Induction of Subgoal Automata for Reinforcement Learning

Daniel Furelos-Blanco[1], Mark Law[1], Alessandra Russo[1], Krysia Broda[1], and Anders Jonsson[2]

[1]Imperial College London, UK [2]Universitat Pompeu Fabra, Barcelona, Spain

Motivation

- Advances to achieve *generalization* and *transfer* between RL tasks are mainly due to *abstractions*.
- Abstract hierarchies have been represented using *automata* in *reinforcement learning (RL)* and *automated planning*.

Problem

Current RL methods use *handcrafted automata*.

Proposed Approach

ISA (Induction of Subgoal Automata)

A method for learning and exploiting a minimal automaton from observation traces perceived by an RL agent.

- Learn an automaton whose transitions are labeled by propositional formulas representing *subgoals*.
- The *automata learning* is formulated as an *inductive logic programming* task.
- The automata can be exploited by RL algorithms.

Tasks

The tasks are *episodic* MDPs $\mathcal{M} = \langle S, A, p, r, \gamma, S_T, S_G \rangle$ where:

- S is a finite set of states,
- A is a finite set of actions,
- $p : S \times A \to \Delta(S)$ is a transition probability function,
- $\gamma \in [0, 1)$ is a discount factor,
- $S_T \subseteq S$ is the set of *terminal states*,
- $S_G \subseteq S_T$ is the set of *goal states*, and
- $r : S \times A \times S \to \mathbb{R}$ is a *reward function* such that

$$r(s, a, s') = \begin{cases} 1 & \text{if } s' \in S_G \\ 0 & \text{otherwise} \end{cases}.$$

- The automaton transitions are defined by a logical formula over a set of *observables* $\mathcal{O}$.
- A *labeling function* $L : S \to 2^{\mathcal{O}}$ maps a state into a subset of observables perceived by the agent.

Example The OfficeWorld domain (Toro Icarte et al., 2018), where $\mathcal{O} = \{☕, ✉, o, A, B, C, D, *\}$.

- Coffee: deliver coffee to the office.
- CoffeeMail: deliver coffee and mail to the office.
- VisitABCD: visit locations A, B, C and D in order.

The tasks terminate when the goal is achieved or a $*$ is broken (this is a dead-end state).

Subgoal Automata

A *subgoal automaton* is a tuple $\mathcal{A} = \langle U, \mathcal{O}, \delta, u_0, u_A, u_R \rangle$ where

- U is a finite set of states,
- $\mathcal{O}$ is a set of observables (or alphabet),
- $\delta : U \times 2^{\mathcal{O}} \to U$ is a deterministic transition function,
- $u_0 \in U$ is a start state,
- $u_A \in U$ is the unique accepting state, and
- $u_R \in U$ is the unique rejecting state.

Learning Subgoal Automata from Traces

Input

- A set of states $U \supseteq \{u_0, u_A, u_R\}$.
- A set of observables $\mathcal{O}$.
- A set of traces $\mathcal{T}_{L,\mathcal{O}} = \langle \mathcal{T}^+_{L,\mathcal{O}}, \mathcal{T}^-_{L,\mathcal{O}}, \mathcal{T}^I_{L,\mathcal{O}} \rangle$.

Output

The automaton's transition function such that the automaton:

- accepts all *positive traces* $\mathcal{T}^+_{L,\mathcal{O}}$,
- rejects all *negative traces* $\mathcal{T}^-_{L,\mathcal{O}}$,
- neither accepts nor rejects *incomplete traces* $\mathcal{T}^I_{L,\mathcal{O}}$.

The automaton learning task is described as an *Inductive Learning from Answer Sets (ILASP)* task:

- The *learned rules* are of two types:

  Facts $\mathtt{ed(X, Y, EDGE\_ID)}$ + Rules $\bar{\delta}(\mathtt{X, Y, EDGE\_ID, T})$

- The actual transitions are defined in terms of the *negative* ones:

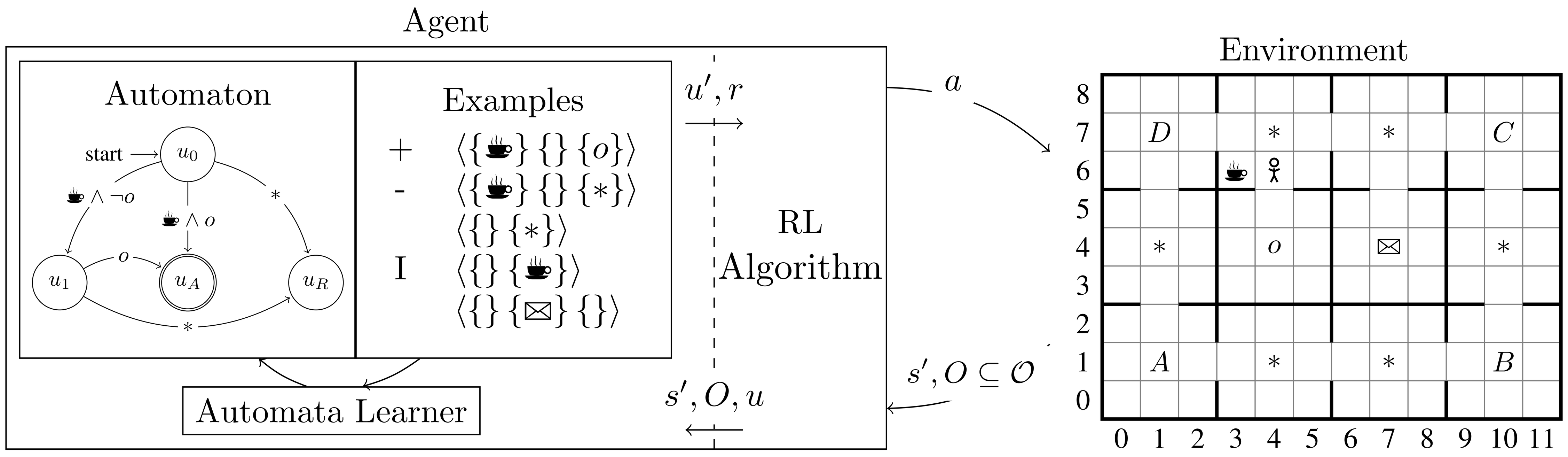
  $\delta(\mathtt{X, Y, EDGE\_ID, T})$:- not $\bar{\delta}(\mathtt{X, Y, EDGE\_ID, T})$.

- Each *trace* is expressed as a set of $\mathtt{obs(O, T)}$ facts.

  $\langle \{☕\}, \{\}, \{o\} \rangle \to \{\mathtt{obs}(☕, 0).\ \mathtt{obs}(o, 2).\}$

Example

$\mathtt{ed}(u_0, u_1, 1).\ \mathtt{ed}(u_0, u_A, 1).$
$\bar{\delta}(u_0, u_1, 1, \mathtt{T})$:- not $\mathtt{obs}(☕, \mathtt{T}), \mathtt{step}(\mathtt{T})$.
$\bar{\delta}(u_0, u_1, 1, \mathtt{T})$:- $\mathtt{obs}(o, \mathtt{T}), \mathtt{step}(\mathtt{T})$.
$\bar{\delta}(u_0, u_A, 1, \mathtt{T})$:- not $\mathtt{obs}(o, \mathtt{T}), \mathtt{step}(\mathtt{T})$.
$\bar{\delta}(u_0, u_A, 1, \mathtt{T})$:- not $\mathtt{obs}(☕, \mathtt{T}), \mathtt{step}(\mathtt{T})$.



Interleaved Learning Algorithm

QRM (Q-Learning for Reward Machines)

- Keep a Q-function for each automaton state.
- Update rule ($r = 1$ if $u' = u_A$):

$$Q_u(s, a) = Q_u(s, a) + \alpha \left(r + \gamma \max_{a'} Q_{u'}(s', a') - Q_u(s, a) \right).$$

- Updates all Q-functions after every step (s, a, s').

ISA Algorithm RL and automata learning are *interleaved*.

- The *initial automaton* does not accept nor reject anything.
- The automaton learner runs when a *counterexample* is found:
  - multiple transitions from the current state u hold, or
  - it does not correctly recognize the MDP state s.
- When a new automaton is learned, all Q-functions are *reset*.

Reward shaping Leverage the *automaton structure*: give extra reward for getting closer to the accepting state.

$$F(u, u') = \gamma \Phi(u') - \Phi(u), \text{ where } \Phi(u) = |U| - d(u, u_A)$$

Experimental Results

- Given *100 random grids*, simultaneously:
  - learn a policy for each of these, and
  - an automaton that generalizes to all of them.
- Use *compressed traces*, e.g. $\langle \{☕\}, \{\}, \{\}, \{o\} \rangle \to \langle \{☕\}, \{\}, \{o\} \rangle$.
- Use *Q-tables* to represent the Q-functions.

Constraints

1. The automata are forced to be *acyclic*.
2. The automaton states must be visited in increasing index order (*symmetry breaking*).

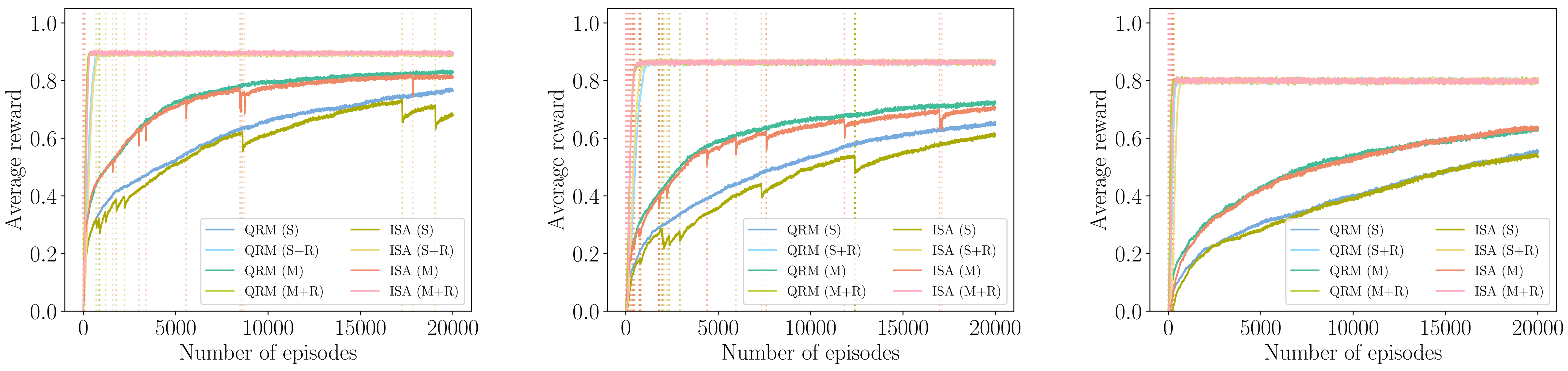


Fig. 3: Average learning curves (10 runs). **S** - single task, **M** - multitask, **R** - reward shaping.

	All	+	-	I
Coffee	6.6 (0.5)	2.2 (0.2)	2.3 (0.2)	2.1 (0.3)
CoffeeMail	34.5 (2.9)	5.5 (0.4)	9.9 (0.9)	19.1 (2.2)
VisitABCD	32.5 (2.1)	1.7 (0.2)	11.6 (0.8)	19.2 (1.7)

Fig. 4: Average number of examples (setting **S**).

- ↑ task complexity → + examples, + time.
- $|\mathcal{T}_{L,\mathcal{O}}| \approxeq$ #paths(u_0, u_A).

	S	S+R	M	M+R
Coffee	0.5 (0.0)	0.4 (0.0)	0.3 (0.0)	0.4 (0.0)
CoffeeMail	43.3 (12.1)	36.9 (6.0)	24.8 (3.6)	24.6 (2.7)
VisitABCD	63.0 (11.4)	68.5 (13.0)	48.4 (8.8)	69.6 (8.1)

Fig. 5: Average ILASP running time.

- ILASP time << Total time.
- There is not a setting consistently better than the others.

Conclusions

- Algorithm for learning subgoals by inducing an automaton from observation traces.
- The automaton structure can be exploited using an existing RL algorithm.
- Performance is comparable to the case where the automaton is given beforehand.

Imperial College London
Universitat Pompeu Fabra Barcelona