A Direct Policy-Search Algorithm for Relational Reinforcement Learning

Samuel Sarjant
Bernhard Pfahringer, Kurt Driessens, Tony Smith

Department of Computer Science
University of Waikato, New Zealand

29th August, 2013

Introduction

- Relational Reinforcement Learning (RRL) is a representational generalisation of Reinforcement Learning.
- Uses *policy* to select *actions* when provided state *observations* to maximise *reward*.
- Value-based RRL affected by number of states and may require predefined abstractions or expert guidance.
- Direct policy-search only needs to encode ideal action, hypothesis-driven learning.
- We use the Cross-Entropy Method (CEM) to learn policies.

Introduction

- Relational Reinforcement Learning (RRL) is a representational generalisation of Reinforcement Learning.
- Uses *policy* to select *actions* when provided state *observations* to maximise *reward*.
- Value-based RRL affected by number of states and may require predefined abstractions or expert guidance.
- Direct policy-search only needs to encode ideal action, hypothesis-driven learning.
- We use the Cross-Entropy Method (CEM) to learn policies.

Introduction

- Relational Reinforcement Learning (RRL) is a representational generalisation of Reinforcement Learning.
- Uses *policy* to select *actions* when provided state *observations* to maximise *reward*.
- Value-based RRL affected by number of states and may require predefined abstractions or expert guidance.
- Direct policy-search only needs to encode ideal action, hypothesis-driven learning.
- We use the Cross-Entropy Method (CEM) to learn policies.

Introduction

- Relational Reinforcement Learning (RRL) is a representational generalisation of Reinforcement Learning.
- Uses *policy* to select *actions* when provided state *observations* to maximise *reward*.
- Value-based RRL affected by number of states and may require predefined abstractions or expert guidance.
- Direct policy-search only needs to encode ideal action, hypothesis-driven learning.
- We use the Cross-Entropy Method (CEM) to learn policies.

Introduction

- Relational Reinforcement Learning (RRL) is a representational generalisation of Reinforcement Learning.
- Uses *policy* to select *actions* when provided state *observations* to maximise *reward*.
- Value-based RRL affected by number of states and may require predefined abstractions or expert guidance.
- Direct policy-search only needs to encode ideal action, hypothesis-driven learning.
- We use the Cross-Entropy Method (CEM) to learn policies.

Cross-Entropy Method

- In broad terms, the Cross-Entropy Method consists of these phases:
  - Generate samples $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$ from a generator and evaluate them $f(\mathbf{x}^{(1)}), \ldots, f(\mathbf{x}^{(n)})$.
  - Alter the generator such that it is more likely to produce the highest valued samples again.
  - Repeat until converged.
- No worse than random, then iterative improvement.
- Multiple generators produce combinatorial samples.

CERRLA

- The Cross-Entropy Relational Reinforcement Learning Agent (CERRLA) applies the CEM to RRL.
- The CEM *generator* consists of multiple distributions of condition-action rules.
- A *sample* is a decision-list (policy) of rules.
- The generator is *altered* to produce the rules used in highest valued policies more often.
- Two parts to CERRLA: **Rule Discovery** and **Probability Optimisation**.

clear(A), clear(B), block(A) $\rightarrow$ *move(A, B)*
above(X, B), clear(X), floor(Y) $\rightarrow$ *move(X, Y)*
above(X, A), clear(X), floor(Y) $\rightarrow$ *move(X, Y)*

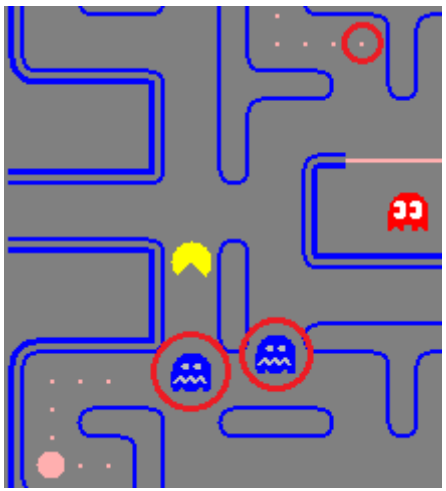
Rule Discovery

- Rules are created by first identifying pseudo-RLGG rules for each action.
- Each rule can then produce more specialised rules by:
  - Adding a single literal to the rule conditions.
  - Replacing a variable with a goal variable.
  - Splitting numerical ranges into smaller partitions.
- All information makes use of lossy inverse substitution.

Example

- *The RLGG for the Blocks World move action is:*
  clear(X), clear(Y), block(X) $\rightarrow$ move(X, Y)
- *Specialisations include: highest(X), floor(Y), X/A, . . .*

Relative Least General Generalisation Rules*



For the `moveTo` action:

1. $edible(g_1), ghost(g_1), distance(g_1, 5), thing(g_1) \rightarrow moveTo(g_1, 5)$
2. $edible(g_2), ghost(g_2), distance(g_2, 8), thing(g_2) \rightarrow moveTo(g_2, 8)$

$RLGG_{1,2}$ $edible(\mathbf{X}), ghost(\mathbf{X}), distance(\mathbf{X}, (5.0 \leq \mathbf{D} \leq 8.0)), thing(\mathbf{X}) \rightarrow moveTo(\mathbf{X}, \mathbf{D})$

3. $distance(d_3, 14), dot(d_3), thing(d_3) \rightarrow moveTo(d_3, 14)$

$RLGG_{1,2,3}$ ~~$edible(\mathbf{X}), ghost(\mathbf{X}),$~~ $distance(\mathbf{X}, (5.0 \leq \mathbf{D} \leq 14.0)), thing(\mathbf{X}) \rightarrow moveTo(\mathbf{X}, \mathbf{D})$

* Closer to LGG, as background knowledge is explicitly known.

Simplification Rules

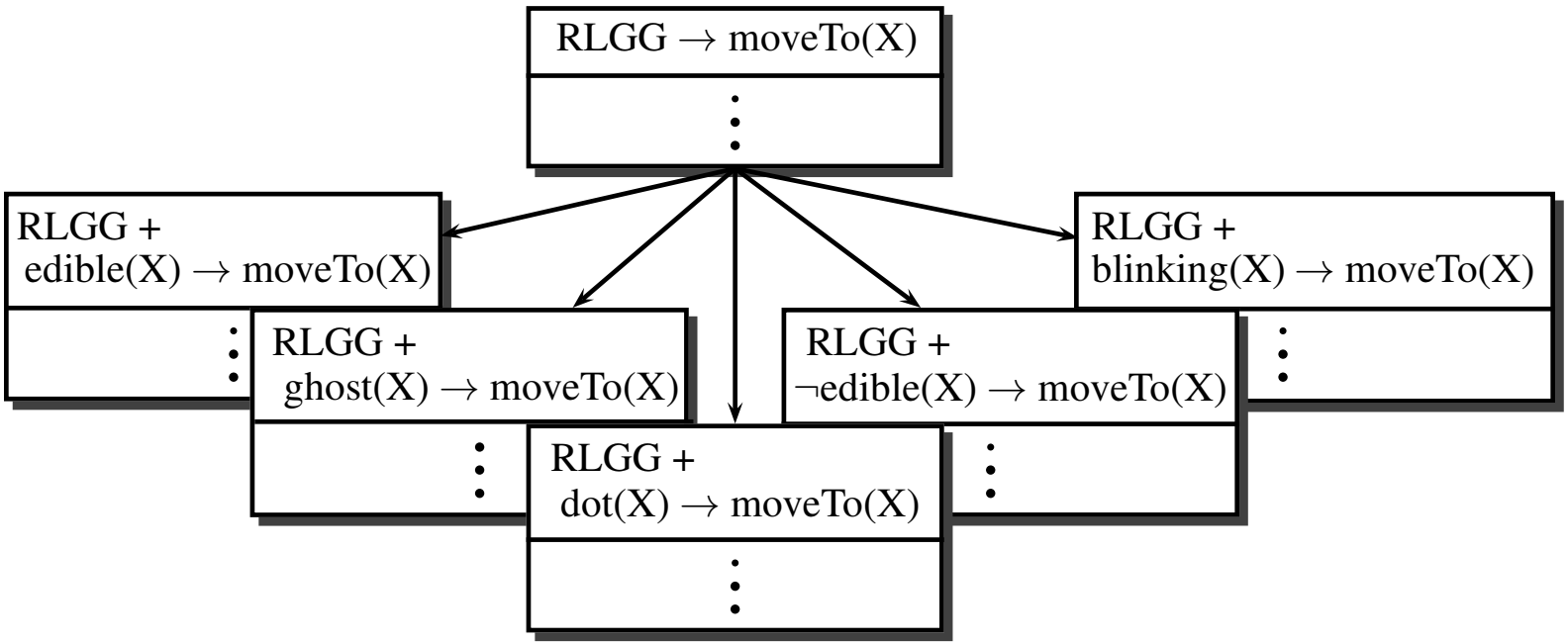
- Simplification rules are also inferred from the environment.
- They are used to remove redundant conditions and identify illegal combinations.
- Use the same RLGG process, but only using state facts.
- We can infer the set of variable form *untrue* conditions for a state to use negated terms in simplification rules.

Example

- *When on(X, Y) is true, above(X, Y) is true*
- $on(X, Y) \Rightarrow above(X, Y)$
- $block(X) \Leftrightarrow not(floor(X))$

Initial Rule Distributions

- Initial rule distributions consist of RLGG distributions and all immediate specialisations.



Probability Optimisation

- A policy consists of multiple rules.
- Each rule comes from a separate distribution.
- Rule usage and position are determined by CEM controlled probabilities.
- Each policy is tested three times.

Distribution A	Distribution B	Distribution C
$a_1 : 0.6$ $a_2 : 0.2$ $a_3 : 0.15$ $\vdots$	$b_1 : 0.33$ $b_2 : 0.33$ $b_3 : 0.33$	$c_1 : 0.7$ $c_2 : 0.05$ $c_3 : 0.05$ $\vdots$
$p(D_A) = 1.0$ $q(D_A) = 0.0$	$p(D_B) = 0.5$ $q(D_B) = 0.5$	$p(D_C) = 0.3$ $q(D_C) = 0.8$

Example policy

a_1
b_3
c_1

Updating Probabilities

- A subset of samples make up the floating elite samples.
- The *observed distribution* is the distribution of rules in the elites.
  - Observed rule probability equal to frequency of rules.
  - Observed $p(D)$ equal to proportion of elite policies using D.
  - Observed $q(D)$ equal to average relative position $[0, 1]$.
- Probabilities are updated in a stepwise fashion towards the observed distribution.

$$p_i \leftarrow \alpha \cdot p'_i + (1 - \alpha) \cdot p_i$$

Updating Probabilities, Contd.

- When a rule is sufficiently probable, it branches, seeding a new candidate rule distribution.
- More and more specialised rules are created until further branches are not useful.
- Stopping Condition: A seed rule cannot branch again.
- Convergence occurs when each distribution converges (no significant updates).

Summary

Initialise the distribution set $\mathbb{D}$
repeat
  Generate a policy π from $\mathbb{D}$
  Evaluate π, receiving average reward R
  Update elite samples E with sample π and value R
  Update $\mathbb{D}$ using E
  Specialise rules (if $\mathbb{D}$ is ready)
until $\mathbb{D}$ has converged

How Well Does It Work?

- Each environment provides a different style of problem to solve.
- E.g. Competing agents, partial information, numerical information.
- The policies produced are not necessarily optimal, but can be competitive.
- The policies are understandable (relative to neural networks, random forests, etc).

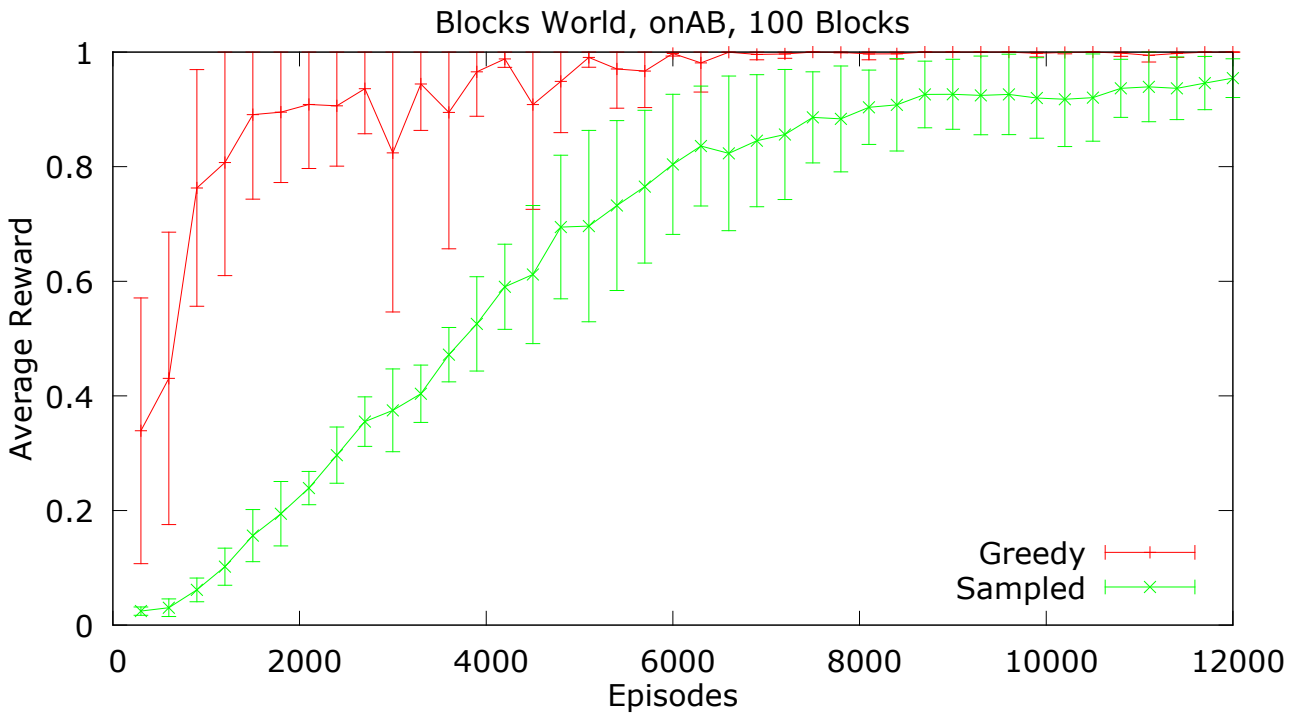
Blocks World

- Blocks World is the standard testing environment for RRL.
- Simple dynamics, single reward state, fully deterministic (usually), single action.

Algorithm	Average Reward		# of Training Episodes ($\times$1000)	
	stack	*onAB*	*stack*	*onAB*
CERRLA	**1.0**	**0.99**	**1.6**	**10.3**
P-RRL	1.0	0.9	0.045	0.045
RRL-TG	0.88	0.92	0.5	12.5
RRL-TG (P learning)	1.0	0.92	30	30
RRL-RIB	0.98	0.9	0.5	2.5
RRL-KBR	1.0	0.98	0.5	2.5
TRENDI	1.0	0.99	0.5	2.5
TREENPPG	—	0.99	—	2
MARLIE	1.0	0.98	2	2
FOXCS	1.0	0.98	20	50

Blocks World

- CERRLA is scale-free.
- Can learn strategies in enormous BW states, but is also hampered in small BW.



Ms. Pac-Man

- Fully observable, variable reward, multiple actions, multiple agents (ghosts).
- Achieves a similar reward to Szita and Lörincz's CEM algorithm.

edible(X), distance(X, Y) $\rightarrow$ *moveTo(X, Y)*
powerDot(X), distance(X, Y) $\rightarrow$ *moveTo(X, Y)*
thing(X), distance(X, Y), not(ghost(X)), not(ghostCentre(X)) $\rightarrow$ *moveTo(X, Y)*
dot(X), distance(X, $(26 <= Y <= 52)$*)* $\rightarrow$ *moveFrom(X, Y)*

Mario

- Partial map observability, variable reward, many actions (non-deterministic action resolution), multiple agents (enemies).
- The environment proved to be challenging, but some learning did take place.

marioPower(fire), *canJumpOn(X)*, *goomba(X)*, *heightDiff(X, ?)*, *width(X, ?)*, *distance(X, Y)* $\rightarrow$ *shootFireball(X, Y, fire)*
canJumpOn(X), *heightDiff(X, ?)*, *distance(X,* $(37.0 \leq Y \leq 304.0)$*)*, *not(powerup(X))*, *not(enemy(X))* $\rightarrow$ *jumpOnto(X, Y)*
. . . 4 more.

Carcassonne

- Variable reward, competing agent(s), many state predicates available.
- CERRLA learns effective behaviour in all different experiment settings.

currentPlayer(X), *controls(X, ?)*, *validLoc(Y, Z, W)*, *numSurroundingTiles(Z,* $(4.5 \leq D_0 \leq 8.0)$*)* $\rightarrow$ *placeTile(X, Y, Z, W)*
currentPlayer(X), *meepleLoc(Y, Z)*, *worth(Z,* $(3.0 \leq D_0 \leq 6.0)$*)*, *not(nextTo(?, ?, Z))* $\rightarrow$ *placeMeeple(X, Y, Z)*
. . . 7 more.

In Conclusion

- When applied to all four environments, CERRLA learns fast, effective, and comprehensible behaviour.
- No human guidance was given. Only the state observations and the predicate definitions were used.
- In the standard Blocks World, it performs near-optimally, and was shown that the number of states does not affect convergence.
- A specialised approach may achieve better performance, but often requires specialised techniques.
- Whereas CERRLA can be applied in any relational environment and learn good behaviour.


Questions?