Dissertation for the degree of doctor of philosophy

ALICE: Analysing & Learning in Complex Environments

Helga Ingimundardóttir

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

Prof. Darrell Whitley
Colorado State University

Prof. Mark Schoenauer
Director of research with INRIA Saclay – Île-de-France

ALICE: Analysing & Learning in Complex Environments

Printed in Iceland by Háskólaprent
ISBN 978-9979-9807-1-1

This is not for you

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

Ágrip

Ágrip á íslensku, lorem ipsum dolor sit amet, consectetuer adipiscing elit. Morbi commodo, ipsum sed pharetra gravida, orci magna rhoncus neque, id pulvinar odio lorem non turpis. Nullam sit amet enim. Suspendisse id velit vitae ligula volutpat condimentum. Aliquam erat volutpat. Sed quis velit. Nulla facilisi. Nulla libero. Vivamus pharetra posuere sapien. Nam consectetuer. Sed aliquam, nunc eget euismod ullamcorper, lectus nunc ullamcorper orci, fermentum bibendum enim nibh eget ipsum. Donec porttitor ligula eu dolor. Maecenas vitae nulla consequat libero cursus venenatis. Nam magna enim, accumsan eu, blandit sed, blandit a, eros.

Quisque facilisis erat a dui. Nam malesuada ornare dolor. Cras gravida, diam sit amet rhoncus ornare, erat elit consectetuer erat, id egestas pede nibh eget odio. Proin tincidunt, velit vel porta elementum, magna diam molestie sapien, non aliquet massa pede eu diam. Aliquam iaculis. Fusce et ipsum et nulla tristique facilisis. Donec eget sem sit amet ligula viverra gravida. Etiam vehicula urna vel turpis. Suspendisse sagittis ante a urna. Morbi a est quis orci consequat rutrum. Nullam egestas feugiat felis. Integer adipiscing semper ligula. Nunc molestie, nisl sit amet cursus convallis, sapien lectus pretium metus, vitae pretium enim wisi id lectus. Donec vestibulum. Etiam vel nibh. Nulla facilisi. Mauris pharetra. Donec augue. Fusce ultrices, neque id dignissim ultrices, tellus mauris dictum elit, vel lacinia enim metus eu nunc.

Contents

Listing of figures

Listing of tables

Listing of Publications

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

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

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

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

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

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

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

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 $\mathcal{A}$
Y	Mapping for algorithm and feature space onto performance space, i.e., $y = Y(a, \boldsymbol{\varphi}(\mathbf{x})) \in \mathcal{Y}$ where $Y : \mathcal{A} \times \mathcal{F} \mapsto \mathcal{Y}$

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
$s(a, j)$	flow between current and previous task on machine M_a

$C_{\max}$ 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

ℓ number of dispatches needed for a complete schedule, $\ell = n \cdot m$

Experimental Settings

S_b preference set added w.r.t. basic ranking

S_f preference set added w.r.t. full subsequent ranking

S_p preference set added w.r.t. partial subsequent ranking

S_a union of all aforementioned rankings, i.e., $S_a = S_b \cup S_f \cup S_p$

Φ^{SPT} training data guided by SPT trajectory

Φ^{LPT} training data guided by LPT trajectory

Φ^{LWR} training data guided by LWR trajectory

Φ^{MWR} training data guided by MWR trajectory

Φ^{OPT} training data guided by (random) optimum trajectory

Φ^{RND} training data guided by a random trajectory

Φ^{CMA} training data guided by trajectory using by CMA-ES obtained weights

Φ^{ALL} union of all aforementioned trajectories, i.e., $S^{ALL} = S^{SPT} \cup S^{LPT} \cup S^{LWR} \cup S^{MWR} \cup S^{OPT} \cup S^{RND} \cup S^{CMA}$

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

opt (known) optimum

sub sub-optimum

bks best known solution

Acronyms

JSP Job Shop scheduling problem

FSP Flow Shop scheduling problem

DR Dispatching rule

SDR Single priority dispatching rule

CDR Composite dispatching rule

BDR Blended dispatching rule

SPT Shortest Processing Time rule

LPT Largest Processing Time rule

LWR Least Work Remaining rule

MWR Most Work Remaining rule

ES Evolution Strategy

CMA Covariance Matrix Adaptation

PREF Linear preference learning model

Acknowledgements

This thesis owes its existence to the help, support, and inspiration of many. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Reykjavík, November 2015
Helga Ingimundardóttir

ACKNOWLEDGEMENTS

PROLOGUE

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

Alice

1 Something different

Off with their head! Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

$$\zeta = \frac{1039}{\pi} \tag{1.1}$$

1.1 Cleverref

For an example of a full page figure, see Fig. 1.3. Figure 1.1 on the other hand is a smaller figure, inline with text. The `cleverref` package is very convenient, as it uses the same formatting throughout, i.e., not *Figure* vs. *figure* inconsistencies. You can also refer to your papers, e.g. Paper I*, with their roman numeral. Always calling `\cref`.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

1.2 First Paragraph

And now I begin my first chapter here ...

Here is an equation**:

$$CIF: \qquad F^{j}_{\circ}(a) \quad = \quad \frac{1}{2\pi\imath} \oint_{\gamma} \frac{F^{j}_{\circ}(z)}{z-a} dz \tag{1.2}$$

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellen-

*Update `endmatter/papers.tex`, `frontmatter/papers.tex` and `papers/papers.bib` to your publications.

**the notation is explained in the nomenclature section

tesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

You can also include pseudocode, such as in Alg. 1. Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, com-

Algorithm 1 Euclid's algorithm

```
1: procedure EUCLID(a, b)                          ▷ The g.c.d. of a and b
2:     r ← a mod b
3:     while r ≠ 0 do                              ▷ We have the answer if r is 0
4:         a ← b
5:         b ← r
6:         r ← a mod b
7:     end while
8:     return b                                    ▷ The gcd is b
9: end procedure
```

modo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

1.3 Mad Hatter's Tea Party

There are many examples of Job-Shop Scheduling Problem (JSP) for real-world application. However, 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. 1.1. There are four guests attending, namely: 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 you mean, be it asking a riddle, telling a story, or singing a song to the group. Our 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 in alphabetical order. And each would rather wait than breaking their habit. Moreover, they tend to be very absent-minded so each task takes them a different amount of time. Let's assume that their processing times and ordering are given in Table 1.1.

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-

Table 1.1: Example of 4×5 Job-Shop

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

party, in order for Alice to be on-time?

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 the makespan, i.e., when Alice can leave.

Let's assume we've come to the party, after 10 operations have already been made (i.e.

Figure 1.1: The Mad Hatter's Tea Party, from Alice's Adventures in Wonderland by Carroll (1865). Illustration by John Tenniel (1820-1914).

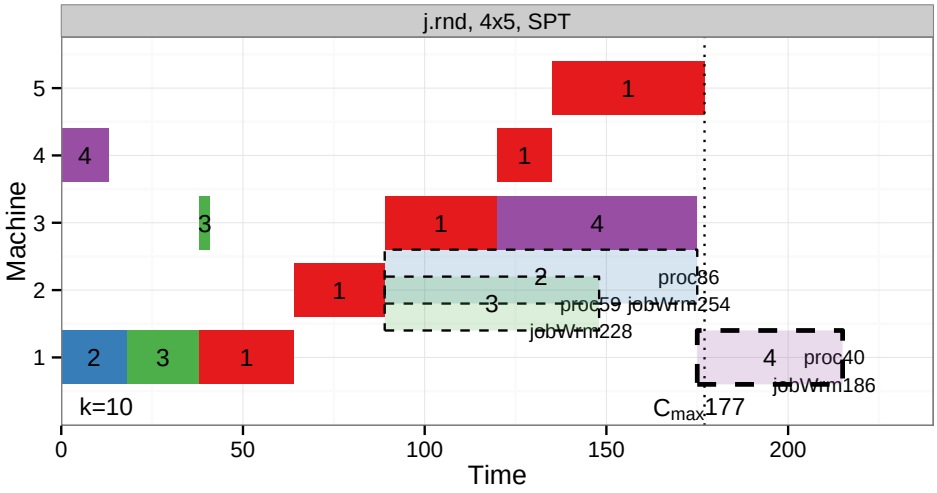


Figure 1.2: Gantt chart of a partial JSP schedule after 10 dispatches: Solid and dashed boxes represent $\boldsymbol{\chi}$ and $\mathcal{L}^{(11)}$, respectively. Current $C_{\max}$ denoted as dotted line.

strikeout entries in Table 1.1), by using the following job sequence,*

$$\boldsymbol{\chi} = \{\chi_i\}_{i=1}^{k-1} = \{J_4, J_2, J_3, J_3, J_1, J_1, J_1, J_1, J_1, J_4\} \tag{1.3}$$

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 1.1) to be dispatched, i.e., $\chi_k \in \mathcal{L}^{(k)}$.

Figure 1.2 illustrates the temporal partial schedule of the dispatching process 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 partial schedule for when a particular job is scheduled next, and *iv)* the current $C_{\max}$ is denoted with a dotted line.

*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 $J_1 \notin \mathcal{L}^{(11)}$.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Figure 1.3 ***(following page)*****:** This is a full page figure using the FPfigure command. It takes up the whole page and the caption appears on the preceding page. Its useful for large figures. Harvard's rules about full page figures are tricky, but you don't have to worry about it because we took care of it for you. For example, the full figure is supposed to have a title in the same style as the caption but without the actual caption. The caption is supposed to appear alone on the preceding page with no other text. You do't have to worry about any of that. We have modified the fltpage package to make it work. This is a lengthy caption and it clearly would not fit on the same page as the figure. Note that you should only use the FPfigure command in instances where the figure really is too large. If the figure is small enough to fit by the caption than it does not produce the desired effect. Good luck with your thesis. I have to keep writing this to make the caption really long. LaTex is a lot of fun. You will enjoy working with it. Good luck on your post doctoral life! I am looking forward to mine.

Figure 1.3: (continued)

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

Alice

References

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

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

REFERENCES

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

Alice

A

Test function suite

Test functions f used in **??** are defined $f : \mathbb{R}^d \mapsto \mathbb{R}$. All benchmark functions with the exception of **?** are described in **?**. They are summarized here for completeness. The original sources of the functions are also cited.

Citations are on https://notendur.hi.is/tpr/software/sres/testfcn.pdf

A.1 Sphere

Sphere function is a convex and unimodal function (cf. Fig. A.1a). Sphere function is defined as follows,

$$f_{\text{Sphere}}(\mathbf{x}) = \sum_{i=1}^{d} x_i^2 \tag{A.1}$$

where $\mathbf{x} \in [-3, 7]^d$. It has a global minimum at $\mathbf{x} = \mathbf{o}$ where $f_{\text{Sphere}}(\mathbf{o}) = 0$.

A.2 Noisy sphere

Noisy sphere function is the sphere function from Appendix A.1 where a Gaussian noise has added to perturb the sphere (cf. Fig. A.1b where $\varepsilon = 0.1$). Noisy sphere function is defined as follows,

$$f_{\text{NoisySphere}}(\mathbf{x}) = f_{\text{Sphere}}(\mathbf{x})\,(1 + \varepsilon\mathcal{N}(0,1)) \tag{A.2}$$

where $\mathbf{x} \in [-3,7]^d$. It has a global minimum at $\mathbf{x} = \mathbf{0}$ where $f_{\text{NoisySphere}}(\mathbf{0}) = 0$.

A.3 Schwefel

Schwefel is ... (cf. Fig. A.1c). Schwefel function is defined as follows,

$$f_{\text{Schwefel}}(\mathbf{x}) = \sum_{i=1}^{d} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{A.3}$$

where $\mathbf{x} \in [-10,10]^d$. It has a global minimum at $\mathbf{x} = \mathbf{0}$ where $f_{\text{Schwefel}}(\mathbf{0}) = 0$.

A.4 Ellipsoid

Ellipsoid is ... (cf. Fig. A.1d). Ellipsoid function is defined as follows,

$$f_{\text{Ellipsoid}}(\mathbf{x}) = \sum_{i=1}^{d} \left(100^{\frac{i-1}{d-1}} x_i \right)^2 \tag{A.4}$$

where $\mathbf{x} \in [-3,7]^d$. It has a global minimum at $\mathbf{x} = \mathbf{0}$ where $f_{\text{Ellipsoid}}(\mathbf{0}) = 0$.

A.5 Rosenbrock

Rosenbrock function is a non-convex function, where the global minimum is inside a long, narrow, parabolic shaped flat valley (cf. Fig. A.1e). To find the valley is trivial. To converge to the global minimum, however, is difficult. Rosenbrock function is defined as follows,

$$f_{\text{Rosenbrock}}(\mathbf{x}) = \sum_{i=1}^{d-1} \left(100 \cdot (x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \right) \tag{A.5}$$

where $\mathbf{x} \in [-5, 5]^d$. It has a global minimum at $\mathbf{x} = \mathbf{1}$ where $f_{\text{Rosenbrock}}(\mathbf{1}) = 0$.

A.6 Ackley

Ackley function is a non-convex function, where the function is a fairly difficult problem due to its large search space and its large number of local minima (cf. Fig. A.1f). Ackley function is defined as follows,

$$\begin{aligned} f_{\text{Ackley}}(\mathbf{x}) &= 20 - 20 \cdot \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{d} x_i^2}\right) \\ &\quad + e - \exp\left(\frac{1}{2}\sum_{i=1}^{d} \cos(2\pi x_i)\right) \end{aligned} \tag{A.6}$$

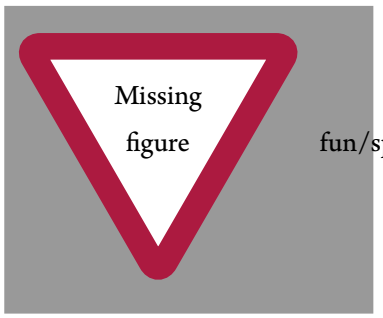
where $\mathbf{x} \in [1, 30]^d$. It has a global minimum at $\mathbf{x} = \mathbf{0}$ where $f_{\text{Ackley}}(\mathbf{0}) = 0$.

A.7 Rastrigin

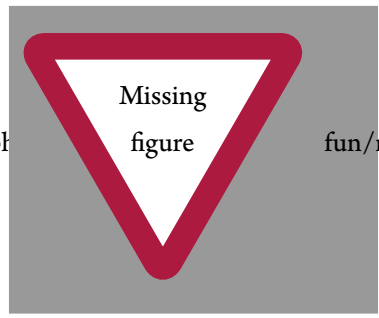
Rastrigin function is based on the sphere function from Appendix A.1 with the addition of cosine modulation in order to produce frequent local minima (cf. Fig. A.1g). Thus, the test function is highly multimodal. However, the location of the minima are regularly distributed. Rastrigin function is defined as follows,

$$f_{\text{Rastrigin}}(\mathbf{x}) = 10n + \sum_{i=1}^{d} \left(x_i^2 - 10\cos(2\pi x_i)\right) \tag{A.7}$$

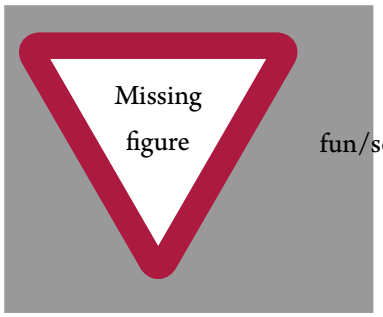
where $\mathbf{x} \in [1, 5]^d$. It has a global minimum at $\mathbf{x} = \mathbf{0}$ where $f_{\text{Rastrigin}}(\mathbf{0}) = 0$.



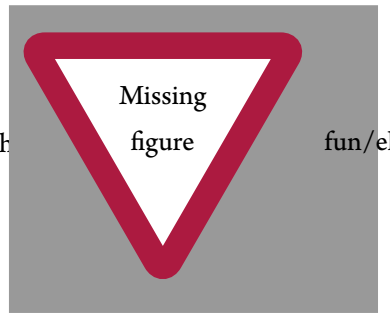
(a) Sphere function



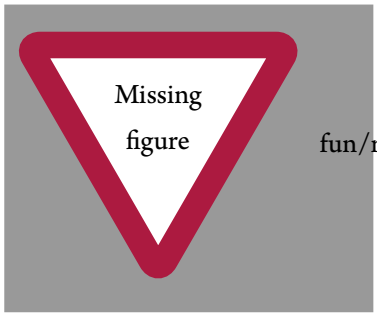
(b) Noisy sphere (with $\varepsilon = 0.1$)



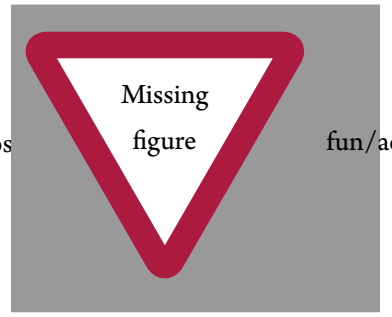
(c) Schwefel



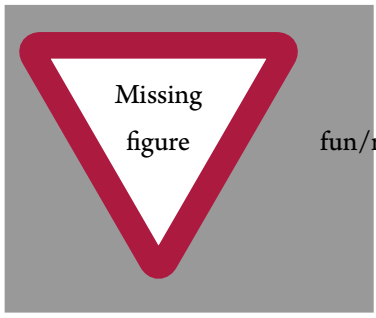
(d) Ellipsoid



(e) Rosenbrock



(f) Ackley



(g) Rastrigin

Figure A.1: Test function suite of two variables in 3D.

PAPERS

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

Alice

I

Supervised Learning Linear Priority Dispatch Rules for Job-Shop Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

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

Learning and Intelligent Optimization

I

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.

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.

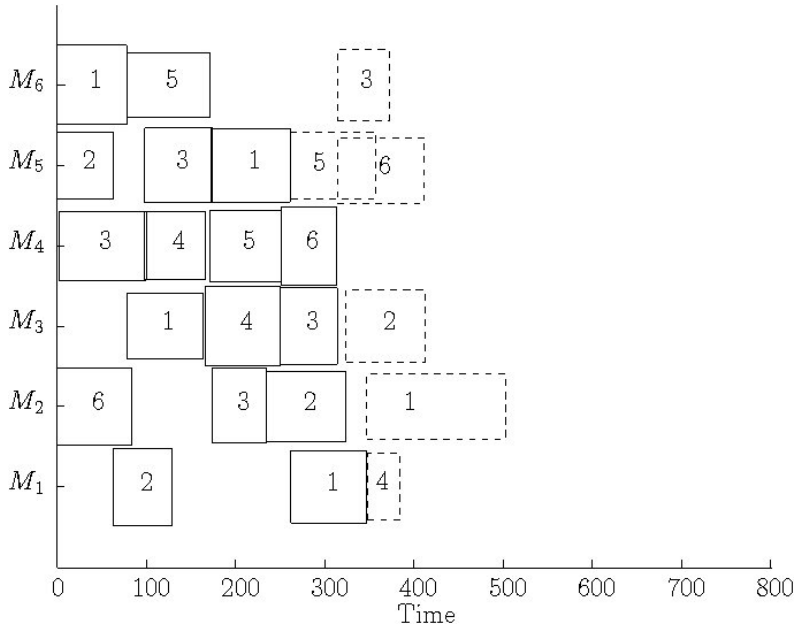


Fig. 1. A schedule being built, the dashed boxes represent six different possible jobs that could be scheduled next using a dispatch rule

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 ϕ 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

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

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.

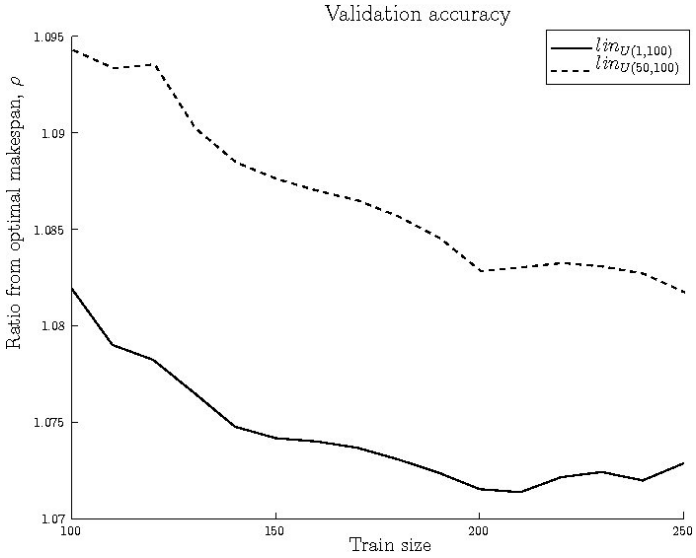


Fig. 2. Ratio from optimum makespan, ρ, for the validation set as a function of size of training set. Solid line represents model $lin_{U(1,100)}$ and dashed line represents model $lin_{U(50,100)}$

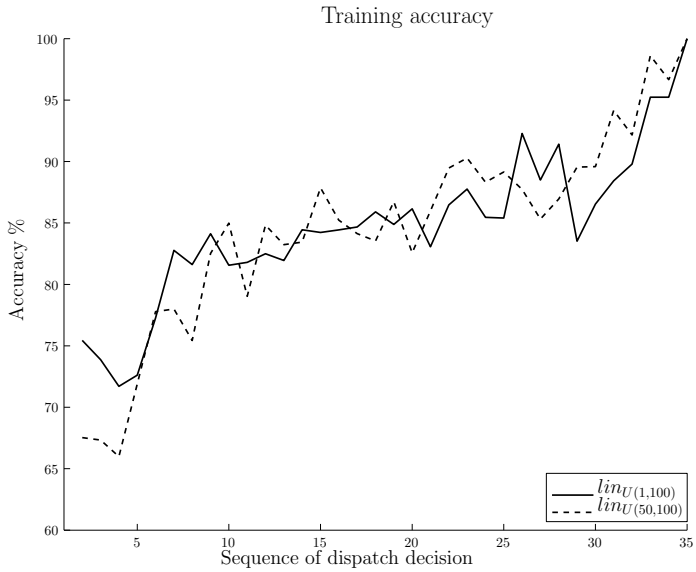


Fig. 3. Training accuracy as a function of sequence of dispatching decisions. Solid line represents model $lin_{U(1,100)}$ and dashed line represents data distributions $lin_{U(50,100)}$

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.

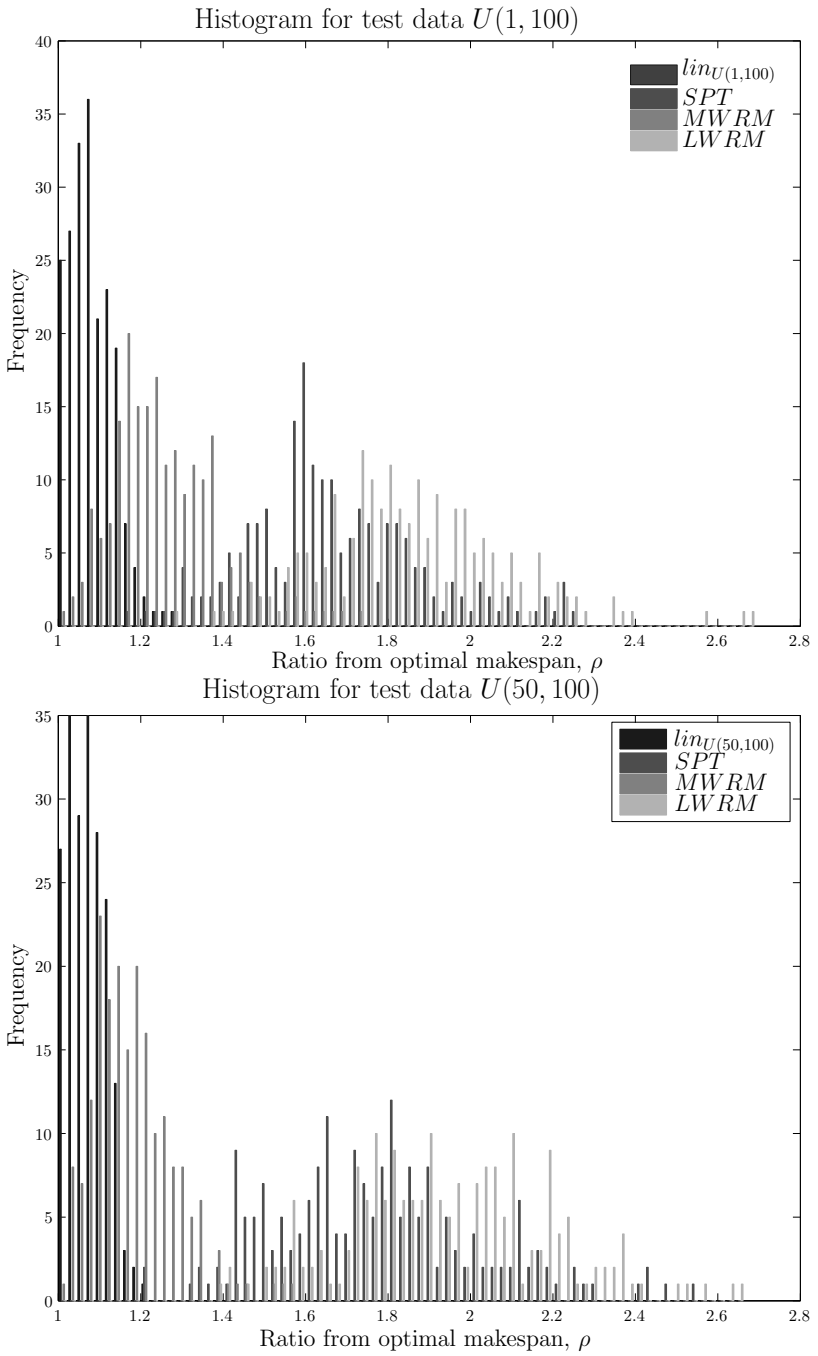


Fig. 4. Histogram of ratio ρ for the dispatching rules $lin_{U(R,100)}$, SPT, $MWRM$ and $LWRM$ for models $lin_{U(1,100)}$ (top) and $lin_{U(50,100)}$ (bottom)

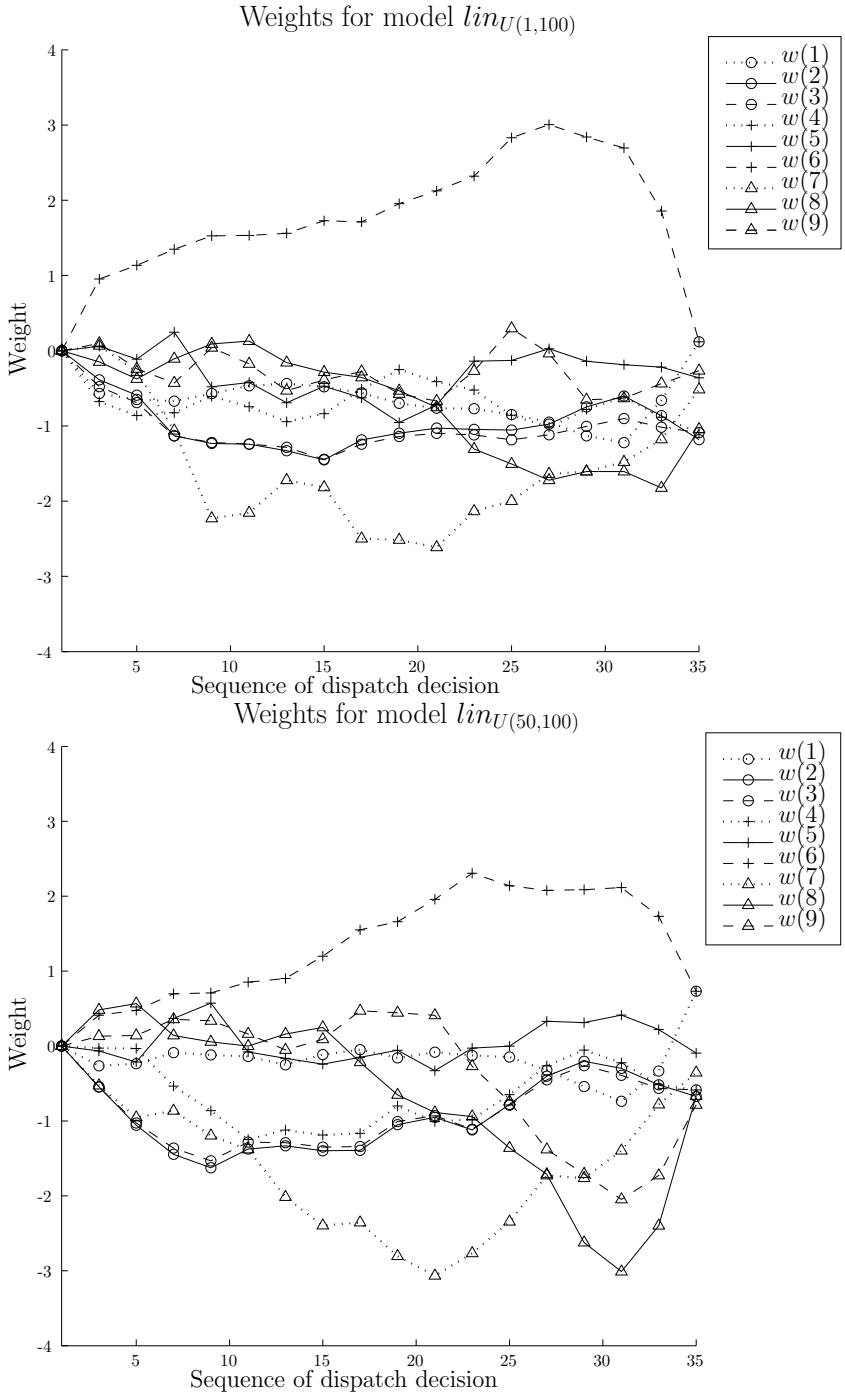


Fig. 5. Weights of features as a function of sequence of dispatching decisions, for test data $U(1, 100)$ (top) and $U(50, 100)$ (bottom)

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

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:

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

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:

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

subject to $t_k \left\langle \mathbf{w} \cdot (\phi(\mathbf{x}_k^{(1)}) - \phi(\mathbf{x}_k^{(2)}) \right\rangle \geq 1 - \xi_k$ 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

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

and that minimizing $\left\langle \mathbf{w} \cdot \mathbf{w} \right\rangle$ 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.

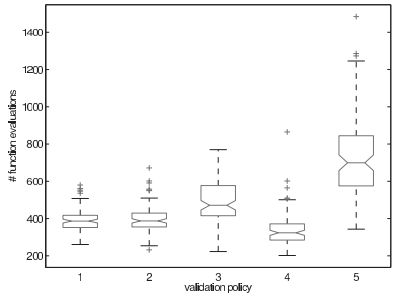


Fig. 1. Anova plot for different validation strategies: 1) prune old individuals, 2) prune bad individuals, 3) adding a pseudo mean candidate individual 4) correctly rank μ best ranked candidate individuals 5) update on every other generation for Rosenbrock's function for dimension $n = 2$

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

0 *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.
1 **for** $t := 1$ to λ **do** *(validate a test individual)*
2 Estimate ranking of $\mathcal{X}$ using h; denoted by $\bar{R}_0$.
3 $\mathbf{x}_B \leftarrow \max_{\mathbf{x}\in\mathcal{X}\setminus\mathcal{Y}} \{\bar{R}_0\}$ (*test individual*).
4 Rank $\mathbf{x}_B$ w.r.t. individuals in $\mathcal{Y}$ using h; denoted by $\bar{R}$.
5 Evaluate $\mathbf{x}_B$ using true fitness function and evaluate its true rank among individuals in $\mathcal{Y}$; denoted by R.
6 $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{\mathbf{x}_B\}$ *(add to training set)*.
7 Compare the rankings $\bar{R}$ and R by computing the rank correlation τ.
8 **if** $\tau > 0.999$ **then**
9 break (*model is sufficiently accurate*)
10 **fi**
11 Update the surrogate h using the new training set $\mathcal{Y}$.
12 **if** μ best individuals of $\bar{R}_0$ have been evaluated **then**
13 break (*model is sufficiently accurate*).
14 **fi**
15 **od**

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

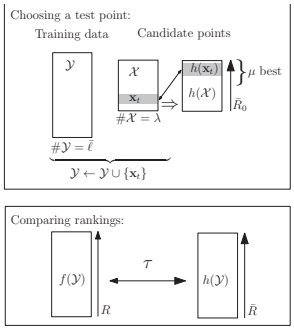


Fig. 3. Schema for the sampling strategy.

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

		Function eval.			Generations			Fitness		
	n	mean	median	sd	mean	median	sd	mean	median	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.

		Function eval.			Generations			Fitness		
	n	mean	median	sd	mean	median	sd	mean	median	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.

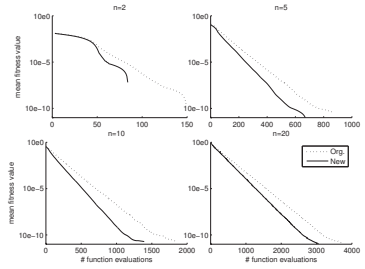


Fig. 4. Mean fitness values versus number of function evaluation by updating surrogate using all (dotted) or μ best (solid) individuals for sphere model.

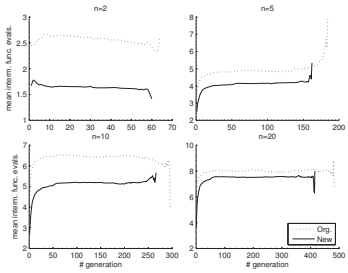


Fig. 5. Mean intermediate function evaluations versus generation by updating surrogate using all (dotted) or μ best (solid) individuals for sphere model.

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

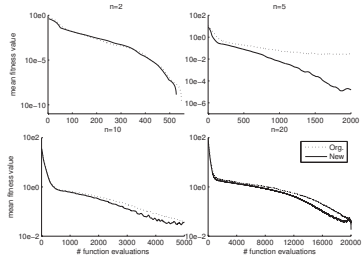


Fig. 6. Mean fitness values versus number of function evaluation by updating surrogate using all (dotted) or μ best (solid) individuals for Rosenbrock's function.

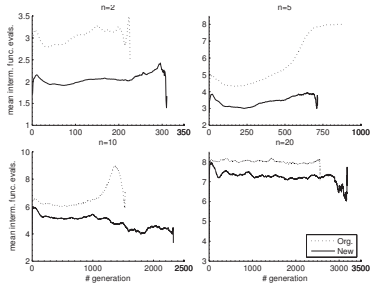


Fig. 7. Mean intermediate function evaluations versus generation by updating surrogate using all (dotted) or μ best (solid) individuals for Rosenbrock's function.

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.

II

This page is intentionally left blank.

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

Alice

III

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

Helga Ingimundardóttir, Tómas Philip Rúnarsson

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

Learning and Intelligent Optimization

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.

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 ·)

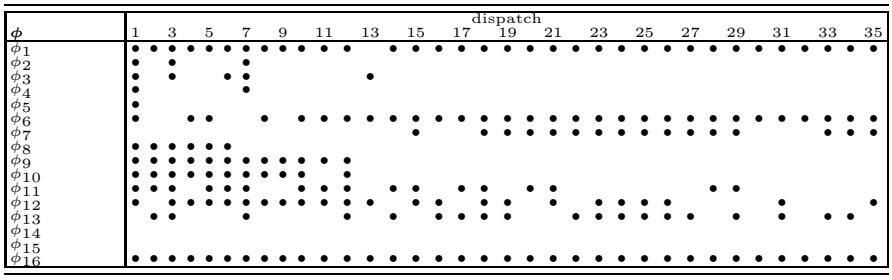


Table 3. Significant correlation (denoted by ·) for *easy* (left) and *hard* (right) problems and resulting ratio from optimality, ρ defined by (1). Commonly significant features across the tables are denoted by •.

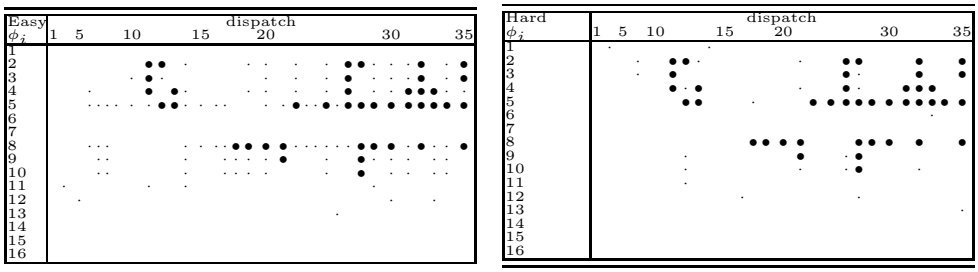


Table 2 reports where data distributions are the same (denoted by ·). From the table one can see that distribution for ϕ_1, ϕ_6, ϕ_{12} and ϕ_{16}) are (more or less) the same throughout the scheduling process. However there is a clear time of divergence for distribution of slacks; step 6 for ϕ_8 and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

Evolutionary Learning of Weighted Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

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

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

You can place todo's for papers in remark

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.

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 applying dispatching rules (DR), e.g., choosing a task corresponding to longest or shortest processing time, most or least successors, or ranked positional weight,

Helga Ingimundardottir
Industrial Eng., Mechanical Eng. and Computer Science, University of Iceland, e-mail: hei2@hi.is

Thomas Philip Runarsson
Industrial Eng., Mechanical Eng. and Computer Science, University of Iceland, e-mail: tpr@hi.is

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. Section 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 [3, 4, 22, 13, 1], 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) [23, 5, 15] 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 $\boldsymbol{\phi}(\mathbf{x}) = \{\phi_1(\mathbf{x}), ..., \phi_k(\mathbf{x})\} \in \mathscr{F}$ 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. [19, 20, 11] 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 Section 3; *b*) feature space $\mathscr{F}$, which is outlined in Section 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 Section 5; *d*) performance space is based on the resulting $C_{\max}$. Different fitness measures are investigated in Section 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.

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 Section 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 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].

[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 1 Problem space distributions used in Section 5. Note, problem instances are synthetic and each problem space is i.i.d. and '–' denotes not available.

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

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).

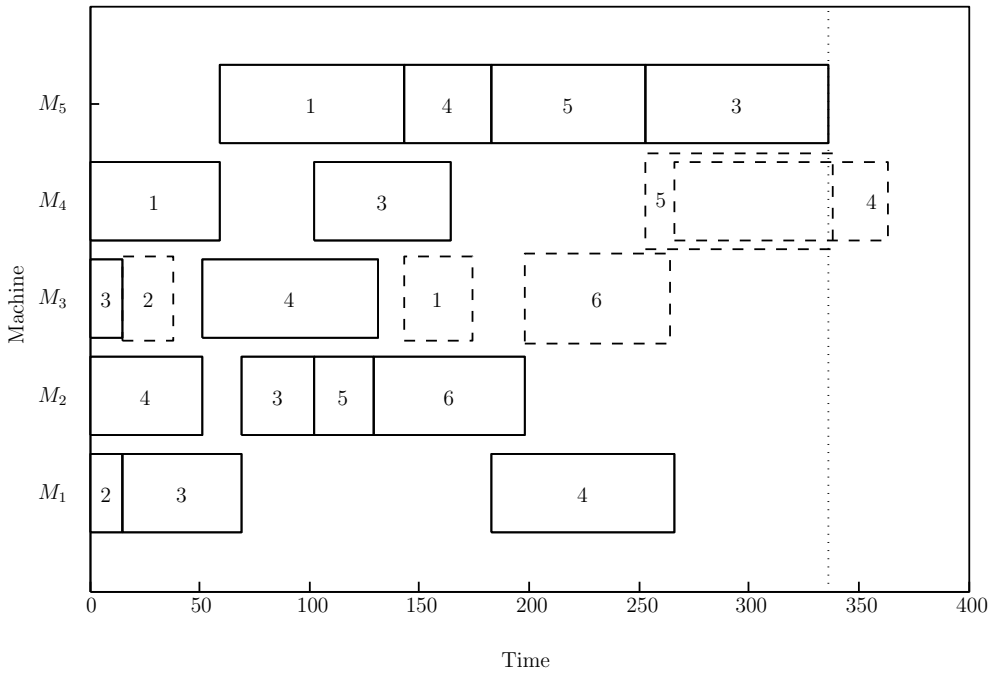


Fig. 1 Gantt chart of a partial JSP schedule after 15 operations: Solid boxes represent previously dispatched jobs, and dashed boxes represent the jobs that could be scheduled next. Current $C_{\max}$ denoted as dotted line.

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

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,

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

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 hours on a cluster for each training set given in Section 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. 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].

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. 4(a) 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.

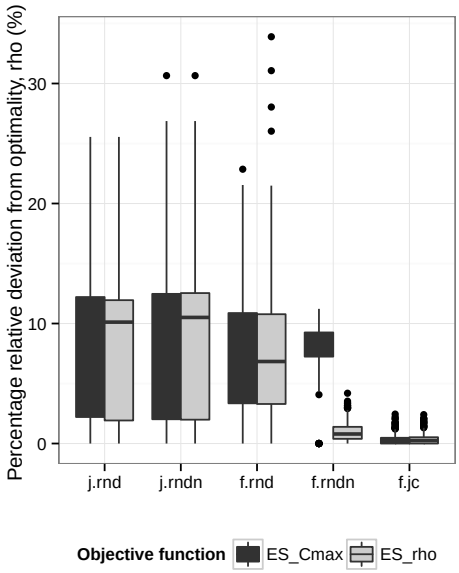


Fig. 2 Box-plot of training data for percentage relative deviation from optimality, defined by Eq. (5), when implementing the final weights obtained from CMA-ES optimisation, using both objective functions from Eqs. (6) and (7), left and right, respectively.

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. 4(b)), 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. $\phi_3 = \phi_1 + \phi_2$.

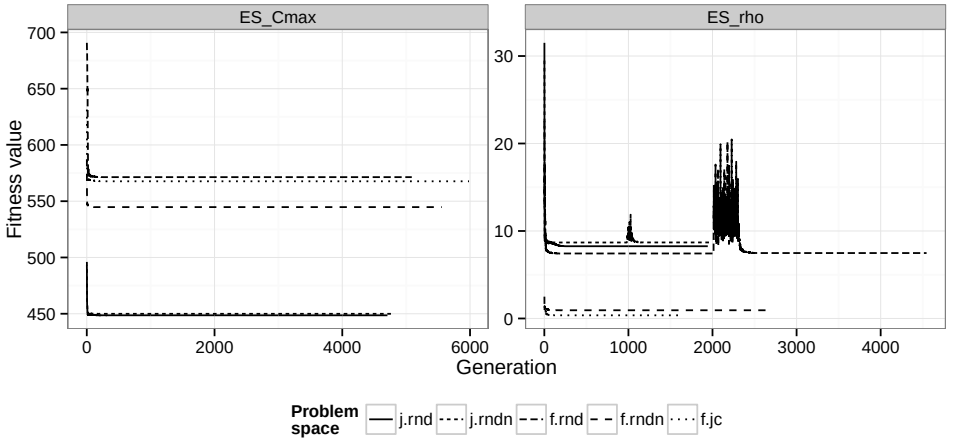


Fig. 3 Fitness for optimising (w.r.t. Eqs. (6) and (7) above and below, receptively), per generation of the CMA-ES optimisation.

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

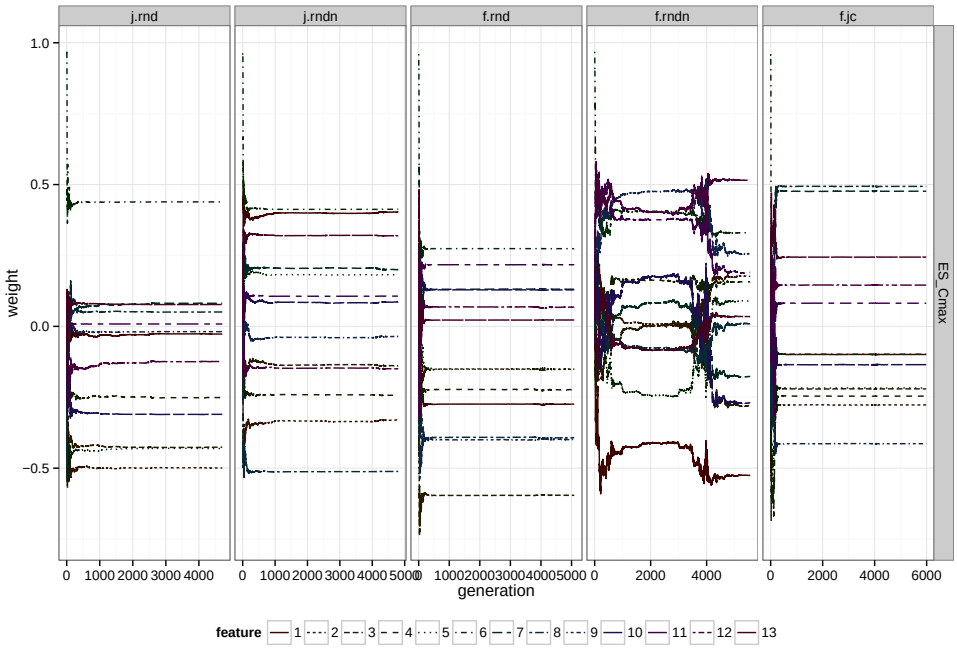
(b) w.r.t. Eq. (7)

$\mathscr{P}$	#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

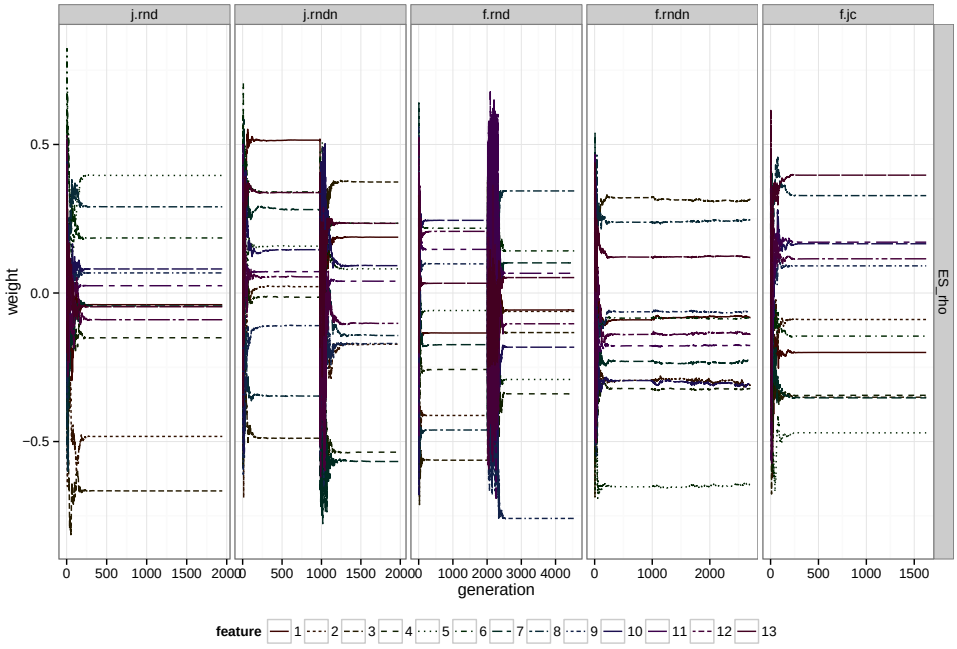
5.3 Scalability

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 Section 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.



(a) minimise w.r.t. Eq. (6)



(b) minimise w.r.t. Eq. (7)

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 $\|\mathbf{w}\| = 1$.

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) $\mathscr{P}^{6\times5}_{j.rndn}$

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

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

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

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

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

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

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

IV

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 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.

Table 5 Main statistics of $C_{\max}$ for various models, using corresponding 10×10 test data.

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

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

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

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

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

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

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

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

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

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

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 hours without converging. Now, 6×5 JSP requires 30 sequential operations where at each time step there are up to 6 jobs to choose from – i.e., its complexity is $\mathscr{O}(n^{n \cdot m})$ 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 Section 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.

References

1. Ak, B., Koc, E.: A Guide for Genetic Algorithm Based on Parallel Machine Scheduling and Flexible Job-Shop Scheduling. Procedia - Social and Behavioral Sciences **62**, 817–823 (2012)
2. 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)
3. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithmsI. Representation. Computers & Industrial Engineering **30**(4), 983–997 (1996)
4. Cheng, R., Gen, M., Tsujimura, Y.: A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. Computers & Industrial Engineering **36**(2), 343–364 (1999)
5. Dhingra, A., Chandna, P.: A bi-criteria M-machine SDST flow shop scheduling using modified heuristic genetic algorithm. International Journal of Engineering, Science and Technology **2**(5), 216–225 (2010)
6. Gurobi Optimization, Inc.: Gurobi optimization (version 5.6.2) [software] (2013). URL http://www.gurobi.com/
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 Spectrum **11**, 3–16 (1989)
9. Ingimundardottir, H., Runarsson, T.P.: Sampling strategies in ordinal regression for surrogate assisted evolutionary optimization. In: Intelligent Systems Design and Applications (ISDA), 2011 11th International Conference on, pp. 1158–1163 (2011)
10. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: C. Coello (ed.) Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 6683, pp. 263–277. Springer, Berlin, Heidelberg (2011)
11. Ingimundardottir, H., Runarsson, T.P.: Determining the characteristic of difficult job shop scheduling instances for a heuristic solution method. In: Y. Hamadi, M. Schoenauer (eds.) Learning and Intelligent Optimization, Lecture Notes in Computer Science, pp. 408–412. Springer, Berlin, Heidelberg (2012)
12. 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)
13. Qing-dao-er ji, R., Wang, Y.: A new hybrid genetic algorithm for job shop scheduling problem. Computers & Operations Research **39**(10), 2291–2299 (2012)
14. Koza, J.R., Poli, R.: Genetic programming. In: E. Burke, G. Kendal (eds.) Introductory Tutorials in Optimization and Decision Support Techniques, chap. 5. Springer (2005)
15. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Learning iterative dispatching rules for job shop scheduling with genetic programming. The International Journal of Advanced Manufacturing Technology (2013)
16. Panwalkar, S.S., Iskander, W.: A survey of scheduling rules. Operations Research **25**(1), 45–61 (1977)
17. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems, 3 edn. Springer Publishing Company, Incorporated (2008)
18. Rice, J.R.: The algorithm selection problem. Advances in Computers **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: T. Sttzle (ed.) Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 5851, pp. 89–103. Springer, Berlin, Heidelberg (2009)
20. Smith-Miles, K., Lopes, L.: Generalising algorithm performance in instance space: A timetabling case study. In: C. Coello (ed.) Learning and Intelligent Optimization, *Lecture Notes in Computer Science*, vol. 6683, pp. 524–538. Springer, Berlin, Heidelberg (2011)
21. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers and Industrial Engineering **54**(3), 453–473 (2008)

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. The International Journal of Advanced Manufacturing Technology **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. Journal of 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 Journal on Computing **14**, 98–123 (2002)

IV

This page is intentionally left blank.

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

The Mock Turtle

V

Generating Training Data for Learning Linear Composite Dispatching Rules for Scheduling

Helga Ingimundardóttir, Tómas Philip Rúnarsson

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

Learning and Intelligent Optimization – Nominated for best paper award

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,

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

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.

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

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.

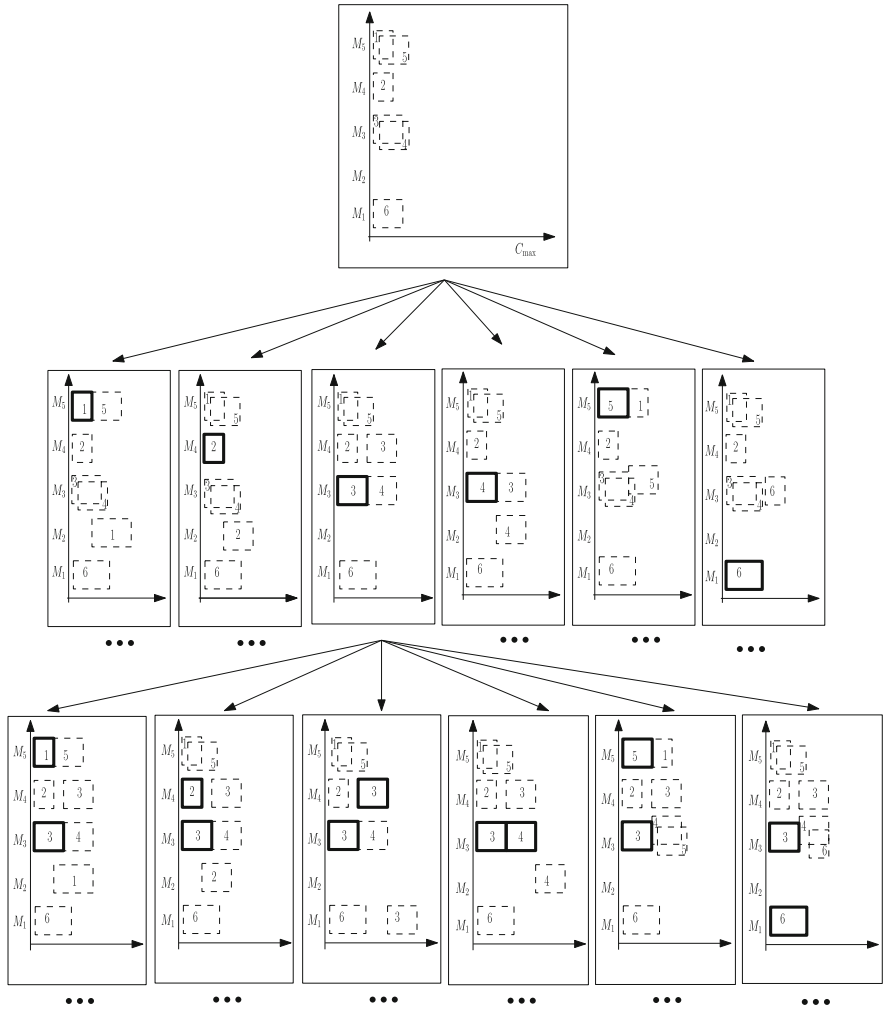


Fig. 1. Partial Tree for JSP for the first two dispatches. Executed dispatches are depicted solid, and all possible dispatches are dashed.

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., $C_{\max}^{(s)} \gtrless C_{\max}^{(o)}$, 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.

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.

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.

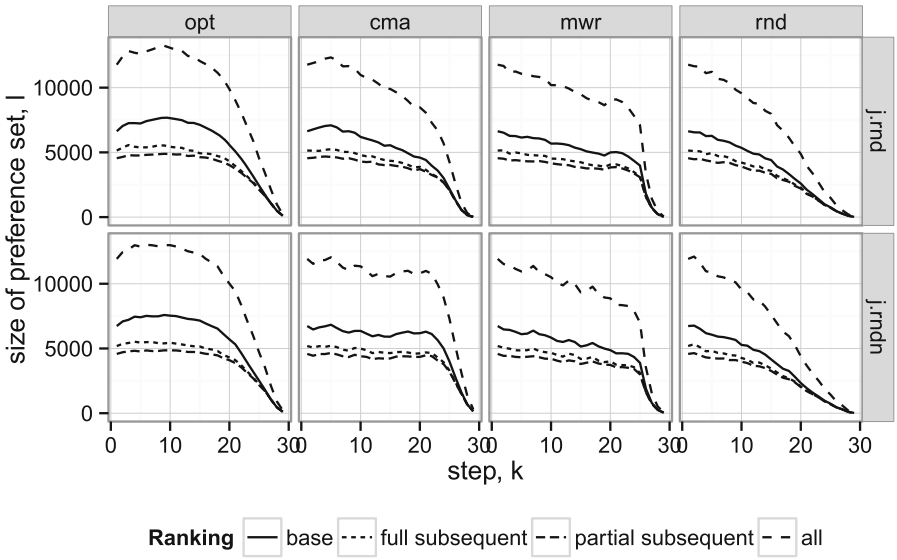


Fig. 2. Size of preference set, $l = |S|$, for different trajectories and ranking strategies obtained from the training set for problem spaces $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$.

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.

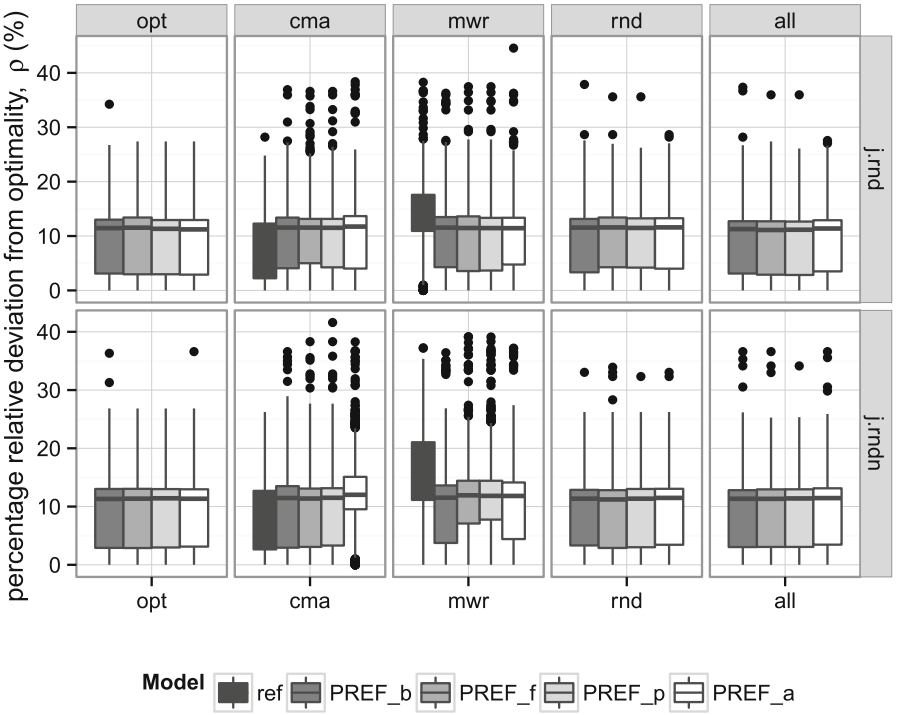


Fig. 3. Box-plot of results for linear ordinal regression model trained on various preference sets using test sets for problem spaces $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$.

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

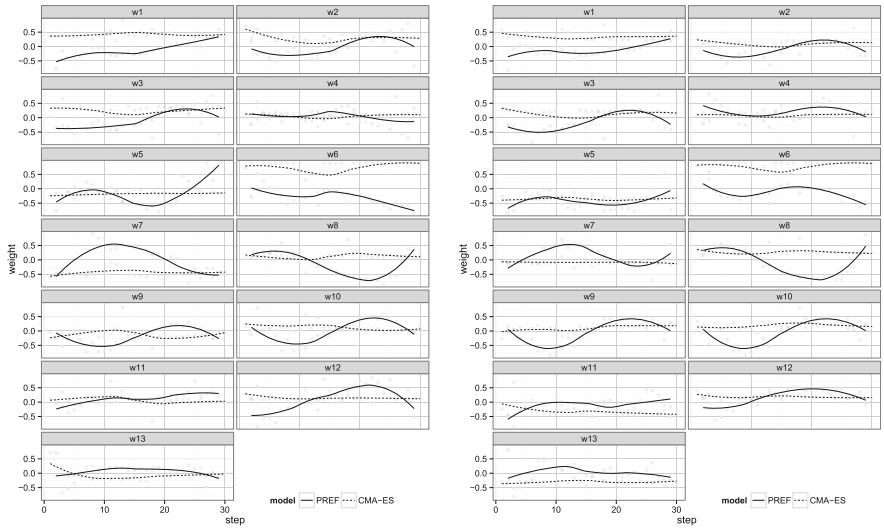
It is particularly interesting there is no statistical difference between PREF^{opt} and PREF^{rnd} for both $\mathcal{P}_{j.rnd}$ and $\mathcal{P}_{j.rndn}$ ranking-models. That is to say, tracking optimal dispatches gives the same performance as completely random dispatches. This indicates that exploring 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.

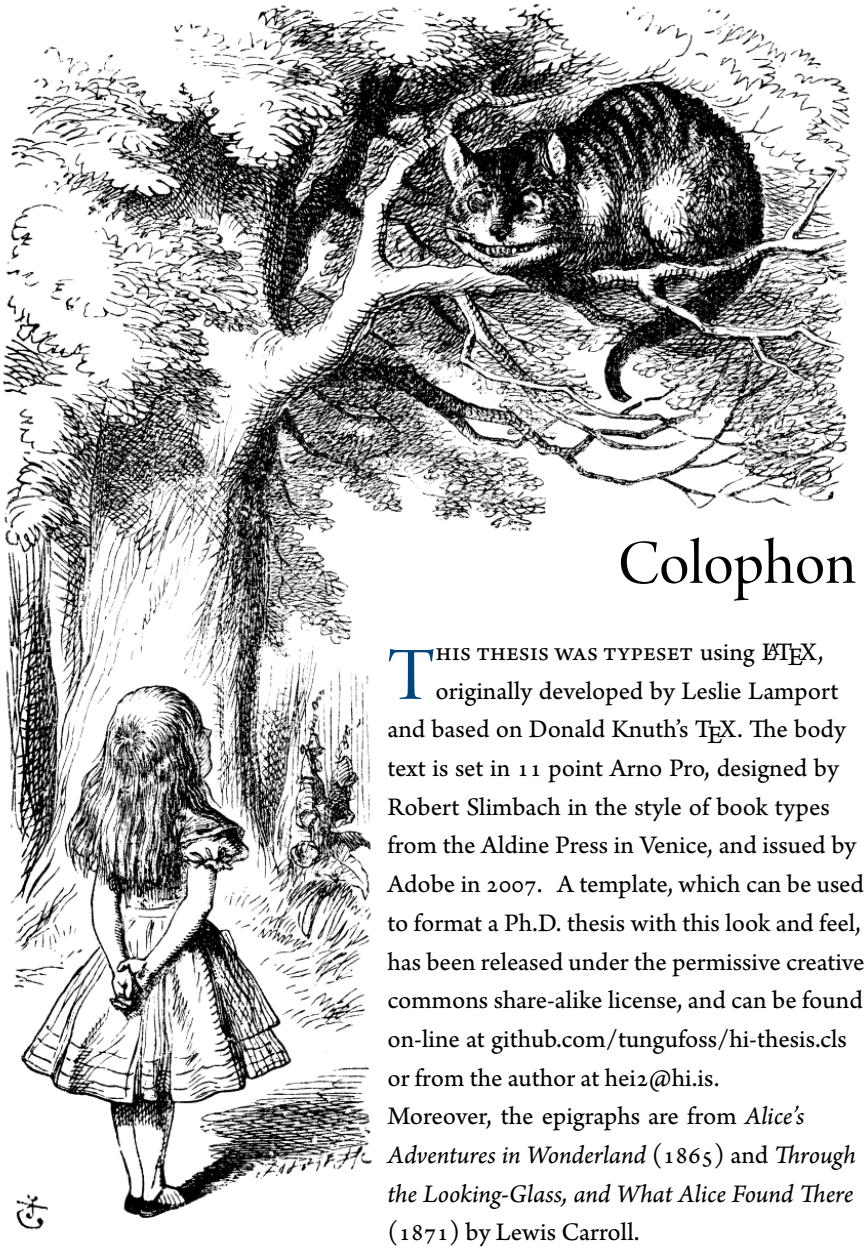
Unlike [8,10,15] learning only optimal training data was not fruitful. However, inspired by the original work of [7], having heuristics guide the generation of training data (while using optimal labelling based on a solver) gave meaningful preference pairs which the learning algorithm could learn. In 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)



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.