ALICE

Analysis & Learning Iterative Consecutive Executions

Helga Ingimundardóttir

University of Iceland

June 30, 2016

Introduction

Motivation:

- The general goal is to train optimisation algorithms using data.

Contribution:

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

Introduction

Motivation:

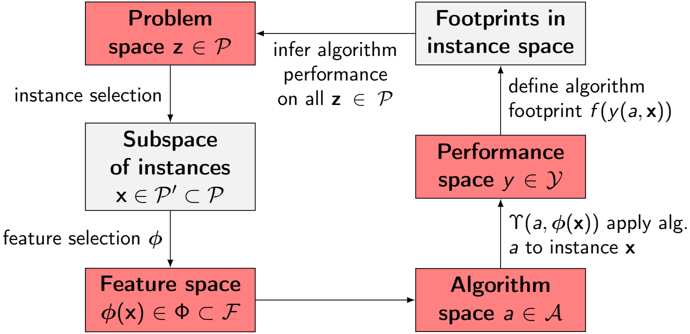
- ⋆ The general goal is to train optimisation algorithms using data.

Contribution:

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

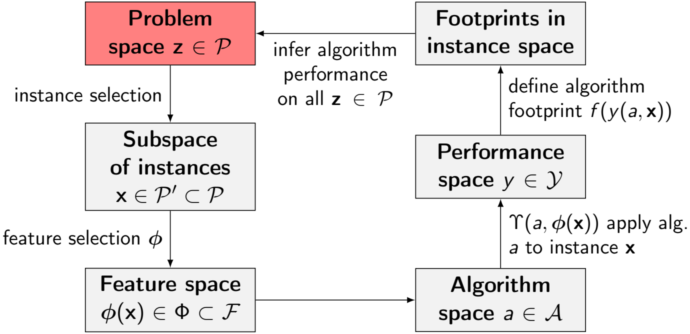
Framework for Algorithm Selection

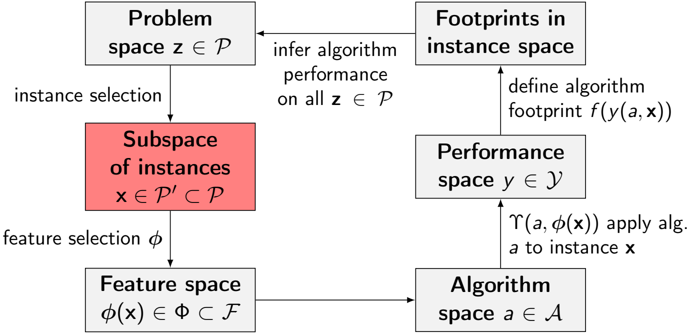
Overview of Rice (1976)



Framework for Algorithm Selection

Overview of Rice (1976)

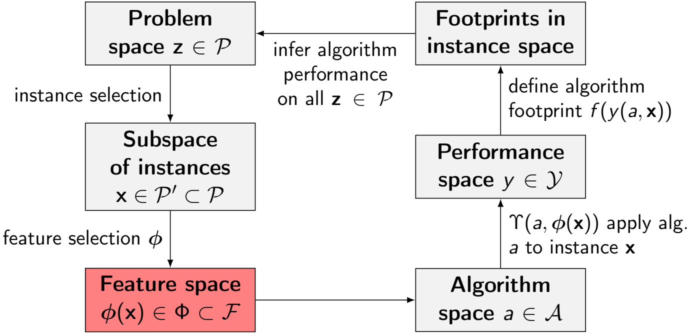




Problem space $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
instance selection
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
define algorithm footprint $f(y(a, \mathbf{x}))$
Performance space $y \in \mathcal{Y}$
feature selection ϕ
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$

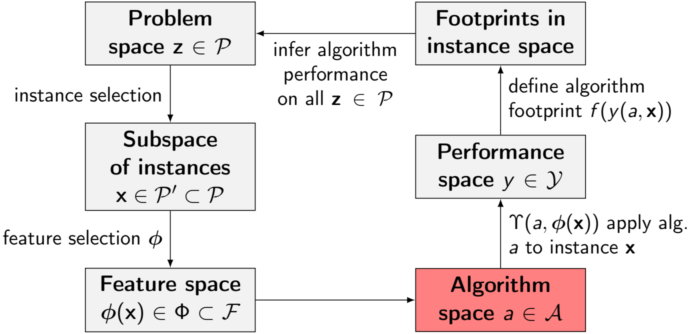
Framework for Algorithm Selection

Overview of Rice (1976)



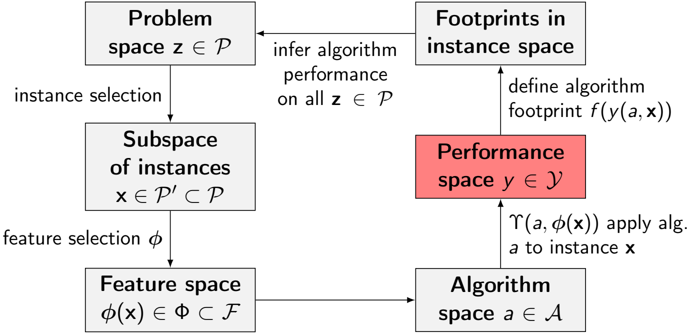
Framework for Algorithm Selection

Overview of Rice (1976)



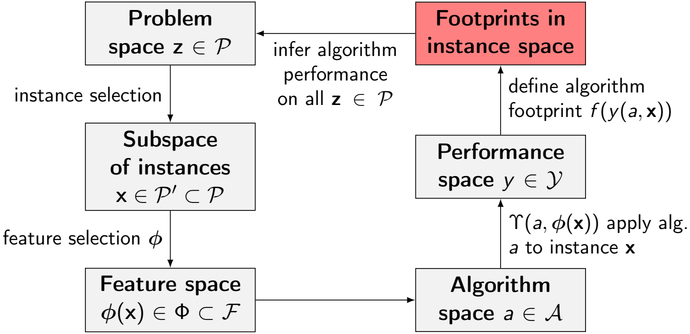
Framework for Algorithm Selection

Overview of Rice (1976)



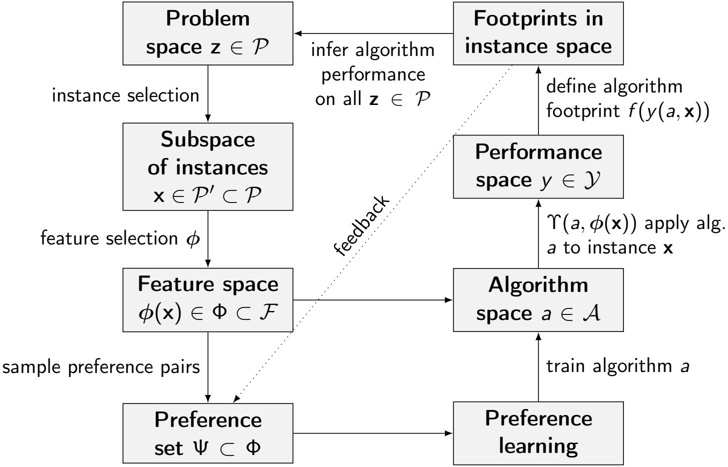
Framework for Algorithm Selection

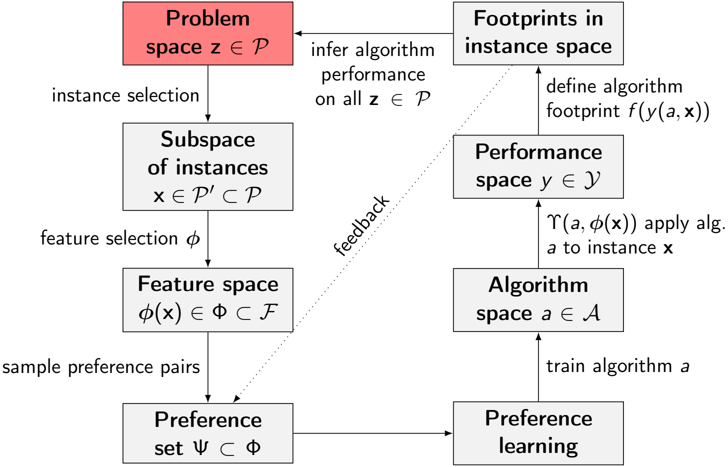
Overview of Rice (1976)



Framework for Algorithm Learning

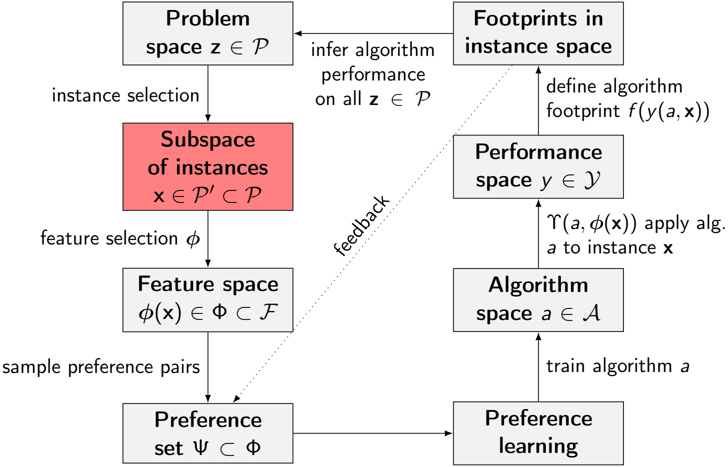
Overview of Rice (1976)

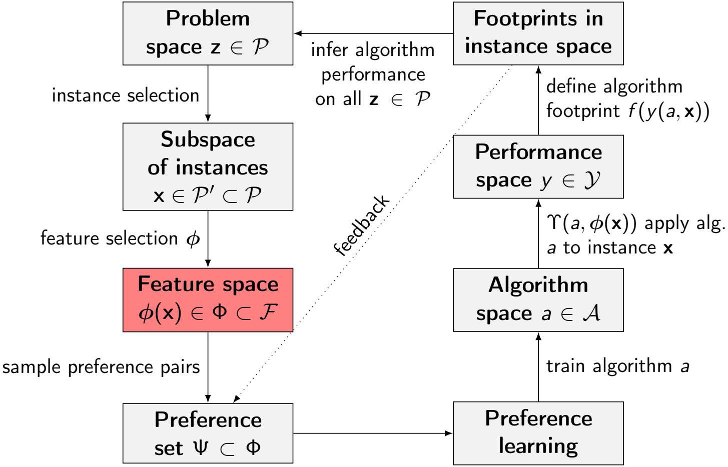


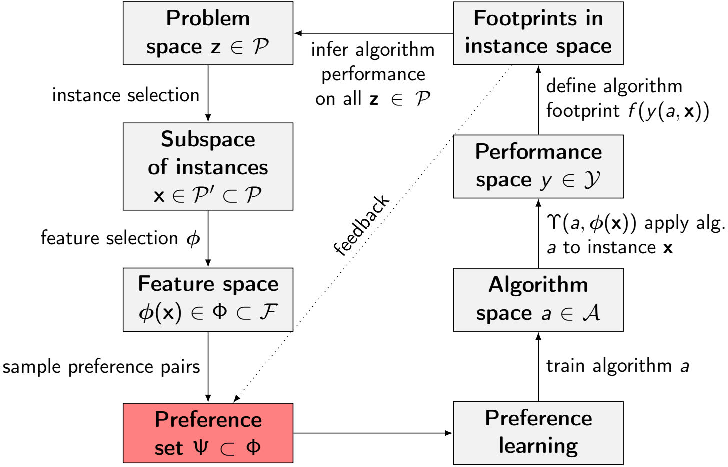

Problem space $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
instance selection
define algorithm footprint $f(y(a, \mathbf{x}))$
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
Performance space $y \in \mathcal{Y}$
feature selection ϕ
feedback
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$
sample preference pairs
train algorithm a
Preference set $\Psi \subset \Phi$
Preference learning

Framework for Algorithm Learning

Outline



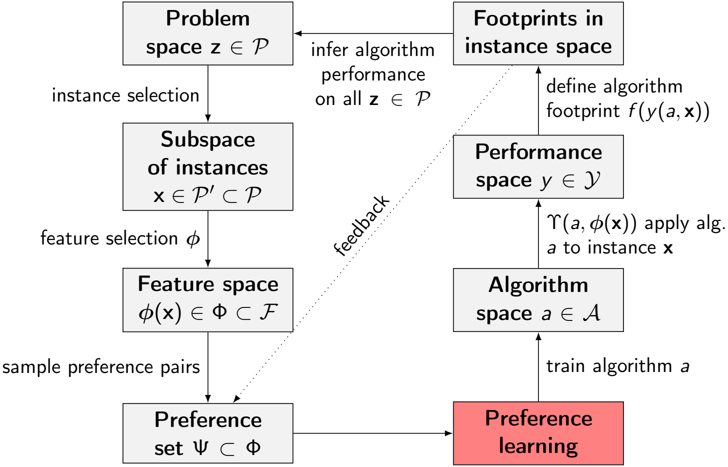

Problem space $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
instance selection
define algorithm footprint $f(y(a, \mathbf{x}))$
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
Performance space $y \in \mathcal{Y}$
feedback
feature selection ϕ
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$
sample preference pairs
train algorithm a
Preference set $\Psi \subset \Phi$
Preference learning

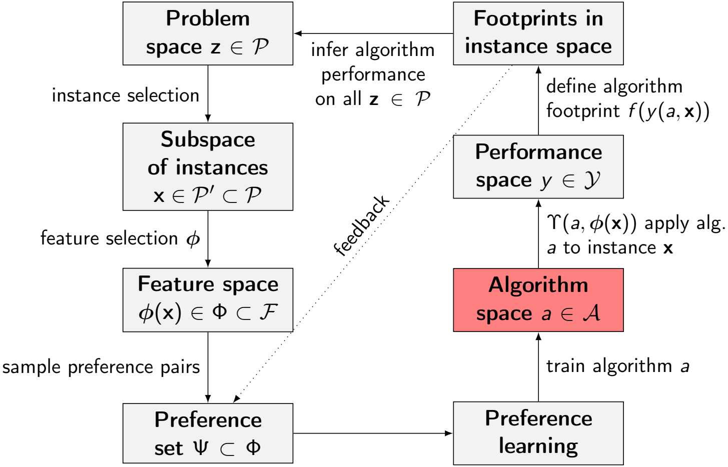


Problem space $\mathbf{z} \in \mathcal{P}$
instance selection
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
feature selection ϕ
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
sample preference pairs
Preference set $\Psi \subset \Phi$
Preference learning
train algorithm a
Algorithm space $a \in \mathcal{A}$
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Performance space $y \in \mathcal{Y}$
define algorithm footprint $f(y(a, \mathbf{x}))$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
feedback

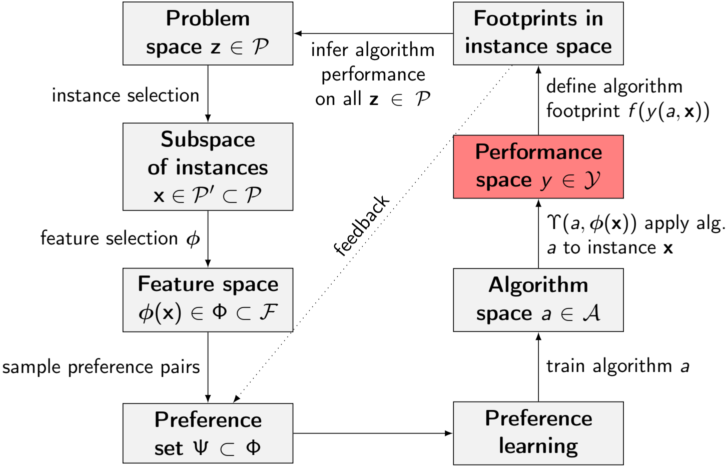
Framework for Algorithm Learning

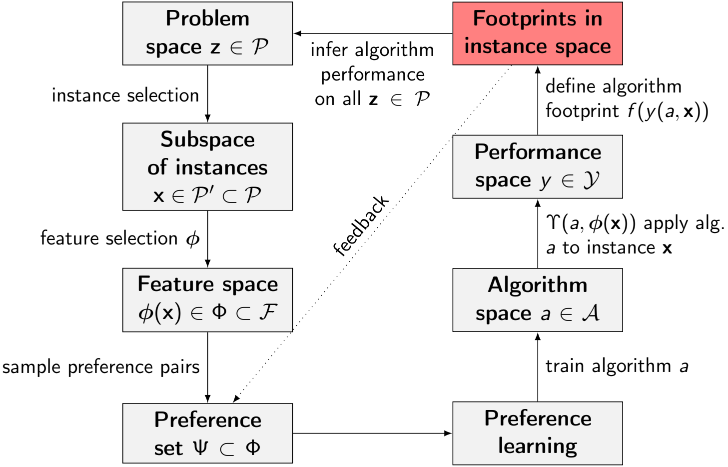
Outline

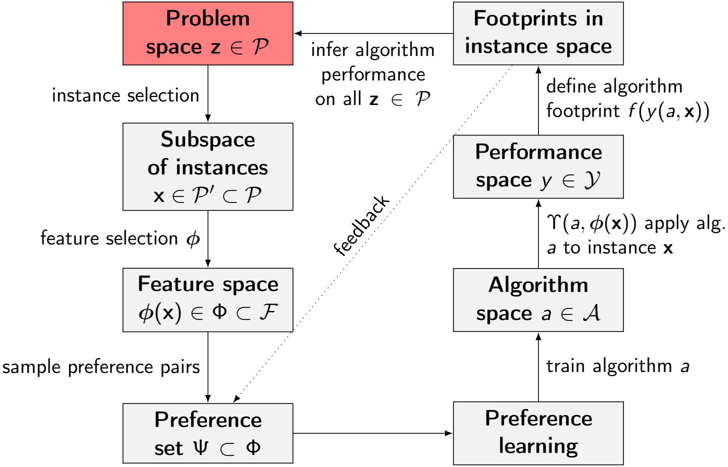



Problem space $\mathbf{z} \in \mathcal{P}$
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
instance selection
define algorithm footprint $f(y(a, \mathbf{x}))$
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
Performance space $y \in \mathcal{Y}$
feedback
feature selection ϕ
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$
sample preference pairs
train algorithm a
Preference set $\Psi \subset \Phi$
Preference learning

Framework for Algorithm Learning
Outline




Problem space $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
instance selection
define algorithm footprint $f(y(a, \mathbf{x}))$
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
Performance space $y \in \mathcal{Y}$
feedback
feature selection ϕ
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$
sample preference pairs
train algorithm a
Preference set $\Psi \subset \Phi$
Preference learning


Problem space $\mathbf{z} \in \mathcal{P}$
instance selection
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
feature selection ϕ
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
sample preference pairs
Preference set $\Psi \subset \Phi$
Preference learning
train algorithm a
Algorithm space $a \in \mathcal{A}$
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Performance space $y \in \mathcal{Y}$
define algorithm footprint $f(y(a, \mathbf{x}))$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
feedback

In this style 10/6

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 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).

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 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).

The attending guests:	They all have to:
J_1) Alice	M_1) have wine or pour tea
J_2) March Hare	M_2) spread butter
J_3) Dormouse	M_3) get a haircut
J_4) Mad Hatter.	M_4) check the time of the broken watch
	M_5) say what they mean.

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

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

The attending guests:	They all have to:
J_1) Alice	M_1) have wine or pour tea
J_2) March Hare	M_2) spread butter
J_3) Dormouse	M_3) get a haircut
J_4) Mad Hatter.	M_4) check the time of the broken watch
	M_5) say what they mean.

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

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

ALICE
Helga
Introduction
Problem Space
Subspace of Instances
Feature Space
Algorithm Space
Performance Space
Footprints in Instance Space
Preference Set
Preference Learning
Conclusions

Start: $k = 0$

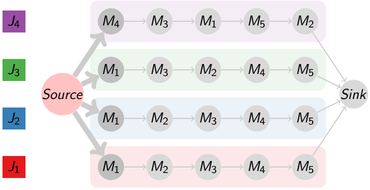


Figure: Disjunctive graph

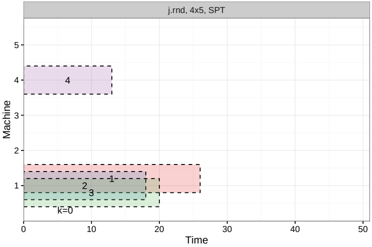


Figure: Gantt chart

Midway: $k = 10$

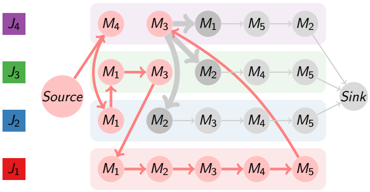


Figure: Disjunctive graph

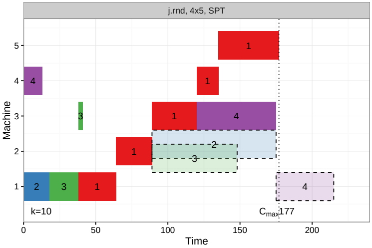


Figure: Gantt chart

Finish: $k = 20$

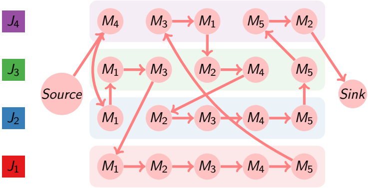


Figure: Disjunctive graph

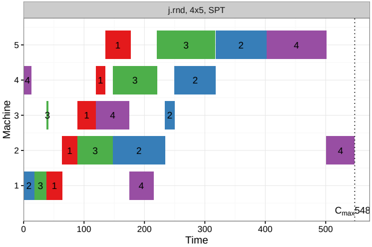
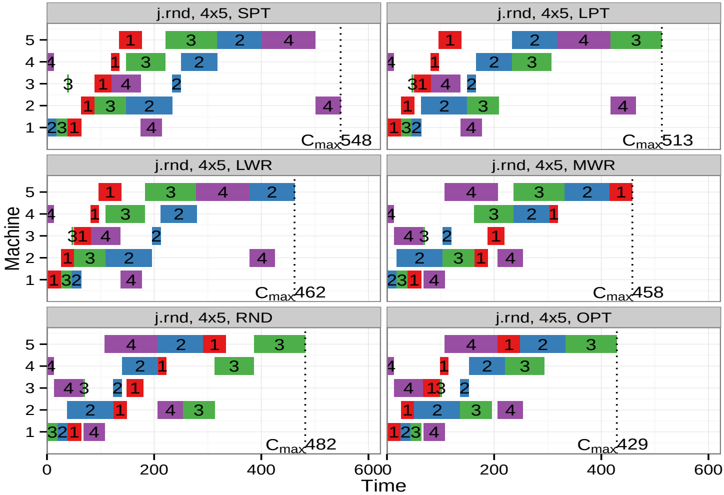
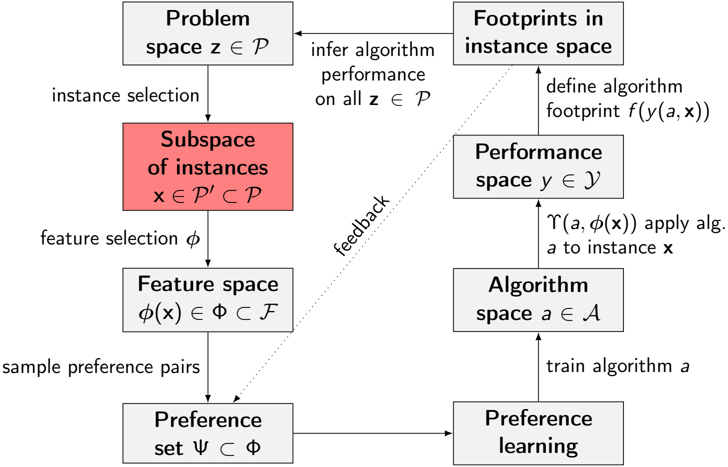


Figure: Gantt chart

Mad Hatter Tea-party

K-solutions





Problem space $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
instance selection
define algorithm footprint $f(y(a, \mathbf{x}))$
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
Performance space $y \in \mathcal{Y}$
feature selection ϕ
feedback
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$
sample preference pairs
train algorithm a
Preference set $\Psi \subset \Phi$
Preference learning

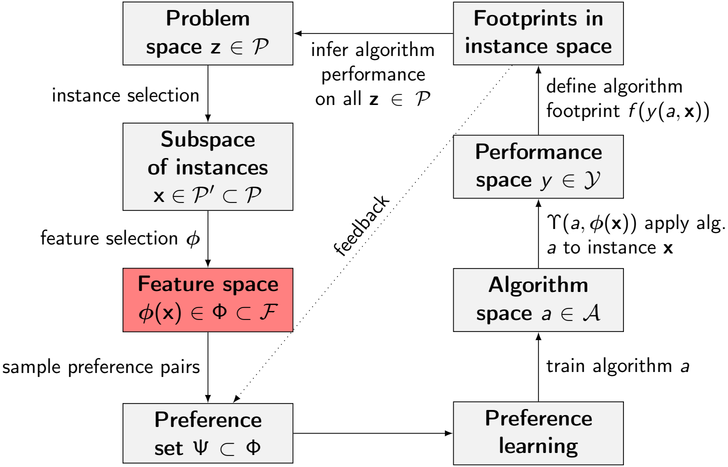
Problem Instance Generators

Based on Watson et al. (2002)

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

Framework for Algorithm Learning

Outline



Feature Space

for job-shop

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}

Feature Space
for job-shop

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}

Feature Space

for job-shop

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}

Trajectory Strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ (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).
- $\star$ ($\Phi^{\text{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{ALL}) union of all of the above.

Following the policy:

- ⋆ (Φ^{OPT}) expert $\pi_\star$.
- ⋆ (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).
- ⋆ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- ⋆ (Φ^{ALL}) union of all of the above.

Trajectory Strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ (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).
- $\star$ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{ALL}) union of all of the above.

Trajectory Strategies for Φ

Following the policy:

- ⋆ (Φ^{OPT}) expert $\pi_\star$.
- ⋆ (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).
- ⋆ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- ⋆ (Φ^{ALL}) union of all of the above.

Trajectory Strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ (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).
- $\star$ ($\Phi^{\text{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{ALL}) union of all of the above.

Trajectory Strategies for Φ

Following the policy:

- $\star$ (Φ^{OPT}) expert $\pi_\star$.
- $\star$ (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).
- $\star$ ($\Phi^{\text{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- $\star$ (Φ^{ALL}) union of all of the above.

Trajectory Strategies for Φ

Following the policy:

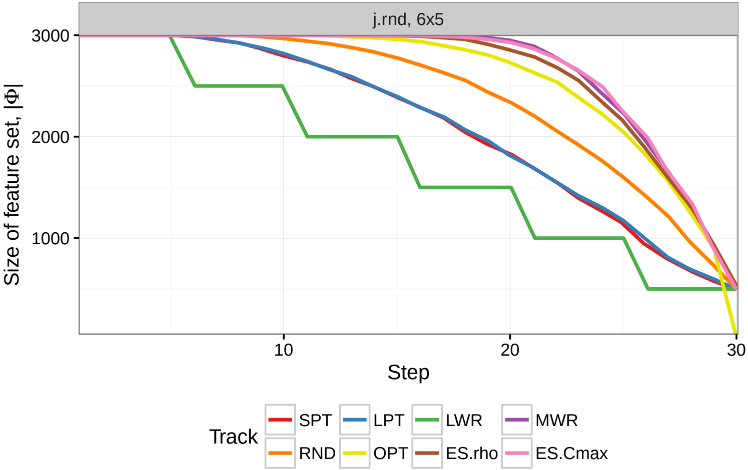
- ⋆ (Φ^{OPT}) expert $\pi_\star$.
- ⋆ (Φ^{SPT}) shortest processing time (SPT).
- ⋆ (Φ^{LPT}) longest processing time (LPT).
- ⋆ (Φ^{LWR}) least work remaining (LWR).
- ⋆ (Φ^{MWR}) most work remaining (MWR).
- ⋆ (Φ^{RND}) random policy (RND).
- ⋆ ($\Phi^{\mathrm{ES}.\rho}$) the policy obtained by optimising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

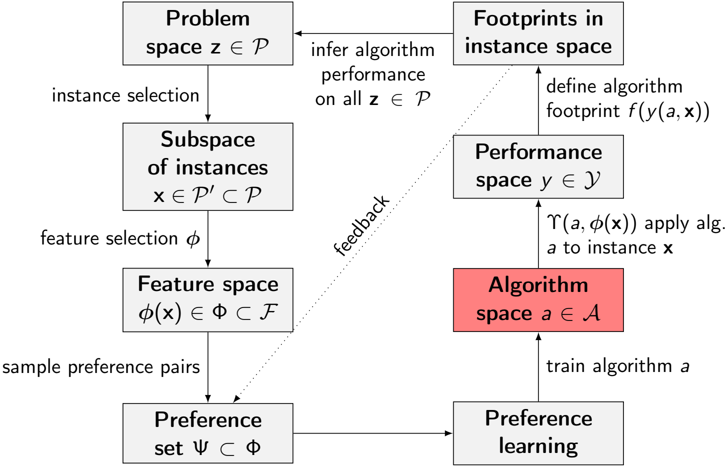
Following the policy:

- (Φ^{OPT}) expert $\pi_\star$.
- (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).
- ($\Phi^{\text{ES}.\rho}$) the policy obtained by optimising with CMA-ES.
- (Φ^{ALL}) union of all of the above.

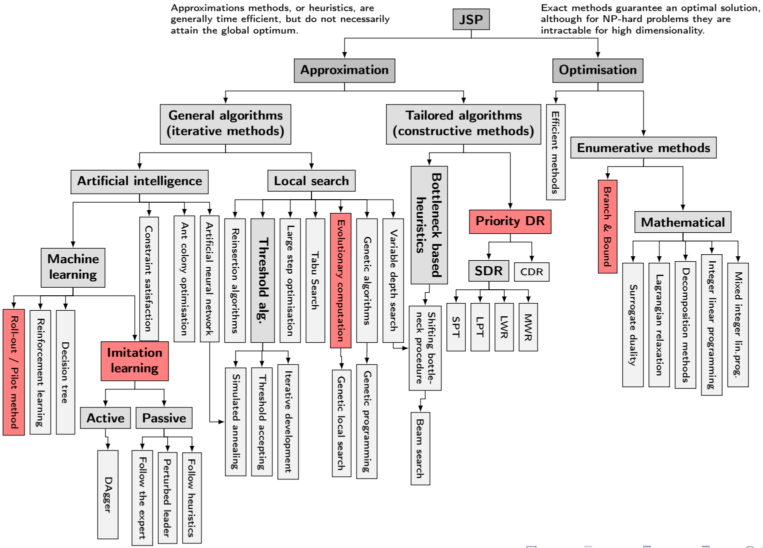
Sampled Size of $|\Phi(k)|$

6×5, $N_{train} = 500$





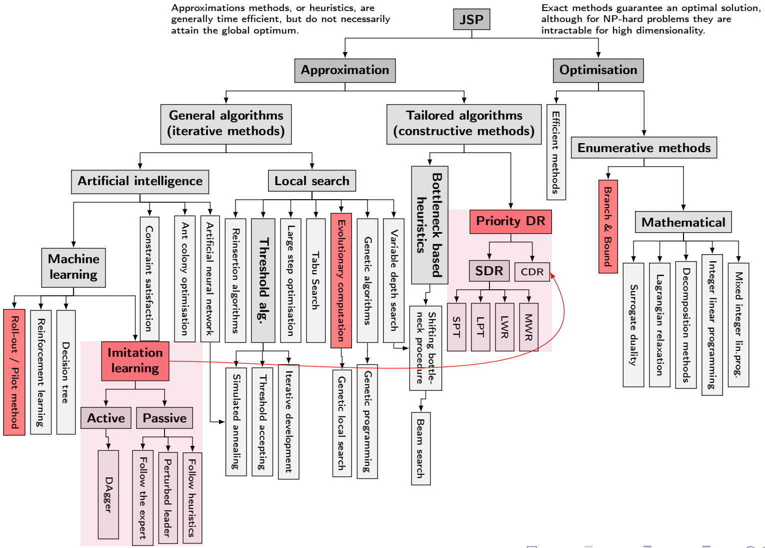
Problem space $\mathbf{z} \in \mathcal{P}$
instance selection
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
feature selection ϕ
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
sample preference pairs
Preference set $\Psi \subset \Phi$
Preference learning
train algorithm a
Algorithm space $a \in \mathcal{A}$
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Performance space $y \in \mathcal{Y}$
define algorithm footprint $f(y(a, \mathbf{x}))$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
feedback



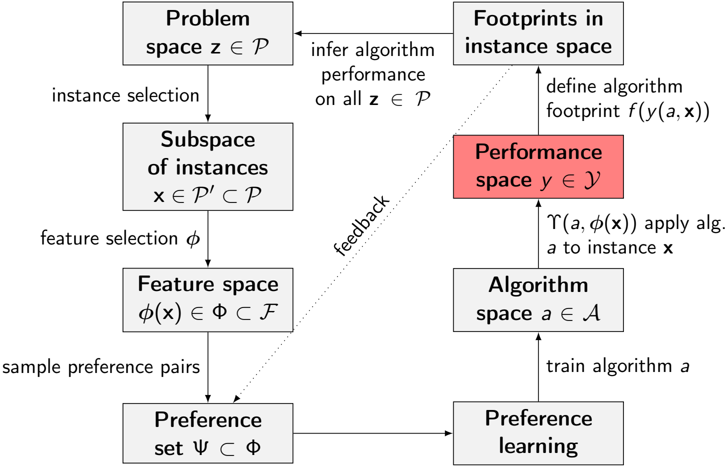
Approximations methods, or heuristics, are generally time efficient, but do not necessarily attain the global optimum.
JSP
Exact methods guarantee an optimal solution, although for NP-hard problems they are intractable for high dimensionality.
Approximation
Optimisation
General algorithms (iterative methods)
Tailored algorithms (constructive methods)
Efficient methods
Enumerative methods
Artificial intelligence
Local search
Bottleneck based heuristics
Priority DR
Branch & Bound
Mathematical
Machine learning
Constraint satisfaction
Ant colony optimisation
Artificial neural network
Reinsertion algorithms
Threshold alg.
Large step optimisation
Tabu Search
Evolutionary computation
Genetic algorithms
Variable depth search
SDR
CDR
Surrogate duality
Lagrangian relaxation
Decomposition methods
Integer linear programming
Mixed integer lin.prog.
Roll-out / Pilot method
Reinforcement learning
Decision tree
Imitation learning
Shifting bottle-neck procedure
SPT
LPT
LWR
MWR
Simulated annealing
Threshold accepting
Iterative development
Genetic local search
Genetic programming
Beam search
Active
Passive
DAgger
Follow the expert
Perturbed leader
Follow heuristics

Various Methods for Solving JSP

Based on Jain and Meeran (1999)



ALICE
Helga
Introduction
Problem Space
Subspace of Instances
Feature Space
Algorithm Space
Performance Space
Footprints in Instance Space
Preference Set
Preference Learning
Conclusions



Performance Measure

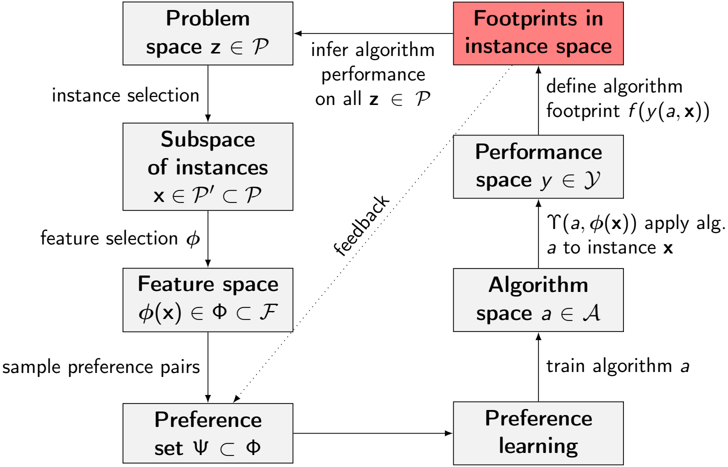
Performance of policy π compared with its optimal makespan, found using an expert policy, $\pi_\star$, is the following loss function:

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

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

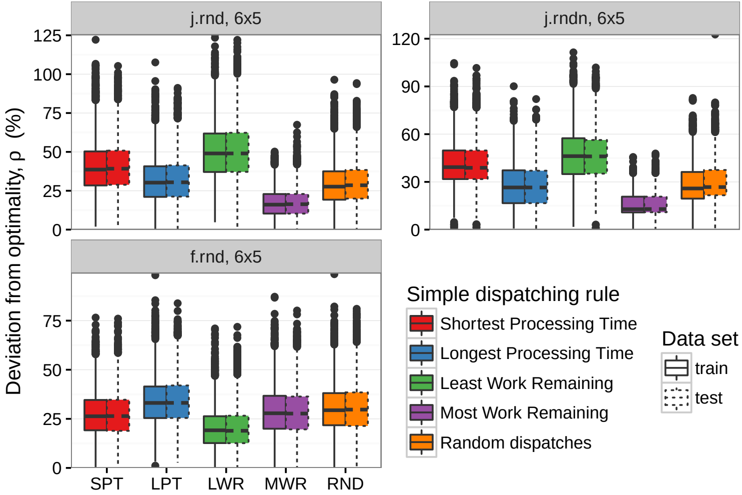
Framework for Algorithm Learning

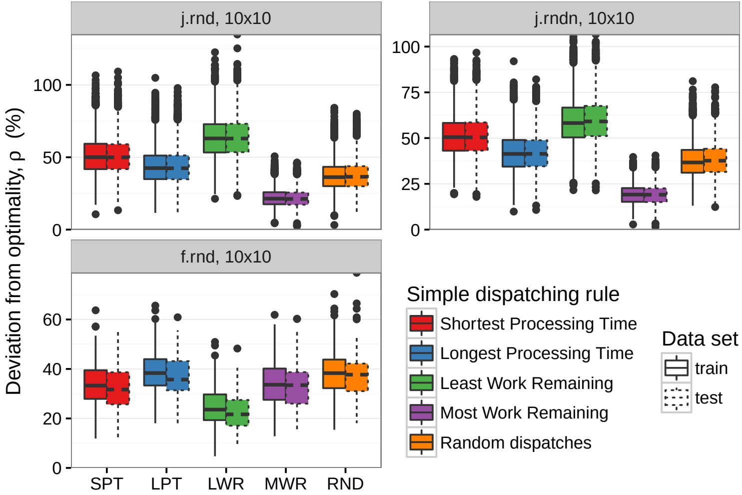
Outline



Deviation from Optimality

ρ

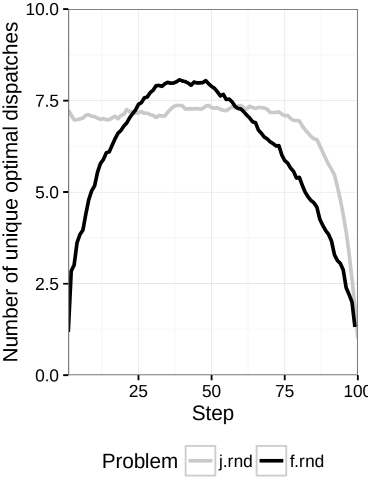


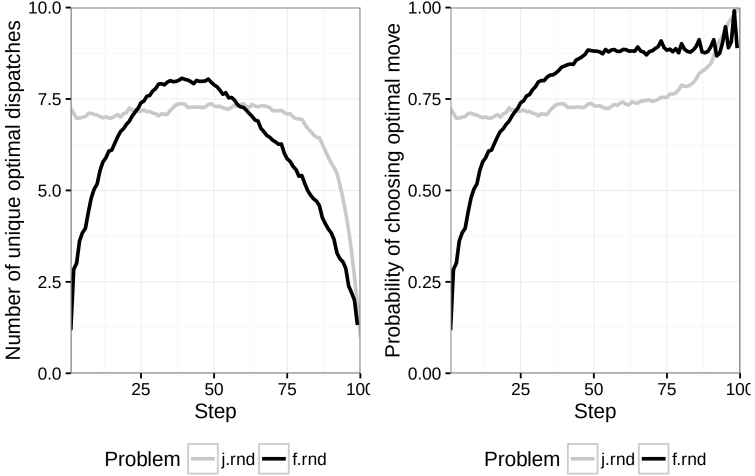


j.rnd, 10x10
j.rndn, 10x10
f.rnd, 10x10
Deviation from optimality, ρ (%)
SPT
LPT
LWR
MWR
RND
Simple dispatching rule
Shortest Processing Time
Longest Processing Time
Least Work Remaining
Most Work Remaining
Random dispatches
Data set
train
test

Making Optimal Decisions

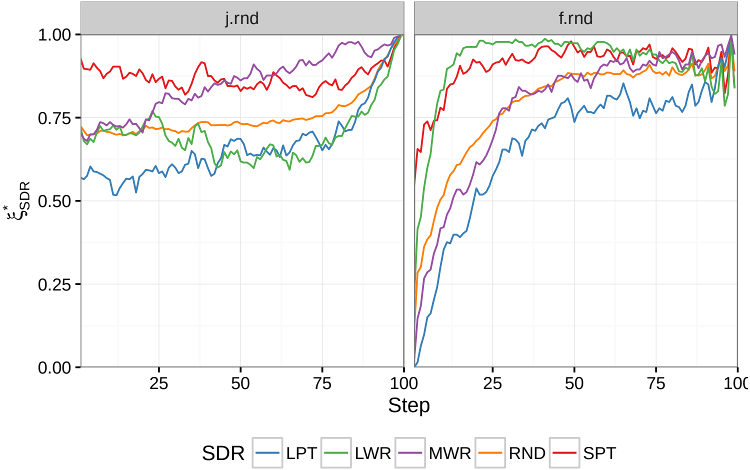
$\xi^{\star}_{\pi}$




Number of unique optimal dispatches
10.0
7.5
5.0
2.5
0.0
25
50
75
100
Step
Problem
j.rnd
f.rnd
Probability of choosing optimal move
1.00
0.75
0.50
0.25
0.00
25
50
75
100
Step
Problem
j.rnd
f.rnd

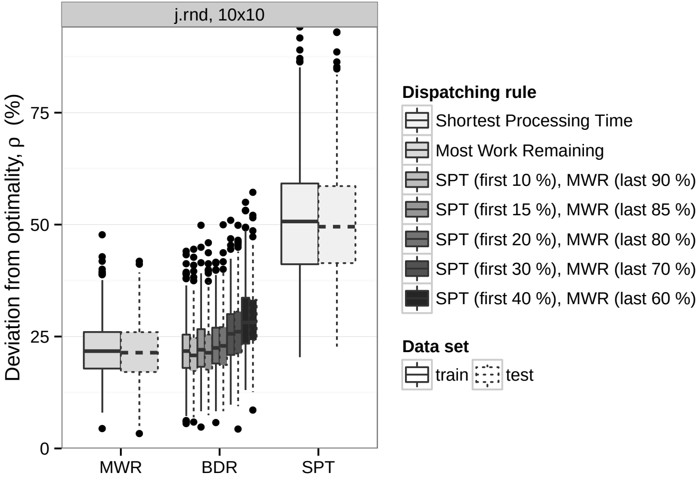
Probability of SDR Being Optimal

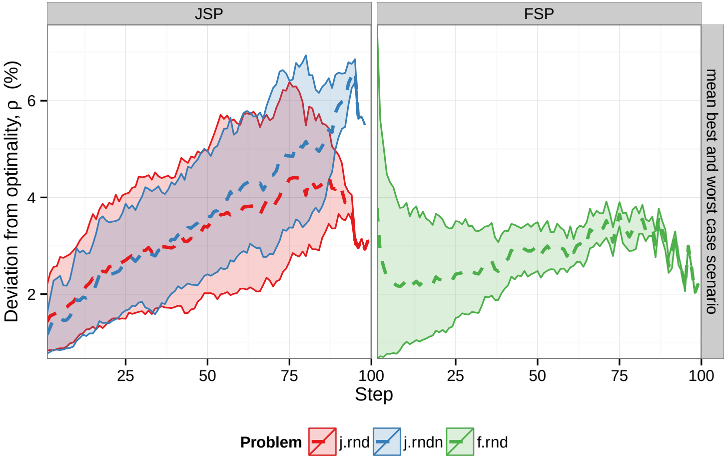
$\xi^{\star}_{\langle \text{SDR} \rangle}$



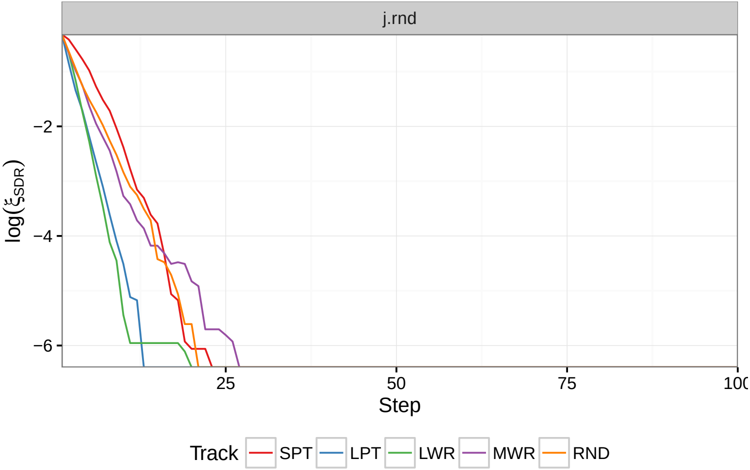
Blended Dispatching Rules

ρ





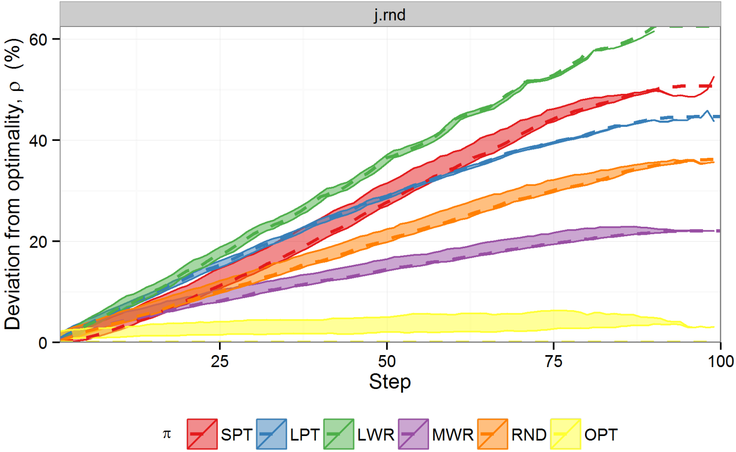
JSP
FSP
mean best and worst case scenario
Deviation from optimality, ρ (%)
2
4
6
25
50
75
100
Step
Problem
j.rnd
j.rndn
f.rnd



j.rnd
log(ξSDR)
−2
−4
−6
25
50
75
100
Step
Track
SPT
LPT
LWR
MWR
RND

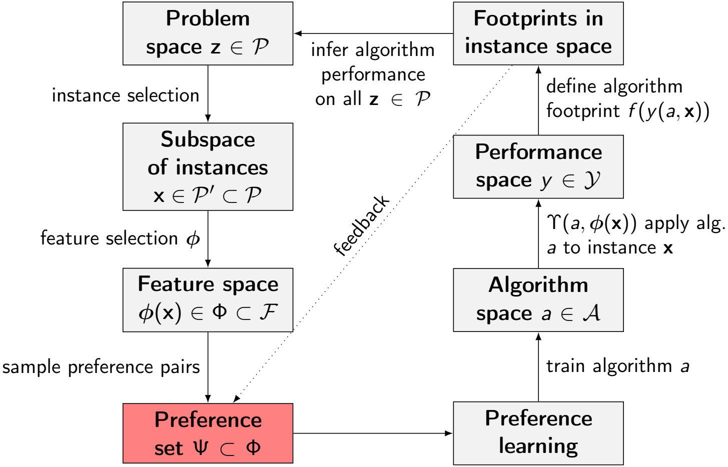
Impact of Sub-optimal Decision

$\{\zeta^{\pi}_{\min}, \zeta^{\pi}_{\max}\}$



Framework for Algorithm Learning

Outline



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:

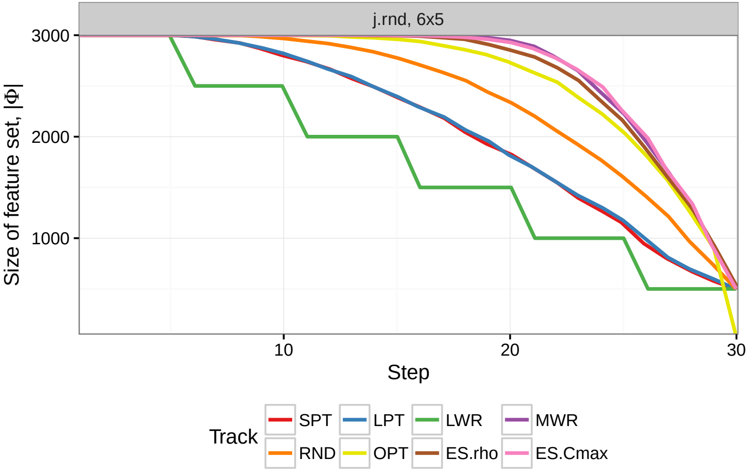
- Linear classification to identify good dispatches, from worse ones.
- Generate feature set, $\Phi \subset \mathcal{F}$, both from
  - optimal solutions, ϕ^o
  - suboptimal solutions, ϕ^s

  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.

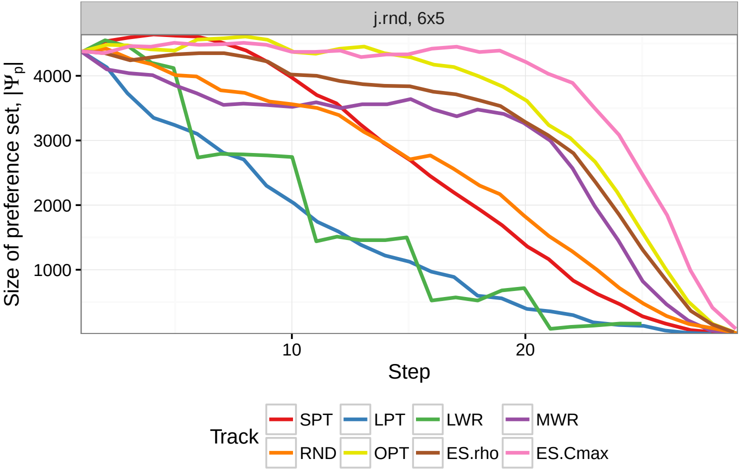
Sampled Size of $|\Phi(k)|$

6×5, $N_{train} = 500$



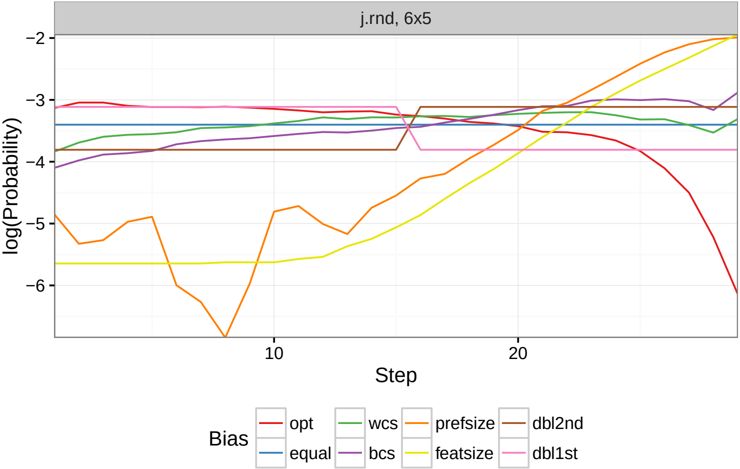
Sampled Size of $|\Psi(k)|$

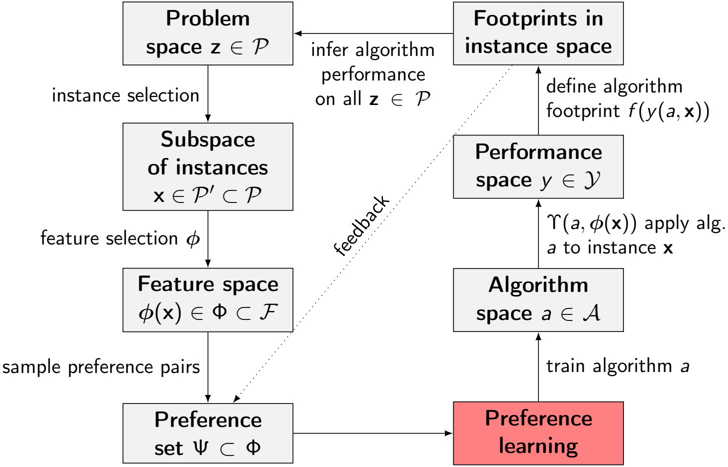
6×5, $N_{train} = 500$



Stepwise Bias Strategies

6×5, $N_{train} = 500$





Problem space $\mathbf{z} \in \mathcal{P}$
Footprints in instance space
infer algorithm performance on all $\mathbf{z} \in \mathcal{P}$
instance selection
define algorithm footprint $f(y(a, \mathbf{x}))$
Subspace of instances $\mathbf{x} \in \mathcal{P}' \subset \mathcal{P}$
Performance space $y \in \mathcal{Y}$
feedback
feature selection ϕ
$\Upsilon(a, \phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
Feature space $\phi(\mathbf{x}) \in \Phi \subset \mathcal{F}$
Algorithm space $a \in \mathcal{A}$
sample preference pairs
train algorithm a
Preference set $\Psi \subset \Phi$
Preference learning

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:

$$h(\phi) = \langle \mathbf{w} \cdot \phi \rangle$$

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.

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:

$$h(\phi) = \langle \mathbf{w} \cdot \phi \rangle$$

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.

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:

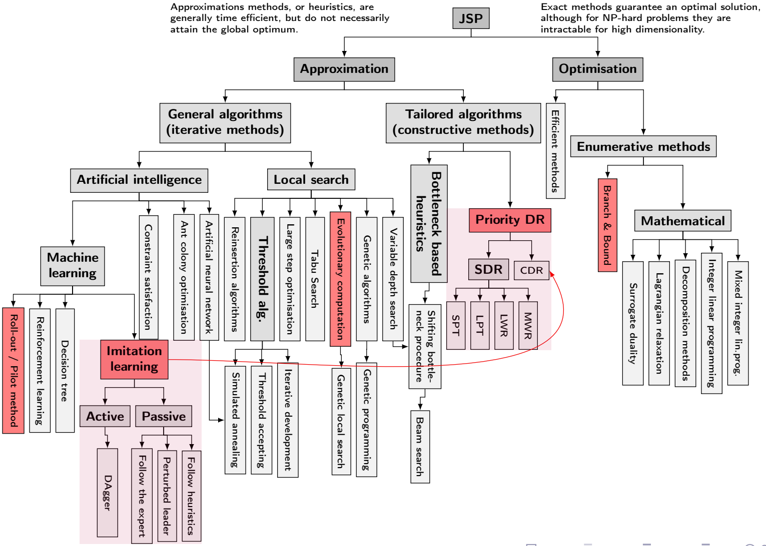
$$h(\phi) = \langle \mathbf{w} \cdot \phi \rangle$$

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.

Various Methods for Solving JSP

Based on Jain and Meeran (1999)



Passive imitation learning (single pass):

- Prediction with expert advice, $\pi_\star$
- Follow the perturbed leader (OPTϵ)
- Follow a heuristic (e.g. SDRs).

Passive imitation learning (single pass):

- $\star$ Prediction with expert advice, $\pi_\star$
- $\star$ Follow the perturbed leader (OPTϵ)
- $\star$ Follow a heuristic (e.g. SDRs).

Passive imitation learning (single pass):

- $\star$ Prediction with expert advice, $\pi_\star$
- $\star$ Follow the perturbed leader (OPTϵ)
- $\star$ Follow a heuristic (e.g. SDRs).

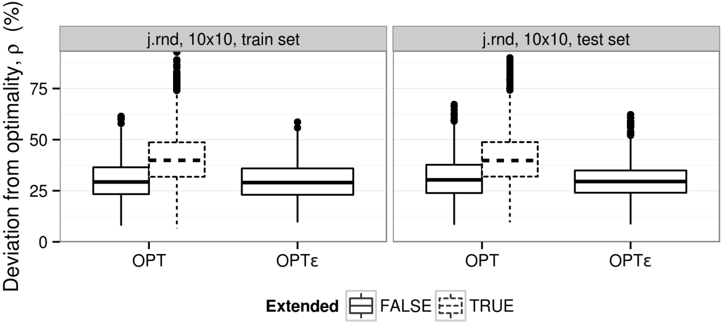
Passive Imitation Learning

Passive imitation learning (single pass):

- ⋆ Prediction with expert advice, $\pi_\star$
- ⋆ Follow the perturbed leader (OPTϵ)
- ⋆ Follow a heuristic (e.g. SDRs).

Passive imitation learning (single pass):

- ⋆ Prediction with expert advice, $\pi_\star$
- ⋆ Follow the perturbed leader (OPTϵ)
- ⋆ Follow a heuristic (e.g. SDRs).



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 aggregated dataset of all previous iterations.

Active Imitation Learning

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 aggregated dataset of all previous iterations.

Active Imitation Learning

Active imitation learning (iterative):

- ⋆ Dataset Aggregation (DAgger)

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

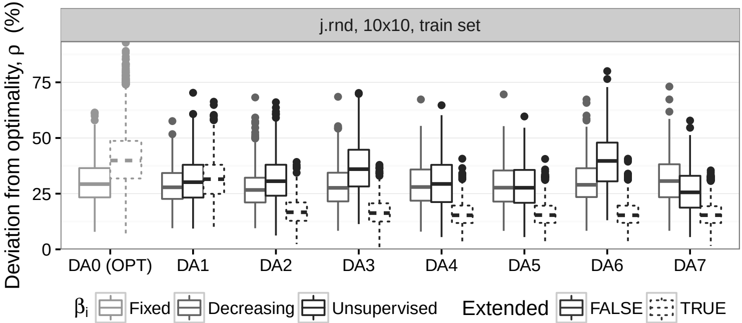
where $\hat{\pi}_{i-1}$ is the previous learned model, and $\hat{\pi}_i$ learns on aggregated dataset of all previous iterations.

Active imitation learning (iterative):

- ⋆ Dataset Aggregation (DAgger)

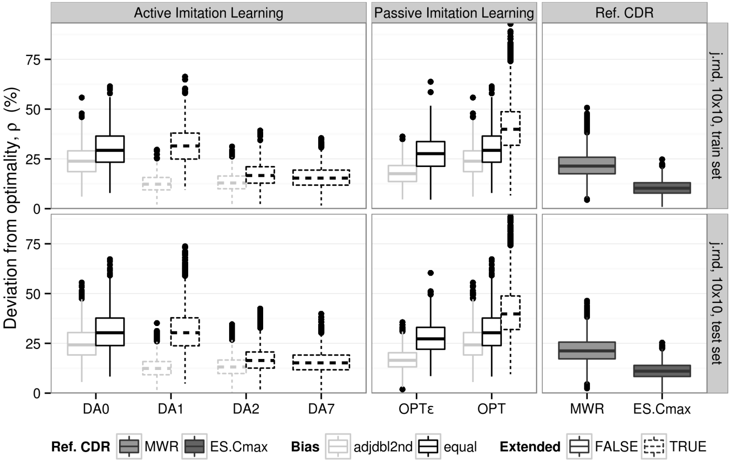
$$\pi_i = \beta_i \pi_\star + (1 - \beta_i)\hat{\pi}_{i-1}$$

where $\hat{\pi}_{i-1}$ is the previous learned model, and $\hat{\pi}_i$ learns on aggregated dataset of all previous iterations.



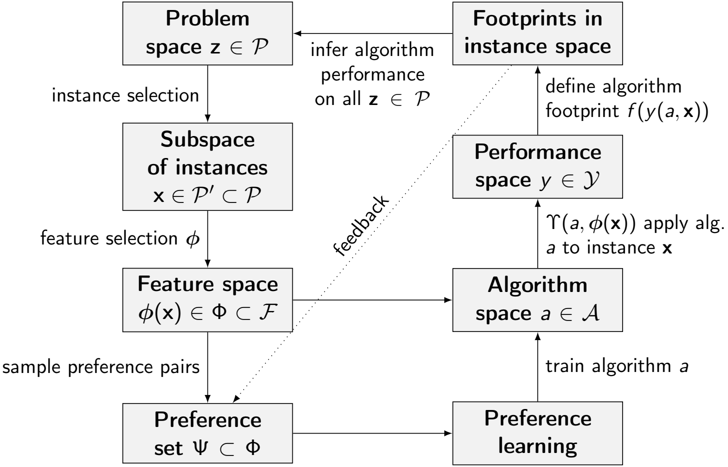
Deviation from Optimality

ρ



Framework for Algorithm Learning

Overview of Rice (1976)



The thesis introduces 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 ALICE Framework I

The thesis introduces 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 ALICE Framework I

The thesis introduces 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 ALICE Framework I

The thesis introduces 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.

ALICE
Helga
Introduction
Problem Space
Subspace of Instances
Feature Space
Algorithm Space
Performance Space
Footprints in Instance Space
Preference Set
Preference Learning
Conclusions

The thesis introduces 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 ALICE Framework I

The thesis introduces 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.

The thesis introduces 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 ALICE 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.
- ⋆ 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 the same problem instances, extend the training set with new instances for quicker convergence of DAgger.
- ⋆ In sequential decision making, all future observations are dependent on previous operations.

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.
- ⋆ 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 the same problem instances, extend the training set with new instances for quicker convergence of DAgger.
- ⋆ In sequential decision making, all future observations are dependent on previous operations.

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.
- ⋆ 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 the same problem instances, extend the training set with new instances for quicker convergence of DAgger.
- ⋆ In sequential decision making, all future observations are dependent on previous operations.

Using ALICE 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.
- ⋆ 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 the same problem instances, extend the training set with new instances for quicker convergence of DAgger.
- ⋆ In sequential decision making, all future observations are dependent on previous operations.

Using ALICE Framework II

Continued from prev. slide:

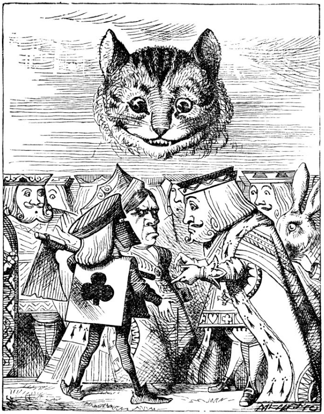
- ⋆ Learning optimal trajectories predominant in literature. Study showed Φ^{OPT} can result in insufficient knowledge.
- ⋆ Following sub-optimal deterministic policies, yet labelling with an optimal solver, improves the guiding policy.
- ⋆ Active update procedure using DAgger ensures sample states the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

Acknowledgements

Funding: University of Iceland's Research Fund.

Doctoral committee:

- ⋆ Prof. Tómas Philip Rúnarsson, University of Iceland (advisor).
- ⋆ Prof. Gunnar Stefánsson, University of Iceland.
- ⋆ Prof. Michèle Sebag, Université Paris-Sud.



Illustrations: Sir John Tenniel (1820–1914)

ALICE

Helga

Introduction
Problem Space
Subspace of Instances
Feature Space
Algorithm Space
Performance Space
Footprints in Instance Space
Preference Set
Preference Learning
Conclusions

Questions?

Helga Ingimundardóttir
hei2@hi.is

Supplementary material:

- ⋆ Shiny application
- ⋆ Github.