EAI 2021

Reasoning Agents

Presentation

Lorenzo Guercio - 1913660

Francesco Starna - 1613660

Graziano Specchi - 1620431

Kevin Munda - 1905663

SAPIENZA
UNIVERSITÀ DI ROMA

Sapienza University of Rome

Reinforcement Learning in Regular Decision Processes

Algorithm and experiments explanation

SAPIENZA

Outline

SAPIENZA

Team

Francesco Starna

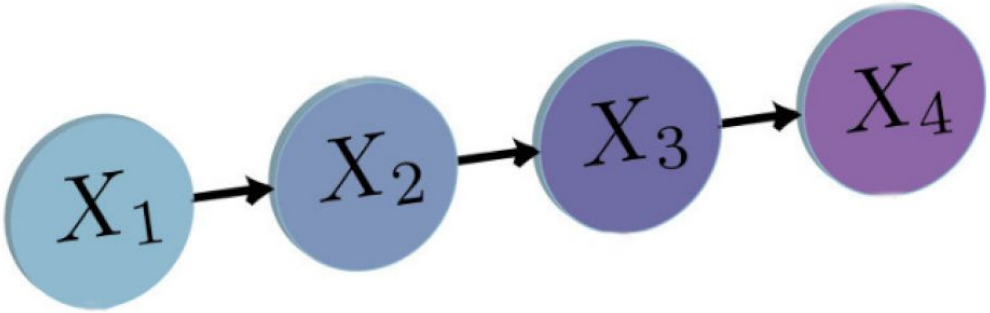
Graziano Specchi

Lorenzo Guercio

Kevin Munda

Motivations

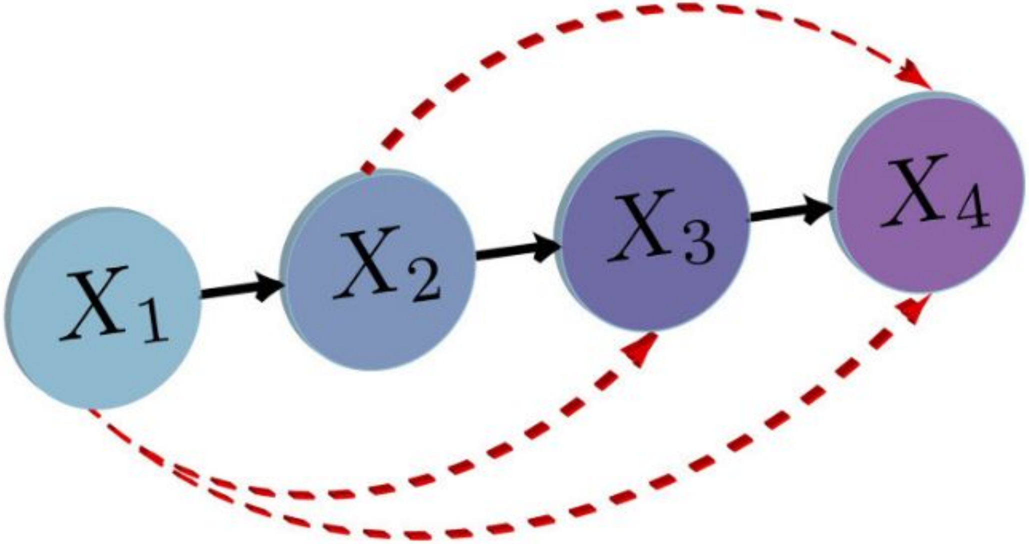
Reinforcement Learning has been using the Markov Assumption



Motivations

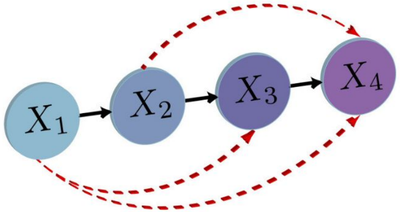
Reinforcement Learning has been using the Markov Assumption

But there are some domains in which it is not possible to neglect the past



Motivations

Reinforcement Learning has been using the Markov Assumption

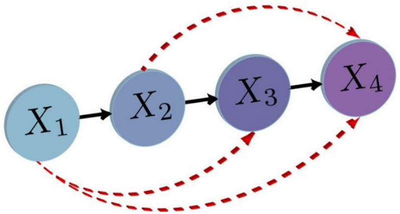


But there are some domains in which it is not possible to neglect the past

Regular Decision Processes have been proposed to overcome this issue

Motivations

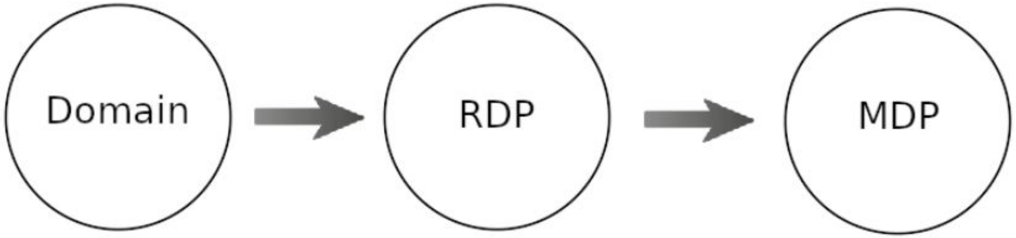
Reinforcement Learning has been using the Markov Assumption



But there are some domains in which it is not possible to neglect the past

Regular Decision Processes have been proposed to overcome this issue

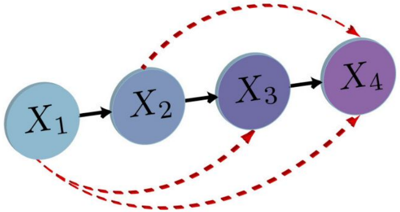
The knowledge of a RDP allows to compute an equivalent Markov Decision Process (MDP)



SAPIENZA

Motivations

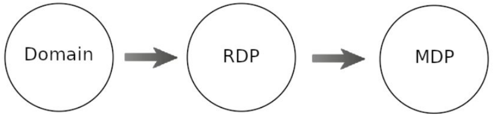
Reinforcement Learning has been using the Markov Assumption



But there are some domains in which it is not possible to neglect the past

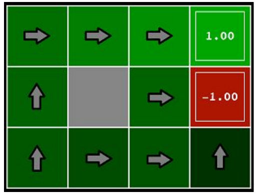
Regular Decision Processes have been proposed to overcome this issue

The knowledge of a RDP allows to compute an equivalent Markov Decision Process (MDP)



The resulting MDP can be solved by using already existing techniques

e.g. Value Iteration



Regular Decision Processes

Non-markovian Decision Process

Rewards depend on the sequence of past states rather than current state

Regular Expression Dependence

Transition and Reward functions are regular functions of history $(a_1 s_1 r_1, a_2 s_2 r_2 \ldots a_n s_n r_n)$

Several Formalisms

LDLf formulas, FST, regular expressions, probabilistic automata

[Regular Decision Processes: A Model for Non-Markovian Domains. 2019]

Reinforcement Learning Algorithm

Representing RDP as PDFA

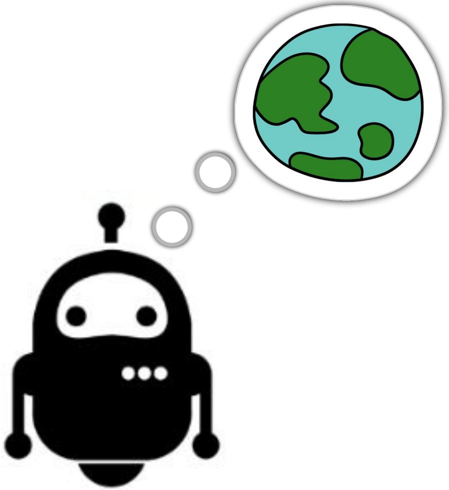
Transducer of dynamics of RDP to probabilistic automata

Learning PDFA

AdaCT learns the hypothesis graph

Solve MDP

Transition function of PDFA to build a MDP



[Efficient PAC Reinforcement Learning in Regular Decision Processes. 2021]

Sampling

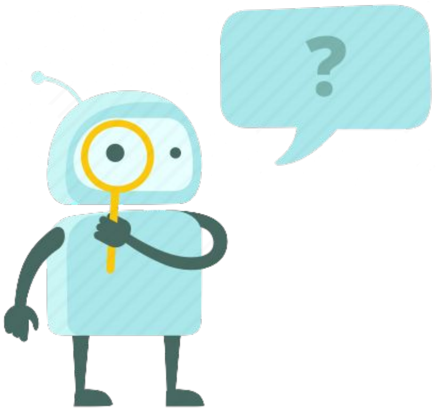
Sampling AdaCT MDP FST

Exploration Policy

Agent explores domain choosing an action uniformly at random

Stop Probability

Add stop action “$” with stop probability 1/(10*n^+1)



Learning Probabilistic Deterministic Finite-Automata

Sampling AdaCT MDP FST

PDFA <Q, Σ, τ, λ, ξ, q_0>

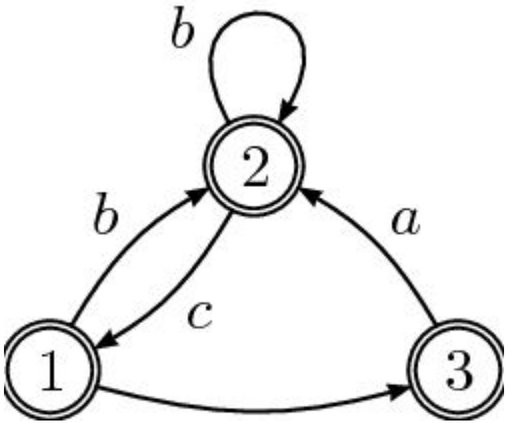
States, alphabet, transition, probability, eos, initial state

Hypothesis Graph

Nodes and edges are states and transition function

AdaCT Algorithm

[Learning probabilistic automata: A study in state distinguishability. 2013]



Learning Probabilistic Deterministic Finite-Automata

Sampling AdaCT MDP FST

Inputs: alphabet size $|\Sigma|$, upper bound n, confidence parameter δ, sample S with N examples

Output: hypothesis graph G

Max. n° steps: $n^*|\Sigma|$

The algorithm guarantees that at each step one transition is added to G

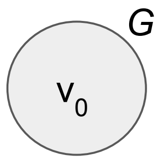
Learning Probabilistic Deterministic Finite-Automata

Sampling AdaCT MDP FST

Two periods:

- **‘Important period’:** the graph built in this period is theoretically guaranteed to be a good representation of the target *T*. Nodes added during this period are called ‘safe nodes’
- **Second period:** transitions added during this period could possibly be wrong

At the start, *G* consists of a single node whose attached multiset equals *S*



Learning Probabilistic Deterministic Finite-Automata

Sampling — AdaCT — MDP — FST

Routine:

- Creation of candidate nodes $u = v_u\, \sigma$ with multiset (initially empty) S_u
- For each string $x\xi = \sigma_0\, \sigma_1 ... \sigma_k$ in the training sample S, traverse the graph matching each observation σ_i :
  - All observations are matched to a state and the example is discarded
  - Observation σ_{i-1} lead to node w but there is no edge out of w labeled by σ_i . The suffix $\sigma_{i+1} \ldots \sigma_k$ is added to the multiset of the candidate
- Choose the candidate $u = v_u\, \sigma$ whose multiset has maximum cardinality
- Check if the candidate is a distinct state of the target calling 'TestDistinct'

Learning Probabilistic Deterministic Finite-Automata

Sampling | AdaCT | MDP | FST

TestDistinct:

- **'Not Clear'**: Choose the safe *v* that has not been distinguished from *u* and add an edge from v_u to *v* labeled by σ
- **All 'Distinct'**: Candidate node *u* is promoted to a safe node. *G* gets a new node labeled with *u* and an edge from v_u to *u* labeled by σ

Algorithm 4: Test Distinct

Input : a candidate node u and a safe node v
Output: $Distinct$ or $NotClear$
$m_u \longleftarrow |S_u|$; $s_u \longleftarrow |prefixes(S_u)|$
$m_v \longleftarrow |S_v|$; $s_v \longleftarrow |prefixes(S_u)|$
$t_{u,v} \longleftarrow \sqrt{\frac{2}{min(m_u,m_v)} \cdot ln\frac{8(s_u+s_v)}{\delta_0}}$
$d \longleftarrow max(L_\infty(\hat{S}_u,\hat{S}_v), prefL_\infty(\hat{S}_u,\hat{S}_v))$
if $d > t_{u,v}$ **then**
  return $Distinct$
else
  return $NotClear$

AdaCT pseudo-code

Sampling — AdaCT — MDP — FST

Algorithm 1: AdaCT

Inputs : alphabet size Σ, upper bound of the number of states of the target n, confidence parameter δ, sample S drawn from the target PDFA T

Output: Hypothesis graph H

```
H ⟵ init hypothesis graph with a node (initial state, multiset S);
M ⟵ n ∗ Σ, maximum steps;
SN ⟵ initialstate, safe nodes array;
for i ≤ M do
    CN ⟵ createCandidateNodesArray;
    if length(CN)=0 then
        return H
    end
    CN ⟵ populateMultisets(initial node);
    N ⟵ select the node with maximum cardinality;
    for j ≤ length(SN) do
        res ⟵ TestDistinct(N, SN_i, H, CN, δ, n, Σ);
        if res = 'Not Clear' then
            Add a transition from the parent of the candidate to the unclear node
        end
    end
    if res = 'Distinct' then
        Take the previous safe node;
        Add the link between the previous safe node and the new safe node;
        Add the new node to the graph;
    end
end
return H
```

Equivalent Markov Decision Process

Sampling — AdaCT — MDP — FST

MDP <Q, A, D^M, R, q_0, γ>

Compute equivalent MDP from PDFA

Theorem

If π_* is an optimal policy for the equivalent MDP, then $\pi_* \tau$ is optimal for the original RDP.

Dynamic Function

Probability function of Multisets

$$\mathbf{D}^{\hat{M}}(q_2, r | q_1, a) = (|A|/(1-p)) \sum_{s:\hat{\tau}'(q_1, asr)=q_2} \hat{\lambda}(q_1, asr).$$

Value Iteration

Solve MDP to find optimal policy π^*

repeat
  $i \leftarrow i+1$
  for each state s **do**
    for each action a **do**
      $V^i(s) = \sum_{s'} \mathrm{T(s'|s,a)}(R(s,a) + \gamma V^{i-1}(s'))$
    end for
  end for
  for each state s **do**
    $\pi^i(s) = \operatorname*{argmax}_a \sum_{s'} \mathrm{T(s'|s,a)}(R(s,a) + \gamma V^i(s')$

Transducer Composition

Sampling · AdaCT · MDP · FST

FST $\langle Q, q_0, S, \tau, A, \theta \rangle$

Transition and Reward functions of RDP
can be represented by Finite-state transducers

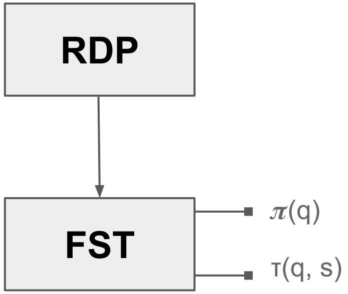
Composition

Find the mapping $\pi(\tau(\cdot))$ dropping actions and rewards

Resulting Transducer

Minimal representation of RDP, where

- $\theta(q) = \pi(q) \rightarrow a$
- $\tau(q, s) \rightarrow q$



Algorithm pseudo-code

Algorithm 5: Reinforcement Learning RL

Input : Actions A, discount factor γ, required precision ϵ, confidence parameter δ, upper bound $\hat{n}$ on transducer states

Output: transducer of policy π

$p \longleftarrow 1/(10 \cdot \hat{n} + 1)$ ⟵ maximum exploration

$X \longleftarrow$ generate episodes under exploration policy π_p

$\hat{\Sigma} \longleftarrow$ symbols in X

$\hat{R}_{max} \longleftarrow$ max reward in X

$\hat{\mathcal{A}} \longleftarrow$ learn PDFA by calling $AdaCT(\hat{n}, |\hat{\Sigma}|, \delta, X)$

$\hat{M} \longleftarrow$ compute MDP induced by $\hat{\mathcal{A}}$ and γ

$m \longleftarrow \lceil \frac{1}{1-\gamma} \cdot ln\left(\frac{2 \cdot \hat{R}_{max}}{\epsilon \cdot (1-\gamma)^2}\right) \rceil$ ⟵ iterations of Value Iteration algorithm

$\pi \longleftarrow$ solve $\hat{M}$ by calling $ValueIteration(\hat{M}, m)$

return transducer composition of π and transition function τ' of $\hat{\mathcal{A}}$

<u>Theorem</u>

Algorithm is PAC-RL w.r.t. the parameters

$$\mathbf{d}_{\mathcal{P}} = \left(|A|, \frac{1}{1-\gamma}, R_{\max}, n, \frac{1}{\rho}, \frac{1}{\mu}, \frac{1}{\eta}\right)$$

RL Algorithm Considerations

Fast Learning

How fast an agent learns a near-optimal policy

Probably Approximately Correct (PAC)

Look for ε-optimal policies with probability $1-\delta$

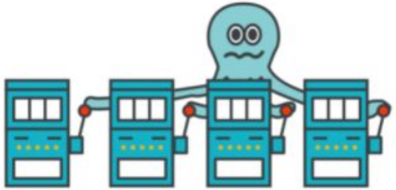
Exponentially-many Episodes Required

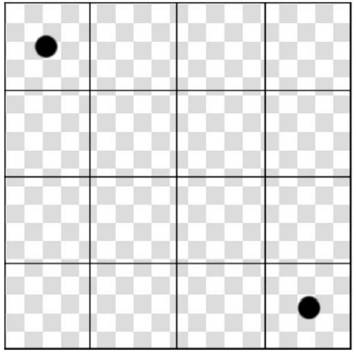
History length proportional to transducer states

Experiments

Rotating MAB

Multi-armed bandit with rotating probabilities



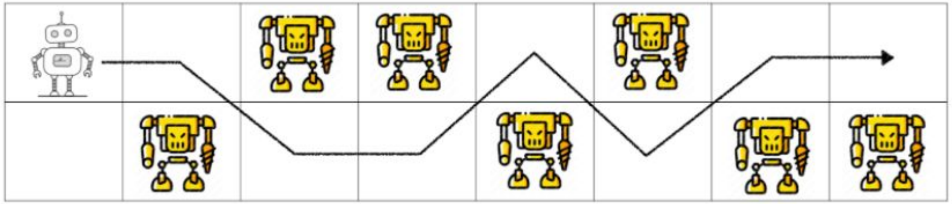


Rotating Maze

Maze domain with position and orientation variables

Enemy Corridor

2 x N corridor with enemies



Parameters

Parameters	γ	ϵ	δ	$\lvert X \rvert$	$\lvert \pi \rvert$	l
Description	discount factor	required accuracy	confidence of success	number of samples	number of learned policies	expected episode length
Values	0.1	0.01	0.1	50000-500000	5-10	10-15

List of parameters used in the RL algorithm

Number of samples: gets higher with the increasing complexity

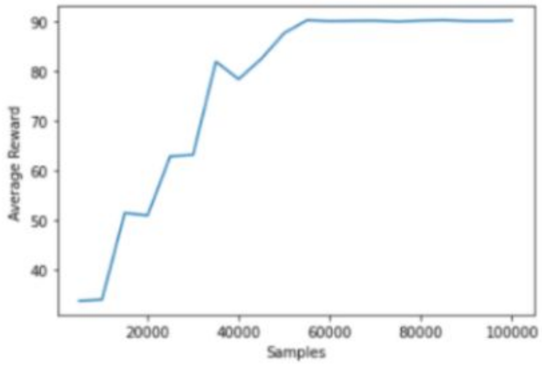
Number of learned policies: algorithm repeated between 5 and 10 times to find the average reward

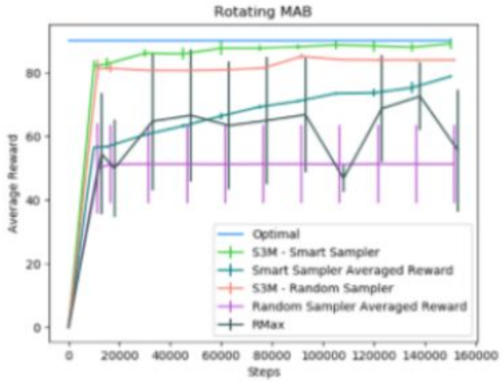
Episode length: indicates the stop probability value ***p = 1/(l+1)***

SAPIENZA

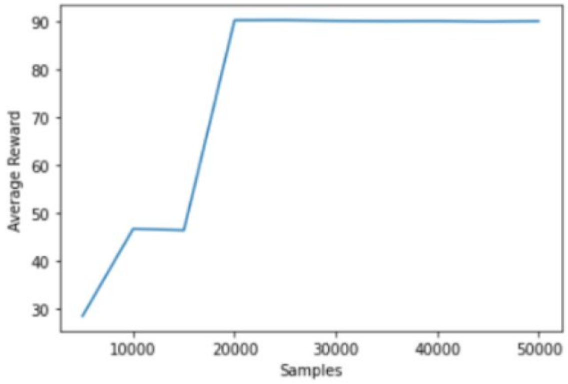
Results: Rotating MAB

MAX average reward: 90



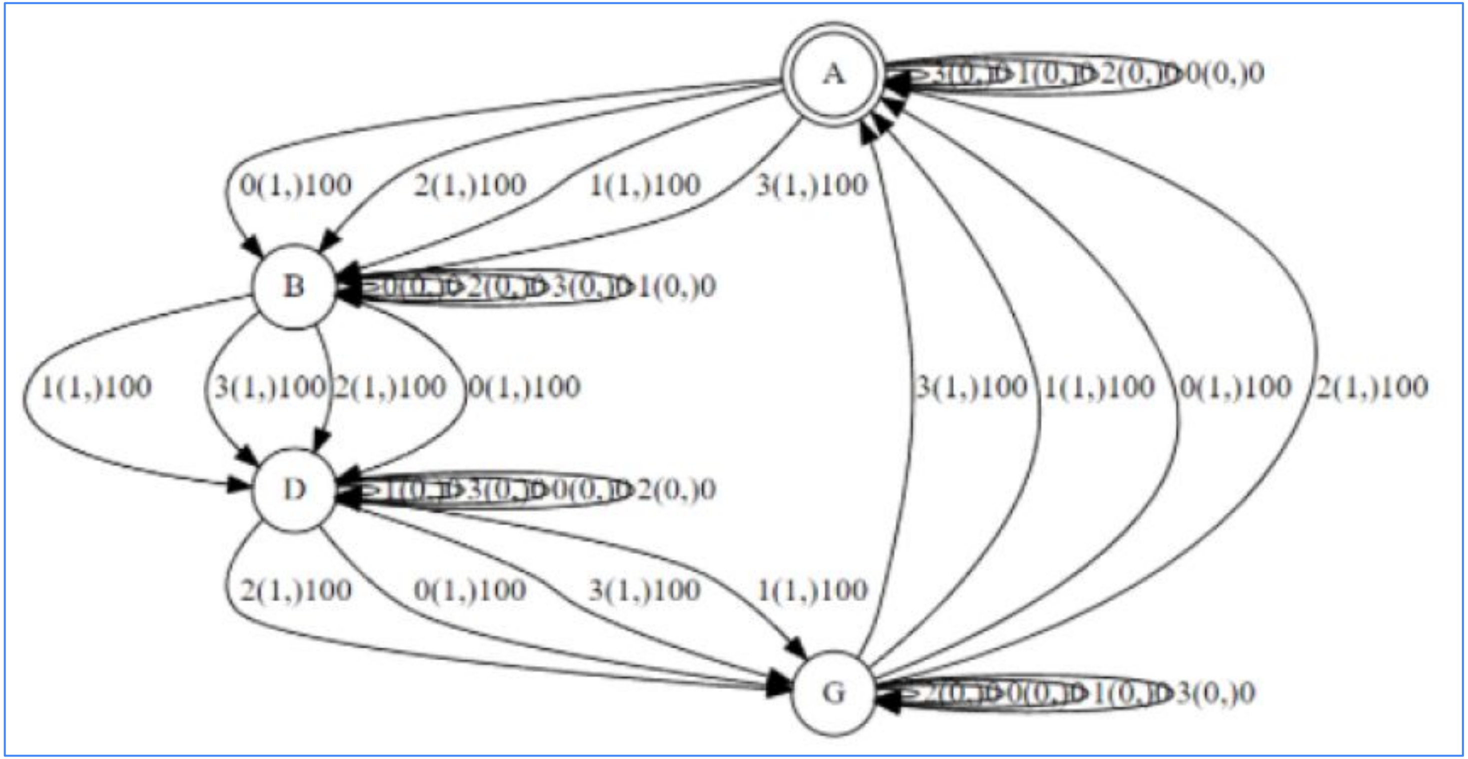


Comparison with the results obtained by Abadi and Brafman (*l = 10*)



Result obtained with expected episode length *l* = 40

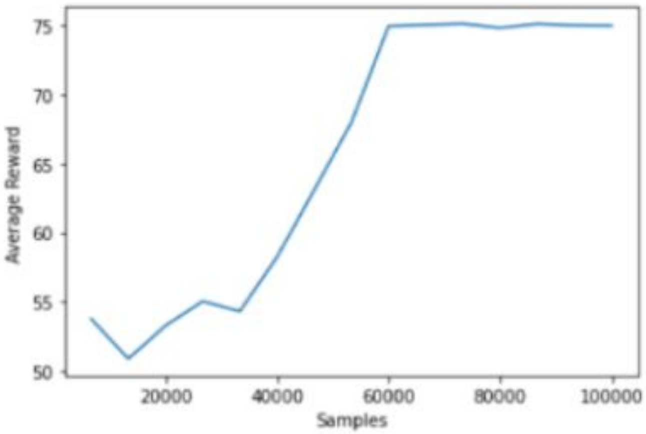
Results: Rotating MAB

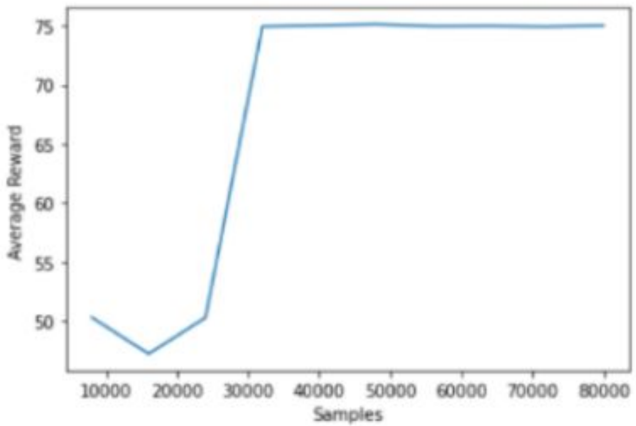


Graph of the Rotating MAB domain

Results: Enemy Corridor

MAX average reward: 75



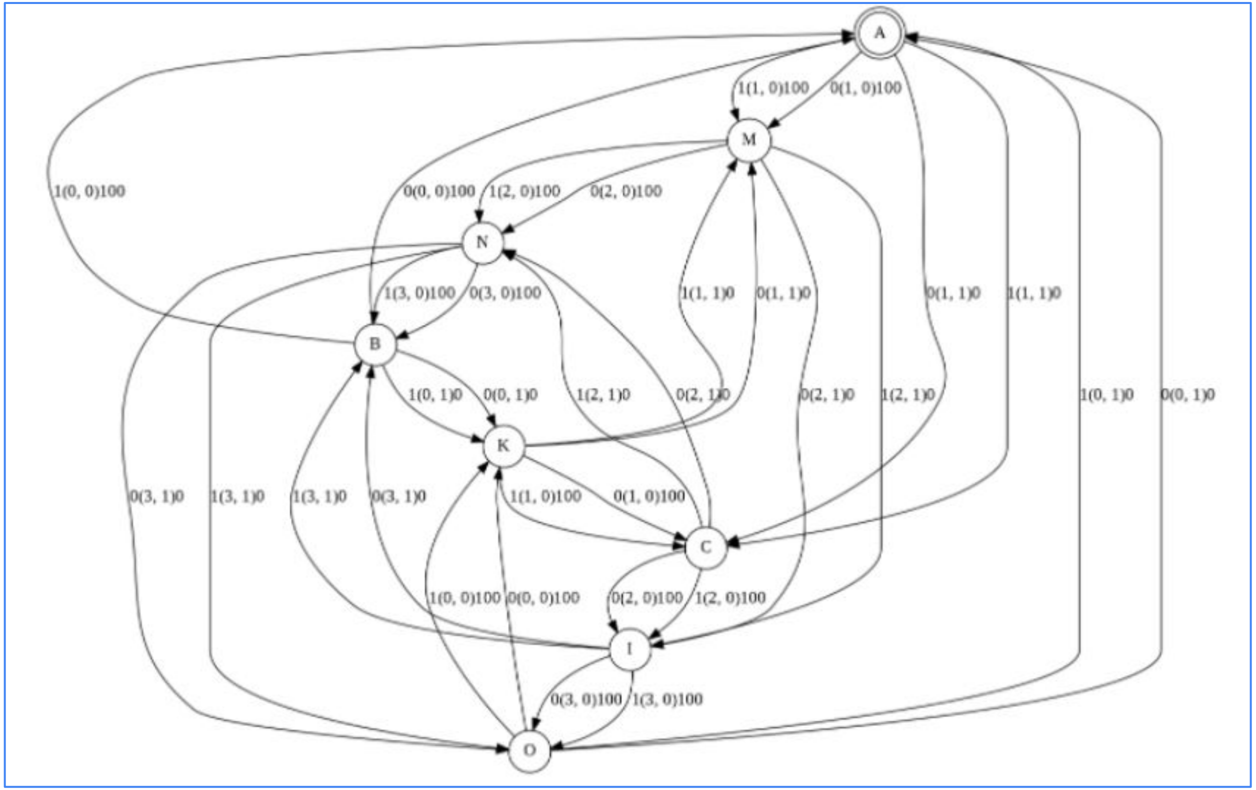


Left image → episode length ***l* = 20**

Right image → episode length ***l* = 80**

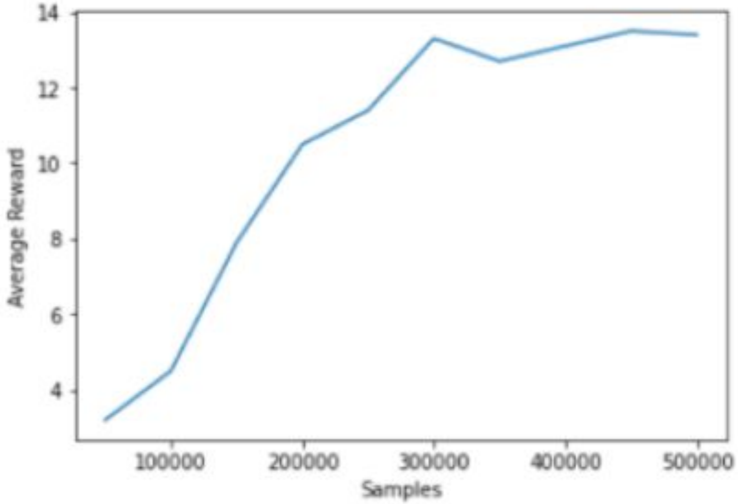
SAPIENZA

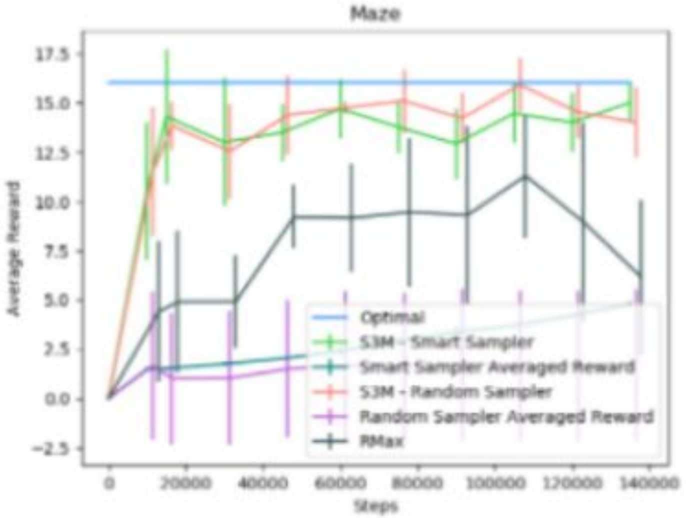
Results: Enemy Corridor



Graph of the Enemy Corridor domain

Results: Rotating Maze





Different environment: reward is only given at the target → lower average reward (MAX: 20)

Worse performances with respect to Abadi and Brafman

High number of samples needed → bad performances

SAPIENZA

Conclusions

PROs

- 1) Optimal performances for Rotating MAB and Enemy Corridor
- 2) Perfect learning of small and medium domains with a sufficient number of samples
- 3) Linear method with respect to the number of samples used

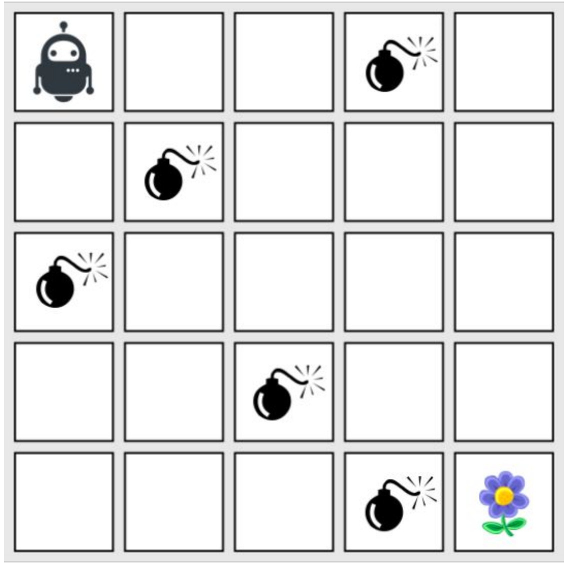
CONs

- 1) Mediocre performances for Rotating Maze
- 2) High computational cost for a big environment, too many samples needed
- 3) Less efficient for environments with many terminal states

Suitable Environment

Minefield

- N x M grid
- Actions → {*Right, Left, Down*}
- Rewards → {-20 bomb, 20*N flower}
- Observations → {*n*, *m*, bool: *bomb*}
- Initial state → (0, 0)
- Final state → (N, M)
- Probabilities → **1/m**, ∀ [1, n-1]; **1/m-1** with n = 0,N.
  When the agent hits the bomb they shift down.



References

Brafman, Ronen I. and De Giacomo, Giuseppe
Regular Decision Processes: A Model for Non-Markovian Domains. 2019
http://www.cvlibs.net/datasets/kitti/

Alessandro Ronca and Giuseppe De Giacomo. 2021
Efficient PAC Reinforcement Learning in Regular Decision Processes.
https://www.researchgate.net/publication/351623733_Efficient_PAC_Reinforcement_Learning_in_Regular_Decision_Processes

Balle, B., Castro, J. and Gavaldà, R. 2013
Learning probabilistic automata: A study in state distinguishability
https://doi.org/10.24963/ijcai.2019/766

SAPIENZA

Thanks