outperformed the original CDRs by using multi-contextual functions based on either job waiting time or machine idle time (similar to ϕ_5 and ϕ_{14} in Table 1), i.e., the CDRs are a combination of either one or both of these key features and then the SDRs. However, there are no combinations of the basic SDRs explored, only those two features. Similarly, using priority rules to combine 12 existing DRs from the literature, [38] had 48 CDR combinations which yielded 48 different models to implement and test. It is intuitive to get a boost in performance by introducing new CDRs, since where one DR might be failing, another could be excelling, so combining them together should yield a better CDR. However, these approaches introduce fairly ad-hoc solutions and there is no guarantee the optimal combination of dispatching rules are found.

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

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

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

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

Similarly, single priority dispatching rules may be described by this linear model. For instance, let all $w_i = 0$, but with following exceptions: $w_1 = -1$ for SPT, $w_1 = +1$ for LPT, $w_7 = -1$ for LWR and $w_7 = +1$ for MWR. Generally, the weights $\mathbf{w}$ are chosen by the designer or the rule apriori. A more attractive approach would be to learn these weights from problem examples directly. The following section will investigate how this may be accomplished.

4 Performance Analysis of Priority Dispatching Rules

In order to create successful dispatching rules, a good starting point is to investigate the properties of optimal solutions and hopefully be able to learn how to mimic the construction of such solutions. For this, optimal solutions (obtained by using a commercial software package [10]) are followed and the probability of SDRs being optimal is inspected. This serves as an indicator of how hard it is to put our objective up as a machine learning problem. However, the end-goal, which is minimising deviation from optimality, ρ, must also take into consideration because of its relationship to stepwise optimality is not fully understood.

In this section the concerns of learning new priority dispatching rules will be addressed. At the same time experimental set-up used in the study are described.

4.1 Problem Instances

The class of problem instances used in our studies is the job-shop scheduling problem described in Section 2. Each instance will have different processing times and machine ordering. Each instance will therefore create different challenges for a priority dispatching rule. Dispatching rules learned will be customised for the problems used for their training. For real world application using historical data would be most appropriate. The aim would be to learn a dispatching rule that works well on average for a given distribution of problem instances. To illustrate the performance difference of priority dispatching rules on different problem distributions within the same class of problems, consider the following three cases. Problem instances for JSP are generated stochastically by fixing the number of jobs and machines to ten. A discrete processing time is sampled independently from a discrete uniform distribution from the interval $I = [u_1, u_2]$, i.e., $\mathbf{p} \sim \mathscr{U}(u_1, u_2)$. The machine order

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

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

is a random permutation of all of the machines in the job-shop. Two different processing times distributions were explored, namely $\mathscr{P}^{n\times m}_{j.rnd}$ where $I = [1,99]$ and $\mathscr{P}^{n\times m}_{j.rndn}$ where $I = [45,55]$. These instances are referred to as random and random-narrow, respectively. In addition, the case where the machine order is fixed and the same for all jobs, i.e. $\sigma_j(a) = a$ for all $J_j \in \mathscr{J}$ and where $\mathbf{p} \sim \mathscr{U}(1,99)$, is also considered. These jobs are denoted by $\mathscr{P}^{n\times m}_{f.rnd}$ and are analogous to $\mathscr{P}^{n\times m}_{j.rnd}$. The problem spaces are summarised in Table 2.

The goal is to minimise the makespan, $C_{\max}$. The optimum makespan is denoted $C^{\pi_\star}_{\max}$ (using the expert policy $\pi_\star$), and the makespan obtained from the scheduling policy π under inspection by $C^{\pi}_{\max}$. Since the optimal makespan varies between problem instances the performance measure is the following:

$$\rho = \frac{C^{\pi}_{\max} - C^{\pi_\star}_{\max}}{C^{\pi_\star}_{\max}} \cdot 100\% \tag{12}$$

which indicates the percentage relative deviation from optimality. Note: Eq. (12) measures the discrepancy between predicted value and true outcome, and is commonly referred to as a loss function, which should be minimised for policy π.

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

4.2 Reconstructing optimal solutions

When building a complete 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 that each machine can handle at most one job at each time, and jobs need to have finished their previous machines

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

Algorithm 1 Pseudo code for constructing a JSP sequence using a deterministic scheduling policy rule, π, for a fixed construction heuristic, Υ.

1: **procedure** SCHEDULEJSP(π, Υ)
2: $\boldsymbol{\chi} \leftarrow \emptyset$ ▷ initial current dispatching sequence
3: **for** $k \leftarrow 1$ **to** $K = n \cdot m$ **do** ▷ at each dispatch iteration
4: **for all** $J_j \in \mathscr{L}^{(k)} \subset \mathscr{J}$ **do** ▷ inspect job-list
5: $\boldsymbol{\phi}^j \leftarrow \boldsymbol{\phi} \circ \Upsilon\left(\boldsymbol{\chi}^j\right)$ ▷ temporal features for J_j
6: $I^{\pi}_j \leftarrow \pi\left(\boldsymbol{\phi}^j\right)$ ▷ priority for J_j
7: **end for**
8: $j^* \leftarrow \operatorname{argmax}_{j \in \mathscr{L}^{(k)}} \{I^{\pi}_j\}$ ▷ choose highest priority
9: $\chi_k \leftarrow J_{j^*}$ ▷ dispatch j^*
10: **end for**
11: **return** $C^{\pi}_{\max} \leftarrow \Upsilon(\boldsymbol{\chi})$ ▷ makespan and final schedule
12: **end procedure**

according to their machine order. Unfinished jobs from the job-list $\mathscr{L}$ are dispatched one at a time according to a deterministic scheduling policy (or heuristic). This process is given as a pseudo-code is given in Algorithm 1. After each dispatch[1] the schedule's current features are updated based on the half-finished schedule, $\boldsymbol{\chi}$. For each possible post-decision state the temporal features are collected (cf. Line 5) forming the feature set, Φ, based on all N_{train} problem instances available, namely:

$$\Phi := \bigcup_{\{\mathbf{x}_i\}_{i=1}^{N_{\text{train}}}} \left\{ \boldsymbol{\phi}^j \,:\, J_j \in \mathscr{L}^{(k)} \right\}_{k=1}^{K} \subset \mathscr{F} \tag{13}$$

where the feature space $\mathscr{F}$ is described in Table 1, and are based on job- and machine-features which are widespread in practice.

It is easy to see that the sequence of task assignments is by no means unique. Inspecting a partial schedule further along in the dispatching process such as in Fig. 1, then let's say J_1 would be dispatched next, and in the next iteration

[1] Dispatch and time step are used interchangeably.

J_2. Now this sequence would yield the same schedule as if J_2 would have been dispatched first and then J_1 in the next iteration, i.e., these are jobs with non-conflicting machines. In this particular scenario, one cannot infer that choosing J_1 is better and J_2 is worse (or vice versa) since they can both yield the same solution. Furthermore, there may be multiple optimal solutions to the same problem instance. Hence not only is the sequence representation 'flawed' in the sense that slight permutations on the sequence are in fact equivalent w.r.t. the end-result, but very varying permutations on the dispatching sequence (although given the same partial initial sequence) can result in very different complete schedules yet can still achieve the same makespan.

The redundancy in building optimal solutions using dispatching rules means that many different dispatches may yield an optimal solution to the problem instance. Let's formalise the probability of optimality (or stepwise classification accuracy) for a given policy π, as:

$$\xi^{\star}_{\pi} := \mathbb{E}_{\pi_\star}\{\pi_\star = \pi\} \tag{14}$$

that is to say the mean likelihood of our policy π being equivalent to the expert policy $\pi_\star$. The probability that a job chosen by a SDR yields an optimal makespan on a step-by-step basis, i.e., $\xi^{\star}_{\langle\text{SDR}\rangle}$, is depicted in Fig. 3. These probabilities vary quite a bit between the different problem instances distributions studied. From Fig. 3 it is observed that $\xi^{\star}_{\text{MWR}}$ has a higher probability than random guessing, in choosing a dispatch which may result in an optimal schedule. This is especially true towards the end of the schedule building process. Similarly, $\xi^{\star}_{\text{LWR}}$ chooses dispatches resulting in optimal schedules with a higher probability. This would appear to be support the idea that the higher the probability of dispatching jobs that may lead to an optimal schedule, the better the SDRs performance, as illustrated by Fig. 2. However, there is a counter example, $\xi^{\star}_{\text{SPT}}$ has a higher probability than random dispatching of selecting a jobs that may lead to an optimal solution. Nevertheless, the random dispatching performs better than SPT on problem instances $\mathscr{P}^{10\times10}_{j.rnd}$ and $\mathscr{P}^{10\times10}_{j.rndn}$.

Looking at Fig. 3, then $\mathscr{P}^{10\times10}_{j.rnd}$ has a relatively high probability (70% and above) of choosing an optimal job at random. However, it is imperative to keep making optimal decisions, because the consequences of making suboptimal dispatches are unknown. To demonstrate this Fig. 4 depicts mean worst and best case scenario of the resulting deviation from optimality, ρ, once off the optimal track, defined as follows:

$$\zeta^{\star}_{\min}(k) := \mathbb{E}_{\pi_\star}\left\{\min_{J_j\in\mathscr{L}^{(k)}}(\rho) \,:\, \forall C^{\boldsymbol{\chi}^j}_{\max} \gneq C^{\pi_\star}_{\max}\right\} \tag{15a}$$

$$\zeta^{\star}_{\max}(k) := \mathbb{E}_{\pi_\star}\left\{\max_{J_j\in\mathscr{L}^{(k)}}(\rho) \,:\, \forall C^{\boldsymbol{\chi}^j}_{\max} \gneq C^{\pi_\star}_{\max}\right\} \tag{15b}$$

Fig. 3: Probability of SDR being optimal, $\xi^{\star}_{\langle\text{SDR}\rangle}$

Fig. 4: Mean deviation from optimality, ρ, (%), for best and worst case scenario of making one suboptimal dispatch (i.e. $\zeta^{\star}_{\min}$ and $\zeta^{\star}_{\max}$), depicted as lower and upper bound, respectively, for $\mathscr{P}^{10\times10}_{j.rnd}$, $\mathscr{P}^{10\times10}_{j.rndn}$ and $\mathscr{P}^{10\times10}_{f.rnd}$. Moreover, mean suboptimal move is given as a dashed line.

Note, that this is given that there is only made one non-optimal dispatch. Generally, there will be more, and then the compound effects of making suboptimal decisions cumulate.

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

4.3 Blended dispatching rules

A naive approach to create a simple blended dispatching rule (BDR) would be to switch between SDRs at a prede-

Fig. 5: Box-plot for deviation from optimality, ρ, (%) for BDR where SPT is applied for the first 10%, 15%, 20%, 30% or 40% of the dispatches, followed by MWR

Fig. 6: Log likelihood of SDR being optimal for $\mathscr{P}_{j.rnd}^{10\times10}$, when following its corresponding SDR trajectory, i.e., $\log\left(\xi_{\langle\text{SDR}\rangle}\right)$

termined time. Observing again Fig. 3, a presumably good BDR for $\mathscr{P}_{j.rnd}^{10\times10}$ would be to start with $\xi^{\star}_{\text{SPT}}$ and then switch over to $\xi^{\star}_{\text{MWR}}$ at around time step $k = 40$, where the SDRs change places in outperforming one another. A box-plot for ρ for the BDR compared with MWR and SPT is depicted in Fig. 5 and its main statistics are reported in Table 3. This simple swap between SDRs does outperform the SPT heuristic, yet doesn't manage to gain the performance edge of MWR. Using SPT downgrades the performance of 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. 3 shows, the inherent structure that's already taking place might make it hard to come across by simple methods. Therefore, it's by no means guaranteed that by simply swapping over to MWR will handle that situation which applying SPT has already created. Figure 5 does however show, that by applying MWR instead of SPT in the latter stages, does help the schedule to be more compact w.r.t. SPT. However, the fact remains that the schedules have diverged too far from what MWR would have been able to achieve on its own.

In Fig. 3 the stepwise optimality was inspected, given that all committed dispatches were based on the optimal trajectory. As mistakes are bound to be made at some points, it is interesting to see how the stepwise optimality evolves for its intended trajectory, thereby updating Eq. (14) to:

$$\xi_{\pi} := \mathbb{E}_{\pi}\left\{\pi_{\star} = \pi\right\} \tag{16}$$

Figure 6 shows the log likelihood for $\xi_{\langle\text{SDR}\rangle}$ using $\mathscr{P}_{j.rnd}^{10\times10}$. There one can see that even though ξ_{SPT} is generally more likely to find optimal dispatches in the initial steps, then shortly after $k = 15$, ξ_{MWR} becomes a contender again. This could explain why our BDR switch at $k = 40$ from Fig. 5 was unsuccessful. However, changing to MWR at $k \leq 20$ is not statistically significant from MWR (boost in mean ρ is at most -0.5%). But as pointed out for Fig. 4, it's not so fatal to make bad moves in the very first dispatches for $\mathscr{P}_{j.rnd}^{10\times10}$, hence little gain with improved classification accuracy in that region. However, after $k > 20$ then the BDR performance starts diverging from that of MWR.

5 Preference Learning

Section 4.3 demonstrated there is something to be gained by trying out different combinations of DRs, however, it is non-trivial. In this section one approach to learning how such combinations is presented. Learning models considered in this study are based on ordinal regression in which the learning task is formulated as learning preferences. In the case of scheduling, learning which operations are preferred to others. Ordinal regression has been previously presented in [33] and in [14] for JSP, and given here for completeness.

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

The approach taken here is to verify analytically, at each time step, by fixing the current temporal schedule as an initial state, whether it is possible to 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 dis-

Table 3: Main statistics for $\mathscr{P}_{j.rnd}^{10\times 10}$ deviation from optimality, ρ, using BDR that changes from SDR at a fixed time step k.

SDR #1	SDR #2	k	Set	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
SPT	–	K	train	20.38	41.15	50.70	51.31	59.18	94.20
SPT	–	K	test	22.75	41.39	49.53	50.52	58.60	93.03
MWR	–	K	train	**4.42**	**17.84**	**21.74**	22.13	26.00	47.78
MWR	–	K	test	**3.37**	**17.07**	21.39	21.65	25.98	**41.80**
SPT	MWR	10	train	5.54	17.98	21.75	**21.99**	**25.43**	**44.02**
SPT	MWR	10	test	5.87	17.29	**20.78**	**21.28**	**24.67**	44.47
SPT	MWR	15	train	4.76	18.24	22.04	22.49	26.65	49.86
SPT	MWR	15	test	7.42	17.60	21.38	21.83	25.45	45.98
SPT	MWR	20	train	5.76	18.98	22.46	23.01	26.97	41.59
SPT	MWR	20	test	8.31	18.64	22.92	23.29	27.10	49.93
SPT	MWR	30	train	9.77	20.89	25.60	25.76	30.01	50.94
SPT	MWR	30	test	4.39	21.20	26.08	26.25	30.58	49.88
SPT	MWR	40	train	13.04	23.42	28.12	28.94	33.67	54.98
SPT	MWR	40	test	8.55	24.20	28.16	28.98	33.20	57.21

VI

patching sequence would have been chosen. That is to say the training data generation takes into consideration when there are multiple optimal solutions[2] to the same problem instance.

Let's label features from Eq. (13) that were considered optimal, $\boldsymbol{\psi}^o = \boldsymbol{\phi}^o - \boldsymbol{\phi}^s$, and suboptimal, $\boldsymbol{\psi}^s = \boldsymbol{\phi}^s - \boldsymbol{\phi}^o$ by $y_o = +1$ and $y_s = -1$ respectively. Then, the preference learning problem is specified by a set of preference pairs:

$$\Psi = \left\{ (\boldsymbol{\psi}^o, +1), (\boldsymbol{\psi}^s, -1) \,:\, \forall (J_o, J_s) \in \mathscr{O}^{(k)} \times \mathscr{S}^{(k)} \right\}_{k=1}^{K} \subset \Phi \times Y \tag{17}$$

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

To summarise, each job is compared against another job of the job-list, $\mathscr{L}^{(k)}$, and if the makespan differs (i.e $C_{\max}^{\pi_*}(\boldsymbol{\chi}^s) \gtrless C_{\max}^{\pi_*}(\boldsymbol{\chi}^o)$) an optimal/suboptimal pair is created. However, if the makespans are identical the pair is omitted since they give the same optimal makespan. This way, only features from a dispatch resulting in a suboptimal solution is labelled undesirable.

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

$$\boldsymbol{\chi}^i \succ \boldsymbol{\chi}^j \quad \Leftrightarrow \quad \pi(\boldsymbol{\chi}^i) > \pi(\boldsymbol{\chi}^j) \tag{18}$$

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

$$\pi(\boldsymbol{\chi}^j) = \sum_{i=1}^{d} w_i \phi_i(\boldsymbol{\chi}^j) = \langle \mathbf{w} \cdot \boldsymbol{\phi}(\boldsymbol{\chi}^j) \rangle. \tag{19}$$

[2] There can be several optimal solutions available for each problem instance. However, it is deemed sufficient to inspect only one optimal trajectory per problem instance as there are $N_{\text{train}} = 300$ independent instances which gives the training data variety.

Logistic regression learns the optimal parameters $\mathbf{w}^* \in \mathbb{R}^d$. For this study, L2-regularised logistic regression from the LIBLINEAR package [7] without bias is used to learn the preference set Ψ, defined by Section 5. Hence the job chosen to be dispatched, J_{j^*}, is the one corresponding to the highest preference estimate, i.e., Eq. (11) where $\pi(\cdot)$ is the classification model obtained by the preference set.

Preliminary experiments for creating step-by-step model was done in [14] resulting in local linear model for each dispatch; a total of K linear models for solving $n \times m$ JSP. However, the experiments there showed that by fixing the weights to its mean value throughout the dispatching sequence results remained satisfactory. A more sophisticated way would be to create a new linear model, where the preference set, Ψ, is the union of the preference pairs across the K dispatches, such as described in Section 5. This would amount to a substantial preference set, and for Ψ to be computationally feasible to learn, Ψ has to be reduced. For this several ranking strategies were explored in [15], the results there showed that it's sufficient to use partial subsequent rankings, namely, combinations of r_i and r_{i+1} for $i \in \{1, \ldots, n'\}$, are added to the preference set, where $r_1 > r_2 > \ldots > r_{n'}$ ($n' \leq n$) are the rankings of the job-list, in such a manner that in the cases that there are more than one operation with the same ranking, only one from that rank is needed to be compared to the subsequent rank. Moreover, for this study, which deals with 10×10 problem instances instead of 6×5, the partial subsequent ranking becomes necessary, as full ranking is computationally infeasible due to its size. Defining the size of the preference set as $l = |\Psi|$, then if l is too large re-sampling to size $l_{\max}$ may be needed to be done in order for the ordinal regression to be computationally feasible.

The training data from [14] was created from optimal solutions of randomly generated problem instances, i.e., traditional *passive imitation learning* (PIL). As JSP is a sequential decision making process, errors are bound to emerge. Due to compound effect of making suboptimal dispatches,

the model leads the schedule astray from learned feature-space, resulting in the new input being foreign to the learned model. Alternatively, training data could be generated using suboptimal solution trajectories as well, as was done in [15], where the training data also incorporated following the trajectories obtained by applying successful SDRs from the literature. The reasoning behind it was that they would be beneficial for learning, as they might help the model to escape from local minima once off the coveted optimal path. Simply aggregating training data obtained by following the trajectories of well-known SDRs yielded better models with lower deviation from optimality, ρ.

Inspired by the work of [30,31], the methodology of generating training data will now be such that it will iteratively improve upon the model, such that the feature-space learned will be representative of the feature-space the eventual model would likely encounter, known as DAgger for *active imitation learning* (AIL). Thereby, eliminating the ad-hoc nature of choosing trajectories to learn, by rather letting the model lead its own way in a self-perpetuating manner until it converges.

Furthermore, in order to boost training accuracy, two strategies were explored:

Boost.1 increasing number of preferences used in training (i.e. varying $l_{\max} \leq |\Psi|$),

Boost.2 introducing more problem instances (denoted EXT in experimental setting).

Note, the following experimental studies will address Boost.2, whereas preliminary experiments for Boost.1 showed no statistical significance in boost of performance. Hence, the default set-up will be $l_{\max} = 5 \cdot 10^5$ which is roughly the amount of features encountered from one pass of sampling a K-stepped trajectory using a fixed policy π for the default $N_{\text{train}} = 300$.

Another way to adjust training accuracy is to give different weight to various time steps. To address this problem, two different stepwise sampling biases (or data balancing techniques) will be considered:

Bias.1 **(equal)** where each time step has equal probability, this was used in [13,15] and serves as a baseline.

Bias.2 **(adjdbl2nd)** where each time step is adjusted to the number of preference pairs for that particular step (i.e. each step now has equal probability irrespective of quantity of encountered features). This is done with re-sampling. In addition, there is superimposed twice as much likelihood of choosing pairs from the latter half of the dispatching process. Then the final sampled data set is divided as follows: $|\{\Psi(k)\}_{k=0}^{\frac{K}{2}-1}| \approx \frac{1}{3} l_{\max}$ and $|\{\Psi(k)\}_{k=\frac{K}{2}}^{K-1}| \approx \frac{2}{3} l_{\max}$.

Remark: as the following sections require repeated collection of training data, and since its labelling is a very time intensive task the remainder of the paper will solely be focusing on $\mathscr{P}_{j.rnd}^{10\times 10}$.

6 Passive Imitation Learning

Using the terms from game-theory used in [4], then our problem is a basic version of the sequential prediction problem where the predictor (or forecaster), π, observes each element of a sequence $\boldsymbol{\chi}$ of jobs, where at each time step $k \in \{1, ..., K\}$, before the k-th job of the sequence is revealed, the predictor guesses its value χ_k on the basis of the previous $k-1$ observations.

6.1 Prediction with Expert Advice

Let us assume one knows the expert policy $\pi^\star$, which can query what is the optimal choice of $\chi_k = j^*$ at any given time step k. Now let's use Eq. (11) to back-propagate the relationship between post-decision states and $\hat{\pi}$ with preference learning via our collected feature set, denoted Φ^{OPT}, i.e., collecting the features set corresponding following optimal tasks J_{j^*} from $\pi^\star$ in Algorithm 1. This baseline sampling trajectory originally introduced in [14] for adding features to the feature set is a pure strategy where at each dispatch an optimal task is dispatched.

By querying the expert policy, $\pi_\star$, the ranking of the job-list, $\mathscr{L}$, is determined such that:

$$r_1 \succ r_2 \succ \cdots \succ r_{n'} \quad (n' \leq n) \tag{20}$$

implies r_1 is preferable to r_2, and r_2 is preferable to r_3, etc. In this study, then it's known that $r \propto C_{\max}^{\pi_\star}$, hence the optimal job-list is the following:

$$\mathscr{O} = \left\{ r_i \,:\, r_i \propto \min_{J_j \in \mathscr{L}} C_{\max}^{\pi_\star(\boldsymbol{\chi}^j)} \right\} \tag{21}$$

found by solving the current partial schedule to optimality using a MIP solver.

When $|\mathscr{O}^{(k)}| > 1$, there can be several trajectories worth exploring. However, only one is chosen at random. This is deemed sufficient as the number of problem instances, N_{train}, is relatively large.

6.2 Follow the Perturbed Leader

By allowing a predictor to randomise it's possible to achieve improved performance [4,11]. This is the inspiration for our next strategy called Follow the Perturbed Leader, denoted OPTε. Its pseudo code is given in Algorithm 2 and describes how the expert policy (i.e. optimal trajectory) from Section 6.1 is subtly "perturbed" with $\varepsilon = 10\%$ likelihood, by choosing a job corresponding to the second best $C_{\max}$ instead of a optimal one with some small probability.

Algorithm 2 Pseudo code for choosing job J_{j^*} following a perturbed leader.

Require: Ranking $r_1 \succ r_2 \succ \cdots > r_{n'}$ $(n' \leq n)$ of $\mathscr{L}$ ▷ query $\pi_\star$
procedure PERTURBEDLEADER($\mathscr{L}, \pi_\star$)
  $\varepsilon \leftarrow 0.1$ ▷ likelihood factor
  $p \leftarrow \mathscr{U}(0,1) \in [0,1]$ ▷ uniform probability
  $\mathscr{O} \leftarrow \{j \in \mathscr{L} : r_j = r_1\}$ ▷ optimal job-list
  $\mathscr{S} \leftarrow \{j \in \mathscr{L} : r_j > r_1\}$ ▷ sub-optimal job-list
  if $p < \varepsilon$ **and** $n' > 1$ **then**
    return $j^* \in \{j \in \mathscr{S} : r_j = r_2\}$ ▷ any second best job
  else
    return $j^* \in \mathscr{O}$ ▷ any optimal job
  end if
end procedure

Fig. 7: Box plot for $\mathscr{P}_{j.rnd}^{10\times10}$ deviation from optimality, ρ, using either expert policy and following perturbed leader.

6.3 Experimental study

Results for Sections 6.1 and 6.2 using $\mathscr{P}_{j.rnd}^{10\times10}$ box-plot of deviation from optimality, ρ, is given in Fig. 7 and main statistics are reported in Table 4. To address Boost.2, the extended training set was simply obtained by iterating over more examples, namely $N^{\text{OPT}}_{\text{train, EXT}} = 1000$. However, one can see that the increased number of varied features dissuades the preference models to achieving a good performance w.r.t. ρ. It's preferable to use the default $N^{\text{OPT}}_{\text{train}} = 300$ and allowing slight perturbations of the optimal trajectory, as done for $\Phi^{\text{OPT}\varepsilon}$. Unfortunately, all this overhead has not managed to surpass MWR in performance, except for $\Phi^{\text{OPT}\varepsilon}$ using Bias.2 with a $\Delta\rho \approx -4.24\%$ boost in mean performance. Otherwise, for Bias.1, there is a loss of $\Delta\rho \approx +6.23\%$ in mean performance. This is likely due to the fact that if equal probability is used for stepwise sampling, then there are hardly any emphasis given to the final dispatches as there a relatively few (compared to previous steps) preference pairs belonging to those final stages. Revisiting Fig. 4, then the band for $\{\zeta^\star_{\min}, \zeta^\star_{\max}\}$ is quite tight, as the problem is immensely constricted and few operations to choose from. However, the empirical evidence from using Bias.2 shows that it is imperative to make right decisions at the very end.

Based on the results from [14] the expert policy is a promising starting point. However, that was for 6×5 dimensionality (i.e. $K = 30$), which is a much simpler problem space. Notice that in Fig. 6 there was virtually no chance for $\xi_\pi(k)$ of choosing a job resulting in optimal makespan after step $k = 28$. Since job-shop is a sequential prediction problem, all future observations are dependent on previous operations. Therefore, learning sampled features that correspond only to optimal or near-optimal schedules isn't of much use when the preference model has diverged too far. Section 4.3 showed that good classification accuracy based on $\xi^\star_\pi$ does not necessarily mean a low mean deviation from optimality, ρ. This is due to the learner's predictions affects future input observations during its execution, which violates the crucial i.i.d. assumptions of the learning approach, and ignoring this interaction leads to poor performance. In fact, [30] proves that assuming the preference model has a training error of ε, then the total compound error (for all K dispatches) the classifier induces itself grows quadratically, $O(\varepsilon K^2)$, for the entire schedule, rather than having linear loss, $O(\varepsilon K)$, if it were i.i.d.

7 Active Imitation Learning

To amend performance from Φ^{OPT}-based models, suboptimal partial schedules were explored in [15] by inspecting the features from successful SDRs, $\Phi^{\langle\text{SDR}\rangle}$, by passively observing a full execution of following the task chosen by the corresponding SDR. This required some trial-and-error as the experiments showed that features obtained by SDR trajectories were not equally useful for learning.

To automate this process, inspiration from AIL presented in [31] is sought, called *Dataset Aggregation* (DAgger) method, which addresses a no-regret algorithm in an on-line learning setting. The novel meta-algorithm for IL learns a deterministic policy guaranteed to perform well under its induced distribution of states. The method is closely related to Follow-the-leader (cf. Section 6), however, with a more sophisticated leverage to the expert policy. In short, it entails the model π_i that queries an expert policy (same as in Section 6.1), $\pi_\star$, it's trying to mimic, but also ensuring the learned model updates itself in an iterative fashion, until it converges. The benefit of this approach is that the feature-states that are likely to occur in practice are also investigated and as such used to dissuade the model from making poor choices. In fact, the method queries the expert about the desired action at individual post-decision states which are both based on past queries, and the learner's interaction with the current environment.

DAgger has been proven successful on a variety of benchmarks [31,32], such as the video games Super Tux

Algorithm 3 Pseudo code for choosing job J_{j^*} using imitation learning (dependent on iteration i) to collect training set $\Phi^{\text{IL}i}$; either by following optimal trajectory, $\pi_\star$, or preference model from previous iterations, $\hat{\pi}_{i-1}$.

Require: $i \geq 0$
Require: Ranking $r_1 \succ r_2 \succ \cdots > r_{n'}$ $(n' \leq n)$ of $\mathscr{L}$ ▷ query $\pi_\star$
procedure ACTIVEIL$(i, \hat{\pi}_{i-1}, \pi_\star)$
  $p \leftarrow \mathscr{U}(0,1) \in [0,1]$ ▷ uniform probability
  if $i > 0$ **then** (unsupervised)
    $\beta_i \leftarrow 0$ ▷ always apply imitation
  else (fixed supervision)
    $\beta_i \leftarrow 1$ ▷ always follow expert policy (i.e. optimal)
  end if
  if $p > \beta_i$ **then**
    return $j^* \leftarrow \text{argmax}_{j \in \mathscr{L}} \{I_j^{\hat{\pi}_{i-1}}\}$ ▷ best job based on $\hat{\pi}_{i-1}$
  else
    $\mathscr{O} \leftarrow \{j \in \mathscr{L} : r_j = r_1\}$ ▷ optimal job-list
    return $j^* \in \mathscr{O}$ ▷ any optimal job
  end if
end procedure

Kart and Super Mario Bros., handwriting recognition and autonomous navigation for large unmanned aerial vehicles. In all cases greatly improving traditional supervised IL approaches.

7.1 DAgger

The policy of AIL at iteration $i > 0$ is a mixed strategy given as follows:

$$\pi_i = \beta_i \pi_\star + (1 - \beta_i)\hat{\pi}_{i-1} \tag{22}$$

where $\pi_\star$ is the expert policy and $\hat{\pi}_{i-1}$ is the learned model from the previous iteration. Note, for the initial iteration, $i = 0$, a pure strategy of $\pi_\star$ is followed. Hence, $\hat{\pi}_0$ corresponds to the preference model from Section 6.1 (i.e. $\Phi^{\text{IL}0} = \Phi^{\text{OPT}}$).

Equation (22) shows that β_i controls the probability distribution of querying the expert policy $\pi_\star$ instead of the previous imitation model, $\hat{\pi}_{i-1}$. The only requirement for $\{\beta_i\}_i^\infty$ according to [31] is that $\lim_{T\to\infty} \frac{1}{T}\sum_{i=0}^{T} \beta_i = 0$ to guarantee finding a policy $\hat{\pi}_i$ that achieves ε surrogate loss under its own state distribution limit.

Algorithm 3 explains the pseudo code for how to collect partial training set, $\Phi^{\text{IL}i}$ for i-th iteration of AIL. Subsequently, the resulting preference model, $\hat{\pi}_i$, learns on the aggregated datasets from all previous iterations, namely:

$$\Phi^{\text{DA}i} = \bigcup_{i'=0}^{i} \Phi^{\text{IL}i'} \tag{23}$$

and its update procedure is detailed in Algorithm 4.

Algorithm 4 DAgger: Dataset Aggregation for JSP

Require: $T \geq 1$
procedure DAGGER$(\pi_\star, \Phi^{\text{OPT}}, T)$
  $\Phi^{\text{IL}0} \leftarrow \Phi^{\text{OPT}}$ ▷ initialize dataset
  $\hat{\pi}_0 \leftarrow$ TRAIN$(\Phi^{\text{IL}0})$ ▷ initial model, equivalent to Section 6.1
  for $i \leftarrow 1$ **to** T **do** ▷ at each imitation learning iteration
    Let $\pi_i = \beta_i \pi_\star + (1 - \beta_i)\hat{\pi}_{i-1}$ ▷ Eq. (22)
    Sample K-step tracks using π_i ▷ cf. ACTIVEIL$(i, \hat{\pi}_{i-1}, \pi_\star)$
    $\Phi^{\text{IL}i} = \{(s, \pi_\star(s))\}$ ▷ visited states for π_i and actions by $\pi_\star$
    $\Phi^{\text{DA}i} \leftarrow \Phi^{\text{DA}i-1} \cup \Phi^{\text{IL}i}$ ▷ aggregate datasets, cf. Eq. (23)
    $\hat{\pi}_{i+1} \leftarrow$ TRAIN$(\Phi^{\text{DA}i})$ ▷ preference model from Eq. (10)
  end for
  return best $\hat{\pi}_i$ on validation ▷ best preference model
end procedure

7.2 Results

Due to time constraints, only $T = 3$ iterations will be inspected. In addition, preliminary experiments using DAgger for JSP favoured a simple parameter-free version of β_i in Eq. (22). Namely, the mixed strategy for $\{\beta_i\}_{i=0}^{T}$ is unsupervised with $\beta_i = I(i = 0)$, where I is the indicator function.[3]

Regarding Boost.2 strategy, Section 6 showed that adding new problem instances did not boost performance for the expert policy (which is equivalent for the initial iteration of DAgger). Hence, for active IL, the extended set is now consists of each iteration encountering N_{train} new problem instances. For a grand total of:

$$N_{\text{train, EXT}}^{\text{DA}i} = N_{\text{train}} \cdot (i + 1) \tag{24}$$

problem instances explored for the aggregated extended training set used for the learning model at iteration i. This way, the extended training data is used sparingly, as labelling for each problem instances is computationally intensive. As a result, the computational budget for DAgger is same regardless whether there are new problem instances used or not, i.e., $|\Phi^{\text{DA}i}| \approx |\Phi_{\text{EXT}}^{\text{DA}i}|$.

Results for $\mathscr{P}_{j.rnd}^{10\times10}$ box-plot of deviation from optimality, ρ, is given in Fig. 8 and main statistics are reported in Table 4. As one can see, DAgger is not fruitful when the same problem instances are continually used. This is due to the fact that there is not enough variance between $\Phi^{\text{IL}i}$ and $\Phi^{\text{IL}(i-1)}$, hence the aggregated feature set $\Phi^{\text{DA}i}$ is only slightly perturbed with each iterations. Which from Section 6.3 showed it was not a very successful modification for the expert policy. Although, it's noted that by introducing suboptimal feature-space the preference model is not as drastically bad as the extended optimal policy, even though $|\Phi^{\text{DA}i}| \approx |\Phi_{\text{EXT}}^{\text{OPT}}|$. However, when using new problem instances at each iterations, the feature set becomes varied enough that situations arise that can be learned to achieve a better represented classification problem which yields a lower mean deviation from optimality, ρ.

[3] $\beta_0 = 1$ and $\beta_i = 0, \forall i > 0$.

VI

Fig. 8: Box plot for $\mathscr{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using DAgger for JSP

Fig. 9: Box plot for $\mathscr{P}^{10\times10}_{j.rnd}$ deviation from optimality, ρ, using either expert policy, DAgger or following perturbed leader strategies. MWR shown for reference.

VI

8 Summary of Imitation Learning

A summary of $\mathscr{P}^{10\times10}_{j.rnd}$ best PIL and AIL models w.r.t. deviation from optimality, ρ, from Sections 6.3 and 7.2, respectively, are illustrated in Fig. 9, and main statistics are given in Table 4. To summarise, the following trajectories were used: *i)* expert policy, trained on Φ^{OPT}; *ii)* perturbed leader, trained on $\Phi^{\text{OPT}\varepsilon}$, and *iii)* imitation learning, trained on $\Phi^{\text{DA}i}_{\text{EXT}}$ for iterations $i = \{1,\ldots,3\}$ using extended training set. As a reference, the single priority dispatching rule MWR is shown at the edges of Fig. 9.

At first one can see that the perturbed leader ever so-slightly improves the mean for ρ, rather than using the baseline expert policy. However, AIL is by far the best improvement. With each iteration of DAgger, the models improve upon the previous iteration: *i)* for Bias.1 with Boost.2 then $i = 1$ starts with increasing $\Delta\rho \approx +1.39\%$. However, after that first iteration there is a performance boost of $\Delta\rho \approx -15.11\%$ after $i = 2$ and $\Delta\rho \approx -0.19\%$ for the final iteration $i = 3$, and *ii)* on the other hand when using Bias.2 with Boost.2, only one iteration is needed, as $\Delta\rho \approx -11.68$ for $i = 1$, and after that it stagnates with $\Delta\rho \approx +0.55\%$ for $i = 2$ and for $i = 3$ it is significantly worse than the previous iteration by $\Delta\rho \approx +0.75\%$. In both cases, DAgger outperforms MWR: *i)* after $i = 3$ iterations by $\Delta\rho \approx -5.31\%$ for Bias.1 with Boost.2, and *ii)* after $i = 1$ iteration by $\Delta\rho \approx -9.31\%$ for Bias.2 with Boost.2. Note, for Bias.1 without Boost.2, then DAgger is unsuccessful, and the aggregated data set downgrades the performance of the previous iterations, making it best to learn solely on the initial expert policy for that model configuration.

Regarding Boost.2, then it's not successful for the expert policy, as ρ increased approximately 10%. This could most likely be counter-acted by increasing $l_{\max}$ to reflect the 700 additional examples. What is interesting though, is that Boost.2 is well suited for AIL, using the same $l_{\max}$ as before. Note, the amount of problems used for $N^{\text{OPT}}_{\text{train, EXT}}$ is equivalent to $T = 2\frac{1}{3}$ iterations of extended DAgger. The new varied data gives the aggregated feature set more information of what is important to learn in subsequent iterations, as those new feature-states are more likely to be encountered 'in practice.' Not only does the AIL converge faster, it also consistently improves with each iterations.

9 Discussion and conclusions

The single priority dispatching rules remain a popular approach to scheduling, as they are simple to implement and quite efficient. Nevertheless, when they are successful and when they fail remains illusive. By inspecting optimal schedules, and investigating the probability that an optimal dispatch could be chosen by chance, and by looking at the impact of choosing sub-optimal dispatches, some light is shed on how SDRs vary in performance. Furthermore, the problem instance space was varied, giving an even better understanding of the behaviour of the SDRs. This analysis, however, also revealed that creating new dispatching rules from data is by no means trivial.

Experiments in Section 6.3 show that following the optimal policy is not without its faults. There are many obstacles to consider in order to improve model configurations. When training the learning model, there is a trade-off between making the over-all best decisions (in terms of highest mean validation accuracy) versus making the right decision on crucial time points in the scheduling process, as Fig. 4 clearly illustrated. Moreover, before training the learned model, the preference set Ψ needs to be re-sampled to size $l_{\max}$. As the effects of making suboptimal choices varies as a function of time, the stepwise bias should rather take into account the disproportional amount of features dur-

Table 4: Main statistics for $\mathscr{P}_{j.rnd}^{10\times10}$ deviation from optimality, ρ, using either expert policy, imitation learning or following perturbed leader strategies.

π^a	T^b	Bias	Set	N_{train}	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
OPT	0	adjdbl2nd	train	300	6.05	18.60	23.85	24.50	29.04	55.81
OPT	0	adjdbl2nd	test	300	5.56	19.16	24.24	25.19	30.42	55.52
OPT	0	equal	train	300	7.87	23.34	29.30	30.73	36.47	61.45
OPT	0	equal	test	300	8.31	23.88	30.32	31.46	37.70	67.24
DA1	1	adjdbl2nd	train	600	2.08	**9.44**	**12.30**	**12.82**	**15.67**	**29.63**
DA1	1	adjdbl2nd	test	300	**0.00**	**9.22**	**12.39**	**12.73**	**15.85**	35.17
DA1	1	equal	train	600	9.47	24.92	31.51	32.12	37.96	66.29
DA1	1	equal	test	300	4.77	23.77	30.34	31.40	37.81	73.73
DA2	2	adjdbl2nd	train	900	**0.93**	10.01	12.91	13.37	16.40	31.19
DA2	2	adjdbl2nd	test	300	0.39	9.84	13.13	13.44	16.62	**34.57**
DA2	2	equal	train	900	2.36	12.82	16.65	17.01	21.06	39.25
DA2	2	equal	test	300	1.72	12.57	16.38	16.89	20.66	42.44
DA3	3	adjdbl2nd	train	1200	0.93	10.45	13.71	14.12	17.15	32.91
DA3	3	adjdbl2nd	test	300	0.87	10.44	13.64	14.08	17.23	34.41
DA3	3	equal	train	1200	0.98	12.50	16.28	16.82	20.67	37.93
DA3	3	equal	test	300	0.26	12.32	16.01	16.52	20.22	41.62
OPTε	0	adjdbl2nd	train	300	4.64	13.63	17.56	18.07	21.66	36.25
OPTε	0	adjdbl2nd	test	300	1.91	13.18	16.48	16.89	20.28	35.60
OPTε	0	equal	train	300	4.52	21.31	27.63	28.04	33.69	63.74
OPTε	0	equal	test	300	8.54	22.03	27.26	27.94	33.02	60.38

[a] For DAgger, then $T = 0$ is conventional expert policy (i.e. DA0 = OPT).
[b] If $T = 0$ then *passive* imitation learning. Otherwise, for $T > 0$ it is considered *active* imitation learning.

VI

ing the dispatching process. As the experimental studies in Sections 6.3, 7.2 and 8 showed, instead of equal probability (i.e. Bias.1) it was much more fruitful to adjust the set to its number of preference and doubling the emphasis on the second half (i.e. Bias.2). However, there are many other stepwise sampling strategies based on our analysis that could have been chosen instead, as here only a simplification of the trend from Fig. 4 was chosen. This also opens up the question of how should validation accuracy be measured? Since the model is based on learning preferences, both based on optimal versus suboptimal, and then varying degrees of sub-optimality. Since ranks are only looked at in a black and white fashion, such that the makespans need to be strictly greater to belong to a higher rank, then it can be argued that some ranks should be grouped together if their makespans are sufficiently close. This would simplify the training set, making it (presumably) have less contradictions and be more appropriate for linear learning. Or simply the validation accuracy could be weighted w.r.t. the difference in makespan. During the dispatching process, there are some significant time points which need to be especially taken care off. Figure 4 showed how making suboptimal decisions is especially critical during the later stages for job-shop, whereas for flow-shop the earlier stages of dispatches are more critical.

Despite the information gathered by following an optimal trajectory, the knowledge obtained is not enough by itself. Since the learning model isn't perfect, it is bound to make a suboptimal dispatch eventually. When it does, the model is in uncharted territory as there is no certainty the samples already collected are able to explain the current situation. For this we propose investigating partial schedules from suboptimal trajectories as well, since the future observations depend on previous predictions. A straight forward approach would be to inspect the trajectories of promising SDRs or CDRs. However, more information is gained when applying AIL inspired by work of [30,31], such that the learned policy following an optimal trajectory is used to collect training data, and the learned model is iteratively updated. This can be done over several iterations, with the benefit being, that the scheduling features that are likely to occur in practice are investigated, and as such used to dissuade the model from making poor choices in the future.

The main drawback of DAgger is that it quite aggressively queries the expert, making it impractical for some problems, especially if it involves human experts. A way to confront that, [18,17] propose frameworks to minimise the expert's labelling effort. Or even circumvent the expert policy altogether by using a 'poorer' reference policy instead (i.e. $\pi_\star$ in Eq. (22) is suboptimal) [5].

This study has been structured around the job-shop scheduling problem, however, it can be easily extended to other types of deterministic optimisation problems that involve sequential decision making. The framework presented here collects snap-shots of the partial schedules by following an optimal trajectory, and verifying the resulting opti-

mal solution from each possible state. From which the stepwise optimality of individual features can be inspected, and its inference could for instance justify omittance in feature selection. Moreover, by looking at the best and worst case scenario of suboptimal dispatches, it is possible to pinpoint vulnerable times in the scheduling process.

References

1. Andresen, M., Engelhardt, F., Werner, F.: LiSA - A Library of Scheduling Algorithms (version 3.0) [software] (2010). URL `http://www.math.ovgu.de/Lisa.html`
2. Burke, E., Petrovic, S., Qu, R.: Case-based heuristic selection for timetabling problems. Journal of Scheduling **9**, 115–132 (2006)
3. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society **64**(12), 1695–1724 (2013)
4. Cesa-Bianchi, N., Lugosi, G.: Prediction, Learning, and Games, chap. 4. Cambridge University Press (2006)
5. Chang, K., Krishnamurthy, A., Agarwal, A., III, H.D., Langford, J.: Learning to search better than your teacher. In: Proceedings of The 32nd International Conference on Machine Learning, pp. 2058–2066 (2015)
6. Chen, T., Rajendran, C., Wu, C.W.: Advanced dispatching rules for large-scale manufacturing systems. The International Journal of Advanced Manufacturing Technology (2013)
7. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research **9**, 1871–1874 (2008)
8. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research **1**(2), 117–129 (1976)
9. Gomes, C.P., Selman, B.: Algorithm portfolios. Artificial Intelligence **126**(1-2), 43–62 (2001)
10. Gurobi Optimization, Inc.: Gurobi optimization (version 6.0.0) [software] (2014). URL `http://www.gurobi.com/`
11. Hannan, J.: Approximation to bayes risk in repeated play. Contributions to the Theory of Games **3**, 97–139 (1957)
12. Haupt, R.: A survey of priority rule-based scheduling. OR Spectrum **11**, 3–16 (1989)
13. Ingimundardottir, H., Runarsson, T.: Evolutionary learning of weighted linear composite dispatching rules for scheduling. In: International Conference on Evolutionary Computation Theory and Applications. SCITEPRESS (2014)
14. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 6683, pp. 263–277. Springer (2011)
15. Ingimundardottir, H., Runarsson, T.P.: Generating training data for learning linear composite dispatching rules for scheduling. In: Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 8994, pp. 236–248. Springer (2015)
16. Jayamohan, M., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. European Journal of Operational Research **157**(2), 307–321 (2004)
17. Judah, K., Fern, A., Dietterich, T.G.: Active imitation learning via reduction to I.I.D. active learning. CoRR **abs/1210.4876** (2012)
18. Kim, B., Pineau, J.: Maximum mean discrepancy imitation learning. In: Robotics: Science and Systems (2013)
19. Korytkowski, P., Rymaszewski, S., Wiśniewski, T.: Ant colony optimization for job shop scheduling using multi-attribute dispatching rules. The International Journal of Advanced Manufacturing Technology (2013)
20. Li, X., Olafsson, S.: Discovering dispatching rules using data mining. Journal of Scheduling **8**, 515–527 (2005)
21. Lu, M.S., Romanowski, R.: Multicontextual dispatching rules for job shops with dynamic job arrival. The International Journal of Advanced Manufacturing Technology (2013)
22. Malik, A.M., Russell, T., Chase, M., Beek, P.: Learning heuristics for basic block instruction scheduling. Journal of Heuristics **14**(6), 549–569 (2008)
23. Meeran, S., Morshed, M.: A hybrid genetic tabu search algorithm for solving job shop scheduling problems: a case study. Journal of intelligent manufacturing **23**(4), 1063–1078 (2012)
24. Mönch, L., Fowler, J.W., Mason, S.J.: Production Planning and Control for Semiconductor Wafer Fabrication Facilities, *Operations Research/Computer Science Interfaces Series*, vol. 52, chap. 4. Springer (2013)
25. Olafsson, S., Li, X.: Learning effective new single machine dispatching rules from optimal scheduling data. International Journal of Production Economics **128**(1), 118–126 (2010)
26. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Operations Research **25**(1), 45–61 (1977)
27. Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B.: Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. International Journal of Production Economics **145**(1), 67–77 (2013)
28. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3 edn. Springer (2008)
29. Rice, J.R.: The algorithm selection problem. Advances in Computers **15**, 65–118 (1976)
30. Ross, S., Bagnell, D.: Efficient reductions for imitation learning. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, vol. 9, pp. 661–668 (2010)
31. Ross, S., Gordon, G.J., Bagnell, D.: A reduction of imitation learning and structured prediction to no-regret online learning. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, vol. 15, pp. 627–635. Journal of Machine Learning Research - Workshop and Conference Proceedings (2011)
32. Ross, S., Melik-Barkhudarov, N., Shankar, K., Wendel, A., Dey, D., Bagnell, J., Hebert, M.: Learning monocular reactive uav control in cluttered natural environments. In: Robotics and Automation, 2013 IEEE Intl. Conference on, pp. 1765–1772 (2013)
33. Runarsson, T.: Ordinal regression in evolutionary computation. In: Parallel Problem Solving from Nature - PPSN IX, *Lecture Notes in Computer Science*, vol. 4193, pp. 1048–1057. Springer (2006)
34. Runarsson, T.P., Schoenauer, M., Sebag, M.: Pilot, rollout and monte carlo tree search methods for job shop scheduling. In: Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp. 160–174. Springer (2012)
35. Russell, T., Malik, A.M., Chase, M., van Beek, P.: Learning heuristics for the superblock instruction scheduling problem. IEEE Trans. on Knowl. and Data Eng. **21**(10), 1489–1502 (2009)
36. Stafford, E.F.: On the Development of a Mixed-Integer Linear Programming Model for the Flowshop Sequencing Problem. Journal of the Operational Research Society **39**(12), 1163–1174 (1988)
37. Xu, L., Hutter, F., Hoos, H., Leyton-Brown, K.: SATzilla-07: The design and analysis of an algorithm portfolio for SAT. Principles and Practice of Constraint Programming (2007)
38. Yu, J.M., Doh, H.H., Kim, J.S., Kwon, Y.J., Lee, D.H., Nam, S.H.: Input sequencing and scheduling for a reconfigurable manufacturing system with a limited number of fixtures. The International Journal of Advanced Manufacturing Technology (2013)

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.