Supervised Learning Linear Composite Dispatch Rules for Scheduling

Case study for JSP and PFSP

Helga Ingimundardóttir
Thomas Philip Runarsson

University of Iceland

April 22^{nd}, 2013

Outline

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

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ

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

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, $C_{\max}$.

PFSP

Permutation flow shop scheduling is the same as JSP except the predefined machine order is homogeneous for all jobs.

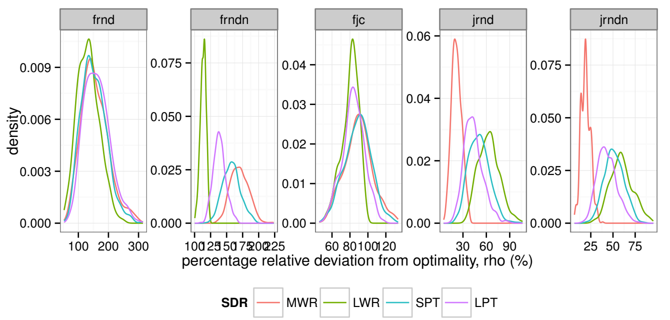
Job Shop Scheduling (2)

Problem space distributions used in experimental studies

type	name	size ($n \times m$)	N_{train}	N_{test}	note
JSP	$\mathcal{P}_{jrnd}^{6\times5}$	6×5	500	500	random
	$\mathcal{P}_{jrndn}^{6\times5}$	6×5	500	500	random-narrow
	$\mathcal{P}_{jrnd}^{10\times10}$	10×10	–	500	random
	$\mathcal{P}_{jrndn}^{10\times10}$	10×10	–	500	random-narrow
PFSP	$\mathcal{P}_{frnd}^{6\times5}$	6×5	500	500	random
	$\mathcal{P}_{frndn}^{6\times5}$	6×5	500	500	random-narrow
	$\mathcal{P}_{fjc}^{6\times5}$	6×5	500	500	job-correlated
	$\mathcal{P}_{frnd}^{10\times10}$	10×10	–	500	random
	$\mathcal{P}_{frndn}^{10\times10}$	10×10	–	500	random-narrow
	$\mathcal{P}_{fjc}^{10\times10}$	10×10	–	500	job-correlated

Job Shop Scheduling (3)

Simple Priority Dispathcing Rules



Job Shop Scheduling (4)

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: $\rho = \frac{C_{\max}^{DR} - C_{\max}^{opt}}{C_{\max}^{opt}} \cdot 100\%$

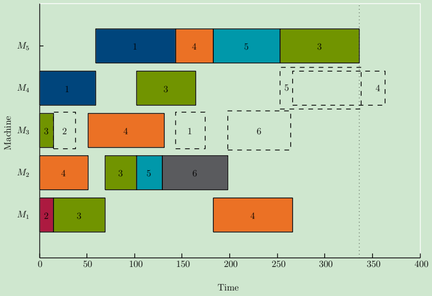
Features for JSSP

Job Shop Scheduling (5)

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

Job Shop Scheduling

Example



A schedule being built at step $k = 16$. The dashed boxes represent five different possible jobs that could be scheduled next using a DR.

Ordinal Regression (1)

Preference learning problem

Specified by a set of preference pairs:

$$S = \left\{ \{\mathbf{z}_o, +1)\}_{k=1}^{\ell}, \{\mathbf{z}_s, -1)\}_{k=1}^{\ell} \mid \forall o \in \mathcal{O}^{(k)}, s \in \mathcal{S}^{(k)} \right\} \subset \Phi \times Y$$

where the set of point/rank pairs are:

- Optimal decision: $\mathbf{z_o} = \phi^{(o)} - \phi^{(s)}$, ranked $+1$
- Sub-optimal decision: $\mathbf{z_s} = \phi^{(s)} - \phi^{(o)}$, ranked -1

Ordinal Regression (2)

- Mapping of points to ranks: $\{h(\cdot) : \Phi \mapsto Y\}$ where

$$\phi_o \succ \phi_s \quad \Leftrightarrow \quad h(\phi_o) > h(\phi_s)$$

- 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:

$$\min_{\mathbf{w}} \quad \tfrac{1}{2}\langle w \cdot w \rangle + C \sum_{j=1}^{|S|} \log\left(1 + e^{-y_j \langle w \cdot z_j \rangle}\right)$$

Generating prefernce set S (1)

A seperate DR for each dispatch iteration

- At each dispatch k a number of data pairs are created
  - for each of the N_{train} problem instance created.
- Deliberately create a separate data set for each dispatch
  - Resulting in ℓ 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.

HÁSKÓLI ÍSLANDS
VERKFRÆÐI- OG NÁTTÚRUVÍSINDASVIÐ

Generating prefernce set S (2)

Previous sampling approach

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

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

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 prefernce set S (3)

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

Evolutionary search

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

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

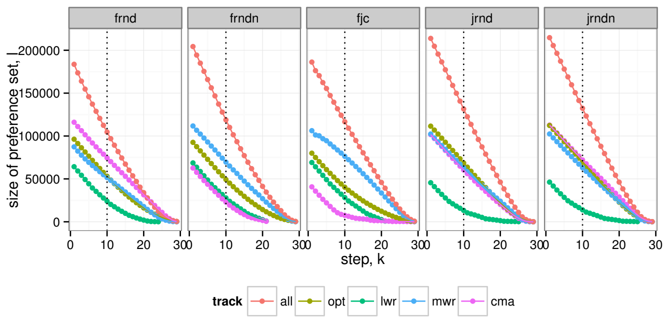
a widely-used evolutionary algorithm, Covariance Matrix Adaptation Evolution Strategy (CMA-ES), is applied to directly minimise the expected relative error, i.e. $\mathbb{E}[\rho]$ (note, could also minimise $\mathbb{E}[C_{\text{max}}]$)

Benefit No need to collect preference set S

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

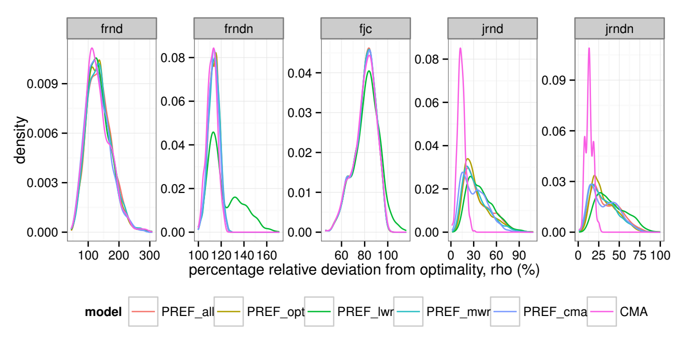
Experiments (1)

Size of preference set S



Experiments (2)

Linear PREF models and CMA-ES obtained weights



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

Summary and conclusions (3)

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 $C_{\max}$

Thank you for your attention

Questions?

Helga Ingimundardóttir, hei2@hi.is