Accelerating Learning by reusing abstract policies in \texttt{gErl}

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

TAMIDA 2013

September 15, 2013
Table of contents

1. Introduction
2. The gErl System
3. Conclusions and Future Work
Introduction

- We present gErl [Martínez-Plumed et al., 2013], a general rule-based learning setting where operators can be defined and customised for each kind of problem.
  - The generalisation/specialization operator to use depends on the structure of the data.
  - Adaptive and flexible rethinking of heuristics, with a model-based reinforcement learning approach.
  - Intrinsic ability to transfer learning (as q-matrices) between related tasks in order to accelerate learning.

http://users.dsic.upv.es/~fmartinez/gerl.html
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]).
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, 1999a, Li and Vitnyi, 2008]).
- **Erlang** functional programming language [Virding et al., 1996].

This is a challenging proposal not sufficiently explored in machine learning.
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 high order**.
- **Unique representation language**, operators, examples, models and background knowledge are represented in the same language.
Rules and Programs

A given problem \((E^+ \text{ and } E^-)\) and a (possible empty) \(BK\).
Flexible architecture which works with populations of rules (unconditional / conditional equations) and programs written in **Erlang**.
Rules and Programs

- Two internal repositories containing **rules** and **programs**.
- Initially, the set of rules $R$ is populated with the positive evidence $E^+$ and the set of programs $P$ is populated defining unitary programs from the rules of $R$.
- Both repositories are updated at each step of the algorithm:
  1. The *Rule Generator* builds new rules ($\rho'$) and they are added to $R$.
  2. By applying the combiners, $\rho'$ is mixed with the programs in $P$ generating a new program $p'$, and it is added to $P$. 
Rules and Programs

Rule Notation

\[ \rho : l \ [\text{when } G] \rightarrow B, r \]

where

- \( l \) and \( r \) are the lhs and the rhs of \( \rho \) (respectively),
- \( G = \{ g_1, g_2, \ldots, g_m \mid m \geq 0 \} \) is a set of conditions or Boolean expressions called guards,
- \( B = b_1, \ldots, b_n \) is the body of \( \rho \), (sequence of equations).
# Rules and Programs

## Rule Notation

$$\rho : l \text{ [when } G] \rightarrow \text{Right}$$

- Let $\mathcal{P} = 2^\mathcal{R}$ be the space of all possible functional programs formed by sets of rules $\rho \in \mathcal{R}$.
- An example $e$ is a ground rule $l \rightarrow r$ being $r$ in normal form and both $l$ and $r$ are ground.
- $e$ is covered by a program $p$ ($p \models e$) if $l$ and $r$ have the same normal form with respect to $p$.
- $p \in \mathcal{P}$ is a solution of a learning problem defined by a $E^+$, a (possibly empty) $E^-$ and a (possibly empty) $K$ if:
  - $K \cup p \models E^+$ (posterior sufficiency or completeness).
  - $K \cup p \not\models E^-$ (posterior satisfiability or consistency).
Rules and Programs

Example

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

Example

member([X|Y],Z) [when true] → member(Y,Z)

member([X|Y],Z)

member(Y,Z)

true

member(Y,Z)

[XY]

X

[Y]

L1.1

L1.2

L

L1

L2

G

true

member(Y,Z)

Rt

Rt1

Rt1.1

Rt1.2

Z

G1

true

Y
The population evolves as in an evolutionary programming setting.
Operators over Rules and Programs

Operators are applied to rules for generating new rules and combined with existing or new programs.
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 \( O \subset O \).
- Then, an operator \( o \in O \) is a function \( o : R \rightarrow 2^R \), where \( O \subseteq O \) denote the set of operators chosen by the user for solving the problem.
- The operator’s aim is to perform modifications over any of subparts of a rule in order to generalise or specialise it.
For defining operators, the system is equipped with meta-level facilities called *meta-operators*. A meta-operator is formally defined as

\[ \mu O :: \text{Pos} \times \mathcal{T}(\Sigma, \mathcal{X}) \rightarrow \mathcal{O} \]

gErI provides the following two meta-operators able to define well-known generalisation and specialisation operators in Machine Learning:

1. **replace**(*Pos*, *Term*): creates an operator that replaces in a rule a term located in the position *Pos* by the term *Term*.
2. **insert**(*Pos*, *Term*): creates an operator that inserts a term *Term* in the position *Pos* of a rule.
3. **delete**(*Pos*): creates an operator that deletes a term located in the position *Pos* of a rule.
Reinforcement Learning-based heuristic to guide the 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 \( \langle S, \mathcal{A}, \tau, \omega \rangle \) where:
  - \( S \): state space \( (\sigma_t = \langle R, P \rangle) \).
  - \( \mathcal{A} : \mathcal{O} \times \mathcal{R} \) \( (a = \langle o, \rho \rangle) \).
  - \( \tau : S \times \mathcal{A} \rightarrow S \).
  - \( \omega : S \times \mathcal{A} \rightarrow \mathbb{R} \).
MML/MDL-based Optimality

According to the MML/MDL [Wallace and Dowe, 1999b] 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):

\[
\text{Cost}(p) = \beta_1 \cdot \text{MsgLen}(p) + \beta_2 \cdot (\text{MsgLen}(e|p))
\]
MML/MDL-based Optimality

According to the MML/MDL [Wallace and Dowe, 1999b] 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):

\[
\text{Cost}(p) = \beta_1 \cdot \text{MsgLen}(p) + \\
\beta_2 \cdot (\text{MsgLen}(\{ e \in E^+ : p \not\models e \}) + \text{MsgLen}(\{ e \in E^- : p \models e \}))
\]
The probably infinite number of states and actions makes the application of classical RL algorithms not feasible:

- **States.** $s_t = \langle \phi_1, \phi_2, \phi_3 \rangle$
  - Global optimality ($\phi_1$):
  - Average Size of Rules ($\phi_2$)
  - Average Size of programs ($\phi_3$)

- **Actions.** $a = \langle o, \varphi_1, \varphi_2, \varphi_3, \varphi_4, \varphi_5, \varphi_6, \varphi_7, \varphi_8 \rangle$
  - Operator ($o$)
  - Size ($\varphi_1$)
  - Positive Coverage Rate ($\varphi_2$).
  - Negative Coverage Rate ($\varphi_3$).
  - NumVars ($\varphi_4$)
  - NumCons ($\varphi_5$)
  - NumFuncs ($\varphi_6$)
  - NumStructs ($\varphi_7$)
  - isRec ($\varphi_8$)

- **Transitions.** Transitions are deterministic. A transition $\tau$ 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. In particular, we use the result returned by equation (??) as reward.
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 \( \tau \) and \( \omega \)) [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

\[
Q_M : S \times A \rightarrow \mathbb{R}
\]

- \texttt{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 \( \hat{a} \) for the state \( \hat{s}_t \) as follows:

\[
a_t = \arg \max_{a \in A} \{ Q_M(\hat{s}_t, \hat{a}) \}
\]
Modelling the state-value function: using a regression model

<table>
<thead>
<tr>
<th>state (s)</th>
<th>action (a)</th>
<th>q</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi_1$</td>
<td>$\Phi_2$</td>
<td>$\phi$</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>140.81</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>$Id_{e^+}$</th>
<th>$e^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>playtennis(overcast, hot, high, weak) → yes</td>
</tr>
<tr>
<td>2</td>
<td>playtennis(rain, mild, high, weak) → yes</td>
</tr>
<tr>
<td>3</td>
<td>playtennis(rain, cool, normal, weak) → yes</td>
</tr>
<tr>
<td>4</td>
<td>playtennis(overcast, cool, normal, strong) → yes</td>
</tr>
<tr>
<td>5</td>
<td>playtennis(sunny, cool, normal, weak) → yes</td>
</tr>
<tr>
<td>6</td>
<td>playtennis(rain, mild, normal, weak) → yes</td>
</tr>
<tr>
<td>7</td>
<td>playtennis(sunny, mild, normal, strong) → yes</td>
</tr>
<tr>
<td>8</td>
<td>playtennis(overcast, mild, high, strong) → yes</td>
</tr>
<tr>
<td>9</td>
<td>playtennis(overcast, hot, normal, weak) → yes</td>
</tr>
</tbody>
</table>

Table 1: Set of positive examples $E^+$ (Playtennis problem)

<table>
<thead>
<tr>
<th>$Id_{e^-}$</th>
<th>$e^-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>playtennis(sunny, hot, high, weak) → yes</td>
</tr>
<tr>
<td>2</td>
<td>playtennis(sunny, hot, high, strong) → yes</td>
</tr>
<tr>
<td>3</td>
<td>playtennis(rain, cool, normal, strong) → yes</td>
</tr>
<tr>
<td>4</td>
<td>playtennis(sunny, mild, high, weak) → yes</td>
</tr>
<tr>
<td>5</td>
<td>playtennis(rain, mild, high, strong) → yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$Id_o$</th>
<th>$o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>replace ($L_1, X_1$)</td>
</tr>
<tr>
<td>2</td>
<td>replace ($L_2, X_2$)</td>
</tr>
<tr>
<td>3</td>
<td>replace ($L_3, X_3$)</td>
</tr>
<tr>
<td>4</td>
<td>replace ($L_4, X_4$)</td>
</tr>
</tbody>
</table>

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

Table 1: Set of positive examples $E^+$ (Playtennis problem)

<table>
<thead>
<tr>
<th>$Id_e$</th>
<th>$e^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>playtennis(overcast, hot, high, weak) → yes</td>
</tr>
<tr>
<td>2</td>
<td>playtennis(rain, mild, high, weak) → yes</td>
</tr>
<tr>
<td>3</td>
<td>playtennis(rain, cool, normal, weak) → yes</td>
</tr>
<tr>
<td>4</td>
<td>playtennis(overcast, cool, normal, strong) → yes</td>
</tr>
<tr>
<td>5</td>
<td>playtennis(sunny, cool, normal, weak) → yes</td>
</tr>
<tr>
<td>6</td>
<td>playtennis(rain, mild, normal, weak) → yes</td>
</tr>
<tr>
<td>7</td>
<td>playtennis(sunny, mild, normal, strong) → yes</td>
</tr>
<tr>
<td>8</td>
<td>playtennis(overcast, mild, high, strong) → yes</td>
</tr>
<tr>
<td>9</td>
<td>playtennis(overcast, hot, normal, weak) → yes</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$Id_\rho$</th>
<th>$p$</th>
<th>$\text{MsgLen}(\rho)$</th>
<th>$\text{Opt}(\rho)$</th>
<th>$\text{Cov}^+ [\rho]$</th>
<th>$\text{Cov}^- [\rho]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>playtennis(overcast, hot, high, weak) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[1]</td>
<td>0[]</td>
</tr>
<tr>
<td>2</td>
<td>playtennis(rain, mild, high, weak) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[2]</td>
<td>0[]</td>
</tr>
<tr>
<td>3</td>
<td>playtennis(rain, cool, normal, weak) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[3]</td>
<td>0[]</td>
</tr>
<tr>
<td>4</td>
<td>playtennis(overcast, cool, normal, strong) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[4]</td>
<td>0[]</td>
</tr>
<tr>
<td>5</td>
<td>playtennis(sunny, cool, normal, weak) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[5]</td>
<td>0[]</td>
</tr>
<tr>
<td>6</td>
<td>playtennis(rain, mild, normal, weak) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[6]</td>
<td>0[]</td>
</tr>
<tr>
<td>7</td>
<td>playtennis(sunny, mild, normal, strong) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[7]</td>
<td>0[]</td>
</tr>
<tr>
<td>8</td>
<td>playtennis(overcast, mild, high, strong) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[8]</td>
<td>0[]</td>
</tr>
<tr>
<td>9</td>
<td>playtennis(overcast, hot, normal, weak) → yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1[9]</td>
<td>0[]</td>
</tr>
</tbody>
</table>
### Example: Playtennis

<table>
<thead>
<tr>
<th>$Id_{\rho}$</th>
<th>$\rho$</th>
<th>$\text{MsgLen}(\rho)$</th>
<th>$\text{Opt}(\rho)$</th>
<th>$\text{Cov}^+ [\rho]$</th>
<th>$\text{Cov}^- [\rho]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>playtennis(overcast, hot, high, weak) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [1]</td>
<td>0 []</td>
</tr>
<tr>
<td>2</td>
<td>playtennis(rain, mild, high, weak) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [2]</td>
<td>0 []</td>
</tr>
<tr>
<td>3</td>
<td>playtennis(rain, cool, normal, weak) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [3]</td>
<td>0 []</td>
</tr>
<tr>
<td>4</td>
<td>playtennis(overcast, cool, normal, strong) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [4]</td>
<td>0 []</td>
</tr>
<tr>
<td>5</td>
<td>playtennis(sunny, cool, normal, weak) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [5]</td>
<td>0 []</td>
</tr>
<tr>
<td>6</td>
<td>playtennis(rain, mild, normal, weak) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [6]</td>
<td>0 []</td>
</tr>
<tr>
<td>7</td>
<td>playtennis(sunny, mild, normal, strong) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [7]</td>
<td>0 []</td>
</tr>
<tr>
<td>8</td>
<td>playtennis(overcast, mild, high, strong) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [8]</td>
<td>0 []</td>
</tr>
<tr>
<td>9</td>
<td>playtennis(overcast, hot, normal, weak) $\rightarrow$ yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [9]</td>
<td>0 []</td>
</tr>
<tr>
<td>10</td>
<td>playtennis(sunny, $X_2$, normal, weak) $\rightarrow$ yes</td>
<td>15.34</td>
<td>158.74</td>
<td>1 [5]</td>
<td>0 []</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$Id_{o}$</th>
<th>$o$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>replace ($L_1, X_1$)</td>
</tr>
<tr>
<td>2</td>
<td>replace ($L_2, X_2$)</td>
</tr>
<tr>
<td>3</td>
<td>replace ($L_3, X_3$)</td>
</tr>
<tr>
<td>4</td>
<td>replace ($L_4, X_4$)</td>
</tr>
</tbody>
</table>

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

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

### Table 5: Matrix $Q$

<table>
<thead>
<tr>
<th>state ($s$)</th>
<th>action ($a$)</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Phi_1$</td>
<td>$\Phi_2$</td>
<td>$\Phi_3$</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>140.81</td>
<td>17.92</td>
<td>1</td>
</tr>
</tbody>
</table>
### Example: Playtennis

<table>
<thead>
<tr>
<th>Id_(\rho)</th>
<th>(\rho)</th>
<th>MsgLen((\rho))</th>
<th>Opt((\rho))</th>
<th>Cov+ [(\rho)]</th>
<th>Cov- [(\rho)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>playtennis(overcast, hot, high, weak) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [1]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>2</td>
<td>playtennis(rain, mild, high, weak) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [2]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>3</td>
<td>playtennis(rain, cool, normal, weak) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [3]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>4</td>
<td>playtennis(overcast, cool, normal, strong) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [4]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>5</td>
<td>playtennis(sunny, cool, normal, weak) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [5]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>6</td>
<td>playtennis(rain, mild, normal, weak) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [6]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>7</td>
<td>playtennis(sunny, mild, normal, strong) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [7]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>8</td>
<td>playtennis(overcast, mild, high, strong) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [8]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>9</td>
<td>playtennis(overcast, hot, normal, weak) (\rightarrow) yes</td>
<td>17.92</td>
<td>161.32</td>
<td>1 [9]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>10</td>
<td>playtennis(sunny, (X_2), normal, weak) (\rightarrow) yes</td>
<td>15.34</td>
<td>158.74</td>
<td>1 [5]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>11</td>
<td>playtennis(overcast, cool, (X_3), strong) (\rightarrow) yes</td>
<td>15.34</td>
<td>158.74</td>
<td>1 [4]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>12</td>
<td>playtennis(overcast, (X_2), normal, weak) (\rightarrow) yes</td>
<td>15.34</td>
<td>158.74</td>
<td>1 [9]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>13</td>
<td>playtennis(rain, (X_2), normal, weak) (\rightarrow) yes</td>
<td>15.34</td>
<td>140.81</td>
<td>2 [3,6]</td>
<td>0 [ ]</td>
</tr>
<tr>
<td>14</td>
<td>playtennis((X_1), hot, high, weak) (\rightarrow) yes</td>
<td>15.34</td>
<td>176.66</td>
<td>1 [1]</td>
<td>1 [1]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>state ((s))</th>
<th>action ((a))</th>
<th>(q)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\Phi_1)</td>
<td>(\Phi_2)</td>
<td>(\Phi_3)</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>140.81</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
<tr>
<td>161.32</td>
<td>17.92</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5: Matrix Q
Accelerating learning by reusing abstract policies in gErl

- While previous transfer work has mainly focused on reducing training time on closely related tasks by transferring from a simple to complex task in a single domain, we show a (potentially) more powerful way of simplifying this transfer task formulating it as an abstraction of states and actions.
- We show how a problem may be broken down in a series of smaller tasks in order to accelerate the learning.
Simple Grammar Problem

A simplified version of the English language phrase grammar problem which is used in our study:

- The problem consists of 40 positive examples $E^+$ and 63 negative examples $E^-$. 
- It is necessary to use a large background knowledge $B$ in order to deal with every kind of word. 
- The hypothesis $\mathcal{H}$ is composed by 5 general rules which cover all the positive evidence (each $h_i$ ($1 \leq i \leq 5$) covers the same number of positive examples).
Simple Grammar Problem

A simplified version of the English language phrase grammar problem which is used in our study:

Positive Examples $E^+$:
- $e_1: s([an, unknown, alien, hits, the, house])$.
- $e_2: s([a, small, boy, walks, a, dog])$.
- $e_3: s([a, dog, walks, into, the, house])$.
  ...

Negative Examples $E^-$:
- $e_4: \neg s([dog, hits, a, boy])$.
- $e_5: \neg s([the, house, a, boy])$.
- $e_6: \neg s([dog, walks, j]. hits])$.
  ...

Hypothesis $H$:
- $h_1: s([S_1,S_2,S_3,S_4,S_5]) \rightarrow np(S_1,S_2), vp(S_3), np(S_4,S_5)$.
- $h_2: s([S_1,S_2,S_3,S_4,S_5,S_6]) \rightarrow np(S_1,S_2), vp(S_3,S_4), np(S_5,S_6)$.
- $h_3: s([S_1,S_2,S_3,S_4,S_5,S_6]) \rightarrow np(S_1,S_2), vp(S_3), np(S_4,S_5,S_6)$.
- $h_4: s([S_1,S_2,S_3,S_4,S_5,S_6,S_7]) \rightarrow np(S_1,S_2,S_3), vp(S_4,S_5), np(S_6,S_7)$.
- $h_5: s(S_1,S_2,S_3,S_4,S_5,S_6) \rightarrow np(S_1,S_2,S_3), vp(S_4), np(S_5,S_6)$.

Background Knowledge $B$:
- $b_1: np(S_1,S_2) \rightarrow det(S_1), noun(S_2)$.
- $b_2: np(S_1,S_2,S_3) \rightarrow det(S_1), adj(S_2), noun(S_3)$.
- $b_3: vp(S_1) \rightarrow verb(S_1)$.
- $b_4: vp(S_1,S_2) \rightarrow verb(S_1), prep(S_2)$.
- $b_5: det(a)$. $b_6: det(an)$. $b_7: det(the)$.
- $b_8: noun(dog)$. $b_9: noun(boy)$.
- $b_{10}: noun(house)$. $b_{11}: noun(alien)$.
- $b_{12}: verb(dog)$. $b_{13}: verb(walks)$.
- $b_{14}: adj(small)$. $b_{15}: adj(unknown)$.
- $b_{16}: prep(\mathit{into})$.
  ...
Simple Grammar Problem

Operators:

- \( \text{replace}(L_{1.i}, Var_i) \) where \( i \in 1..7 \)
- \( \text{insert}(R_t, np(L_{1.i}, L_{1.i+1})) \) where \( i \in 1..6 \)
- \( \text{insert}(R_t, np(L_{1.i}, L_{1.i+1}, L_{1.i+2})) \) where \( i \in 1..5 \)
- \( \text{insert}(R_t, vp(L_{1.i}) \) where \( i \in 1..7 \)
- \( \text{insert}(R_t, vp(L_{1.i}L_{1.i+1})) \) where \( i \in 1..6 \)
To analyse the ability of the system to improve the learning when reusing past policies, we perform two experiments:

1. Taking 5 random samples of 20 examples from the original problem but keeping the same distribution of the original problem, and learning from them a policy. Each policy learned is used to learn with the rest of 20 positive instances not selected previously in each sample.

2. Taking 5 random samples of 20 examples from the original problem, but now without taking into account the distribution of examples and learning from them a policy.

We use these policies learned to bias the learning of the rest of 20 positive instances not selected in each sample.
Simple Grammar Problem

![Graph 1](image1.png)

- **Steps vs. Hypothesis (Solution)**
  - (a) Hypothesis (Solution)
    - Steps: 40, 20, 20 + M_{past}
    - h1, h2, h3, h4, h5

![Graph 2](image2.png)

- **Steps vs. Hypothesis (Solution)**
  - (b) #Positive Examples used
    - Steps: 40, 20, 20 + M_{past}
    - h1, h2, h3, h4, h5
Conclusions and Future Work

- In this paper we have presented the main features of the gErl system, specially its reusing policy ability which is based on a representation of knowledge as a tuple of abstract characteristics.
- We have used our system to solve a simple problem showing that the total learning time is reduced when policy reuse is applied.
Accelerating learning by reusing abstract policies in 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.
Search-based structured prediction.

Similarity functions for structured data. an application to decision trees. 

Incremental learning of functional logic programs. 


References IV

Inverse entailment and Progol.

A note on inductive generalization.
Machine Intelligence, 5.

Reinforcement Learning: An Introduction.
MIT Press.

Reinforcement learning: An introduction.
MIT press.
References V


