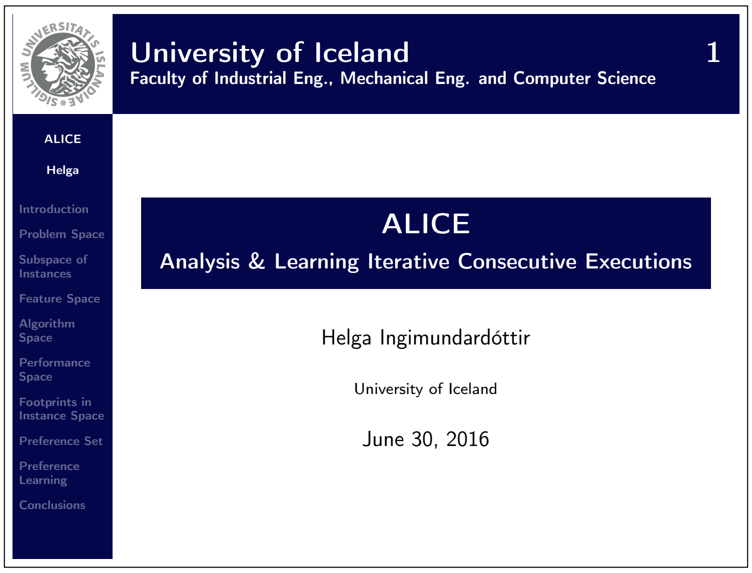


ALICE
Helga
Introduction
Problem Space
Subspace of Instances
Feature Space
Algorithm Space
Performance Space
Footprints in Instance Space
Preference Set
Preference Learning
Conclusions

ALICE
Analysis & Learning Iterative Consecutive Executions

Helga Ingimundardóttir

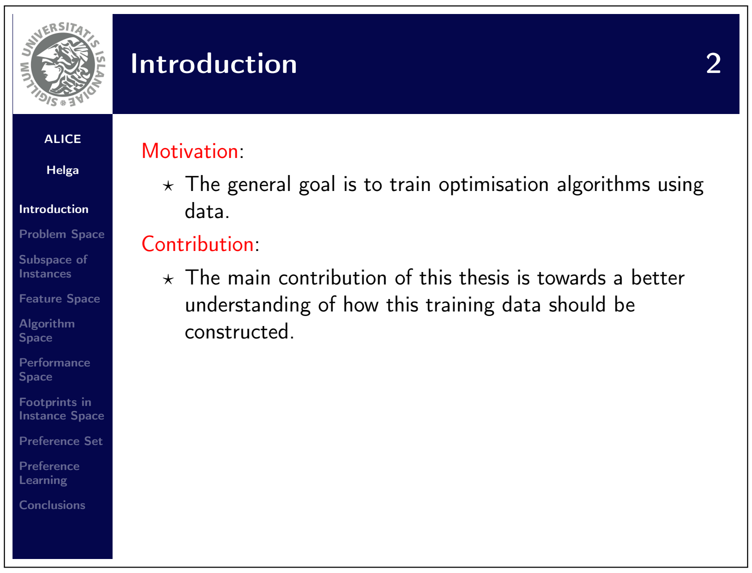
University of Iceland

June 30, 2016

- I would like to start by thanking the Faculty of Industrial Engineering, Mechanical Engineering and Computer Science for accepting my thesis for defence

$$\begin{array}{ll} \textbf{A} & \textbf{Analysis} \text{ or footprints} \\ \textbf{L} & \text{Preference } \textbf{Learning} \\ \textbf{I} & \textbf{Iterative} \text{ feedback improvements} \end{array}$$

$$\left.\begin{array}{ll} \textbf{C} & \textbf{Consecutive} \\ \textbf{E} & \textbf{Executions} \end{array}\right\} \begin{array}{l} \text{the search operator} \\ \text{in algorithm space} \end{array}$$



Introduction

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

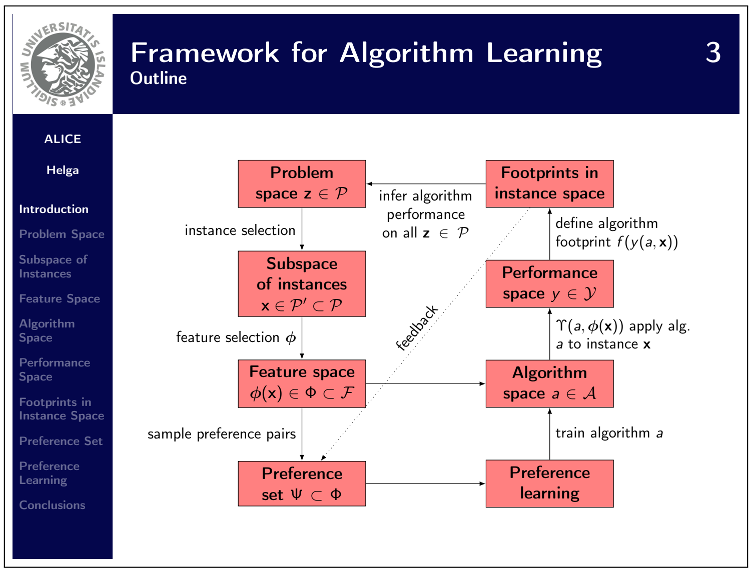
Motivation:

⋆ 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.

- I will start with a brief introduction
- The motivation is to train an optimisation algorithm using data
- This can be done for an arbitrary problem domain **forwards**
- The main contribution of the thesis is towards a better understanding of how this training should be constructed
- The quality of the training data will rewarded with a better learned algorithm



Framework for Algorithm Learning

Outline

3

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

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
$\Upsilon(a,\phi(\mathbf{x}))$ apply alg. a to instance $\mathbf{x}$
feedback
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

- The thesis presents a framework call ALICE to go about this

- It is built on Rice's framework for algorithm selection from 1976

- Here is it presented as a flow chart, emphasised with pink blocks. The additional blocks are for clarification of what's going on **forwards**:

i) We start with the problem space – this could be some arbitrary problem domain **forwards**

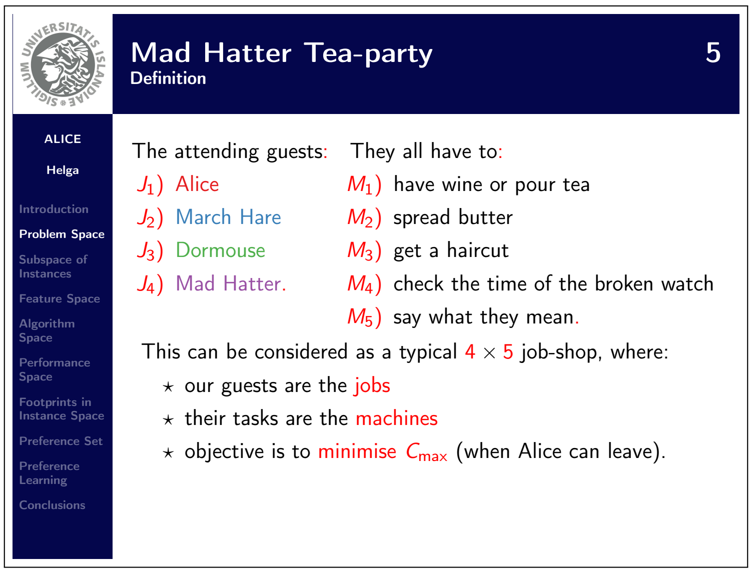
ii) Generally we will work with simulated data, so this will limit the problem space to a subspace of instances **forwards**

iii) Then we collect features. Features are used to describe the quality of a solution **forwards**

iv) From there we look at the algorithm space. This could be any kind algorithm you would want to test for your problem space. It could be for example a machine learning technique or evolutionary strategy. **forwards**

v) So take an algorithm and apply it to the features resulting for a given set of instances and we get a performance **forwards** *vi)* Now we can infer with the interaction of an algorithm and problem space using footprints in instance space (also known as landmarking) This could for explain why an algorithm is successful or unsuccessful for these types of problems .

- Adding on to Rice's framework **forwards** the following represents a framework for algorithm learning. The additional block are for the learning phase in the flow chart
- We would start in the same fashion as before **forwards** x3
- But from the collected features **forwards** we will create preferences that will serve as input **forwards** for the preference learning
- The outcome of which is **forwards** an algorithm, and we can continue **forwards** our framework as before and **forwards** redo our analysis.
- From here we can use the footprints analysis to iteratively improve the learning model via feedback to the preference set
- This has been a brief overview, and now I will address each block in more detail.



Mad Hatter Tea-party
Definition

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

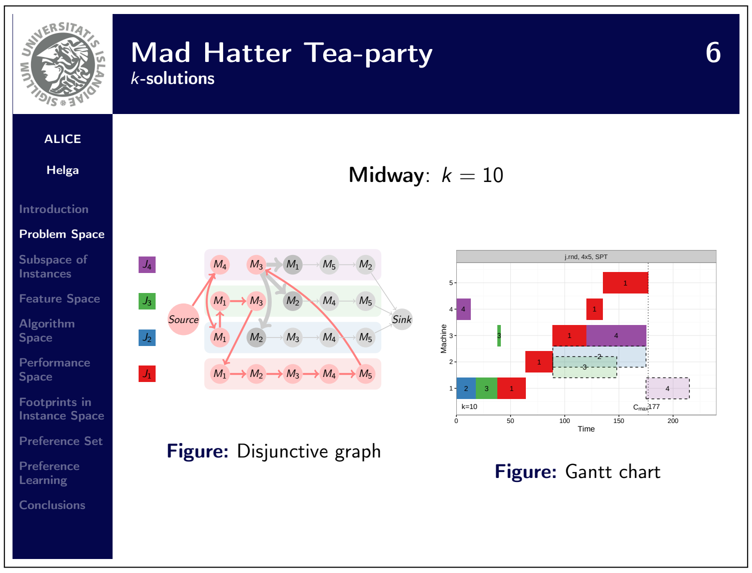
The attending guests:	They all have to:
J_1) Alice	M_1) have wine or pour tea
J_2) March Hare	M_2) spread butter
J_3) Dormouse	M_3) get a haircut
J_4) Mad Hatter.	M_4) check the time of the broken watch
	M_5) say what they mean.

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

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

- Starting with Problem Space (OVERVIEW) **forwards** The thesis uses job-shop scheduling problem as a case study, and I will explain using an hypothetical example based around **forwards** The Mad Hatter's Tea Party
- Here we can see four guest: Alice, March Hare, Dourmouse and the host Mad Hatter
- All of them have to perform 5 different activities and the time it takes can very between guests **forwards**
- So this can be considered as a 4-job 5-machine job-shop subject to the constraints
- **forwards** none of the guests can multi task and each must follow a predetermined order for their tasks (e.g. March Hare insists on doing them alphabetically) and they would rather wait than breaking habit
- **forwards** and that a machine can handle at most one job at a time (e.g. two can't simultaneously pour from the same teapot)
- We know Alice needs to see the Red Queen ASAP however she's polite and won't leave until everyone has finished their activity

- **forwards** so the objective is to schedule the jobs so as to minimise the maximum completion time (so-called makespan) – when Alice can leave



Mad Hatter Tea-party

k-solutions

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

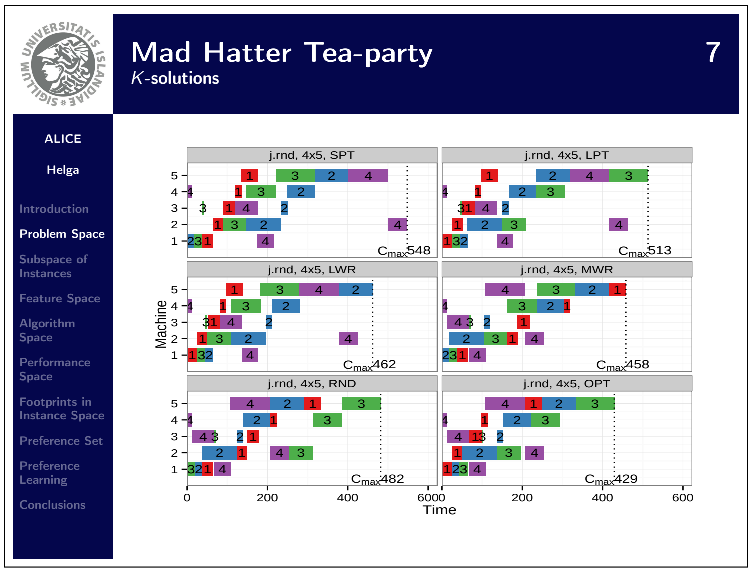
Midway: $k = 10$

Figure: Disjunctive graph

Figure: Gantt chart

- We arrived at the tea party with a clean slate.
- The disjunctive grasp has a swim-lane for each guest that yields the predetermined ordering, so we go from left to right – but we can hop between lanes.
- We need to visit all of the nodes (guest performing an activity) in the graph with a Hamiltonian path (a total of 20 operations).
- What we need to do know is to select from the available jobs (grey in DG) which one should we select next based on which one of these jobs has the highest priority – this is essentially a DR
- The Gantt chart then translates the dispatched operations into physical time or *seconds* (eg at what o'clock should Alice start/finish pouring tea)
- **forwards** Here we are further along the dispatching seq. and we can see the partial schedule to emerge.
- Solids are already chosen (pink in DG) and dashed are potential outcomes for the available jobs (Notice Alice is finished, so only 3 op. currently remain)

- **forwards** We do these executions consecutively until we've visited all nodes in the DG. So this is an example of a completed solution.
- Here Mad Hatter is the last to finish, so the makespan is at 548 seconds (9min).
- Notice that Alice finished much earlier and there is a bottleneck for taking turns in speaking (M_5), is there some other way to arrange these tasks so Alice could leave sooner?



Mad Hatter Tea-party

K-solutions

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

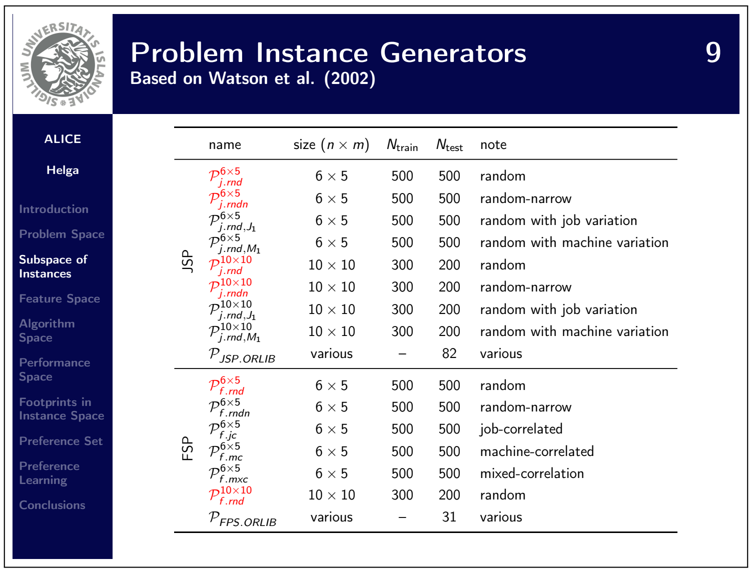
- Here the top four are the outcomes for our Tea-Party using commonly used single priority dispatching rule from the literature. They are simple to use yet surprisingly effective. They are called SPT, LPT, LWR and MWR.

- Previous slides showed SPT being constructed with just over 9min, but MWR would have taken $7\frac{1}{2}$min

- A random solution is shown in the lower left-hand-corner (8min)

- However an optimal solution (obtained by expert policy or *narrator*) would have been for example the one shown in the lower right hand corner, with Alice leaving in 7min

- What we want to now learn is a deterministic policy (or heuristic) that closely resembles the expert policy in the right-hand-corner (from these trials, MWR is the best heuristic policy)



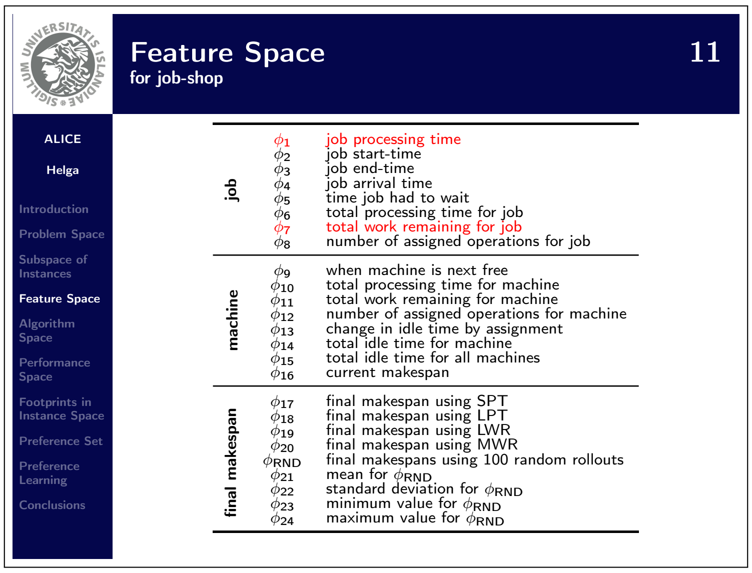
Problem Instance Generators

Based on Watson et al. (2002)

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

	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

- Moving on, given the problem space job-shop we will use simulated data so this is the subspace of instances (OVERVIEW) **forwards**
- The thesis address various problem generators
- There are 5 types of processing time distributions (eg seconds it takes Alice to pour a cup of tea)
- There are also 2 variations of machine ordering – if it distinct between all jobs it is called job-shop or if it is the same for all jobs it is called flow-shop
- Moreover, there are 2 problem sizes explored 6×5 that require 30 operations and 10×10 requiring 100 operations.
- **forwards** However, only the highlighted subspaces will be explored in this presentation



Feature Space

for job-shop

11

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

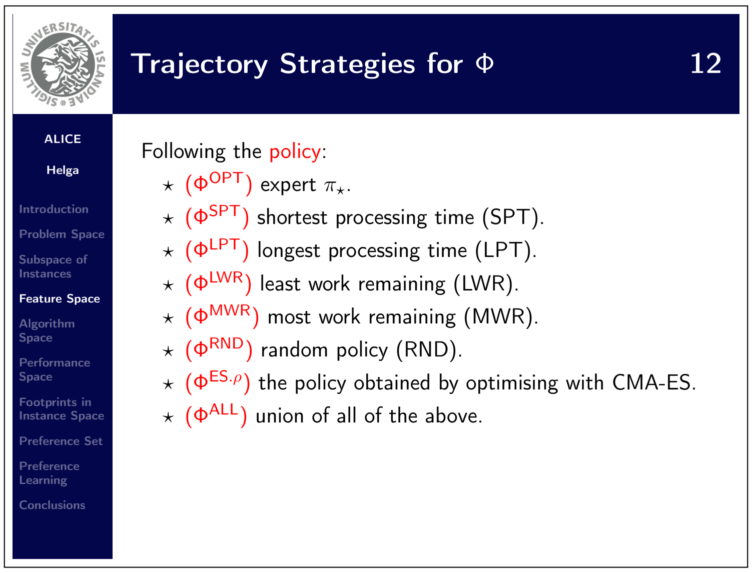
Preference Set

Preference Learning

Conclusions

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}

- Now we can start looking into features for job-shop (OVERVIEW) **forwards**
- Features are a way to measure the quality of our solution which is being built
- Here we can see 16 one-step lookahead features and 8 rollout features, but the presentation will only consider one-step lookahead
- They are inspired from commonly used single priority dispatching rule
- **forwards** ϕ_1 corresponds to shortest/longest proc. time
- **forwards** ϕ_7 corresponds to least/most work remaining



Trajectory Strategies for Φ

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

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.

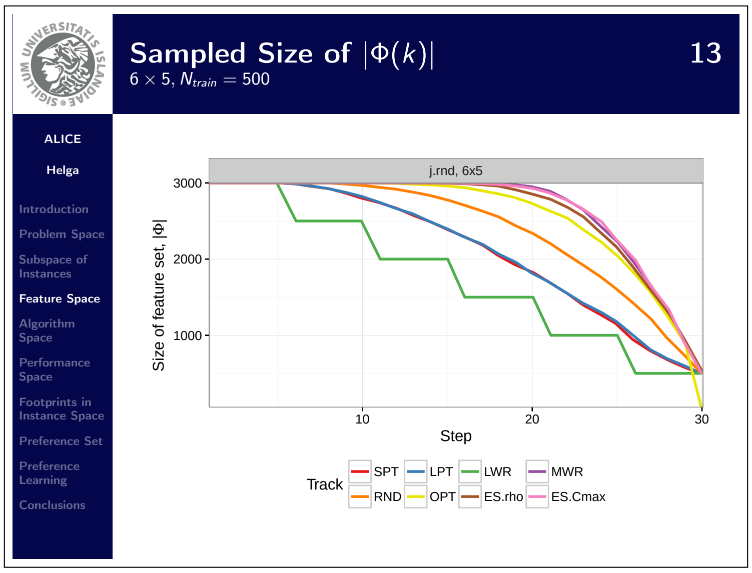
- We can sample features from k-solutions using some policy to guide a trajectory through feature space

- **forwards** that could obtained from an expert policy – so inspecting the features encountered while creating an optimal solution

- **forwards** or we can use some deterministic single priority dispatching rule like SPT **forwards** or LPT **forwards** LWR **forwards** MWR **forwards** or some random trajectory.

- **forwards** instead of following single priority dispatching rule we could also use some more sophisticated heuristic, such as composite priority dispatching rule found by optimising with evolutionary search

- **forwards** and we can aggregate features across different trajectory policies



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

6×5, $N_{train} = 500$

13

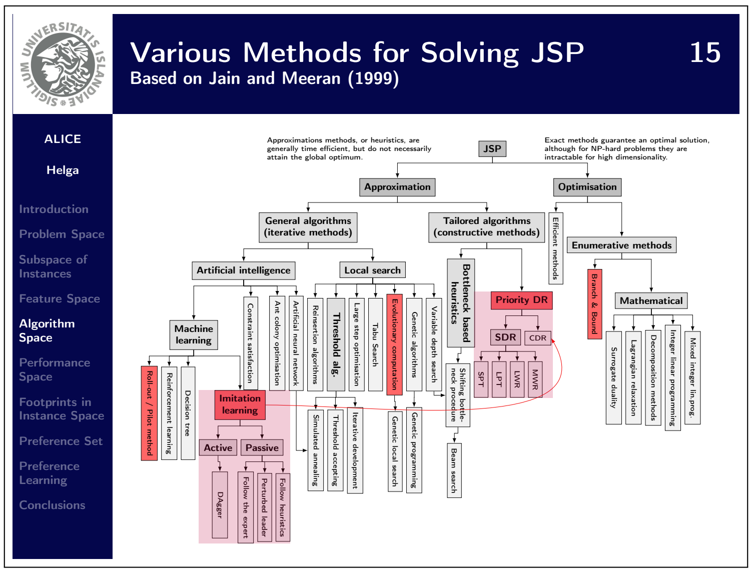
ALICE

Helga

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

- Remark from our Tea-party example, that Alice finished all of her activities quite early ($k < 10$), so even though there are 4-jobs, they can number of available operations we can collect features from diminishes as the dispatching sequence progresses

- Here we can see the amount of features encountered evolving with the dispatching step for the aforementioned trajectories



Various Methods for Solving JSP

15

Based on Jain and Meeran (1999)

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

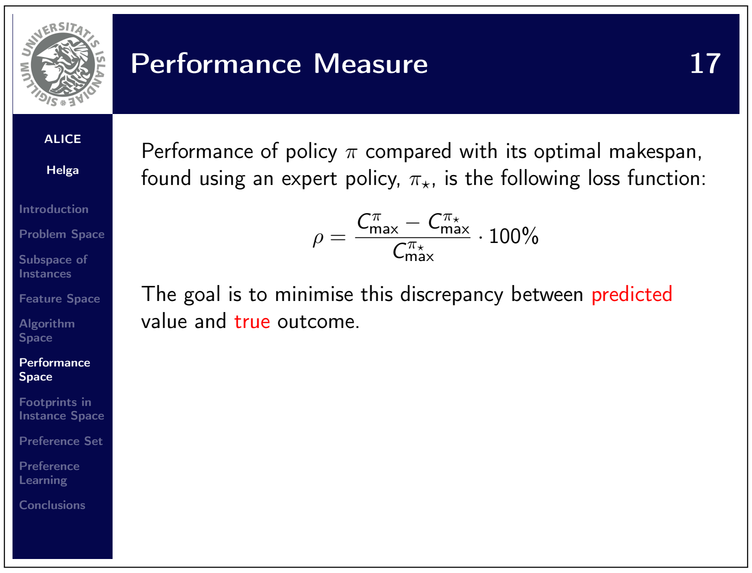
Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

- Moving on to Algorithm space (OVERVIEW) **forwards**
- There have been various methods used to solve job-shop – here we can see a tree representation of the nature of those techniques based on the work from Jain and Meeran in 1999
- I've highlighted the methods addressed in the thesis:
- Branch and bound is the optimisation algorithm I use for labelling my features
- single priority dispatching rule have already been described (ie SPT, LPT, LWR, MWR)
- and composite priority dispatching rule is a combination of those simpler rules
- I used evolutionary search to find appropriate weights for a CDR
- Roll-outs were used to design more comprehensive features with respect to the schedule's overall performance ($\{\phi_i\}_{i=17}^{24}$)
- **forwards** My contribution lies in the *new branch* called imitation learning that to ES learns weights for CDR – this will be address more thoroughly later



Performance Measure

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

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.

- Performance space (OVERVIEW) **forwards**
- is the loss function that describes the discrepancy between the predicted value using a certain policy compared to the true optimum outcome obtained using expert advice
- therefore the lower the mean for the deviation of optimality the better the policy

Deviation from Optimality

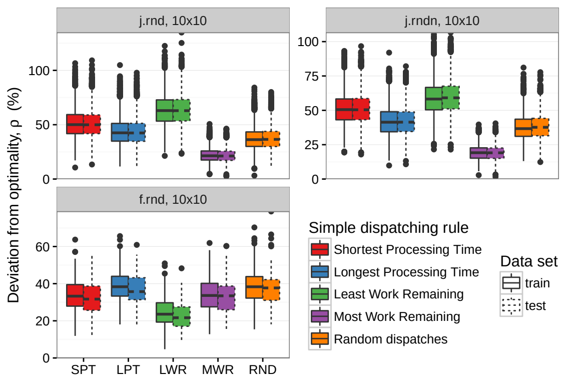
19

ρ

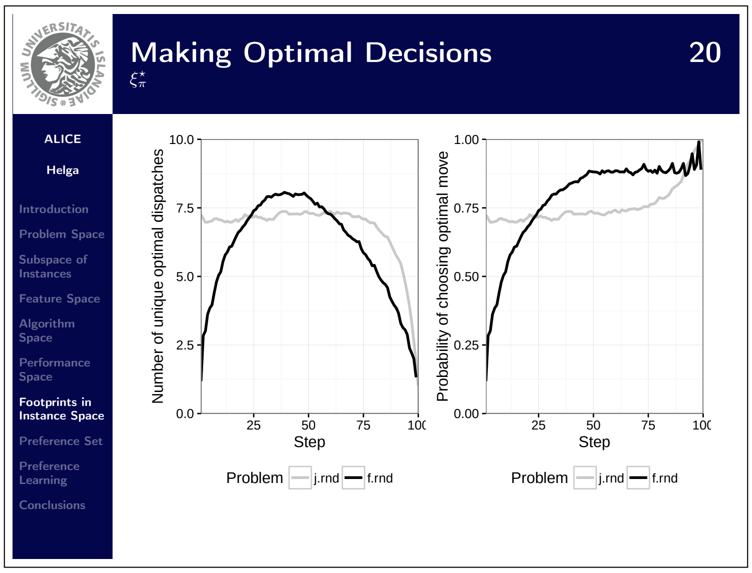
ALICE

Helga

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



- Regarding footprints in instance space I will start by inferring the interaction of the single priority dispatching rules from before to the previously discussed problem generators (OVERVIEW) **forwards**
- Here we can see a box-plot for deviation from optimality, ρ
- for job-shop it is best to use MWR of these rules as it will give me the lowest mean of deviation
- on the other hand its counterpart LWR is much better for flow-shop
- Here we can see that despite the same processing time distributions the change in machine ordering definition we see MWR is better than LWR for job-shop and vice versa for flow-shop
- **forwards** Result hold when going to a higher dimensionality
- So this is only based on results from the final dispatching step, but we would like to know when in the dispatching sequence this performance discrepancy starts to occur



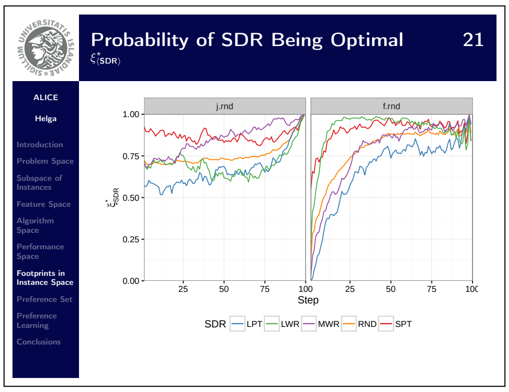
Making Optimal Decisions

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

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

- Here we are looking at a 10 $\times$ 10 problem spaces and inspecting given an optimal trajectory how many optimal moves are available at any given dispatching step. As we already saw, the number of total moves (optimal and suboptimal) diminish as k increases

- **forwards** so here we have the number of moves translated in terms of probability of choosing optimal move – basically what are the odds you would at random select the optimal move (given you do everything else correct)



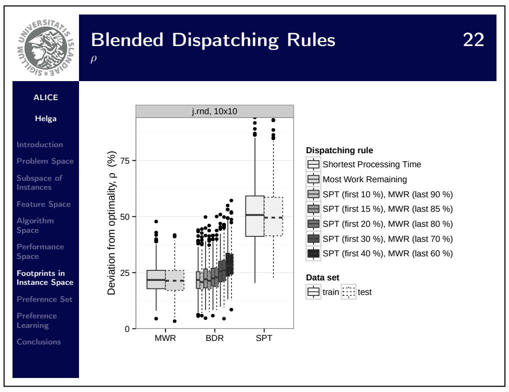
Probability of SDR Being Optimal 21

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

ALICE

Helga

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



Blended Dispatching Rules 22

ρ

ALICE

Helga

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

j.rnd, 10x10
Deviation from optimality, ρ (%)
0
25
50
75
MWR
BDR
SPT
Dispatching rule
Shortest Processing Time
Most Work Remaining
SPT (first 10 %), MWR (last 90 %)
SPT (first 15 %), MWR (last 85 %)
SPT (first 20 %), MWR (last 80 %)
SPT (first 30 %), MWR (last 70 %)
SPT (first 40 %), MWR (last 60 %)
Data set
train
test

- Now let's look at the odds of selecting the optimal move using a deterministic policy, such as a single priority dispatching rule

- Here we see for job-shop that SPT is the most likely policy to select the optimal move, or at least until step $k = 40$ when MWR becomes the better option

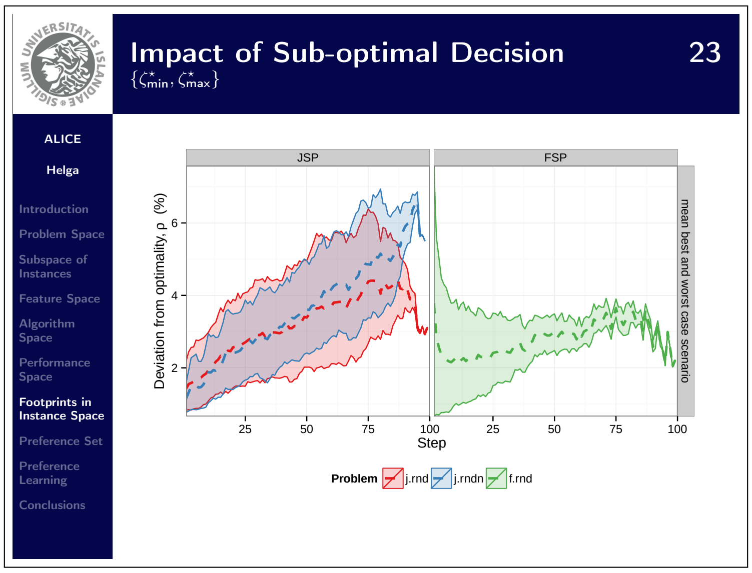
- Similar thing are going on for flow-shop except SPT starts of best until $k = 10$ where LWR surpasses in probability

- Based on this information, I want to create a dispatching rule that starts with SPT until step 40 and then switches to uses MWR from there on out

- **forwards** I call this a blended dispatching rule

- This switching of policies does improve the upon SPT immensely, however it still doesn't manage to achieve the same greatness MWR did on its own.

- Now why is that? **backwards** So instead of looking at the probability of selecting the optimal move, let's look at what happens if you make a sub-optimal move **forwards**



Impact of Sub-optimal Decision

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

23

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

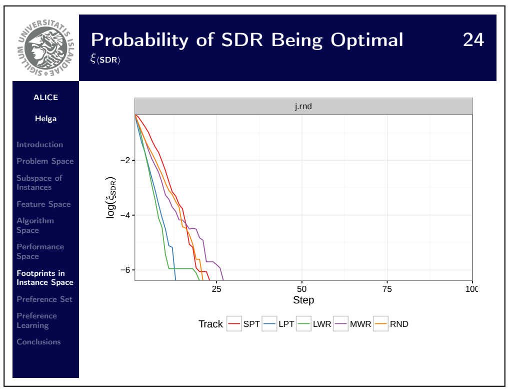
- Here see the impact in terms of deviation from optimality, ρ, (so not probability of optimality) if we make one misstep – everything else is done to optimality

- The lower bound is then the best case scenario and upper bound is the worst case scenario – with the dashed line as the mean impact

- Notice that for job-shop it isn't imperative to make a bad decision in the initial phases of dispatching, it's more important to focus on doing the right thing towards the end

- The opposite is true for flow-shop, where the absolute first moves are critical towards final outcome.

- This can be explained by the nature of flow-shop where having the same machine ordering constrict the search space in such a way it's hard to recover from previous mistakes. Whereas the randomness in machine ordering job-shop natively takes care of that

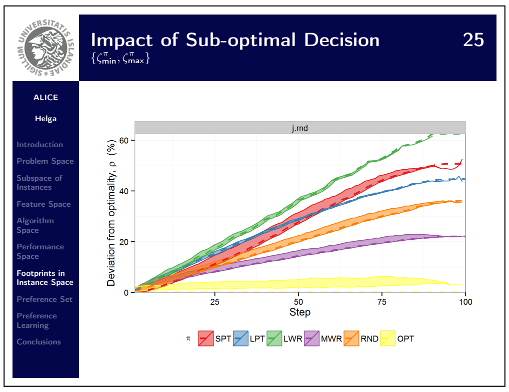


Probability of SDR Being Optimal

24

$\xi\langle\text{SDR}\rangle$

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



Impact of Sub-optimal Decision

25

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

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

- In sequential decision making, all future observations are dependent on previous operations, therefore since we are bound to make mistakes, analysis based on optimal solutions is no longer valid.

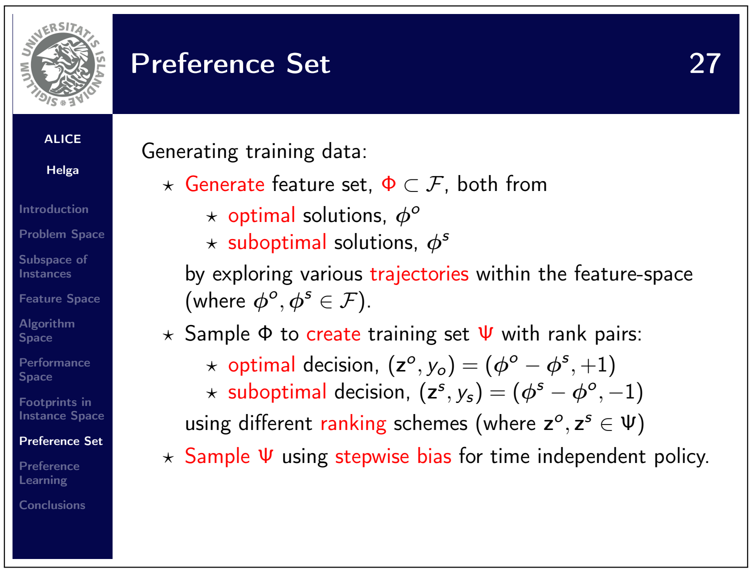
- Therefore let's update the measures based on the probability of choosing an optimal move for a given single priority dispatching rule during its dispatching process

- Here we see the compound effects of making suboptimal decisions

- Notice that for $k < 15$ SPT is above the MWR probability, and this is why switching here is good decision – although not statically significant from MWR (but as we remember the impact on ρ is the least during that phase)

- **forwards** Analogous to before (shown in yellow) here we see the impact of not following our proposed policy (whose ρ evolution is dashed line). So the band between that line and the lower bound is the possible improvement we could learn from that trajectory

- In fact a BDR switch at $k = 40$ never had a chance of improving SPT as the temporal makespan was already higher than the final makespan for MWR and had 60 operation left to dispatch


Preference Set

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

Generating training data:

- ⋆ 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.

- With the information obtained by our footprint analysis we can start generating training data that will serve as input in preference learning (OVERVIEW) **forwards**

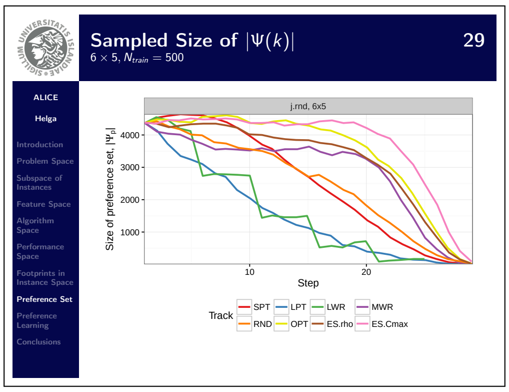
- The objective will be identifying good dispatching/features from worse ones

- **forwards** From the feature set we have both optimal and suboptimal solutions –this could be the state that results in Alice pouring tea vs. Dormouse singing twinkle-twinkle little star

- **forwards** By using branch and bound I have determined that Alice is the preferred option of the two, so I can create the preference pair $\{\phi_{\mathrm{A}} - \phi_{\mathrm{D}}, +1\}$ and $\{\phi_{\mathrm{D}} - \phi_{\mathrm{A}}, -1\}$. The combination of all optimal/suboptimal pairs creates the preference set Ψ. And there are various ranking schemes of how to select the minimum mount of pairs without loss of information

- Furthermore, since we will eventually like to create a policy that could work for the OR-Library (which can be of a much greater size then I am currently training on) I like to make a dispatching step independent model.

- Remember the amount of features encountered for a given trajectory



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

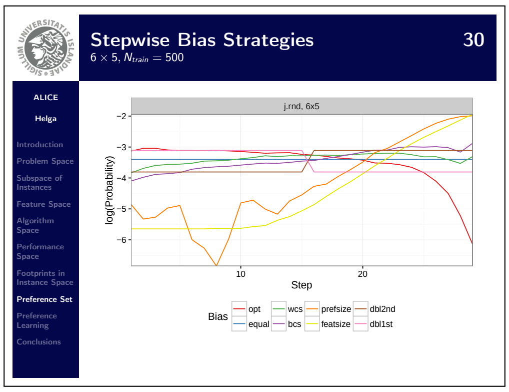
6×5, $N_{train} = 500$

29

ALICE

Helga

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



Stepwise Bias Strategies

6×5, $N_{train} = 500$

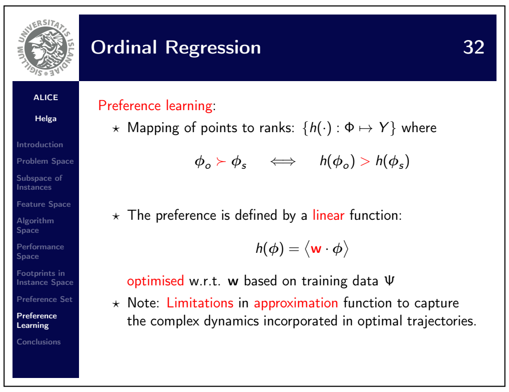
30

ALICE

Helga

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

- Here we see the resulting amount of preference pairs collected
- Bearing in mind for job-shop the impact w.r.t. ρ was the greater influenced during the latter stages of dispatching the emphasis in learning should be there also – however there are fewer samples to learn from, so we need to balance that out with a stepwise bias strategy
- **forwards** Here are some ad-hoc strategies for stepwise sampling based on the footprints analysis



Ordinal Regression

32

ALICE

Helga

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

Preference learning:

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

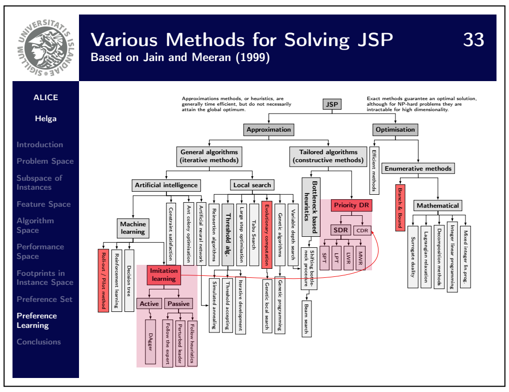
$$\phi_o \succ \phi_s \quad \Longleftrightarrow \quad h(\phi_o) > h(\phi_s)$$

- ⋆ 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)

33

ALICE

Helga

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

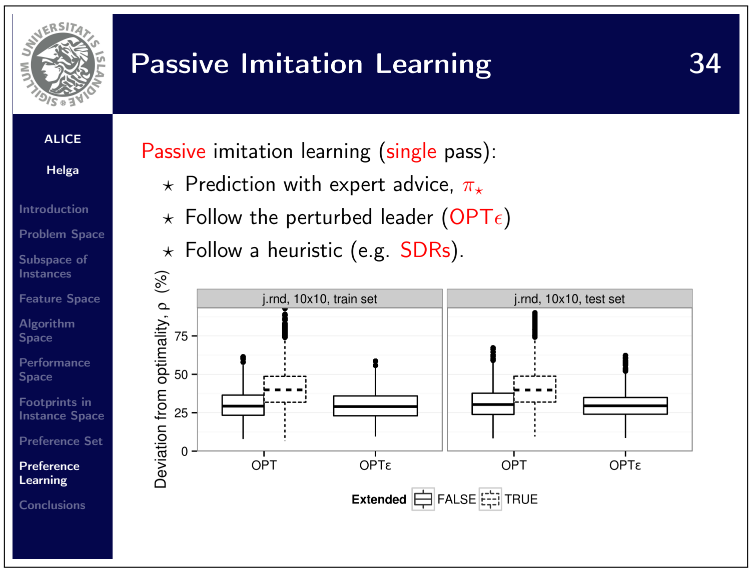
- Preference learning is a methodology in order to create a mapping for pairs to ranks, such that optimal feature is preferred to a suboptimal feature if and only if the function evaluation of a optimal feature is higher than the function evaluation of a suboptimal feature

- **forwards** In the thesis I only consider the mapping to be a linear function, so the learning algorithm is optimising with respect to weights $\mathbf{w}$ using the preference set

- the resulting model is effectively a composite priority dispatching rule

- **forwards** Note this is only a linear approximation used to capture the complex underlying dynamics – you could easily substitute the inner product to kernel methods or a more state-of-the-art algorithm

- **forwards** Revisiting the algorithm space again, the focus is now on the imitation learning branch which is divided into two sub-branches: Passive and Active



ALICE

Helga

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

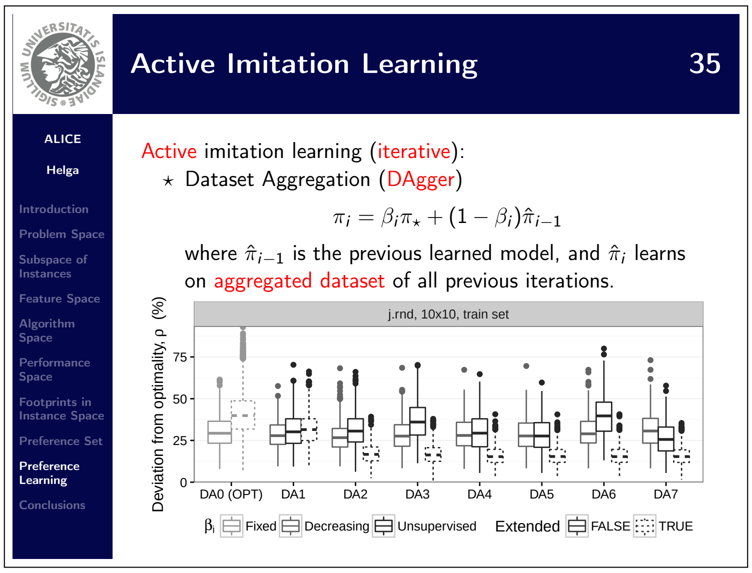
Passive Imitation Learning

34

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 only needs one collection of training set
- **forwards** I can start by learning on Ψ_p^{OPT} – that is prediction using expert advice
- **forwards** or I can perturb that trajectory with an ϵ likelihood
- **forwards** or I could training data from some deterministic policy
- In all of these cases I'm always labelling the using expert advice (gurobi), is only a matter of where does my feature set come from
- **forwards** Here are box-plots for top two strategies, and we see that by introducing suboptimal states spaces to the training set a lower ρ is achieved – implying learning for optimal trajectories is insufficient, at least in the case of a linear approximation function it is unattainable to learn those optimal state spaces
- Furthermore, using more problem instances (but always same sized $|\Psi_p^{\mathrm{OPT}}| = l_{\max}$, but more variety of problem instances) gives a worse result



Active Imitation Learning

35

ALICE

Helga

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

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.

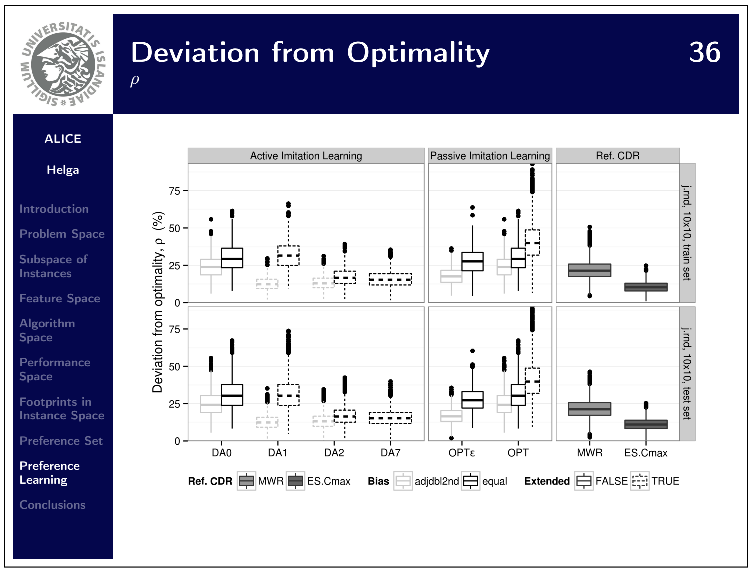
- Instead of using some heuristics (here a SDRs chosen ad-hoc) to guide your training data a more sophisticated approach would be to use active imitation leaning, that in an iterative fashion

- **forwards** I would create a model using Dataset Aggregation (DAgger) that essentially continues with the learned policy using expert advice (from prev. slide) – this would be iteration 0

- I would then collect the features encountered by my learned model and retrain – that way the learned model is more likely to encounter features that it has learned

- and there can be a combination of the learned model from the previous iteration and the expert, where β controls how you let go of the expert and solely rely on learned policies (fixed-supervised-unsupervised)

- **forwards** In the case of DAgger performance becomes (slightly) better with each iteration of DAgger, however for after the 2nd iteration you need to use new problem instances get more variety of new features (in contrast to PIL).



Deviation from Optimality

ρ

36

ALICE

Helga

Introduction

Problem Space

Subspace of Instances

Feature Space

Algorithm Space

Performance Space

Footprints in Instance Space

Preference Set

Preference Learning

Conclusions

- You don't have to have a sampling bias strategy to get a good result using DAgger as it's is automatically inbuilt in DAgger to sample the space. However, if you do use a good sampling bias convergence is quicker

- On the left we have two references (not imitation learning). MWR was the best SDR for job-shop – and AIL and PIL approaches were significantly better

- CMA-ES is a brute force approach where an optimisation can week in computational effort – where an iteration of DAgger could take less than a day – nevertheless the brute force approach still managed to find a better weights than AIL

Using ALICE Framework I

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.

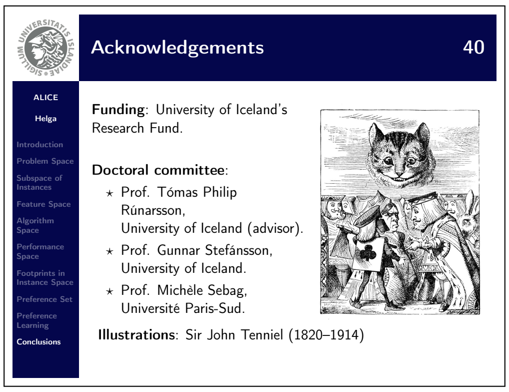
It is non intuitive how to go about collecting training data.

Using ALICE Framework II

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

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.



ALICE

Helga

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

Acknowledgements

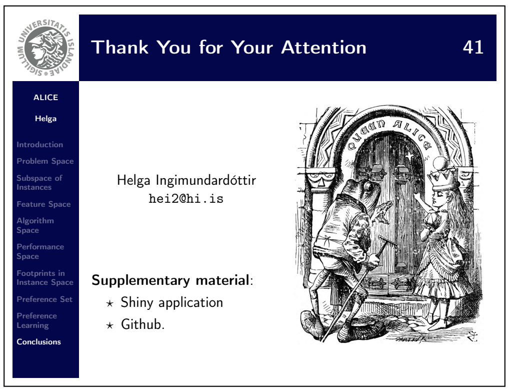
40

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

Thank You for Your Attention

41

Helga Ingimundardóttir
`hei2@hi.is`

Supplementary material:

- ⋆ Shiny application
- ⋆ Github.

QUEEN ALICE

- Invite guests to PhD party!