Policy Reuse in a General Learning Framework

Fernando Martínez-Plumed, Cèsar Ferri, José Hernández-Orallo, María José Ramírez-Quintana

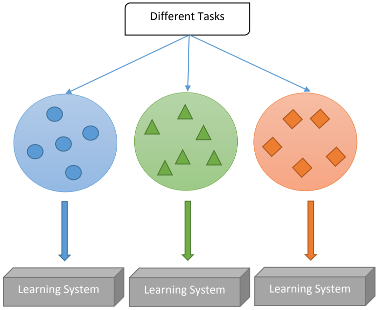
CAEPIA 2013

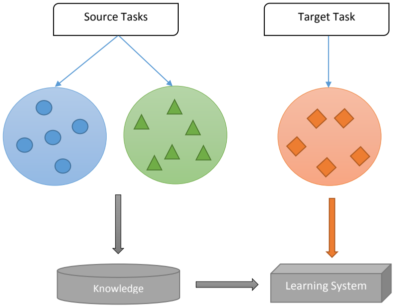
September 15, 2013

Table of contents

Introduction

- The reuse of knowledge which has been acquired in previous learning processes in order to improve or accelerate the learning of future tasks is an appealing idea.
- The knowledge transferred between tasks can be viewed as a bias in the learning of the target using the information learned in the source task





Introduction

- Research on transfer learning has attracted more and more attention since 1995 in different names and areas:
  - Learning to learn
  - Life-long learning
  - Knowledge-transfer
  - Inductive transfer
  - Multitask learning
  - Knowledge consolidation
  - Incremental/cumulative learning
  - Meta-learning
  - Reinforcement Learning.
  - Reframing

Introduction

- Research on transfer learning has attracted more and more attention since 1995 in different names and areas:
  - Learning to learn
  - Life-long learning
  - Knowledge-transfer
  - Inductive transfer
  - Multitask learning
  - Knowledge consolidation
  - Incremental/cumulative learning
  - Meta-learning
  - **Reinforcement Learning.**
  - Reframing

Introduction

Reinforcement Learning.
The knowledge is transfered in several ways ([Taylor and Stone, 2009] for a survey):

- Modifying the learning algorithm [Fernandez and Veloso, 2006, Mehta, 2005].
- Biasing the initial action-value function [J.Carroll, 2002].
- Mapping between actions and/or states [Liu and Stone, 2006, Price and Boutilier, 2003].

Introduction

- We present a **general rule-based learning setting** where **operators can be defined and customised for each kind of problem**.
  - The generalisation/especializazation operator to use depends on the structure of the data.
  - Adaptive and flexible rethinking of heuristics, with a model-based reinforcement learning approach.

gErl
GENERAL LEARNING SYSTEM

http://users.dsic.upv.es/~fmartinez/gerl.html

gErl

Flexible architecture [Lloyd, 2001] (1/2):

- Designing customised systems for applications with complex data.
- Operators can be modified and finetuned for each problem. **Different** to:
    - **Specialized systems** (Incremental models [Daumé III and Langford, 2009, Maes et al., 2009]) .
    - **Feature transformations** (kernels [Gärtner, 2005] or distances [Estruch et al., 2006]).
    - **Fixed operators** (Plotkin's lgg [Plotkin, 1970], Inverse Entailment [Muggleton, 1995], Inverse narrowing and CRG [Ferri et al., 2001]).

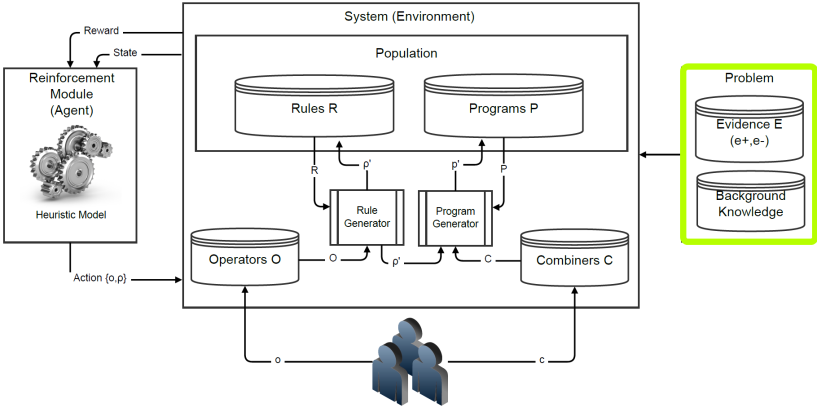
gErl

Flexible architecture [Lloyd, 2001] (2/2):

- Population of rules and programs evolved as in an evolutionary programming setting (LCS [Holmes et al., 2002]).
- Reinforcement Learning-based heuristic.
- Optimality criteria (MML/MDL) [Wallace and Dowe, 1999]).
- **Erlang** functional programming language [Virding et al., 1996].

This is a challenging proposal not sufficiently explored in machine learning.

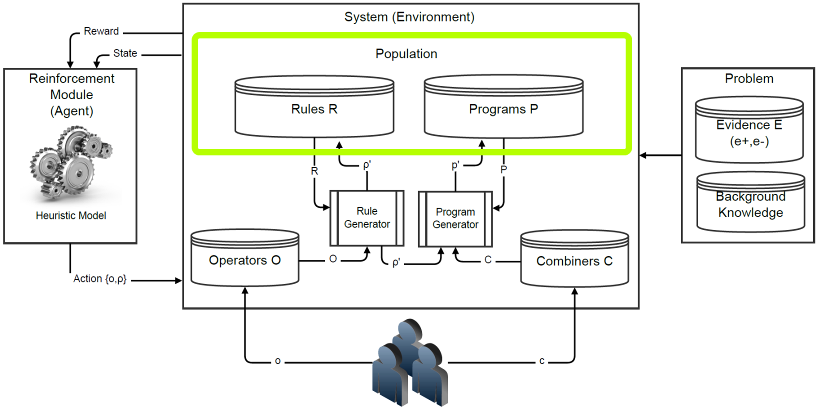
Architecture



A given problem (E^+ and E^-) and a (possible empty) BK.

$member([1,2,3],3) \rightarrow true$

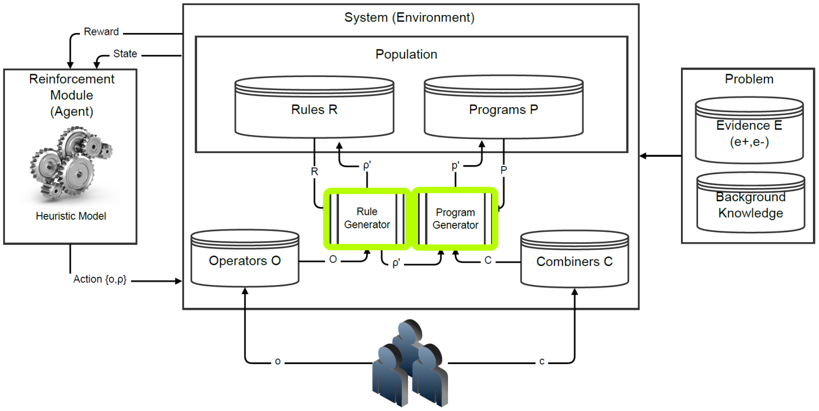
Architecture



Flexible architecture which works with populations of rules (unconditional / conditional equations) and programs written in **Erlang**.

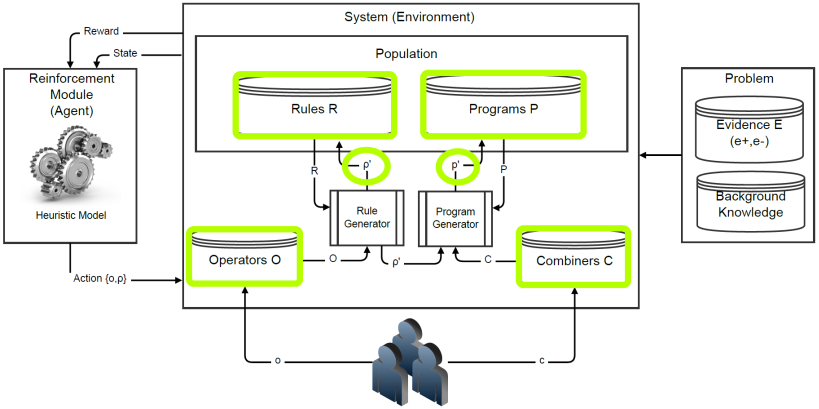
$$member([X|Y], Z) \text{ when true} \rightarrow member(Y, Z)$$

Architecture



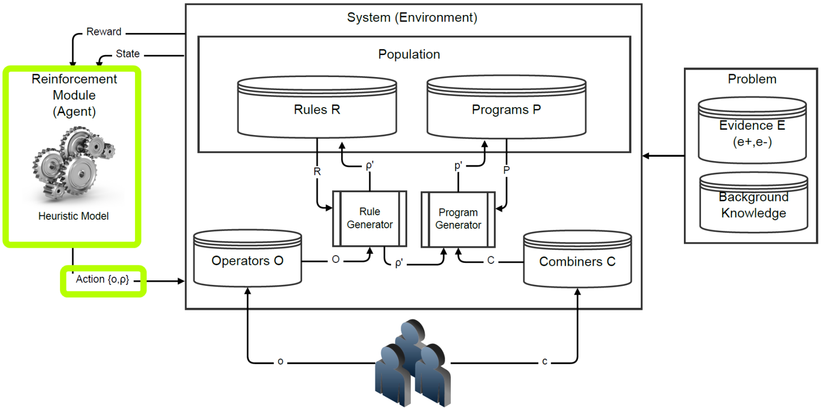
The population **evolves** as in an evolutionary programming setting.

Architecture



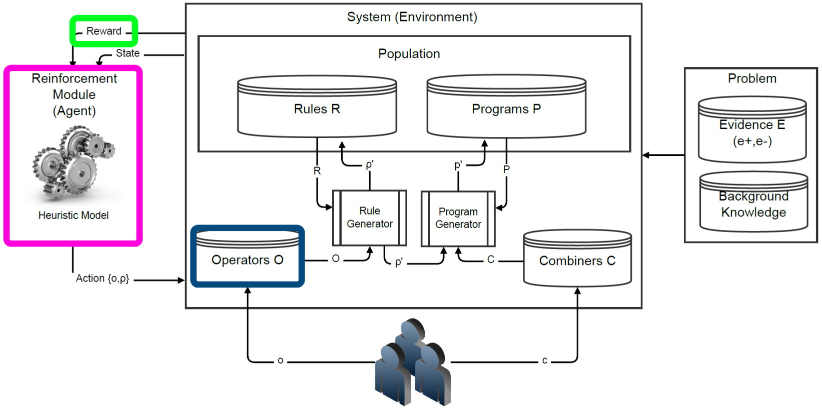
Operators are applied to rules for generating new rules and combined with existing or new programs.

Architecture



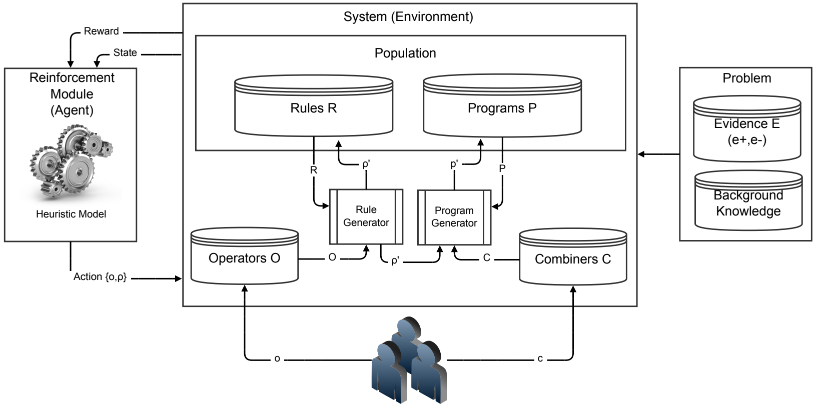
Reinforcement Learning-based heuristic to guide the learning.

Architecture



Appropriate **operators** + MML based **optimality criteria** + **Reinforcement Learning**-based heuristic.

Architecture



As a result, this architecture can be seen as a 'meta-learning system', that is, as a 'system for writing machine learning systems'.

Why **Erlang**?

Erlang/OTP [Virding et al., 1996] is a functional programming language developed by **Ericsson** and was designed from the ground up for writing scalable, fault-tolerant, distributed, non-stop and soft-realtime applications.

- **Free and open-source language** with a large community of developers behind.
- **Reflection and higher order**.
- **Unique representation language**, operators, examples, models and background knowledge are represented in the same language.

Operators over Rules and Programs

- The definition of customized **operators** is one of the key concepts of our proposal.
- In gErl, the set of rules R is transformed by applying a set of *operators* $O \subset \mathcal{O}$.
- Operators perform modifications over any of subparts of a rule in order to generalise or specialise it.
- gErl provides two **meta-operators** able to define well-known generalisation and specialisation operators in Machine Learning

RL-based heuristics

- **Heuristics must be overhauled** as decisions about the operator that must be used (over a rule) at each particular state of the learning process.
- A Reinforcement Learning (Rl) [Sutton and Barto, 1998] approach suits perfectly for our purposes.
- Our decision problem is a four-tuple $\langle \mathcal{S}, \mathcal{A}, \tau, \omega \rangle$ where:
    - S: state space ($s_t = \langle R, P \rangle$).
    - $\mathcal{A} : \mathcal{O} \times \mathcal{R}$ (a= $\langle o, \rho \rangle$).
    - $\tau : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$.
    - $\omega : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

MML/MDL-based Optimality

According to the MDL/MML philosophy, the optimality of a program *p* is defined as the weighted sum of two simpler heuristics, namely, a complexity-based heuristic (which measures the complexity of *p*) and a coverage heuristic (which measures how well *p* fits the evidence):

Cost

$$Cost(p) = \beta_1 \cdot MsgLen(p) + \beta_2 \cdot (MsgLen(e|p))$$

MML/MDL-based Optimality

According to the MDL/MML philosophy, the optimality of a program p is defined as the weighted sum of two simpler heuristics, namely, a complexity-based heuristic (which measures the complexity of p) and a coverage heuristic (which measures how well p fits the evidence):

Cost

$$\begin{aligned} Cost(p) = {} & \beta_1 \cdot MsgLen(p) + \\ & \beta_2 \cdot (MsgLen(\{e \in E^+ : p \not\models e\}) + MsgLen(\{e \in E^- : p \models e\})) \end{aligned}$$

RL-based heuristics

The probably infinite number of states and actions makes the application of classical RL algorithms not feasible:

- **States.** $\dot{s}_t = \langle \phi_1, \phi_2, \phi_3 \rangle$
  1. *Global optimality* (ϕ_1):
  2. *Average Size of Rules* (ϕ_2)
  3. *Average Size of programs* (ϕ_3)
- **Actions.** $\dot{a} = \langle o, \varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7, \varphi_8 \rangle$
  1. *Operator (o)*
  2. *Size (φ_1)*
  3. *Positive Coverage Rate (φ_2)*.
  4. *Negative Coverage Rate (φ_3)*.
  5. *NumVars (φ_4)*
  6. *NumCons (φ_5)*
  7. *NumFuncs (φ_6)*
  8. *NumStructs (φ_7)*
  9. *isRec (φ_8)*
- **Transitions.** Transitions are deterministic. A transition τ evolves the current sets of rules and programs by applying the operators selected (together with the rule) and the combiners.
- **Rewards.** The optimality criteria seen above is used to feed the rewards.

Modelling the state-value function: using a regression model

- We use a hybrid between value-function methods (which update a state-value matrix) and model-based methods (which learn models for τ and ω) [Sutton, 1998].
- Generalise the *state-value* function $Q(s, a)$ of the Q-learning [Watkins and Dayan, 1992] (which returns quality *values*, $q \in \mathbb{R}$) by a supervised model

$$Q_M : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

- gErl uses linear regression by default for generating Q_M, which is retrained periodically from Q.
- Q_M is used to obtain the best action $\dot{a}$ for the state $\dot{s}_t$ as follows:

$$a_t = \arg\max_{a \in \mathcal{A}} \{Q_M(\dot{s}_t, \dot{a})\}$$

Modelling the state-value function: using a regression model

state (s)			***action*** (a)									***q***
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	
161.32	17.92	1	1	17.92	0.11	0	0	4	2	0	0	1
...												
161.32	17.92	1	4	17.92	0.11	0	0	4	2	0	0	1
140.81	17.92	1	2	15.33	0.11	0	1	3	2	0	0	0.82
161.32	17.92	1	3	15.33	0.11	0	1	3	2	0	0	0.82
161.32	17.92	1	2	15.33	0.11	0	1	3	2	0	0	0.82
161.32	17.92	1	2	15.33	0.22	0	1	3	2	0	0	0.85
161.32	17.92	1	1	15.33	0.11	0.2	1	3	2	0	0	0.79

- Once the system has started, at each step, Q is updated using the following formula:

$$Q[s_t, a_t] \leftarrow \alpha \left[w_{t+1} + \gamma \max_{a_{t+1}} Q_M(s_{t+1}, a_{t+1}) \right] + (1-\alpha) Q[s_t, a_t] \tag{1}$$

Example: Playtennis

Id_{e^+}	e^+
1	$playtennis(overcast, hot, high, weak) \rightarrow yes$
2	$playtennis(rain, mild, high, weak) \rightarrow yes$
3	$playtennis(rain, cool, normal, weak) \rightarrow yes$
4	$playtennis(overcast, cool, normal, strong) \rightarrow yes$
5	$playtennis(sunny, cool, normal, weak) \rightarrow yes$
6	$playtennis(rain, mild, normal, weak) \rightarrow yes$
7	$playtennis(sunny, mild, normal, strong) \rightarrow yes$
8	$playtennis(overcast, mild, high, strong) \rightarrow yes$
9	$playtennis(overcast, hot, normal, weak) \rightarrow yes$

Table 1: Set of positive examples E^- (*Playtennis* problem)

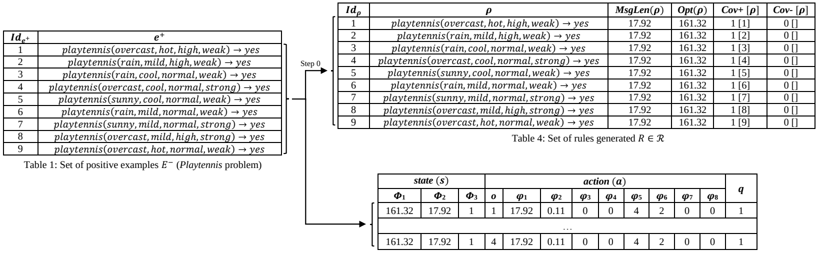
Id_{e^-}	e^-
1	$playtennis(sunny, hot, high, weak) \rightarrow yes$
2	$playtennis(sunny, hot, high, strong) \rightarrow yes$
3	$playtennis(rain, cool, normal, strong) \rightarrow yes$
4	$playtennis(sunny, mild, high, weak) \rightarrow yes$
5	$playtennis(rain, mild, high, strong) \rightarrow yes$

Table 2: Set of negative examples E^- (*Playtennis* problem)

Id_o	o
1	$replace\ (L_1, X_1)$
2	$replace\ (L_2, X_2)$
3	$replace\ (L_3, X_3)$
4	$replace\ (L_4, X_4)$

Table 3: Set of operators $O \in \mathcal{O}$

Example: Playtennis



Id_{e^+}	e^+
1	$playtennis(overcast, hot, high, weak) \rightarrow yes$
2	$playtennis(rain, mild, high, weak) \rightarrow yes$
3	$playtennis(rain, cool, normal, weak) \rightarrow yes$
4	$playtennis(overcast, cool, normal, strong) \rightarrow yes$
5	$playtennis(sunny, cool, normal, weak) \rightarrow yes$
6	$playtennis(rain, mild, normal, weak) \rightarrow yes$
7	$playtennis(sunny, mild, normal, strong) \rightarrow yes$
8	$playtennis(overcast, mild, high, strong) \rightarrow yes$
9	$playtennis(overcast, hot, normal, weak) \rightarrow yes$

Table 1: Set of positive examples E^- (*Playtennis* problem)

Id_ρ	ρ	$MsgLen(\rho)$	$Opt(\rho)$	Cov+ $[\rho]$	Cov- $[\rho]$
1	$playtennis(overcast, hot, high, weak) \rightarrow yes$	17.92	161.32	1 [1]	0 []
2	$playtennis(rain, mild, high, weak) \rightarrow yes$	17.92	161.32	1 [2]	0 []
3	$playtennis(rain, cool, normal, weak) \rightarrow yes$	17.92	161.32	1 [3]	0 []
4	$playtennis(overcast, cool, normal, strong) \rightarrow yes$	17.92	161.32	1 [4]	0 []
5	$playtennis(sunny, cool, normal, weak) \rightarrow yes$	17.92	161.32	1 [5]	0 []
6	$playtennis(rain, mild, normal, weak) \rightarrow yes$	17.92	161.32	1 [6]	0 []
7	$playtennis(sunny, mild, normal, strong) \rightarrow yes$	17.92	161.32	1 [7]	0 []
8	$playtennis(overcast, mild, high, strong) \rightarrow yes$	17.92	161.32	1 [8]	0 []
9	$playtennis(overcast, hot, normal, weak) \rightarrow yes$	17.92	161.32	1 [9]	0 []

Table 4: Set of rules generated $R \in \mathcal{R}$

state (s)			action (a)									q
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	
161.32	17.92	1	1	17.92	0.11	0	0	4	2	0	0	1
…												
161.32	17.92	1	4	17.92	0.11	0	0	4	2	0	0	1

Example: Playtennis

Id_ρ	ρ	$MsgLen(\rho)$	$Opt(\rho)$	$Cov+$ [ρ]	$Cov-$ [ρ]
1	$playtennis(overcast, hot, high, weak) \rightarrow yes$	17.92	161.32	1 [1]	0 []
2	$playtennis(rain, mild, high, weak) \rightarrow yes$	17.92	161.32	1 [2]	0 []
3	$playtennis(rain, cool, normal, weak) \rightarrow yes$	17.92	161.32	1 [3]	0 []
4	$playtennis(overcast, cool, normal, strong) \rightarrow yes$	17.92	161.32	1 [4]	0 []
5	$playtennis(sunny, cool, normal, weak) \rightarrow yes$	17.92	161.32	1 [5]	0 []
6	$playtennis(rain, mild, normal, weak) \rightarrow yes$	17.92	161.32	1 [6]	0 []
7	$playtennis(sunny, mild, normal, strong) \rightarrow yes$	17.92	161.32	1 [7]	0 []
8	$playtennis(overcast, mild, high, strong) \rightarrow yes$	17.92	161.32	1 [8]	0 []
9	$playtennis(overcast, hot, normal, weak) \rightarrow yes$	17.92	161.32	1 [9]	0 []
10	$playtennis(sunny, X_2, normal, weak) \rightarrow yes$	15.34	158.74	1 [5]	0 []

Table 4: Set of rules generated $R \in \mathcal{R}$

Id_o	o
1	$replace\ (L_1, X_1)$
2	$replace\ (L_2, X_2)$
3	$replace\ (L_3, X_3)$
4	$replace\ (L_4, X_4)$

Table 3: Set of operators $O \in \mathcal{O}$

$$a_{t=1} = \arg\max_{a \in \mathcal{A}} \{Q_M(\dot{s}_t, a)\} = \langle 2, 5 \rangle$$

state (s)			action (a)									q
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	
161.32	17.92	1	1	17.92	0.11	0	0	4	2	0	0	1
…												
161.32	17.92	1	4	17.92	0.11	0	0	4	2	0	0	1
140.81	**17.92**	**1**	**2**	**15.33**	**0.11**	**0**	**1**	**3**	**2**	**0**	**0**	**0.82**

Step 1

Table 5: Matrix Q

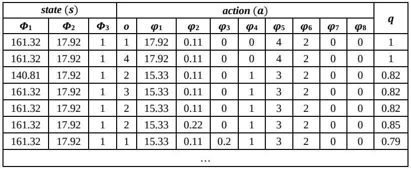
Example: Playtennis

Id_ρ	ρ	$MsgLen(\rho)$	$Opt(\rho)$	Cov+ [ρ]	Cov- [ρ]
1	$playtennis(overcast, hot, high, weak) \rightarrow yes$	17.92	161.32	1 [1]	0 []
2	$playtennis(rain, mild, high, weak) \rightarrow yes$	17.92	161.32	1 [2]	0 []
3	$playtennis(rain, cool, normal, weak) \rightarrow yes$	17.92	161.32	1 [3]	0 []
4	$playtennis(overcast, cool, normal, strong) \rightarrow yes$	17.92	161.32	1 [4]	0 []
5	$playtennis(sunny, cool, normal, weak) \rightarrow yes$	17.92	161.32	1 [5]	0 []
6	$playtennis(rain, mild, normal, weak) \rightarrow yes$	17.92	161.32	1 [6]	0 []
7	$playtennis(sunny, mild, normal, strong) \rightarrow yes$	17.92	161.32	1 [7]	0 []
8	$playtennis(overcast, mild, high, strong) \rightarrow yes$	17.92	161.32	1 [8]	0 []
9	$playtennis(overcast, hot, normal, weak) \rightarrow yes$	17.92	161.32	1 [9]	0 []
10	$playtennis(sunny, X_2, normal, weak) \rightarrow yes$	15.34	158.74	1 [5]	0 []
11	$playtennis(overcast, cool, X_3, strong) \rightarrow yes$	15.34	158.74	1 [4]	0 []
12	$playtennis(overcast, X_2, normal, weak) \rightarrow yes$	15.34	158.74	1 [9]	0 []
13	$playtennis(rain, X_2, normal, weak) \rightarrow yes$	15.34	140.81	2 [3,6]	0 []
14	$playtennis(X_1, hot, high, weak) \rightarrow yes$	15.34	176.66	1 [1]	1 [1]

Table 4: Set of rules generated $R \in \mathcal{R}$

state (s)			*action* (a)										q	
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8			
161.32	17.92	1	1	17.92	0.11	0	0	4	2	0	0	1		
…														
161.32	17.92	1	4	17.92	0.11	0	0	4	2	0	0	1		
140.81	17.92	1	2	15.33	0.11	0	1	3	2	0	0	0.82	Step 1	
161.32	17.92	1	3	15.33	0.11	0	1	3	2	0	0	0.82	Step 2	
161.32	17.92	1	2	15.33	0.11	0	1	3	2	0	0	0.82	Step 3	
161.32	17.92	1	2	15.33	0.22	0	1	3	2	0	0	0.85	Step 4	
161.32	17.92	1	1	15.33	0.11	0.2	1	3	2	0	0	0.79	Step 5	

Reusing Past Policies



state (s)			*action* (a)									q
Φ_1	Φ_2	Φ_3	o	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8	
161.32	17.92	1	1	17.92	0.11	0	0	4	2	0	0	1
161.32	17.92	1	4	17.92	0.11	0	0	4	2	0	0	1
140.81	17.92	1	2	15.33	0.11	0	1	3	2	0	0	0.82
161.32	17.92	1	3	15.33	0.11	0	1	3	2	0	0	0.82
161.32	17.92	1	2	15.33	0.11	0	1	3	2	0	0	0.82
161.32	17.92	1	2	15.33	0.22	0	1	3	2	0	0	0.85
161.32	17.92	1	1	15.33	0.11	0.2	1	3	2	0	0	0.79
...												

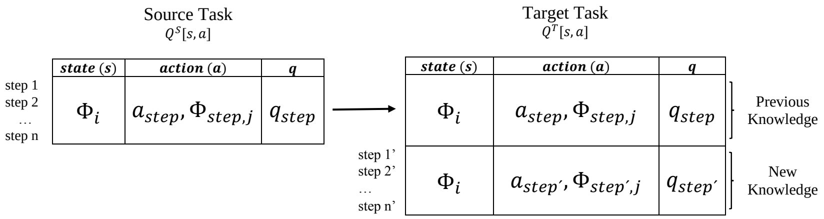
The abstract representation of states and actions (the ϕ and φ features) which allows the system does not start from the scratch and reuse the optimal information:

- Actions successfully applied to certain states (from the previous task) when it reaches a similar (with similar features) new state.

Due this abstract representation, how different are the source and target task does not matter.

Reusing Past Policies

- The table Q^S can be viewed as knowledge acquired during the learning process that can be transferred to a new situation.
- When gErl learns the new task, Q^S is used to train a new model Q^T_M[1].
- Q^S is used from the first learning step and it is afterwards updated with the new information acquired using the model Q^T_M.



Source Task

$Q^S[s,a]$

	state (s)	***action*** (a)	q
step 1, step 2, …, step n	Φ_i	$a_{step}, \Phi_{step,j}$	q_{step}

Target Task

$Q^T[s,a]$

	state (s)	***action*** (a)	q	
	Φ_i	$a_{step}, \Phi_{step,j}$	q_{step}	Previous Knowledge
step 1', step 2', …, step n'	Φ_i	$a_{step'}, \Phi_{step',j}$	$q_{step'}$	New Knowledge

[1]We don't transfer the Q^S_M model since it may not have been retrained with the last information added to the table Q^S (because of the periodicity of training).

An illustrative example of Transfer Knowledge

List processing problems as a structured prediction domain:

1. $d \rightarrow c$: replaces "d" by "c".
   ($trans([t, r, a, d, e]) \rightarrow [t, r, a, c, e]$)
2. $e \rightarrow ing$: replaces "e" by "ing" located at the last position of a list. ($trans([t, r, a, d, e]) \rightarrow [t, r, a, d, i, n, g]$)
3. $d \rightarrow pez$: replaces "d" by "pez" located at any position of a list. ($trans([t, r, a, d, e]) \rightarrow [t, r, a, p, e, z, e]$)
4. $Prefix_{over}$: adds the prefix "$over$".
   ($trans([t, r, a, d, e]) \rightarrow [o, v, e, r, t, r, a, d, e]$)
5. $Suffix_{mark}$: adds the suffix "$mark$".
   ($trans([t, r, a, d, e]) \rightarrow [t, r, a, d, e, m, a, r, k]$)

An illustrative example of Transfer Knowledge

Since we want to analyse the ability of the system to improve the learning process when reusing past policies:

1. we will solve each of the previous problems separately and,
2. then we will reuse the policy learnt solving one problem to solve the rest (including itself).

- The set of operators used consists of the user-defined operators and a small number of non-relevant operators (20).
- To make the experiments independent of the operator index, we will set up 5 random orders for them.
- Each problem has 20 positive instances e^+ and no negative ones.

An illustrative example of Transfer Knowledge

	$l \rightarrow c$	$e \rightarrow ing$	$d \rightarrow pez$	$Prefix_{over}$	$Suffix_{mark}$
Steps	108.68	76.76	74.24	61.28	62.28

Table: Results not reusing previous policies (average number of steps).

	Problem				
PCY from	$l \rightarrow c$	$e \rightarrow ing$	$d \rightarrow pez$	$Prefix_{over}$	$Suffix_{mark}$
$l \rightarrow c$	**65.68**	**58**	70, 64	**48.84**	**49.12**
$e \rightarrow ing$	**66.48**	**50.04**	**56.4**	**45.2**	**45.36**
$d \rightarrow pez$	**56.36**	**49.6**	**57.32**	52.24	**45.84**
$Prefix_{over}$	**58.8**	**48.96**	**60.6**	**43.8**	**46.88**
$Suffix_{mark}$	102, 72	**64.4**	67.32	56.16	**57.48**
Average	70.01	54.2	62.46	49.25	48.94

Table: Results reusing policies (average number of steps).

- From each problem we extract 5 random samples of ten positive instances in order to learn a policy from them with each of the five order of operators (5 problems $\times$ 5 samples $\times$ 5 operator orders $=$ 125 different experiments).

Conclusions and Future Work

- One of the problems of reusing knowledge from previous learning problems to new ones is the representation and abstraction of this knowledge.
- In this paper we have investigated how policy reuse can be useful (even in cases where the problems have no operators in common), simply because some abstract characteristics of two learning problems are similar at a more general level.

Conclusions and Future Work

There are many other things to explore in the context of gErl:

- Include features for the operators.
- Measure of similarity between problems (would help us to better understand when the system is able to detect these similarities).
- Apply the ideas in this paper to other kinds of systems (LCS, RL and other evolutionary techniques).
- Apply this ideas to other psychonometrics (IQ tests):
  - Odd-one-out problems.
  - Raven's matrices.
  - Thurstone Letter Series.

Thanks

THANKS

References I

[Daumé III and Langford, 2009] Daumé III, H. and Langford, J. (2009).
Search-based structured prediction.

[Estruch et al., 2006] Estruch, V., Ferri, C., Hernández-Orallo, J., and Ramírez-Quintana, M. J. (2006).
Similarity functions for structured data. an application to decision trees.
Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 10(29):109–121.

[Fernandez and Veloso, 2006] Fernandez, F. and Veloso, M. (2006).
Probabilistic policy reuse in a Reinforcement Learning agent.
In *AAMAS âĂŹ06*, pages 720–727. ACM Press.

References II

[Ferri et al., 2001] Ferri, C., Hernández-Orallo, J., and Ramírez-Quintana, M. (2001).
Incremental learning of functional logic programs.
In *FLOPS*, pages 233–247.

[Gärtner, 2005] Gärtner, T. (2005).
Kernels for Structured Data.
PhD thesis, Universitat Bonn.

[Holmes et al., 2002] Holmes, J. H., Lanzi, P., and Stolzmann, W. (2002).
Learning classifier systems: New models, successful applications.

Information Processing Letters.

References III

[J.Carroll, 2002] J.Carroll (2002).
Fixed vs Dynamic Sub-transfer in Reinforcement Learning.
In *ICMLA'02*. CSREA Press.

[Liu and Stone, 2006] Liu, Y. and Stone, P. (2006).
Value-function-based transfer for reinforcement learning using structure mapping.
AAAI, pages 415–20.

[Lloyd, 2001] Lloyd, J. W. (2001).
Knowledge representation, computation, and learning in higher-order logic.

[Maes et al., 2009] Maes, F., Denoyer, L., and Gallinari, P. (2009).

Structured prediction with reinforcement learning.
Machine Learning Journal, 77(2-3):271–301.

References IV

[Mehta, 2005] Mehta, N. (2005).
Transfer in variable-reward hierarchical reinforcement learning.
In *In Proc. of the Inductive Transfer workshop at NIPS*.

[Muggleton, 1995] Muggleton, S. (1995).
Inverse entailment and Progol.
New Generation Computing.

[Plotkin, 1970] Plotkin, G. (1970).
A note on inductive generalization.
Machine Intelligence, 5.

[Price and Boutilier, 2003] Price, B. and Boutilier, C. (2003).
Accelerating Reinforcement Learning through implicit imitation.
Journal of Artificial Intelligence Research, 19:2003.

References V

[Sutton, 1998] Sutton, R. (1998).
Reinforcement Learning: An Introduction.
MIT Press.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998).
Reinforcement learning: An introduction.
MIT press.

[Taylor and Stone, 2009] Taylor, M. and Stone, P. (2009).
Transfer learning for Reinforcement Learning domains: A survey.
Journal of Machine Learning Research, 10(1):1633–1685.

[Virding et al., 1996] Virding, R., Wikström, C., and Williams, M. (1996).
Concurrent programming in ERLANG (2nd ed.).
Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK.

References VI

[Wallace and Dowe, 1999] Wallace, C. S. and Dowe, D. L. (1999).
Minimum message length and kolmogorov complexity.
Computer Journal, 42:270–283.

[Watkins and Dayan, 1992] Watkins, C. and Dayan, P. (1992).
Q-learning.
Machine Learning, 8:279–292.