Supervised Learning Linear Composite Dispatch
Rules for Scheduling
Case study for JSP and PFSP

Helga Ingimundardóttir
Thomas Philip Runarsson

University of Iceland

April 22\textsuperscript{nd}, 2013
Outline

Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Motivation

General Goal

- General goal is how to search for *good* solutions for an arbitrary problem domain.
- Automate the design of optimization algorithms.
- Use of randomly sampled problem instances and their corresponding optimal vs. suboptimal solutions.
Case Study: JSP and PFSP

Abstract

- Framework for creating dispatching rules for JSP and PFSP.
- Supervised learning based on optimal and sub-optimal solutions.
- Training data is randomly generated problem instances and their optimal solutions. Method is purely data-driven.
- Linear classification to identify good dispatches from worse ones.
- Robust for higher dimensions.

Keywords: Scheduling ● Composite dispatching rules ● JSP ● PFSP ● Generating training data ● Sampling ● Ranking ● Scalability ● Ordinial Regression ● Evolutionary Search
Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Job Shop Scheduling (1)

**JSP**

Simple job shop scheduling problem is where \( n \) jobs are scheduled on a set of \( m \) machines, subject to constraints:

- each job must follow a predefined machine order,
- that a machine can handle at most one job at a time.

**Objective:** schedule the jobs so as to minimize the maximum completion time, i.e. makespan, \( C_{\text{max}} \).

**PFSP**

Permutation flow shop scheduling is the same as JSP except the predefined machine order is homogeneous for all jobs.
## Problem space distributions used in experimental studies

<table>
<thead>
<tr>
<th>type</th>
<th>name</th>
<th>size ((n \times m))</th>
<th>(N_{\text{train}})</th>
<th>(N_{\text{test}})</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSP</td>
<td>(P_{6 \times 5}^{\text{jrnd}})</td>
<td>(6 \times 5)</td>
<td>500</td>
<td>500</td>
<td>random</td>
</tr>
<tr>
<td></td>
<td>(P_{6 \times 5}^{\text{jrndn}})</td>
<td>(6 \times 5)</td>
<td>500</td>
<td>500</td>
<td>random-narrow</td>
</tr>
<tr>
<td></td>
<td>(P_{10 \times 10}^{\text{jrnd}})</td>
<td>(10 \times 10)</td>
<td>–</td>
<td>500</td>
<td>random</td>
</tr>
<tr>
<td></td>
<td>(P_{10 \times 10}^{\text{jrndn}})</td>
<td>(10 \times 10)</td>
<td>–</td>
<td>500</td>
<td>random-narrow</td>
</tr>
<tr>
<td>PFSP</td>
<td>(P_{6 \times 5}^{\text{frnd}})</td>
<td>(6 \times 5)</td>
<td>500</td>
<td>500</td>
<td>random</td>
</tr>
<tr>
<td></td>
<td>(P_{6 \times 5}^{\text{frndn}})</td>
<td>(6 \times 5)</td>
<td>500</td>
<td>500</td>
<td>random-narrow</td>
</tr>
<tr>
<td></td>
<td>(P_{6 \times 5}^{\text{fjc}})</td>
<td>(6 \times 5)</td>
<td>500</td>
<td>500</td>
<td>job-correlated</td>
</tr>
<tr>
<td></td>
<td>(P_{10 \times 10}^{\text{frnd}})</td>
<td>(10 \times 10)</td>
<td>–</td>
<td>500</td>
<td>random</td>
</tr>
<tr>
<td></td>
<td>(P_{10 \times 10}^{\text{frndn}})</td>
<td>(10 \times 10)</td>
<td>–</td>
<td>500</td>
<td>random-narrow</td>
</tr>
<tr>
<td></td>
<td>(P_{10 \times 10}^{\text{fjc}})</td>
<td>(10 \times 10)</td>
<td>–</td>
<td>500</td>
<td>job-correlated</td>
</tr>
</tbody>
</table>
Job Shop Scheduling (3)

Simple Priority Dispatching Rules

- frnd
- frndn
- fjc
- jrnd
- jrndn

Percentage relative deviation from optimality, \( \rho \) (%)

Density

SDR, MWR, LWR, SPT, LPT
Dispatching rules (DR) for constructing JSSP

- Starts with an empty schedule and adds on one job at a time.
- When a machine is free the DR inspects the waiting/available jobs and selects the job with the highest priority.
- Complete schedule consists of $\ell = n \times m$ sequential dispatches.
- At each dispatch $k$ features $\phi(k)$ for the temporal schedule are calculated.
- Performance of DR is compared with its optimal makespan, as percentage relative deviation from optimality: $\rho = \frac{C_{\text{max}}^{\text{DR}} - C_{\text{opt}}^{\text{max}}}{C_{\text{opt}}^{\text{max}}} \cdot 100\%$
# Job Shop Scheduling (5)

## Features for JSSP

<table>
<thead>
<tr>
<th>( \phi )</th>
<th>Feature description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \phi_1 )</td>
<td>processing time for job on machine</td>
</tr>
<tr>
<td>( \phi_2 )</td>
<td>start-time</td>
</tr>
<tr>
<td>( \phi_3 )</td>
<td>end-time</td>
</tr>
<tr>
<td>( \phi_4 )</td>
<td>when machine is next free</td>
</tr>
<tr>
<td>( \phi_5 )</td>
<td>current makespan</td>
</tr>
<tr>
<td>( \phi_6 )</td>
<td>work remaining</td>
</tr>
<tr>
<td>( \phi_7 )</td>
<td>most work remaining</td>
</tr>
<tr>
<td>( \phi_8 )</td>
<td>slack time for this particular machine</td>
</tr>
<tr>
<td>( \phi_9 )</td>
<td>slack time for all machines</td>
</tr>
<tr>
<td>( \phi_{10} )</td>
<td>slack time weighted w.r.t. number of operations already assigned</td>
</tr>
<tr>
<td>( \phi_{11} )</td>
<td>time job had to wait</td>
</tr>
<tr>
<td>( \phi_{12} )</td>
<td>size of slot created by assignment</td>
</tr>
<tr>
<td>( \phi_{13} )</td>
<td>total processing time for job</td>
</tr>
</tbody>
</table>
A schedule being built at step $k = 16$. The dashed boxes represent five different possible jobs that could be scheduled next using a DR.
Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Ordinal Regression (1)

Preference learning problem

Specified by a set of preference pairs:

\[ S = \left\{ \left\{ z_0, +1 \right\}, \left\{ z_s, -1 \right\} \right\}_{k=1}^{\ell} \mid \forall o \in O^{(k)}, s \in S^{(k)} \}
\subset \Phi \times Y \]

where the set of point/rank pairs are:

- Optimal decision: \( z_o = \phi(o) - \phi(s) \), ranked +1
- Sub-optimal decision: \( z_s = \phi(s) - \phi(o) \), ranked -1
Ordinal Regression (2)

- Mapping of points to ranks: \( \{h(\cdot) : \Phi \mapsto Y\} \) where
  \[
  \phi_o \succ \phi_s \iff h(\phi_o) > h(\phi_s)
  \]

- The preference is defined by a linear function, i.e. PREF model:
  \[
  h(\phi) = \sum_{i=1}^{d} w_i \phi = \langle w \cdot \phi \rangle.
  \]

- Logistic regression learns the optimal parameters \( w \) by solving:
  \[
  \min_w \quad \frac{1}{2} \langle w \cdot w \rangle + C \sum_{j=1}^{|S|} \log \left( 1 + e^{-y_j \langle w \cdot z_j \rangle} \right)
  \]
Generating preference set $S$ (1)

A separate DR for each dispatch iteration

- At each dispatch $k$ a number of data pairs are created
  - for each of the $N_{\text{train}}$ problem instance created.
- Deliberately create a separate data set for each dispatch
  - Resulting in $\ell$ linear scheduling rules for solving a $n \times m$ JSSP.

Defining the size of the training set as $l = |\Phi|$, gives the size of the preference set as $|S| = 2l$.

- If $l$ is too large, than sampling needs to be done.
Generating preference set $S$ (2)

**Previous sampling approach**

The strategy was to follow some single optimal job $j \in \mathcal{O}^{(k)}$, thus creating $|\mathcal{O}^{(k)}| \cdot |S^{(k)}|$ feature pairs at each dispatch $k$, resulting in a training size of:

$$I = \sum_{q=1}^{N_{\text{train}}} \left( \sum_{k=1}^{\ell} |\mathcal{O}^{(k)}| \cdot |S^{(k)}| \right)$$

For the data distribution considered there, this simple sampling was sufficient for a favourable outcome. However for a considerably harder data distribution this strategy did not work well.

**Trajectory sampling strategies explored for $S$,**
### Generating preference set $S$ (3)

<table>
<thead>
<tr>
<th>$S^{opt}$</th>
<th>follow some (random) optimal task</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S^{cma}$</td>
<td>follow the task corresponding to highest priority, computed with fixed weights $w$, which were obtained by optimising with CMA-ES.</td>
</tr>
<tr>
<td>$S^{mwr}$</td>
<td>follow the SDR most work remaining (MWR).</td>
</tr>
<tr>
<td>$S^{lwr}$</td>
<td>similar to $S^{mwr}$ except for least work remaining (LWR).</td>
</tr>
<tr>
<td>$S^{all}$</td>
<td>union of all of the above.</td>
</tr>
</tbody>
</table>
Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Evolutionary search

Instead of using logistic regression for to find the weights $w$ for linear preference function:

$$h(\phi) = \sum_{i=1}^{d} w_i \phi = \langle w \cdot \phi \rangle.$$ 

a widely-used evolutionary algorithm, Covariance Matrix Adaptation Evolution Strategy (CMA-ES), is applied to directly minimise the expected relative error, i.e. $\mathbb{E}[\rho]$ (note, could also minimise $\mathbb{E}[C_{\text{max}}]$)

**Benefit**  No need to collect preference set $S$

**Drawback**  Computationally expensive to evaluate $\mathbb{E}[\rho]$
Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Experiments (1)

Size of preference set $S$

![Graph showing the size of preference set $S$ for different scenarios and tracks over steps $k$.]
Experiments (2)

Linear PREF models and CMA-ES obtained weights

![Graph showing percentage relative deviation from optimality for different models]
Introduction

Job Shop Scheduling

Preference models

Evolutionary search with CMA-ES

Experiments

Summary and conclusions
Summary and conclusions (1)

- Introduced a framework for learning linear composite dispatch rules for scheduling.
- The approaches find linear weights by either direct optimisation with CMA-ES or via preference learning by collecting preference pairs whilst sampling the state space of the schedule strategically.
Summary and conclusions (2)

CMA-ES optimisation

Benefits:
• Does not rely on optimal solutions
• Scalable

Drawbacks:
• Computationally expensive.
• Limited to linear preference function \( h(\cdot) \)

Future Work:
• Mediate evolutionary search by use of surrogate models which indirectly estimate mean expected error w.r.t. current population without a loss in performance
PREF models

Benefits:
• Scalable
• Robust to different data distributions

Drawbacks:
• Must know the optimal solution of the problem a priori to correctly classify optimal decisions from suboptimal ones

Future work:
• Easily adaptable to non-linear preferences function, i.e. project the feature space onto a higher dimension thereby updating $h(\cdot)$ to a kernel based function which should yield lower expected $C_{\text{max}}$
Thank you for your attention

Questions?

Helga Ingimundardóttir, hei2@hi.is