ALICE

Analysis & Learning Iterative Consecutive Executions

Helga Ingimundardóttir

University of Iceland

June 30, 2016

Introduction

Motivation:

⋆ The general goal is to train optimisation algorithms, for an arbitrary problem domain, using data.

Contribution:

⋆ The main contribution of this thesis is towards a better understanding of how this training data should be constructed.

Introduction

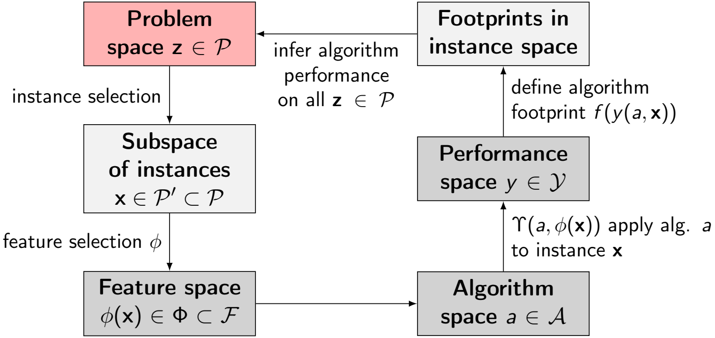
Motivation:

* ⋆ The general goal is to train optimisation algorithms, for an arbitrary problem domain, using data.

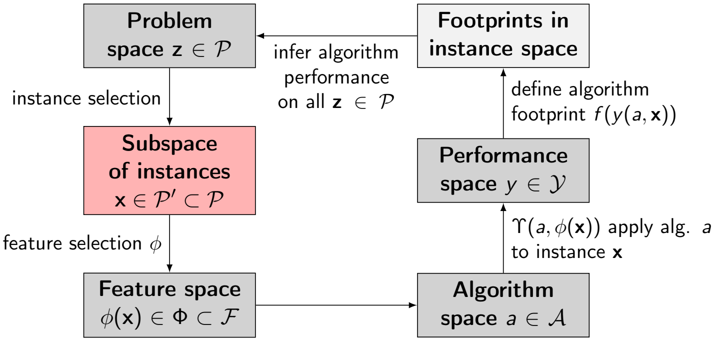
Contribution:

* ⋆ The main contribution of this thesis is towards a better understanding of how this training data should be constructed.

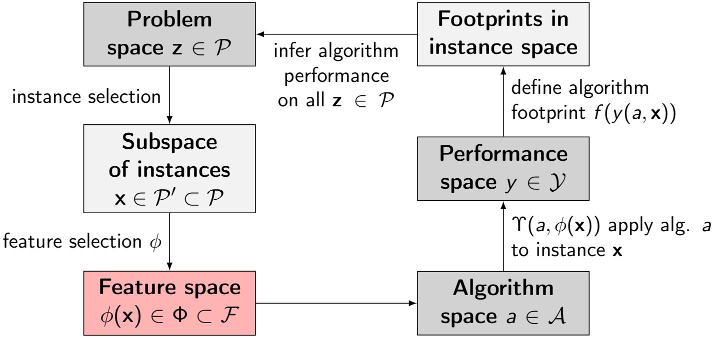
Framework for Algorithm Learning



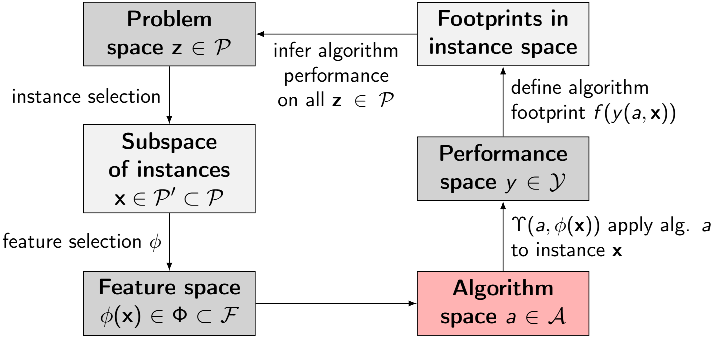
Framework for Algorithm Learning



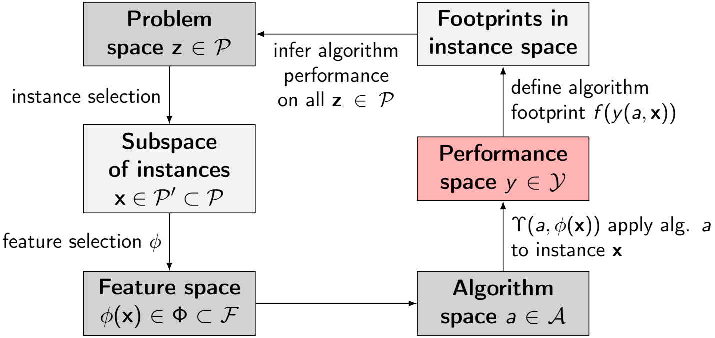
Framework for Algorithm Learning



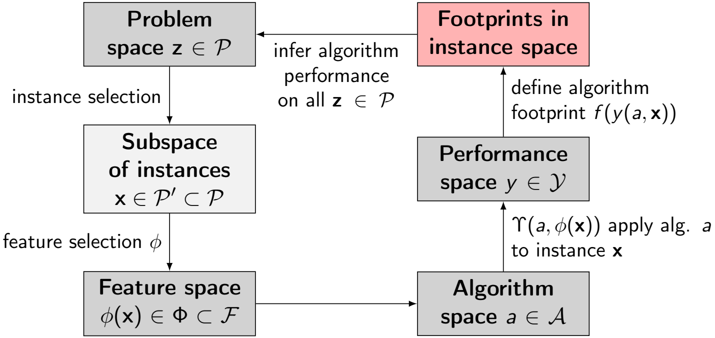
Framework for Algorithm Learning



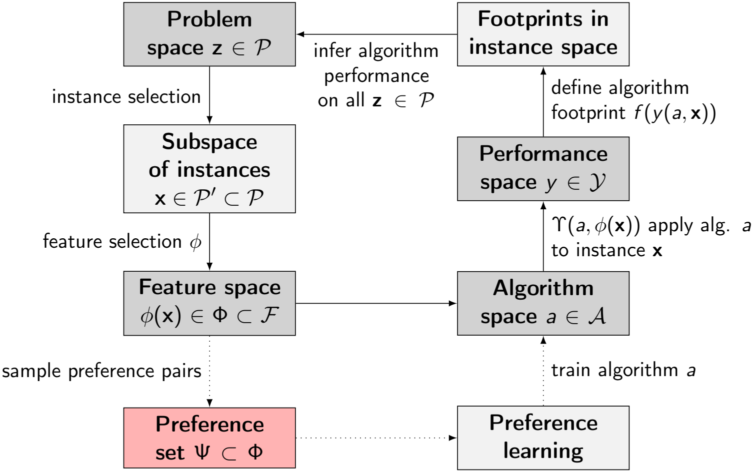
Framework for Algorithm Learning



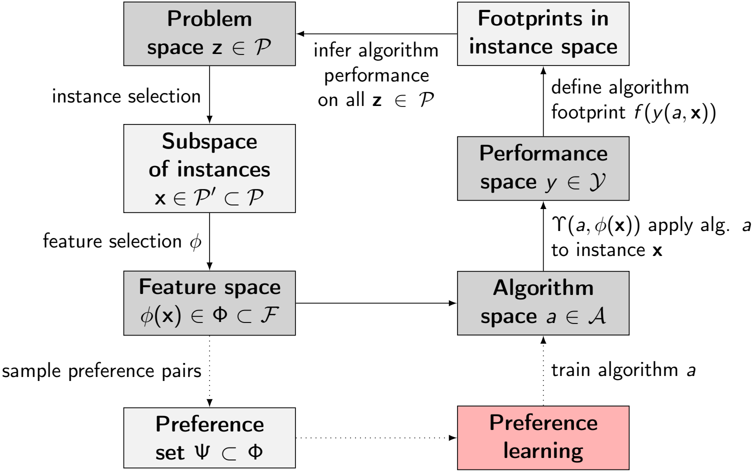
Framework for Algorithm Learning



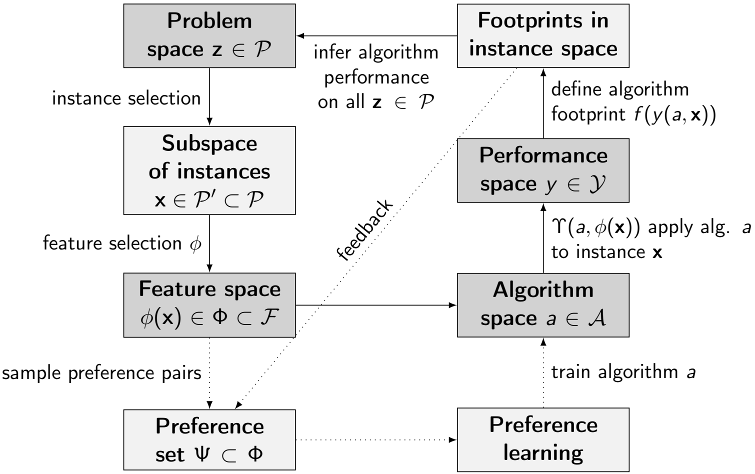
Framework for Algorithm Learning



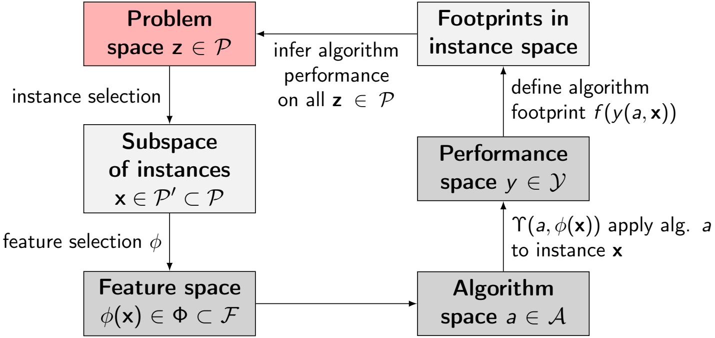
Framework for Algorithm Learning



Framework for Algorithm Learning



Framework for Algorithm Learning



Job Shop Scheduling (JSP)

Simple job-shop 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 minimise the maximum completion time, i.e., makespan, $C_{\max}$.

Dispatching rules

Dispatching rules (DR) are consecutive executions found by:

- ⋆ Starting with an empty schedule and adding on one operation at a time.
- ⋆ When a machine is free the DR inspects the available jobs and selects the one with the highest priority.
- ⋆ Complete schedule consists of $\ell = n \cdot m$ sequential dispatches.
- ⋆ At each dispatch k, features $\phi(k)$ for the temporal schedule are calculated.

Mad Hatter tea-party (definition)

The attending guests:

J_1) Alice
J_2) March Hare
J_3) Dormouse
J_4) Mad Hatter.

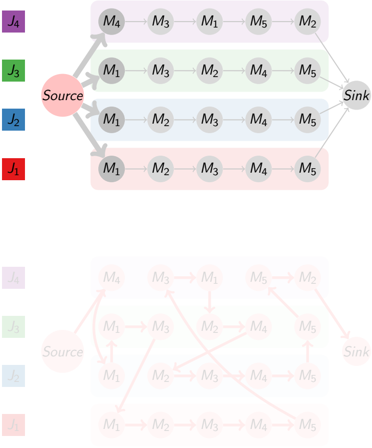
They all have to:

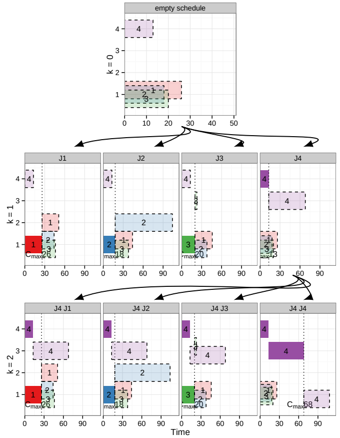
M_1) have wine or pour tea
M_2) spread butter
M_3) get a haircut
M_4) check the time of the broken watch
M_5) say what they mean.

This can be considered as is a typical 4×5 job-shop, where:

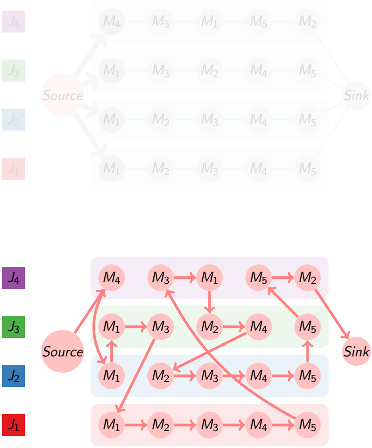
- ⋆ our guests are the jobs
- ⋆ their tasks are the machines
- ⋆ objective is to minimise $C_{\max}$ (when Alice can leave).

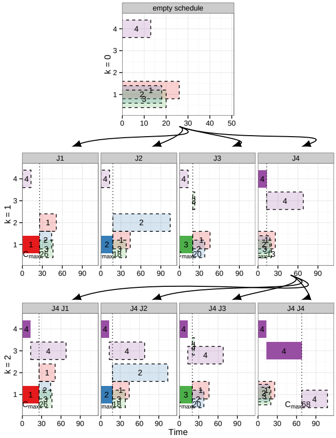
Mad Hatter tea-party (action states)



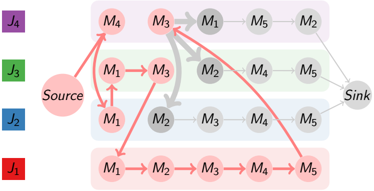


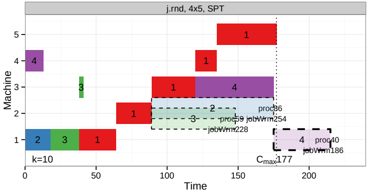
Mad Hatter tea-party (action states)



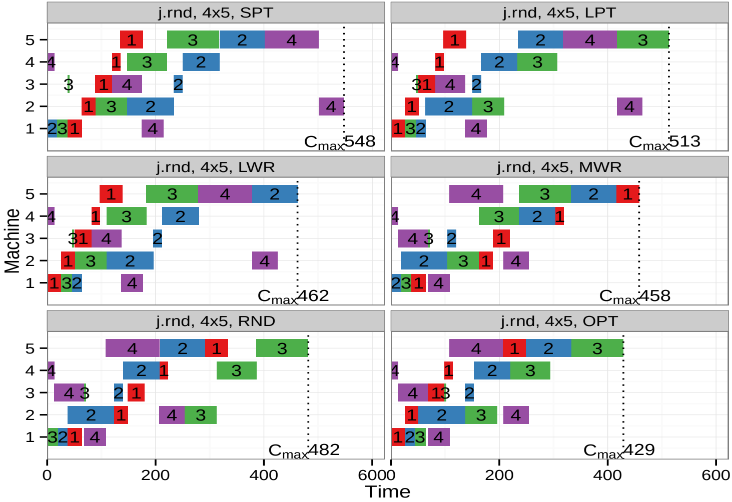


Mad Hatter tea-party (action states)

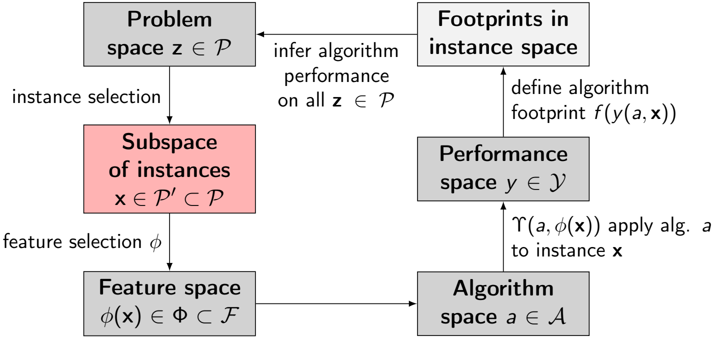




Mad Hatter tea-party (SDRs)



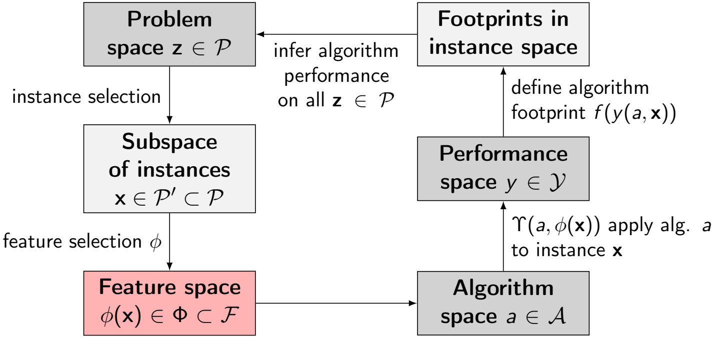
Framework for Algorithm Learning



Problem instance generators

	name	size ($n \times m$)	N_{train}	N_{test}	note
JSP	$\mathcal{P}_{j.rnd}^{6\times5}$	6×5	500	500	random
	$\mathcal{P}_{j.rndn}^{6\times5}$	6×5	500	500	random-narrow
	$\mathcal{P}_{j.rnd,J_1}^{6\times5}$	6×5	500	500	random with job variation
	$\mathcal{P}_{j.rnd,M_1}^{6\times5}$	6×5	500	500	random with machine variation
	$\mathcal{P}_{j.rnd}^{10\times10}$	10×10	300	200	random
	$\mathcal{P}_{j.rndn}^{10\times10}$	10×10	300	200	random-narrow
	$\mathcal{P}_{j.rnd,J_1}^{10\times10}$	10×10	300	200	random with job variation
	$\mathcal{P}_{j.rnd,M_1}^{10\times10}$	10×10	300	200	random with machine variation
	$\mathcal{P}_{JSP.ORLIB}$	various	–	82	various
FSP	$\mathcal{P}_{f.rnd}^{6\times5}$	6×5	500	500	random
	$\mathcal{P}_{f.rndn}^{6\times5}$	6×5	500	500	random-narrow
	$\mathcal{P}_{f.jc}^{6\times5}$	6×5	500	500	job-correlated
	$\mathcal{P}_{f.mc}^{6\times5}$	6×5	500	500	machine-correlated
	$\mathcal{P}_{f.mxc}^{6\times5}$	6×5	500	500	mixed-correlation
	$\mathcal{P}_{f.rnd}^{10\times10}$	10×10	300	200	random
	$\mathcal{P}_{FPS.ORLIB}$	various	–	31	various

Framework for Algorithm Learning



Features for JSP

job	ϕ_1	job processing time
	ϕ_2	job start-time
	ϕ_3	job end-time
	ϕ_4	job arrival time
	ϕ_5	time job had to wait
	ϕ_6	total processing time for job
	ϕ_7	total work remaining for job
	ϕ_8	number of assigned operations for job
machine	ϕ_9	when machine is next free
	ϕ_{10}	total processing time for machine
	ϕ_{11}	total work remaining for machine
	ϕ_{12}	number of assigned operations for machine
	ϕ_{13}	change in idle time by assignment
	ϕ_{14}	total idle time for machine
	ϕ_{15}	total idle time for all machines
	ϕ_{16}	current makespan
final makespan	ϕ_{17}	final makespan using SPT
	ϕ_{18}	final makespan using LPT
	ϕ_{19}	final makespan using LWR
	ϕ_{20}	final makespan using MWR
	ϕ_{RND}	final makespans using 100 random rollouts
	ϕ_{21}	mean for ϕ_{RND}
	ϕ_{22}	standard deviation for ϕ_{RND}
	ϕ_{23}	minimum value for ϕ_{RND}
	ϕ_{24}	maximum value for ϕ_{RND}

ALICE

Helga

Trajectory strategies for Φ

Following the policy:

- (Φ^{OPT}) expert $\pi_\star$.
- ($\Phi^{ES.\rho}$) the policy obtained by optimising with CMA-ES.
- (Φ^{SPT}) shortest processing time (SPT).
- (Φ^{LPT}) longest processing time (LPT).
- (Φ^{LWR}) least work remaining (LWR).
- (Φ^{MWR}) most work remaining (MWR).
- (Φ^{RND}) random policy (RND).
- (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{SPT}) shortest processing time (SPT).
- $\star$ (Φ^{LPT}) longest processing time (LPT).
- $\star$ (Φ^{LWR}) least work remaining (LWR).
- $\star$ (Φ^{MWR}) most work remaining (MWR).
- $\star$ (Φ^{RND}) random policy (RND).
- $\star$ (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

Following the policy:

- ⋆ (Φ^{OPT}) expert $\pi_\star$.
- ⋆ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- ⋆ (Φ^{SPT}) shortest processing time (SPT).
- ⋆ (Φ^{LPT}) longest processing time (LPT).
- ⋆ (Φ^{LWR}) least work remaining (LWR).
- ⋆ (Φ^{MWR}) most work remaining (MWR).
- ⋆ (Φ^{RND}) random policy (RND).
- ⋆ (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{SPT}) shortest processing time (SPT).
- $\star$ (Φ^{LPT}) longest processing time (LPT).
- $\star$ (Φ^{LWR}) least work remaining (LWR).
- $\star$ (Φ^{MWR}) most work remaining (MWR).
- $\star$ (Φ^{RND}) random policy (RND).
- $\star$ (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ ($\Phi^{\mathsf{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{SPT}) shortest processing time (SPT).
- $\star$ (Φ^{LPT}) longest processing time (LPT).
- $\star$ (Φ^{LWR}) least work remaining (LWR).
- $\star$ (Φ^{MWR}) most work remaining (MWR).
- $\star$ (Φ^{RND}) random policy (RND).
- $\star$ (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

Following the policy:

- ⋆ (Φ^{OPT}) expert $\pi_{\star}$.
- ⋆ $(\Phi^{\mathrm{ES}.\rho})$ the policy obtained by optimising with CMA-ES.
- ⋆ (Φ^{SPT}) shortest processing time (SPT).
- ⋆ (Φ^{LPT}) longest processing time (LPT).
- ⋆ (Φ^{LWR}) least work remaining (LWR).
- ⋆ (Φ^{MWR}) most work remaining (MWR).
- ⋆ (Φ^{RND}) random policy (RND).
- ⋆ (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

Following the policy:

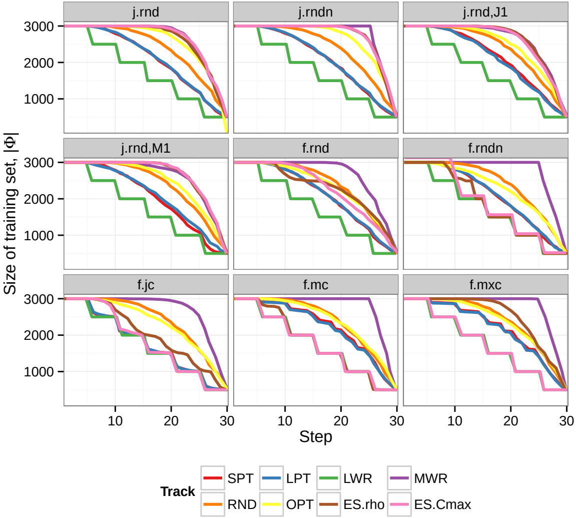
- (Φ^{OPT}) expert $\pi_\star$.
- $(\Phi^{\mathrm{ES}.\rho})$ the policy obtained by optimising with CMA-ES.
- (Φ^{SPT}) shortest processing time (SPT).
- (Φ^{LPT}) longest processing time (LPT).
- (Φ^{LWR}) least work remaining (LWR).
- (Φ^{MWR}) most work remaining (MWR).
- (Φ^{RND}) random policy (RND).
- (Φ^{ALL}) union of all of the above.

Trajectory strategies for Φ

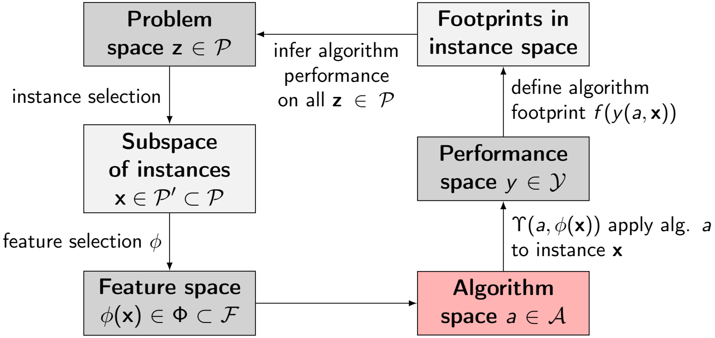
Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{SPT}) shortest processing time (SPT).
- $\star$ (Φ^{LPT}) longest processing time (LPT).
- $\star$ (Φ^{LWR}) least work remaining (LWR).
- $\star$ (Φ^{MWR}) most work remaining (MWR).
- $\star$ (Φ^{RND}) random policy (RND).
- $\star$ (Φ^{ALL}) union of all of the above.

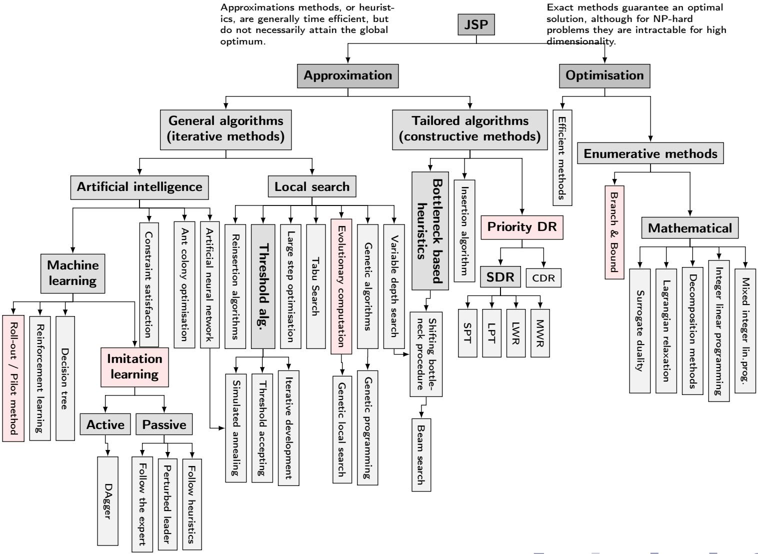
Sampled size of $|\Phi|$ $(6 \times 5, N_{train} = 500)$



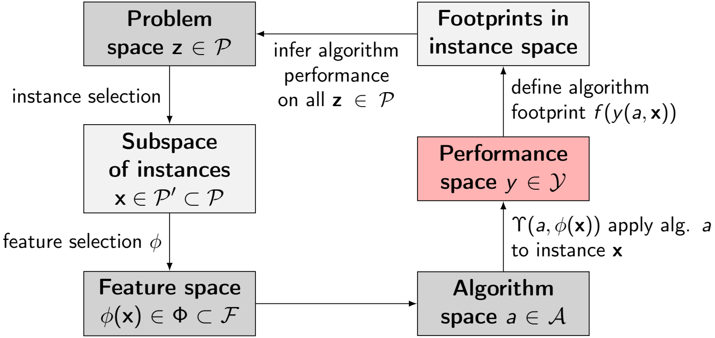
Framework for Algorithm Learning



Various Methods for Solving JSP



Framework for Algorithm Learning



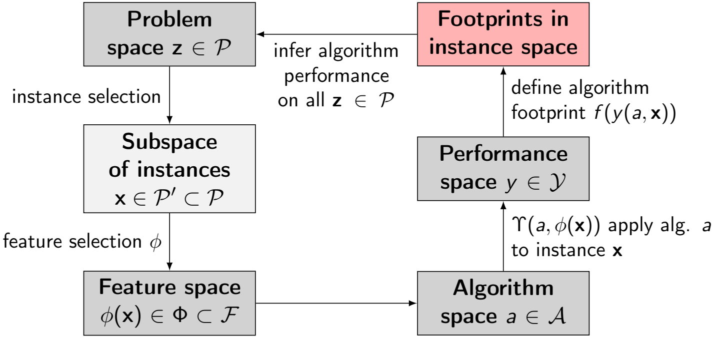
Performance measure

Performance of policy π compared with its optimal makespan, found using an expert policy, $\pi_\star$, is the following loss function:

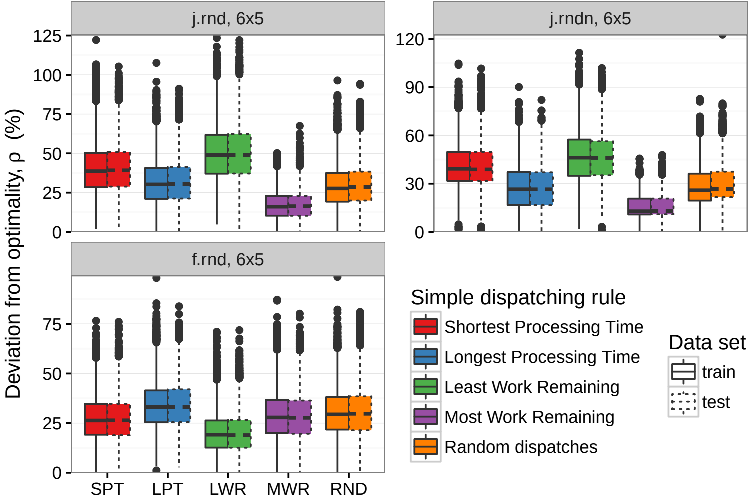
$$\rho = \frac{C_{\max}^{\pi} - C_{\max}^{\pi_\star}}{C_{\max}^{\pi_\star}} \cdot 100\%$$

The goal is to minimise this discrepancy between predicted value and true outcome.

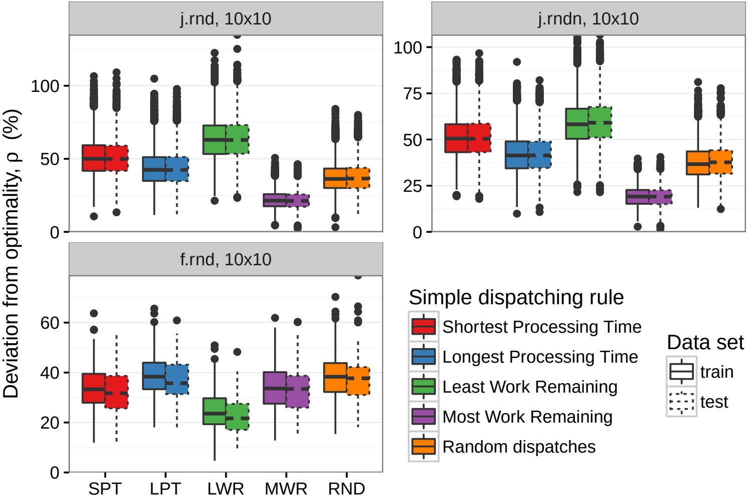
Framework for Algorithm Learning



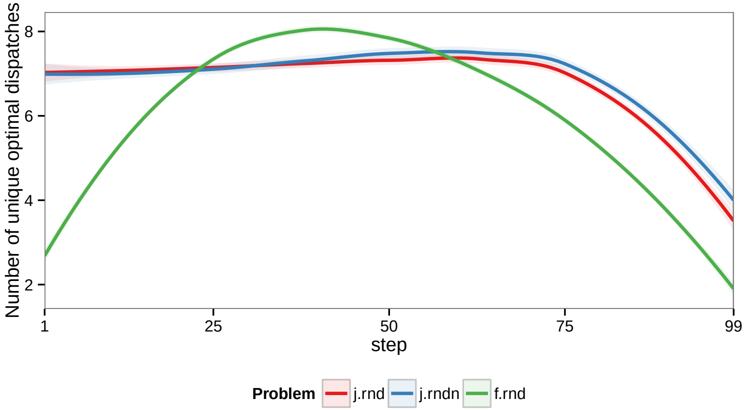
Deviation from optimality, ρ



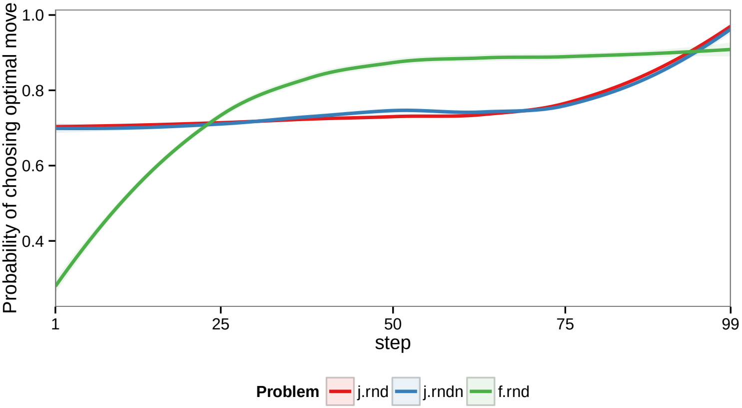
Deviation from optimality, ρ



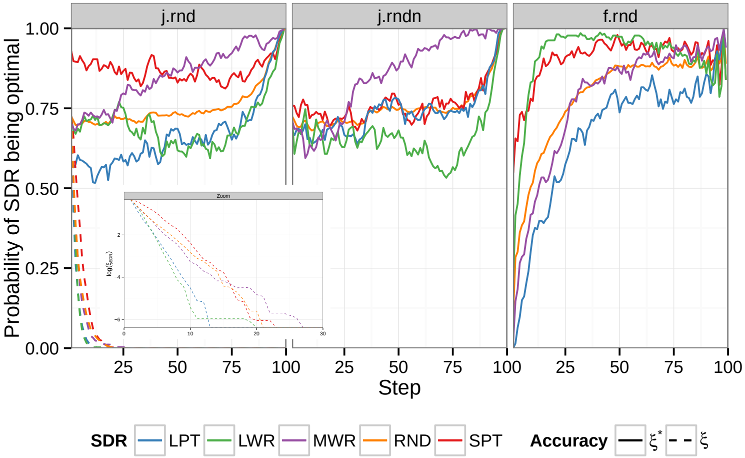
Making optimal decisions, $\xi_\pi^\star$



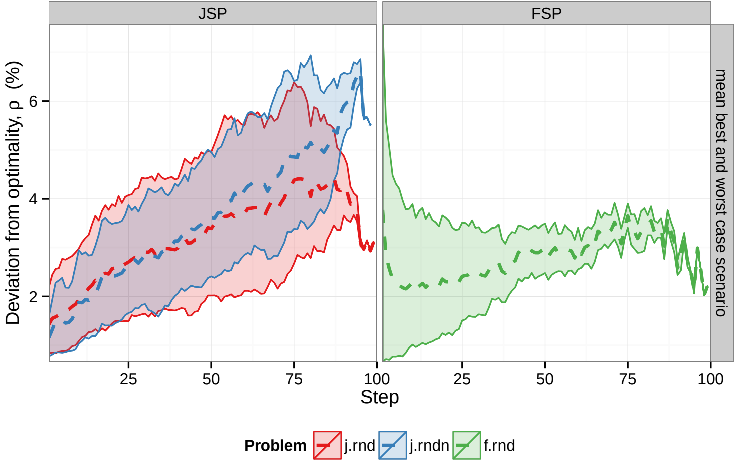
Making optimal decisions, $\xi^{\star}_{\pi}$



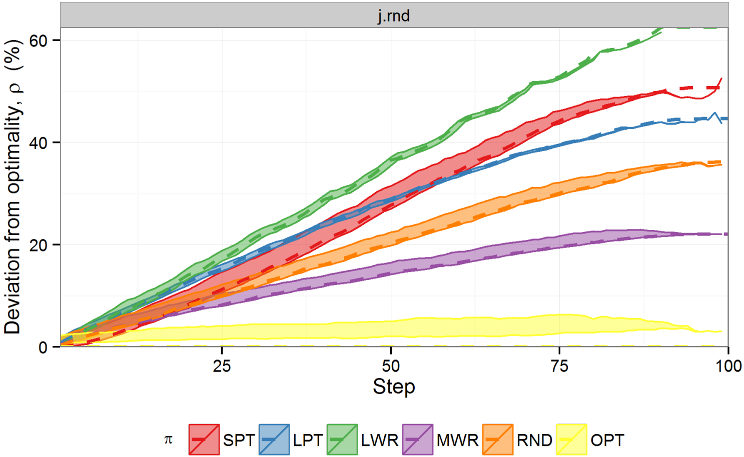
Making optimal decisions, ξ_π



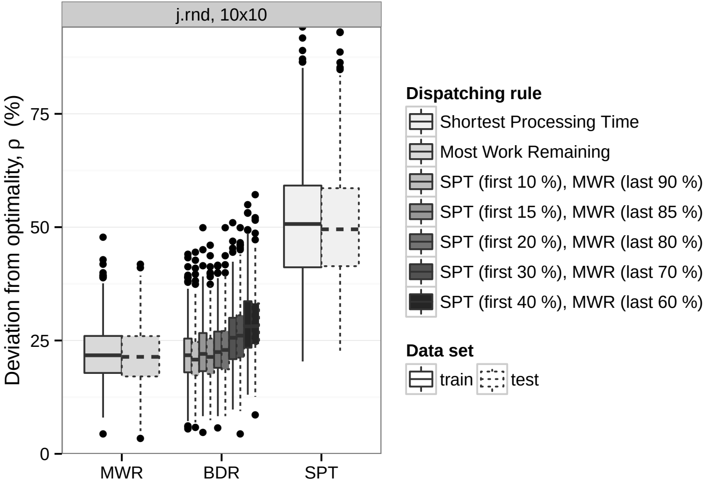
Impact of suboptimal decision, $\{\zeta^\star_{\min}, \zeta^\star_{\max}\}$



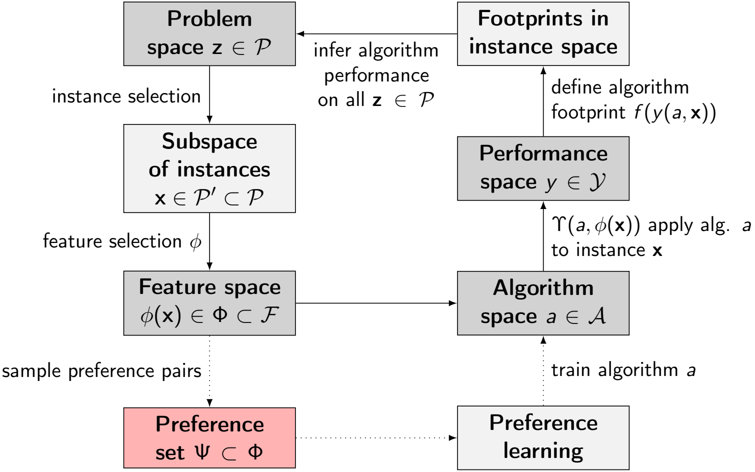
Impact of suboptimal decision, $\{\zeta^{\pi}_{\min}, \zeta^{\pi}_{\max}\}$



Blended dispatching rule



Framework for Algorithm Learning



Generating training data

ALICE framework for creating dispatching rules:

- ⋆ Linear classification to identify good dispatches, from worse ones.
- ⋆ Generate feature set, $\Phi \subset \mathcal{F}$, both from
  - ⋆ optimal solutions, ϕ^o
  - ⋆ suboptimal solutions, ϕ^s

  by exploring various trajectories within the feature-space (where $\phi^o, \phi^s \in \mathcal{F}$).
- ⋆ Sample Φ to create training set Ψ with rank pairs:
  - ⋆ optimal decision, $(\mathbf{z}^o, y_o) = (\phi^o - \phi^s, +1)$
  - ⋆ suboptimal decision, $(\mathbf{z}^s, y_s) = (\phi^s - \phi^o, -1)$

  using different ranking schemes (where $\mathbf{z}^o, \mathbf{z}^s \in \Psi$)
- ⋆ Sample Ψ using stepwise bias for time independent policy.

Generating training data

ALICE framework for creating dispatching rules:

- ⋆ Linear classification to identify good dispatches, from worse ones.
- ⋆ Generate feature set, $\Phi \subset \mathcal{F}$, both from
  - ⋆ optimal solutions, ϕ^o
  - ⋆ suboptimal solutions, ϕ^s

  by exploring various trajectories within the feature-space (where $\phi^o, \phi^s \in \mathcal{F}$).
- ⋆ Sample Φ to create training set Ψ with rank pairs:
  - ⋆ optimal decision, $(\mathbf{z}^o, y_o) = (\phi^o - \phi^s, +1)$
  - ⋆ suboptimal decision, $(\mathbf{z}^s, y_s) = (\phi^s - \phi^o, -1)$

  using different ranking schemes (where $\mathbf{z}^o, \mathbf{z}^s \in \Psi$)
- ⋆ Sample Ψ using stepwise bias for time independent policy.

Generating training data

ALICE framework for creating dispatching rules:

- Linear classification to identify good dispatches, from worse ones.
- Generate feature set, $\Phi \subset \mathcal{F}$, both from
  - optimal solutions, ϕ^o
  - suboptimal solutions, ϕ^s

  by exploring various trajectories within the feature-space (where $\phi^o, \phi^s \in \mathcal{F}$).
- Sample Φ to create training set Ψ with rank pairs:
  - optimal decision, $(\mathbf{z}^o, y_o) = (\phi^o - \phi^s, +1)$
  - suboptimal decision, $(\mathbf{z}^s, y_s) = (\phi^s - \phi^o, -1)$

  using different ranking schemes (where $\mathbf{z}^o, \mathbf{z}^s \in \Psi$)
- Sample Ψ using stepwise bias for time independent policy.

Generating training data

ALICE framework for creating dispatching rules:

- ⋆ Linear classification to identify good dispatches, from worse ones.
- ⋆ Generate feature set, $\Phi \subset \mathcal{F}$, both from
  - ⋆ optimal solutions, ϕ^o
  - ⋆ suboptimal solutions, ϕ^s

  by exploring various trajectories within the feature-space (where $\phi^o, \phi^s \in \mathcal{F}$).
- ⋆ Sample Φ to create training set Ψ with rank pairs:
  - ⋆ optimal decision, $(\mathbf{z}^o, y_o) = (\phi^o - \phi^s, +1)$
  - ⋆ suboptimal decision, $(\mathbf{z}^s, y_s) = (\phi^s - \phi^o, -1)$

  using different ranking schemes (where $\mathbf{z}^o, \mathbf{z}^s \in \Psi$)
- ⋆ Sample Ψ using stepwise bias for time independent policy.

Ranking schemes for Ψ

Sampling rankings of available jobs where where $r_1 > r_2 > \cdots > r_{n'}$ $(n' \leq n)$ with respect to:

- Ψ_b all opt rankings r_1 vs. all possible subopt rankings r_i, $i \in \{2, ..., n'\}$
- Ψ_f full subsequent rankings, i.e., all combinations of r_i and r_{i+1} for all $i \in \{1, ..., n'\}$.
- Ψ_p partial subsequent rankings, similar to Ψ_f except if there are more than one operation with the same rank, only one is needed to be compared to subsequent rank $(\Psi_p \subset \Psi_f)$.
- Ψ_a union of all of the above.

Ranking schemes for Ψ

Sampling rankings of available jobs where where $r_1 > r_2 > \cdots > r_{n'}$ $(n' \leq n)$ with respect to:

- Ψ_b all opt rankings r_1 vs. all possible subopt rankings r_i, $i \in \{2, ..., n'\}$
- Ψ_f full subsequent rankings, i.e., all combinations of r_i and r_{i+1} for all $i \in \{1, ..., n'\}$.
- Ψ_p partial subsequent rankings, similar to Ψ_f except if there are more than one operation with the same rank, only one is needed to be compared to subsequent rank $(\Psi_p \subset \Psi_f)$.
- Ψ_a union of all of the above.

Ranking schemes for Ψ

Sampling rankings of available jobs where where $r_1 > r_2 > \cdots > r_{n'}$ $(n' \leq n)$ with respect to:

- Ψ_b all opt rankings r_1 vs. all possible subopt rankings r_i, $i \in \{2, ..., n'\}$
- Ψ_f full subsequent rankings, i.e., all combinations of r_i and r_{i+1} for all $i \in \{1, ..., n'\}$.
- Ψ_p partial subsequent rankings, similar to Ψ_f except if there are more than one operation with the same rank, only one is needed to be compared to subsequent rank $(\Psi_p \subset \Psi_f)$.
- Ψ_a union of all of the above.

Ranking schemes for Ψ

Sampling rankings of available jobs where where $r_1 > r_2 > \cdots > r_{n'}$ $(n' \leq n)$ with respect to:

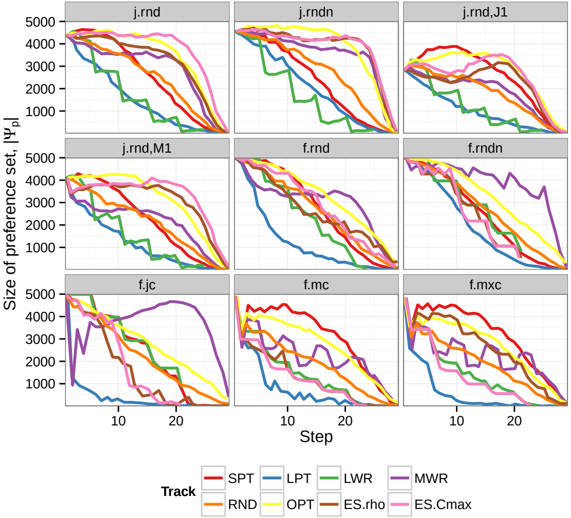
- Ψ_b all opt rankings r_1 vs. all possible subopt rankings r_i, $i \in \{2, ..., n'\}$
- Ψ_f full subsequent rankings, i.e., all combinations of r_i and r_{i+1} for all $i \in \{1, ..., n'\}$.
- Ψ_p partial subsequent rankings, similar to Ψ_f except if there are more than one operation with the same rank, only one is needed to be compared to subsequent rank $(\Psi_p \subset \Psi_f)$.
- Ψ_a union of all of the above.

Ranking schemes for Ψ

Sampling rankings of available jobs where where $r_1 > r_2 > \cdots > r_{n'}$ $(n' \leq n)$ with respect to:

- Ψ_b all opt rankings r_1 vs. all possible subopt rankings r_i, $i \in \{2, ..., n'\}$
- Ψ_f full subsequent rankings, i.e., all combinations of r_i and r_{i+1} for all $i \in \{1, ..., n'\}$.
- Ψ_p partial subsequent rankings, similar to Ψ_f except if there are more than one operation with the same rank, only one is needed to be compared to subsequent rank $(\Psi_p \subset \Psi_f)$.
- Ψ_a union of all of the above.

Sampled size of $|\Psi|$ $(6 \times 5, N_{train} = 500)$



Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- $\star$ (equal) equal probability.
- $\star$ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\text{RND}}$.
- $\star$ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- $\star$ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- $\star$ (featsize) inversely proportional to $|\Phi^{\text{OPT}}|$
- $\star$ (prefsize) inversely proportional to $|\Psi_p^{\text{OPT}}|$
- $\star$ (dbl1st) twice as much weight on the first half of the dispatches.
- $\star$ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\mathrm{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\mathrm{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\mathrm{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\mathrm{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\mathrm{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\mathrm{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\text{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\text{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\text{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- $\star$ (equal) equal probability.
- $\star$ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\text{RND}}$.
- $\star$ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- $\star$ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- $\star$ (featsize) inversely proportional to $|\Phi^{\text{OPT}}|$
- $\star$ (prefsize) inversely proportional to $|\Psi_p^{\text{OPT}}|$
- $\star$ (dbl1st) twice as much weight on the first half of the dispatches.
- $\star$ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\mathrm{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\mathrm{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\mathrm{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\text{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\text{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\text{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

Sampling stepwise bias for preference pairs:

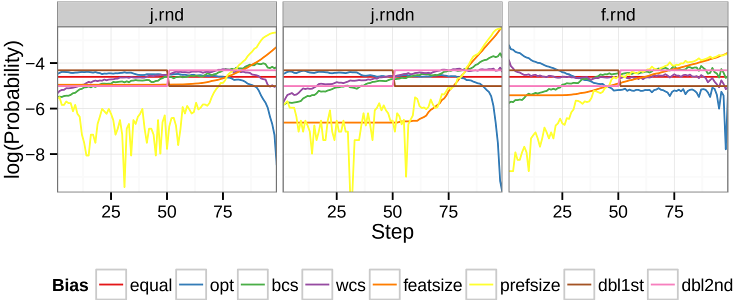
- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\mathrm{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\mathrm{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\mathrm{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias for sampling Ψ

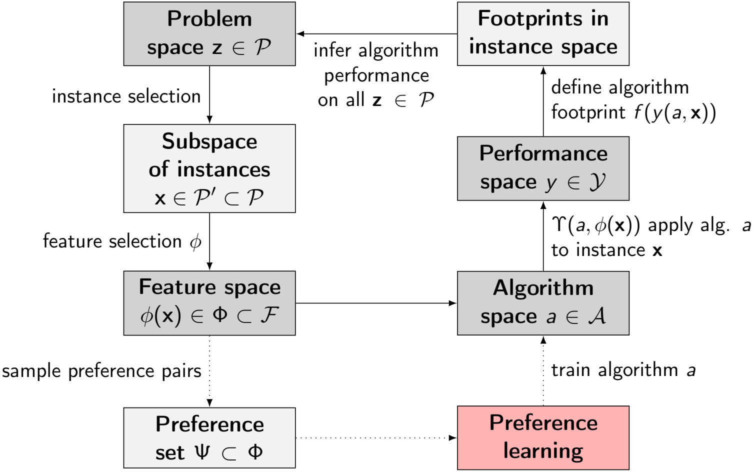
Sampling stepwise bias for preference pairs:

- ⋆ (equal) equal probability.
- ⋆ (opt) inverse optimality for random dispatches – $1 - \xi^{\star}_{\mathrm{RND}}$.
- ⋆ (bcs) best case scenario for mean ρ – $\zeta^{\star}_{\min}$.
- ⋆ (wcs) worst case scenario for mean ρ – $\zeta^{\star}_{\max}$.
- ⋆ (featsize) inversely proportional to $|\Phi^{\mathrm{OPT}}|$
- ⋆ (prefsize) inversely proportional to $|\Psi_p^{\mathrm{OPT}}|$
- ⋆ (dbl1st) twice as much weight on the first half of the dispatches.
- ⋆ (dbl2nd) twice as much weight on the second half of the dispatches.

Stepwise bias strategies



Framework for Algorithm Learning



Ordinal Regression

Preference learning:

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

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

- ⋆ The preference is defined by a linear function:

$$h(\phi) = \sum_{i=1}^{d} w_i \phi$$

optimised w.r.t. $\mathbf{w}$ based on training data Ψ

- ⋆ Note: Limitations in approximation function to capture the complex dynamics incorporated in optimal trajectories.

Ordinal Regression

Preference learning:

- 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:

$$h(\phi) = \sum_{i=1}^{d} w_i \phi$$

optimised w.r.t. $\mathbf{w}$ based on training data Ψ

- Note: Limitations in approximation function to capture the complex dynamics incorporated in optimal trajectories.

Ordinal Regression

Preference learning:

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

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

- ⋆ The preference is defined by a linear function:

$$h(\phi) = \sum_{i=1}^{d} w_i \phi$$

  optimised w.r.t. $\mathbf{w}$ based on training data Ψ

- ⋆ Note: Limitations in approximation function to capture the complex dynamics incorporated in optimal trajectories.

Learning methods

Passive imitation learning (single pass):

- ⋆ Prediction with expert advice, $\pi_\star$ – Gurobi
- ⋆ Follow the perturbed leader (OPTϵ).

Active imitation learning (iterative):

- ⋆ Dataset Aggregation (DAgger)

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

where $\hat{\pi}_{i-1}$ is the previous learned model, and $\hat{\pi}_i$ learns on

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

aggregated dataset of all previous iterations.

Learning methods

Passive imitation learning (single pass):

- ⋆ Prediction with expert advice, $\pi_\star$ – Gurobi
- ⋆ Follow the perturbed leader (OPTϵ).

Active imitation learning (iterative):

- ⋆ Dataset Aggregation (DAgger)

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

where $\hat{\pi}_{i-1}$ is the previous learned model, and $\hat{\pi}_i$ learns on

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

aggregated dataset of all previous iterations.

Learning methods

Passive imitation learning (single pass):

- ⋆ Prediction with expert advice, $\pi_\star$ – Gurobi
- ⋆ Follow the perturbed leader (OPTϵ).

Active imitation learning (iterative):

- ⋆ Dataset Aggregation (DAgger)

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

where $\hat{\pi}_{i-1}$ is the previous learned model, and $\hat{\pi}_i$ learns on

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

aggregated dataset of all previous iterations.

Learning methods

Passive imitation learning (single pass):

- $\star$ Prediction with expert advice, $\pi_\star$ – Gurobi
- $\star$ Follow the perturbed leader (OPTϵ).

Active imitation learning (iterative):

- $\star$ Dataset Aggregation (DAgger)

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

where $\hat{\pi}_{i-1}$ is the previous learned model, and $\hat{\pi}_i$ learns on

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

aggregated dataset of all previous iterations.

DAgger

Require: $T \geq 1$

procedure DAgger($\pi_\star, \Phi^{\text{IL}0}, T$)

  $\hat{\pi}_0 \leftarrow$ Train($\Phi^{\text{IL}0}$) ▷ initial model, iff $\Phi^{\text{IL}0} = \Phi^{\text{OPT}}$

  for $i \leftarrow 1$ **to** T **do** ▷ at each imitation learning iteration

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

    Sample a K-solution using π_i ▷ IL($i, \hat{\pi}_{i-1}, \pi_\star$)

    $\Phi^{\text{IL}i} = \{(s, \pi_\star(s))\}$ ▷ visited by π_i and actions by $\pi_\star$

    $\Phi^{\text{DA}i} \leftarrow \Phi^{\text{DA}i-1} \cup \Phi^{\text{IL}i}$ ▷ aggregate datasets

    $\hat{\pi}_{i+1} \leftarrow$ Train($\Phi^{\text{DA}i}$) ▷ preference model

  end for

  return best $\hat{\pi}_i$ on validation ▷ best preference model

end procedure

DAgger

Require: $T \geq 1$

procedure DAgger($\pi_\star, \Phi^{\mathrm{IL}0}, T$)

  $\hat{\pi}_0 \leftarrow \mathrm{Train}(\Phi^{\mathrm{IL}0})$ ▷ initial model, iff $\Phi^{\mathrm{IL}0} = \Phi^{\mathrm{OPT}}$

  for $i \leftarrow 1$ **to** T **do** ▷ at each imitation learning iteration

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

    Sample a K-solution using π_i ▷ $\mathrm{IL}(i, \hat{\pi}_{i-1}, \pi_\star)$

    $\Phi^{\mathrm{IL}i} = \{(s, \pi_\star(s))\}$ ▷ visited by π_i and actions by $\pi_\star$

    $\Phi^{\mathrm{DA}i} \leftarrow \Phi^{\mathrm{DA}i-1} \cup \Phi^{\mathrm{IL}i}$ ▷ aggregate datasets

    $\hat{\pi}_{i+1} \leftarrow \mathrm{Train}(\Phi^{\mathrm{DA}i})$ ▷ preference model

  end for

  return best $\hat{\pi}_i$ on validation ▷ best preference model

end procedure

DAgger

Require: $T \geq 1$

```
procedure DAgger(π⋆, Φ^IL0, T)
    π̂_0 ← Train(Φ^IL0)                          ▷ initial model, iff Φ^IL0 = Φ^OPT
    for i ← 1 to T do                           ▷ at each imitation learning iteration
        Let π_i = β_i π⋆ + (1 − β_i) π̂_{i−1}
        Sample a K-solution using π_i           ▷ IL(i, π̂_{i−1}, π⋆)
        Φ^ILi = {(s, π⋆(s))}                    ▷ visited by π_i and actions by π⋆
        Φ^DAi ← Φ^DAi−1 ∪ Φ^ILi                 ▷ aggregate datasets
        π̂_{i+1} ← Train(Φ^DAi)                  ▷ preference model
    end for
    return best π̂_i on validation               ▷ best preference model
end procedure
```

DAgger

Require: $T \geq 1$

```
procedure DAgger(π⋆, Φ^IL0, T)
    π̂_0 ← Train(Φ^IL0)                         ▷ initial model, iff Φ^IL0 = Φ^OPT
    for i ← 1 to T do                           ▷ at each imitation learning iteration
        Let π_i = β_i π⋆ + (1 − β_i) π̂_{i−1}
        Sample a K-solution using π_i             ▷ IL(i, π̂_{i−1}, π⋆)
        Φ^ILi = {(s, π⋆(s))}                     ▷ visited by π_i and actions by π⋆
        Φ^DAi ← Φ^DAi−1 ∪ Φ^ILi                   ▷ aggregate datasets
        π̂_{i+1} ← Train(Φ^DAi)                   ▷ preference model
    end for
    return best π̂_i on validation               ▷ best preference model
end procedure
```

DAgger

Require: $T \geq 1$

```
procedure DAgger(π⋆, Φ^{IL0}, T)
    π̂_0 ← Train(Φ^{IL0})                         ▷ initial model, iff Φ^{IL0} = Φ^{OPT}
    for i ← 1 to T do                             ▷ at each imitation learning iteration
        Let π_i = β_i π⋆ + (1 − β_i) π̂_{i−1}
        Sample a K-solution using π_i              ▷ IL(i, π̂_{i−1}, π⋆)
        Φ^{ILi} = {(s, π⋆(s))}                     ▷ visited by π_i and actions by π⋆
        Φ^{DAi} ← Φ^{DAi−1} ∪ Φ^{ILi}              ▷ aggregate datasets
        π̂_{i+1} ← Train(Φ^{DAi})                   ▷ preference model
    end for
    return best π̂_i on validation                 ▷ best preference model
end procedure
```

DAgger

Require: $T \geq 1$

procedure DAgger($\pi_\star, \Phi^{\text{IL}0}, T$)
  $\hat{\pi}_0 \leftarrow \text{Train}(\Phi^{\text{IL}0})$ ▷ initial model, iff $\Phi^{\text{IL}0} = \Phi^{\text{OPT}}$
  for $i \leftarrow 1$ **to** T **do** ▷ at each imitation learning iteration
    Let $\pi_i = \beta_i \pi_\star + (1 - \beta_i)\hat{\pi}_{i-1}$
    Sample a K-solution using π_i ▷ $\text{IL}(i, \hat{\pi}_{i-1}, \pi_\star)$
    $\Phi^{\text{IL}i} = \{(s, \pi_\star(s))\}$ ▷ visited by π_i and actions by $\pi_\star$
    $\Phi^{\text{DA}i} \leftarrow \Phi^{\text{DA}i-1} \cup \Phi^{\text{IL}i}$ ▷ aggregate datasets
    $\hat{\pi}_{i+1} \leftarrow \text{Train}(\Phi^{\text{DA}i})$ ▷ preference model
  end for
  return best $\hat{\pi}_i$ on validation ▷ best preference model
end procedure

DAgger

Require: $T \geq 1$

procedure DAgger($\pi_\star, \Phi^{\text{IL}0}, T$)
    $\hat{\pi}_0 \leftarrow \text{Train}(\Phi^{\text{IL}0})$ ▷ initial model, iff $\Phi^{\text{IL}0} = \Phi^{\text{OPT}}$
    for $i \leftarrow 1$ **to** T **do** ▷ at each imitation learning iteration
        Let $\pi_i = \beta_i \pi_\star + (1 - \beta_i)\hat{\pi}_{i-1}$
        Sample a K-solution using π_i ▷ $\text{IL}(i, \hat{\pi}_{i-1}, \pi_\star)$
        $\Phi^{\text{IL}i} = \{(s, \pi_\star(s))\}$ ▷ visited by π_i and actions by $\pi_\star$
        $\Phi^{\text{DA}i} \leftarrow \Phi^{\text{DA}i-1} \cup \Phi^{\text{IL}i}$ ▷ aggregate datasets
        $\hat{\pi}_{i+1} \leftarrow \text{Train}(\Phi^{\text{DA}i})$ ▷ preference model
    end for
    return best $\hat{\pi}_i$ on validation ▷ best preference model
end procedure

DAgger

Require: $T \geq 1$

procedure DAgger($\pi_\star, \Phi^{\text{IL0}}, T$)

  $\hat{\pi}_0 \leftarrow \text{Train}(\Phi^{\text{IL0}})$ ▷ initial model, iff $\Phi^{\text{IL0}} = \Phi^{\text{OPT}}$

  for $i \leftarrow 1$ **to** T **do** ▷ at each imitation learning iteration

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

    Sample a K-solution using π_i ▷ $\text{IL}(i, \hat{\pi}_{i-1}, \pi_\star)$

    $\Phi^{\text{IL}i} = \{(s, \pi_\star(s))\}$ ▷ visited by π_i and actions by $\pi_\star$

    $\Phi^{\text{DA}i} \leftarrow \Phi^{\text{DA}i-1} \cup \Phi^{\text{IL}i}$ ▷ aggregate datasets

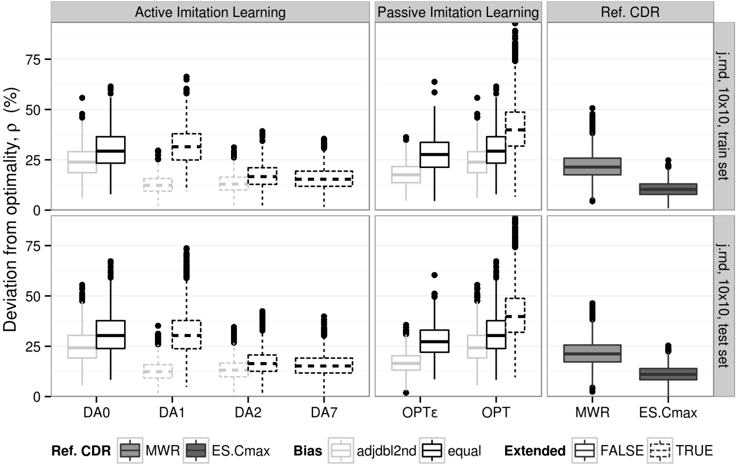
    $\hat{\pi}_{i+1} \leftarrow \text{Train}(\Phi^{\text{DA}i})$ ▷ preference model

  end for

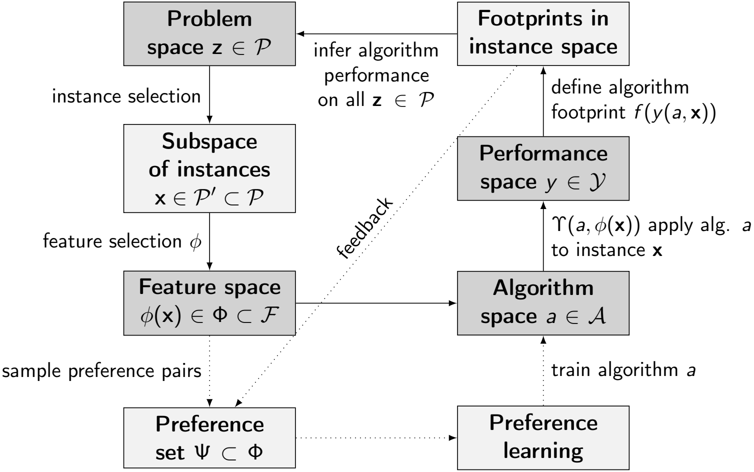
  return best $\hat{\pi}_i$ on validation ▷ best preference model

end procedure

Deviation from optimality, ρ



Framework for Algorithm Learning



Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- For a given problem domain, use a suitable problem generator to train and test on.
- Define features to grasp the essence of visited k-solutions
- Success is highly dependent on the preference pairs introduced to the system:
  - Ψ_p reduces the preference set without loss of performance.
  - Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- ⋆ For a given problem domain, use a suitable problem generator to train and test on.
- ⋆ Define features to grasp the essence of visited k-solutions
- ⋆ Success is highly dependent on the preference pairs introduced to the system:
  - ⋆ Ψ_p reduces the preference set without loss of performance.
  - ⋆ Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- ⋆ For a given problem domain, use a suitable problem generator to train and test on.
- ⋆ Define features to grasp the essence of visited k-solutions
- ⋆ Success is highly dependent on the preference pairs introduced to the system:
  - ⋆ Ψ_p reduces the preference set without loss of performance.
  - ⋆ Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- ⋆ For a given problem domain, use a suitable problem generator to train and test on.
- ⋆ Define features to grasp the essence of visited *k*-solutions
- ⋆ Success is highly dependent on the preference pairs introduced to the system:
  - ⋆ Ψ_p reduces the preference set without loss of performance.
  - ⋆ Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- For a given problem domain, use a suitable problem generator to train and test on.
- Define features to grasp the essence of visited k-solutions
- Success is highly dependent on the preference pairs introduced to the system:
  - Ψ_p reduces the preference set without loss of performance.
  - Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- ⋆ For a given problem domain, use a suitable problem generator to train and test on.
- ⋆ Define features to grasp the essence of visited k-solutions
- ⋆ Success is highly dependent on the preference pairs introduced to the system:
  - ⋆ Ψ_p reduces the preference set without loss of performance.
  - ⋆ Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework I

The thesis introduced a framework for learning (linear) composite priority dispatching rule – using job-shop as a case-study – with the following guidelines:

- For a given problem domain, use a suitable problem generator to train and test on.
- Define features to grasp the essence of visited k-solutions
- Success is highly dependent on the preference pairs introduced to the system:
  - Ψ_p reduces the preference set without loss of performance.
  - Stepwise bias is needed to balance time dependent Ψ_p in order to create time independent models.

It is non intuitive how to go about collecting training data.

Using Analysis & Learning Iterative Consecutive Executions framework II

Continued from prev. slide:

- ⋆ Learning optimal trajectories predominant in literature. Study showed Φ^{OPT} can result in insufficient knowledge.
- ⋆ Following sub-optimal deterministic policies, yet labelling with an optimal solver, improves the guiding policy.
- ⋆ In sequential decision making, all future observations are dependent on previous operations. Active update procedure using DAgger ensures sample states the learned model is likely to encounter is integrated to $\Psi_p^{\text{DA}i}$.
- ⋆ Instead of reusing same problem instances, extend the training set with new instances for quicker convergence of DAgger.

Using Analysis & Learning Iterative Consecutive Executions framework II

Continued from prev. slide:

- ⋆ Learning optimal trajectories predominant in literature. Study showed Φ^{OPT} can result in insufficient knowledge.
- ⋆ Following sub-optimal deterministic policies, yet labelling with an optimal solver, improves the guiding policy.
- ⋆ In sequential decision making, all future observations are dependent on previous operations. Active update procedure using DAgger ensures sample states the learned model is likely to encounter is integrated to $\Psi_p^{\mathrm{DA}i}$.
- ⋆ Instead of reusing same problem instances, extend the training set with new instances for quicker convergence of DAgger.

Using Analysis & Learning Iterative Consecutive Executions framework II

Continued from prev. slide:

- ⋆ Learning optimal trajectories predominant in literature. Study showed Φ^{OPT} can result in insufficient knowledge.
- ⋆ Following sub-optimal deterministic policies, yet labelling with an optimal solver, improves the guiding policy.
- ⋆ In sequential decision making, all future observations are dependent on previous operations. Active update procedure using DAgger ensures sample states the learned model is likely to encounter is integrated to $\Psi_p^{\mathrm{DA}i}$.
- ⋆ Instead of reusing same problem instances, extend the training set with new instances for quicker convergence of DAgger.

Using Analysis & Learning Iterative Consecutive Executions framework II

Continued from prev. slide:

- ⋆ Learning optimal trajectories predominant in literature. Study showed Φ^{OPT} can result in insufficient knowledge.
- ⋆ Following sub-optimal deterministic policies, yet labelling with an optimal solver, improves the guiding policy.
- ⋆ In sequential decision making, all future observations are dependent on previous operations. Active update procedure using DAgger ensures sample states the learned model is likely to encounter is integrated to $\Psi_p^{\mathrm{DA}i}$.
- ⋆ Instead of reusing same problem instances, extend the training set with new instances for quicker convergence of DAgger.

Using Analysis & Learning Iterative Consecutive Executions framework II

Continued from prev. slide:

- ⋆ Learning optimal trajectories predominant in literature. Study showed Φ^{OPT} can result in insufficient knowledge.
- ⋆ Following sub-optimal deterministic policies, yet labelling with an optimal solver, improves the guiding policy.
- ⋆ In sequential decision making, all future observations are dependent on previous operations. Active update procedure using DAgger ensures sample states the learned model is likely to encounter is integrated to $\Psi_p^{\mathrm{DA}i}$.
- ⋆ Instead of reusing same problem instances, extend the training set with new instances for quicker convergence of DAgger.

Future work

Main conclusions:

- ⋆ .

Thank you for your attention

Helga Ingimundardóttir
hei2@hi.is

Supplementary material:

- ⋆ Shiny application
- ⋆ Github.