Accelerating Decision Making Under Partial Observability Using Learned Action Priors

by Ntokozo Jay Mabena

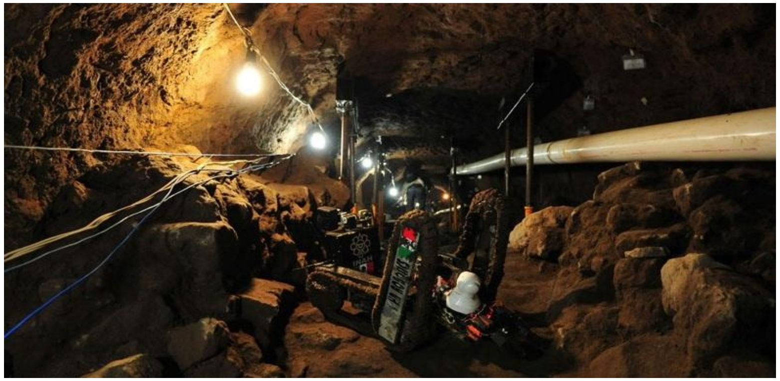
Supervised by Benjamin Rosman

Contents

Introduction

- Operational Uncertainty
  - Ambiguity in a robot's self-perceived state
  - adds ambiguity into the robot's state of operation.
  - Root cause of unsafe or risky behaviour

- Autonomous robots are overwhelmed with contingencies i.e



- Dynamic Environments
  - Handling uncertainty is essential
    - Control errors
    - limited sensing accuracy
    - inaccurate models of the environment
- Decision Making under imperfect state information
    - Depends on all states
    - Requires a sensing proficiency

- Disadvantage
  - Computationally expensive!!!

- Solution
  - POMDPs

Markov Decision Processes

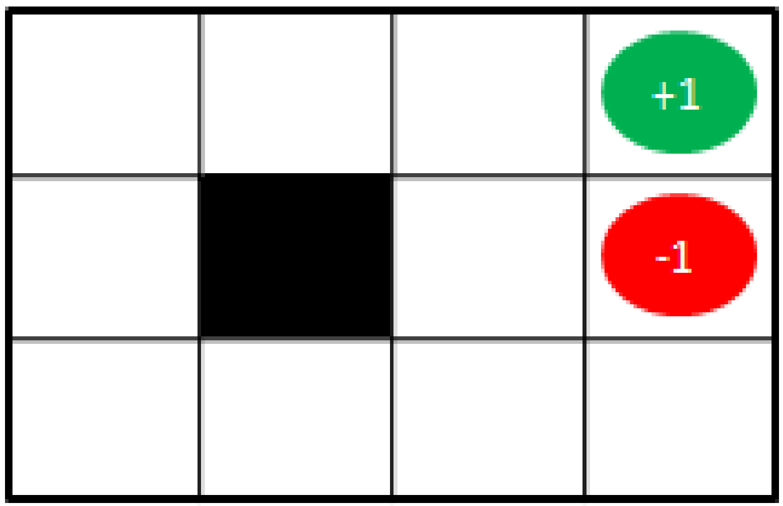
- Defined as a tuple $(S,\ A,\ T,\ R,\ \gamma)$ where
  - S *states*,
  - A actions,
  - $T: S \times A \times S \rightarrow [0,1]$,
  - $R: S \times A \times S \rightarrow \mathbb{R}$,
  - $\gamma \in [0,1]$
- States are fully observable

Goal of a learning agent

- Compute the policy $\pi(s)$: $S \rightarrow A$

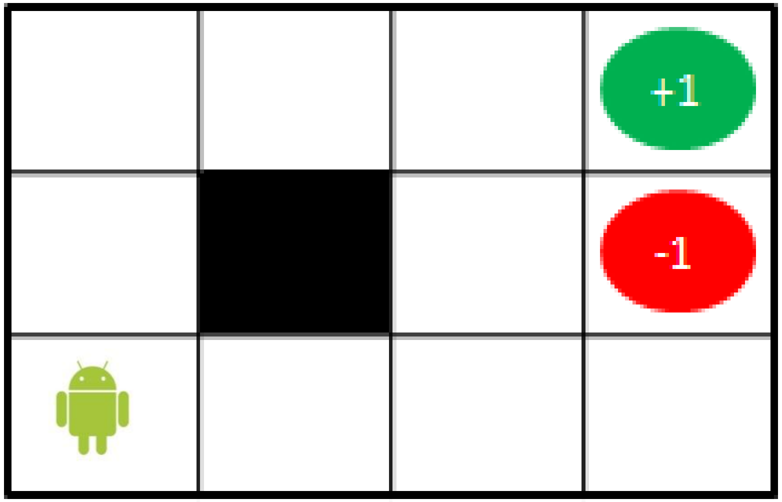
Example:

- Domain



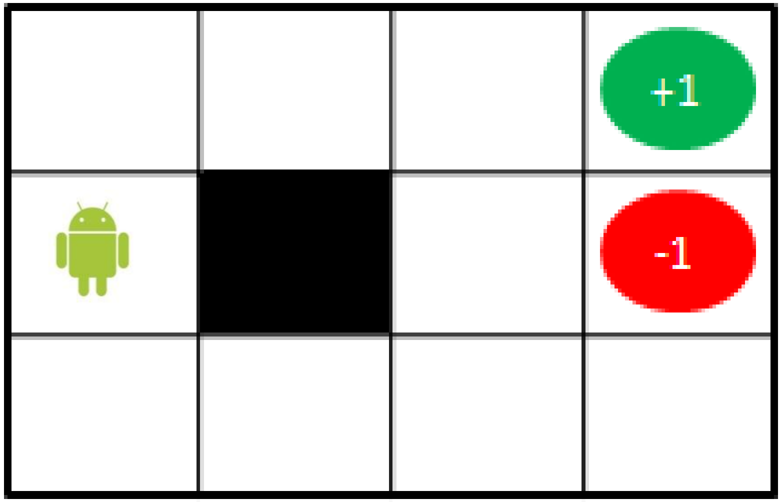
Example:

- Policy $\pi(s)$



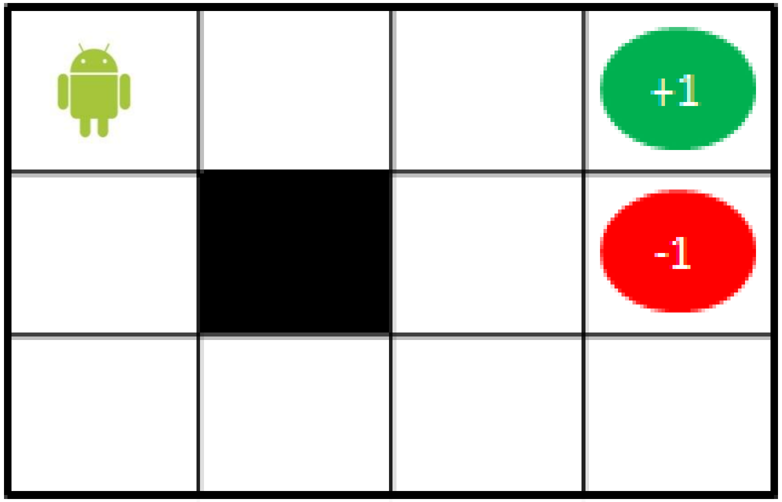
Example:

- Policy $\pi(s)$

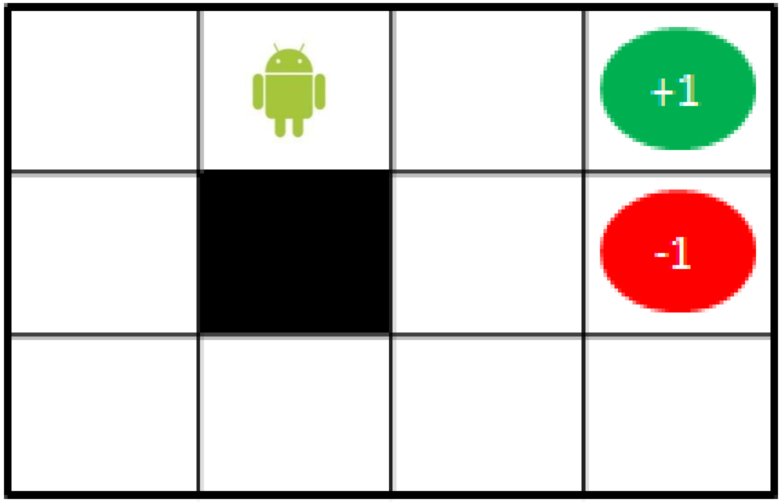


Example:

- Policy $\pi(s)$

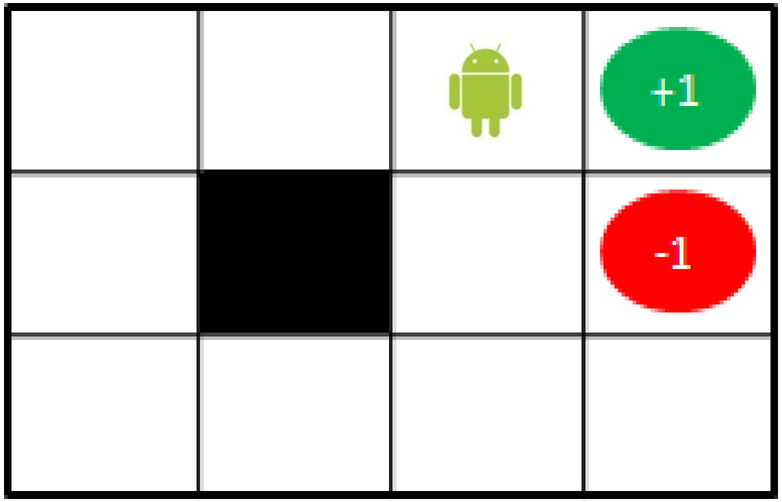


- Example:
  - Policy $\pi(s)$



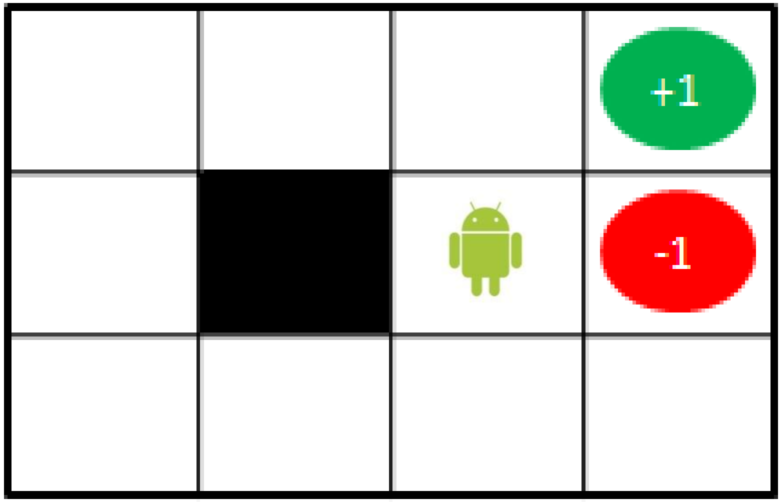
Example:

- Policy $\pi(s)$

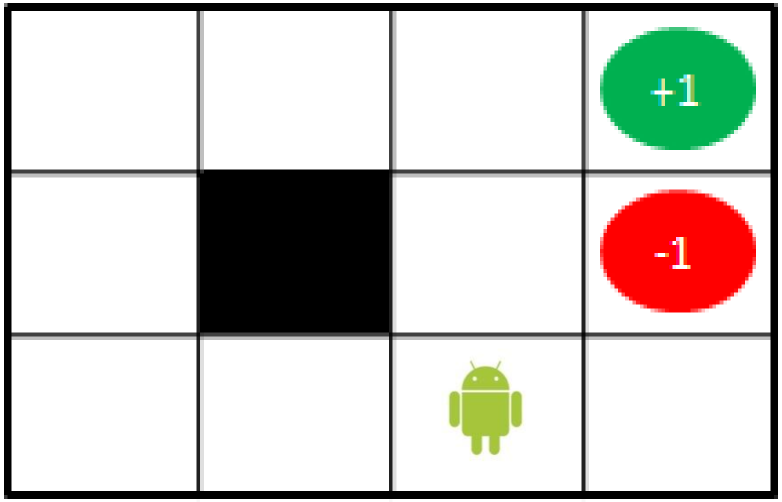


Example:

- Policy $\pi(s)$

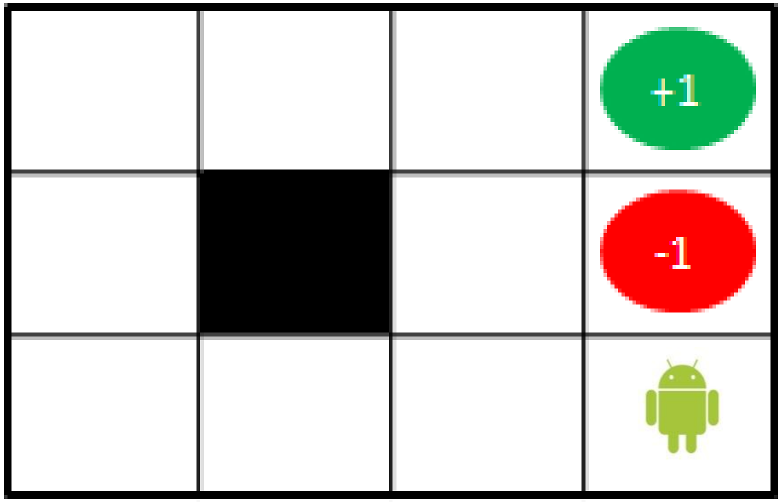


- Example:
  - Policy $\pi(s)$



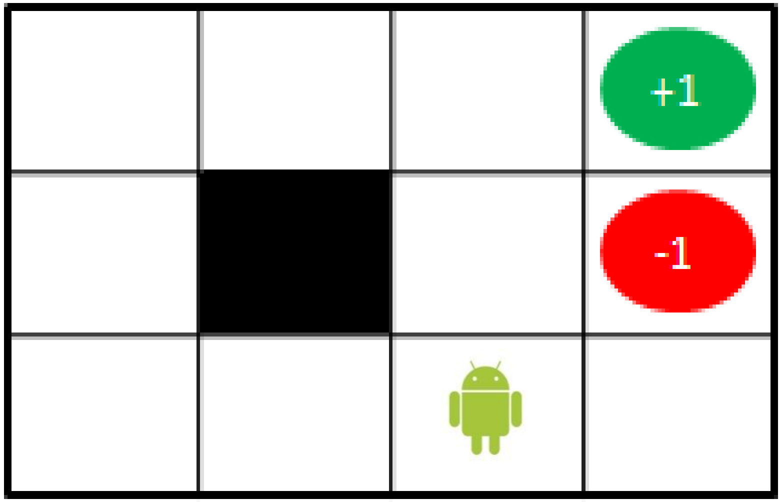
Example:

- Policy $\pi(s)$



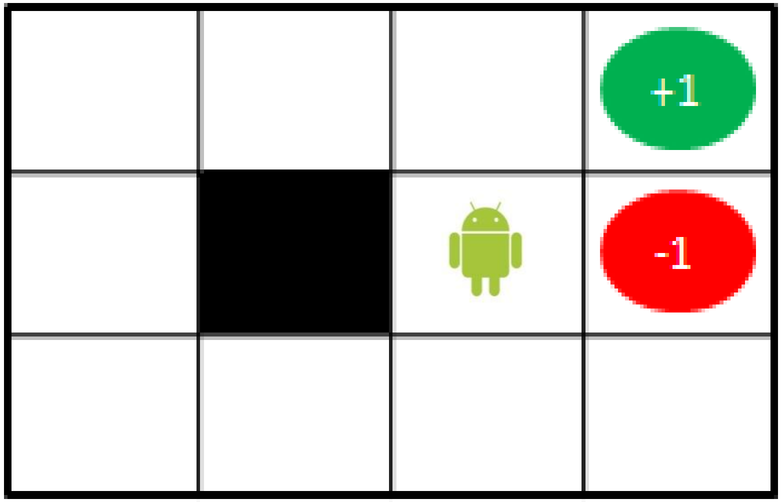
Example:

- Policy $\pi(s)$



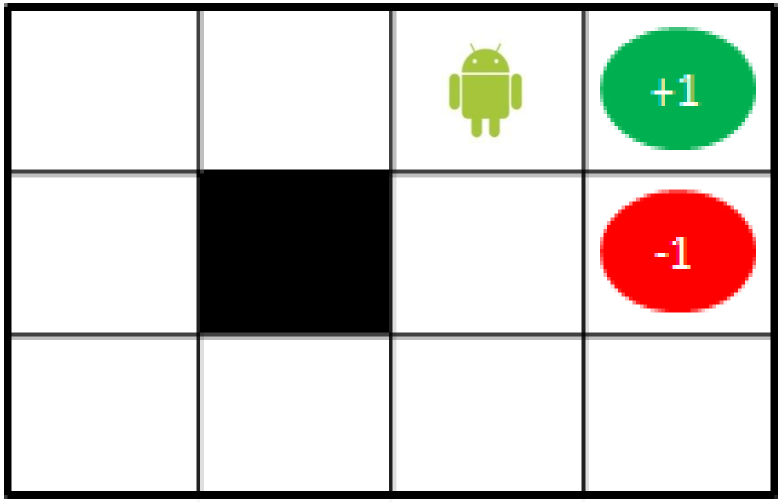
Example:

- Policy $\pi(s)$



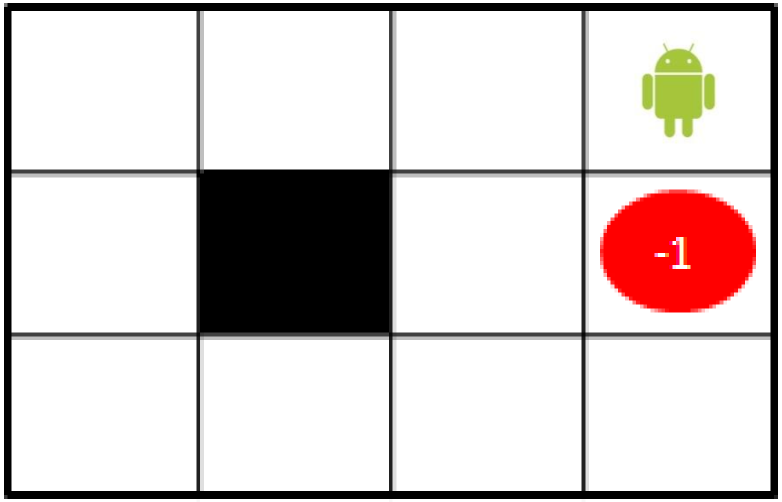
Example:

- Policy $\pi(s)$



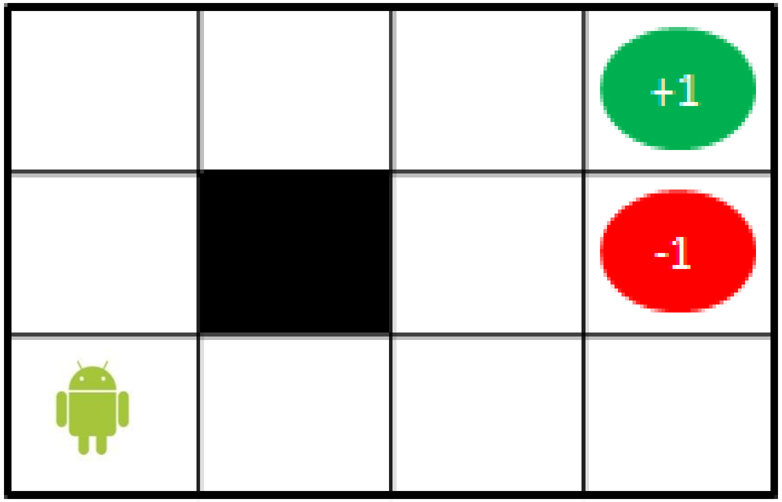
Example:

- Policy $\pi(s)$

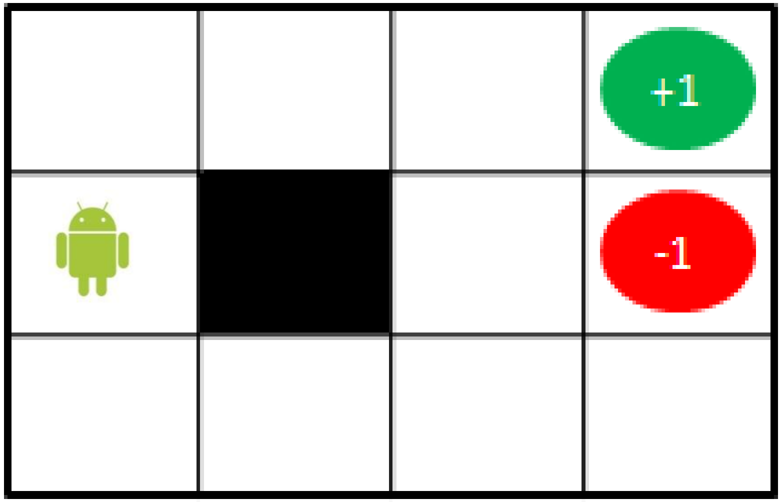


Example:

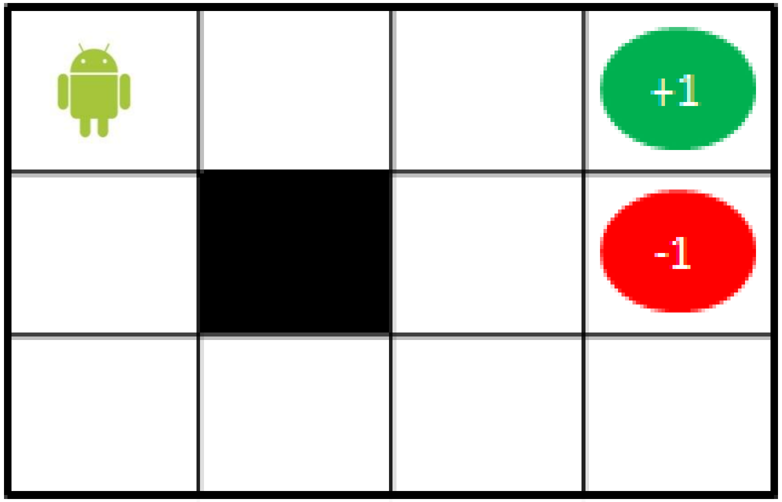
- Optimal policy $\pi^*(s)$



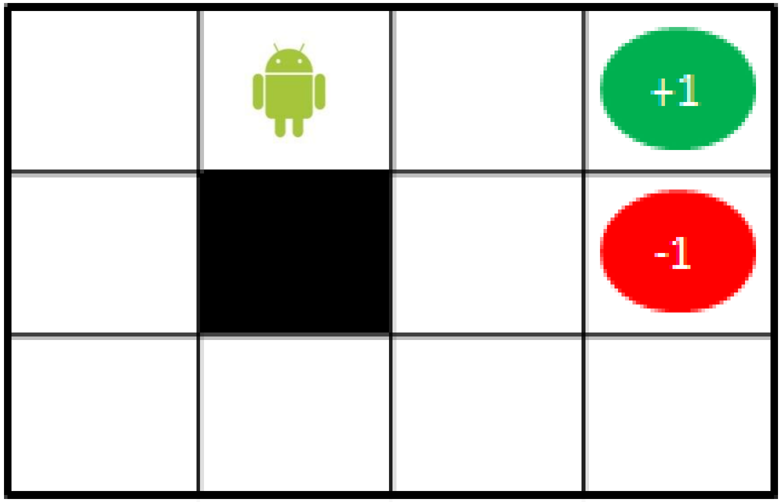
- Example:
    - Optimal policy $\pi^*(s)$



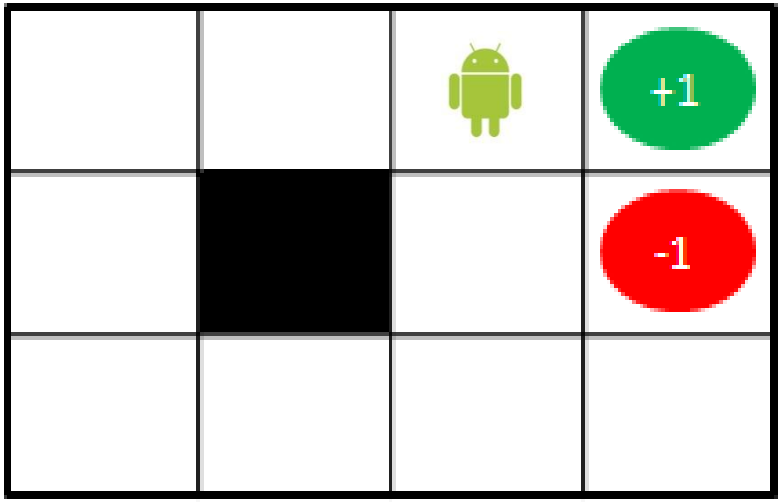
- Example:
  - Optimal policy $\pi^*(s)$



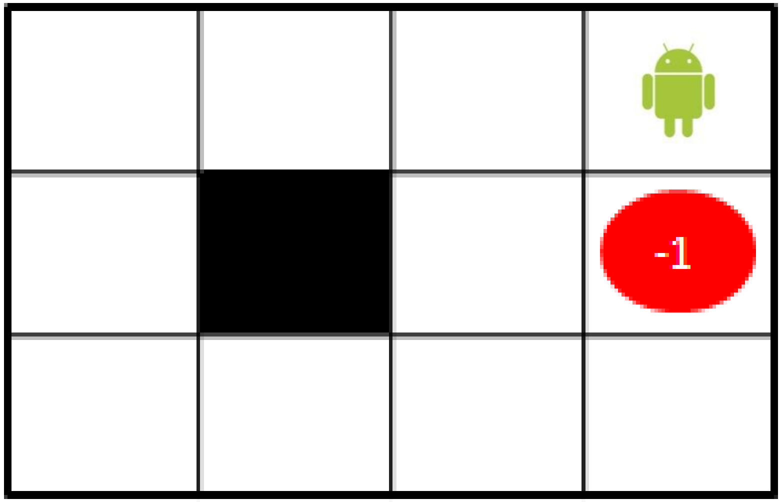
- Example:
  - Optimal policy $\pi^*(s)$



- Example:
  - Optimal policy $\pi^*(s)$

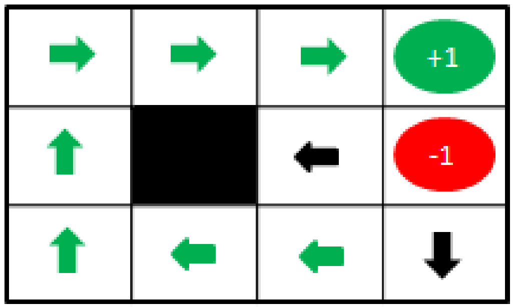


- Example:
  - Optimal policy $\pi^*(s)$

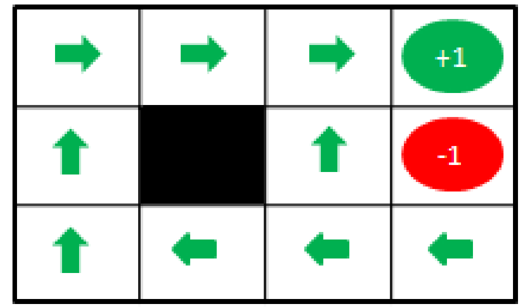


Example:

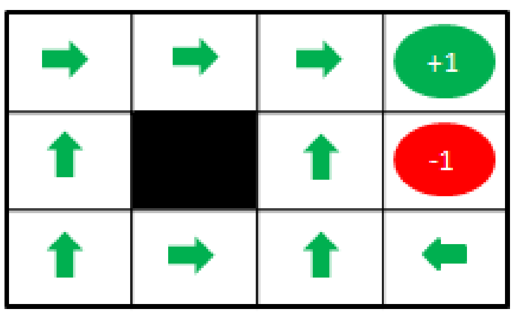
- Policy changes with due to living reward



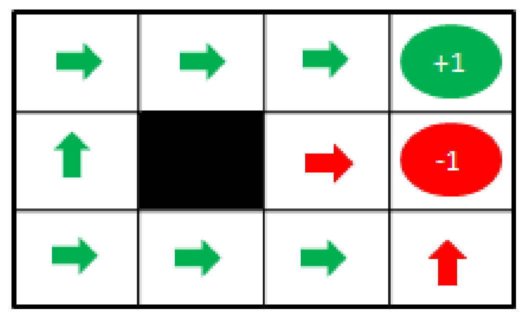
Living Reward = -0.01



Living Reward = -0.03



Living Reward = -0.04



Living Reward = -2

- Value functions
  - Assigns a value to every state
- Value of a state
  - Expected reward/return of starting at that state and following a particular policy $\pi(s)$

$$V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \left[R(s, \pi(s), s') + \gamma V^{\pi}(s') \right]$$

Value of an action in a state

- Expected reward/return of starting at a state s, taking that action a, and then following a particular policy $\pi(s)$

$$Q^{\pi}(s,a)=\sum_{s'\in S}T(s,a,s')\left[R(s,a,s')+\gamma V^{\pi}(s')\right]$$

Optimal sought after quantities

$$V^{*}(s)=\sum_{s'\in S}T\left(s,\pi^{*}(s),s'\right)\left[R\left(s,\pi^{*}(s),s'\right)+\gamma V^{\pi^{*}}(s')\right]$$

$$Q^{*}(s,a)=\sum_{s'\in S}T(s,a,s')\left[R(s,a,s')+\gamma V^{*}(s')\right]$$

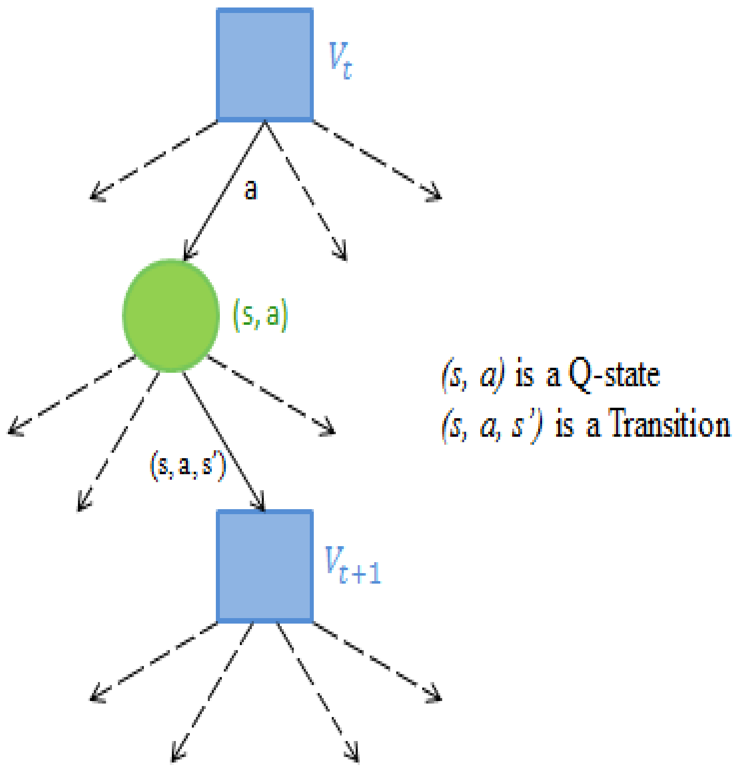
$$\pi^{*}(s)=\arg\max_{a}Q^{*}(s,a)$$

Value Iteration

- The value of a state at time t is computed to be

$$V_{t+1}(s) = \max_{a} \sum_{s' \in S} T(s,a,s')\left[r(s,a,s') + \gamma V_t(s')\right]$$

where $V_0(s) = 0$

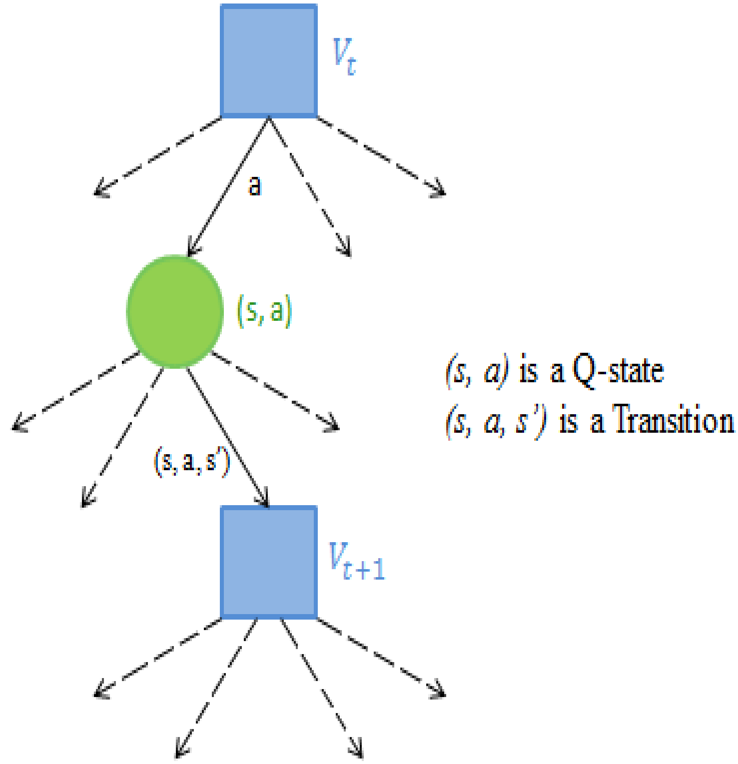


Value Iteration

- The value of a state at time t for a policy π is computed to be

$$V_{t+1}^{\pi}(s) = \max_{a} \sum_{s' \in S} T(s, \pi(s), s')\left[r(s, \pi(s), s') + \gamma V_t^{\pi}(s')\right]$$

where $V_0(s) = 0$



Reinforcement Learning

- Uses a trial and error approach to finding a policy

- Agent Learns from experience

- Q-Learning
  - Model free algorithm
  - Exploration vs Exploitation
  - Learns an optimal policy

- Q-Learning
  - Agent acts randomly in domain with probability $1 - \varepsilon$
  - Exploits current policy with probability ε

Algorithm ε-greedy Q-learning

Initialise $Q(s,a)$ arbitrarily
for every episode $k = 1 \dots K$ **do**
    Choose initial state s
    repeat
        $$a \leftarrow \begin{cases} \arg\max_a Q(s,a) & \text{w.p. } 1-\epsilon \\ a \in A & \text{w.p. } \epsilon \end{cases}$$
        Take action a, observe r, s'
        $Q(s,a) \leftarrow Q(s,a) + \alpha^Q \left[r(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$
        $s \leftarrow s'$
    until s is terminal
end for
return $Q(s,a)$

Action Priors

- Their purpose is to provide knowledge about which actions are rational in particular scenarios
- This knowledge is established by considering the statistics of action choices over the lifetime of the agent
- Correspond to general common sense behaviours

- Perception-based action priors depend on the agent's sensory features

- Acquiring Perception based Action Priors
  - Gathered by solving many tasks in the same or similar domain
  - Maintains α-counts which are dependent on observations

$$\alpha_{o(s)}^{new}(a) \leftarrow \begin{cases} \alpha_{o(s)}(a) + 1 & \text{if } a = \arg\max_a Q^{new}(s, a) \\ \alpha_{o(s)}(a) & \text{otherwise} \end{cases}$$

  - Action prior is $\theta_o(a) \sim \mathrm{Dir}(\alpha_o(a))$

Partially Observable Markov Decision Processes

- Defined as a tuple $(S,\ A,\ O\ ,\ T,\ \ Z,\ R,\ \gamma)$ where
  - S *states*,
  - A actions,
  - O observations,
  - $T: S \times A \times S \rightarrow [0,1]$,
  - $Z: S \times A \times O \rightarrow [0,1]$,
  - $R: S \times A \times S \rightarrow \mathbb{R}$,
  - $\gamma \in [0,1]$
- States are partially observable

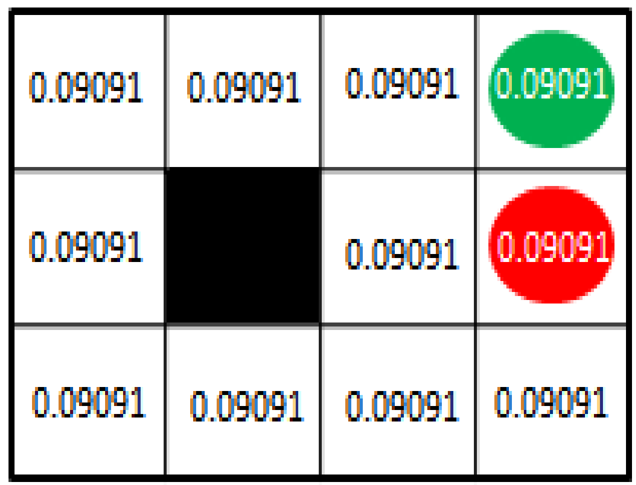
- Solving a POMDP is very similar to solving an MDP

- Similarities
  - State transitions are still stochastic
  - Value function is a function of our current state
  - We still perform Bellman backups to compute V

- Differences
  - Agent maintains a probability distribution of where it may be in space
  - Agent can make (stochastic) observations from its current belief

- Belief State
  - Probability distribution over world states

- Example
  - Uniform Belief state



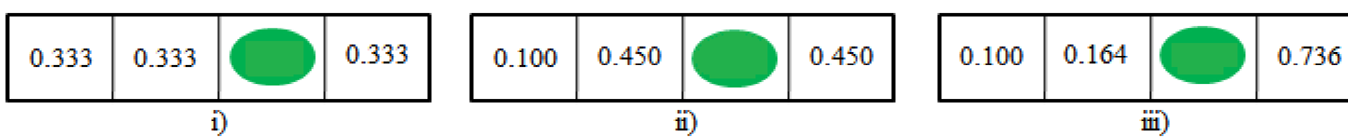
Belief transitions

- After taking action a and observing o, transitions are computed using

$$b'(s') = \frac{Z(s',a,o)\sum_{s\in S} T(s,a,s')b(s)}{P(o\mid b,a)}$$

known as the Belief Update formula

Example



Belief over states.

Rewards

- The reward of taking an action from some belief is the reward function over the belief state distribution

$$r(b,a)=\sum_{s\in S} R(s,a,)b(s)$$

Value Functions

- The value of a belief is computed using

$$V(b)=\sum_{s\in S} V(s)b(s)$$

Value Functions

- We can express

$$V(b)=\sum_{s\in S}V(s)b(s)$$

more compactly using

$$V(b)=\alpha\cdot b$$

- Where

$$b=\langle P(s_1\,|\,b),P(s_2\,|\,b),\ldots,P(s_n\,|\,b)\rangle$$

$$\alpha=\langle V(s_1),V(s_2),\ldots,V(s_n)\rangle$$

is called an *alpha-vector*

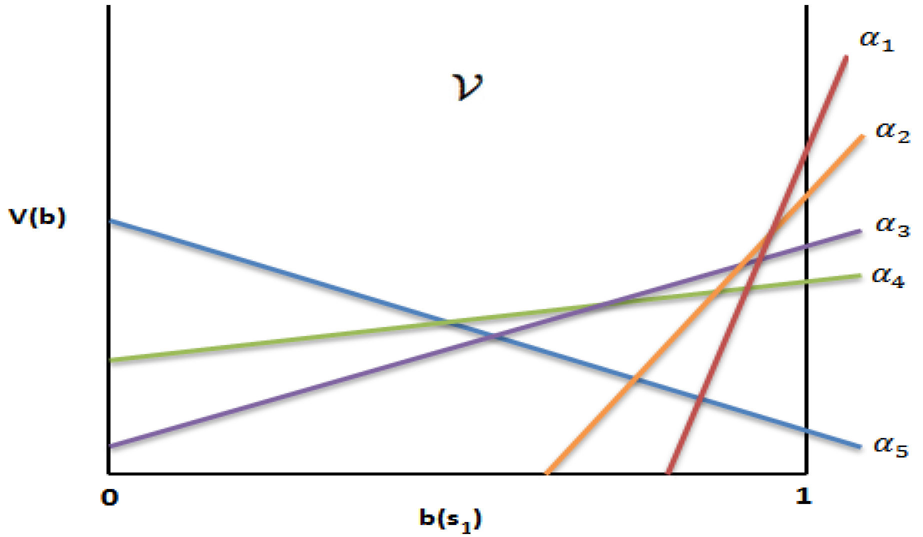
Value Functions

- The optimal value of a belief is computed using

$$V^*(b) = \max_\alpha(\alpha \cdot b)$$

- The value function of a POMDP can be represented using linear line segments representing alpha-vectors

Example



Value Iteration

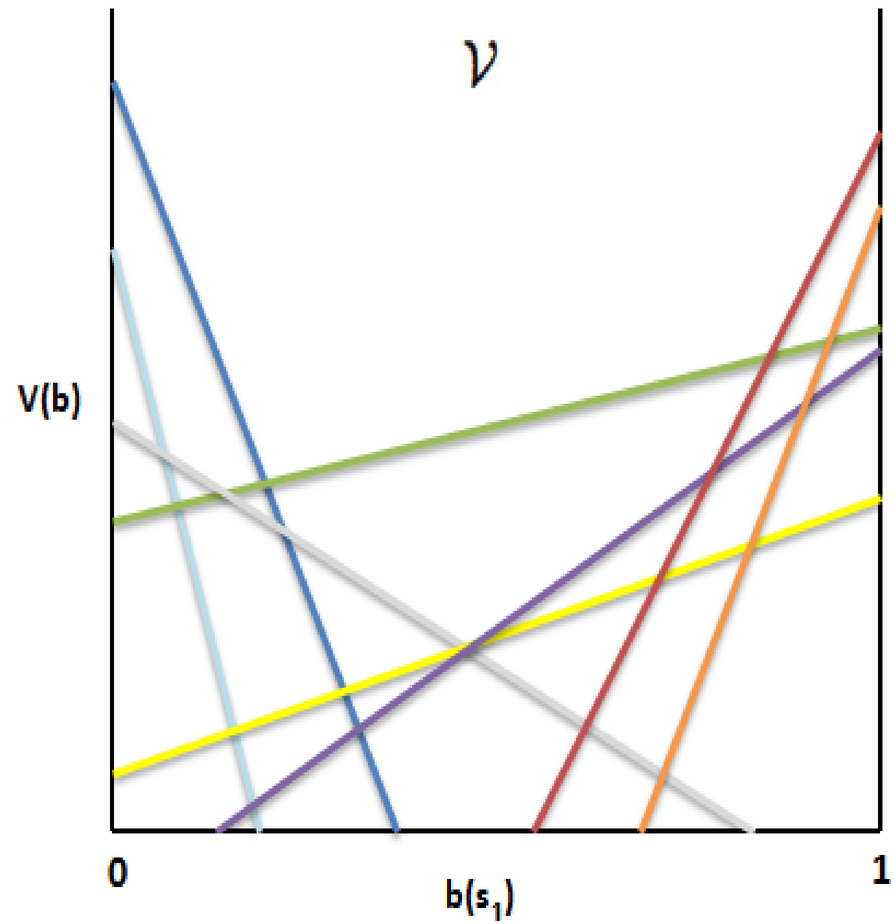
- The value of a belief at time t is computed to be

$$V_{t+1}(b) = \max_a \left[\sum_{s \in S} R(s,a) b(s) + \sum_{o \in O} P(o \mid b, a) \gamma V_t(b') \right]$$

- Here we compute $\mathcal{V}_{t+1}$, the parsimonious representation of V_{t+1} from $\mathcal{V}_t$, the parsimonious representation of V_t

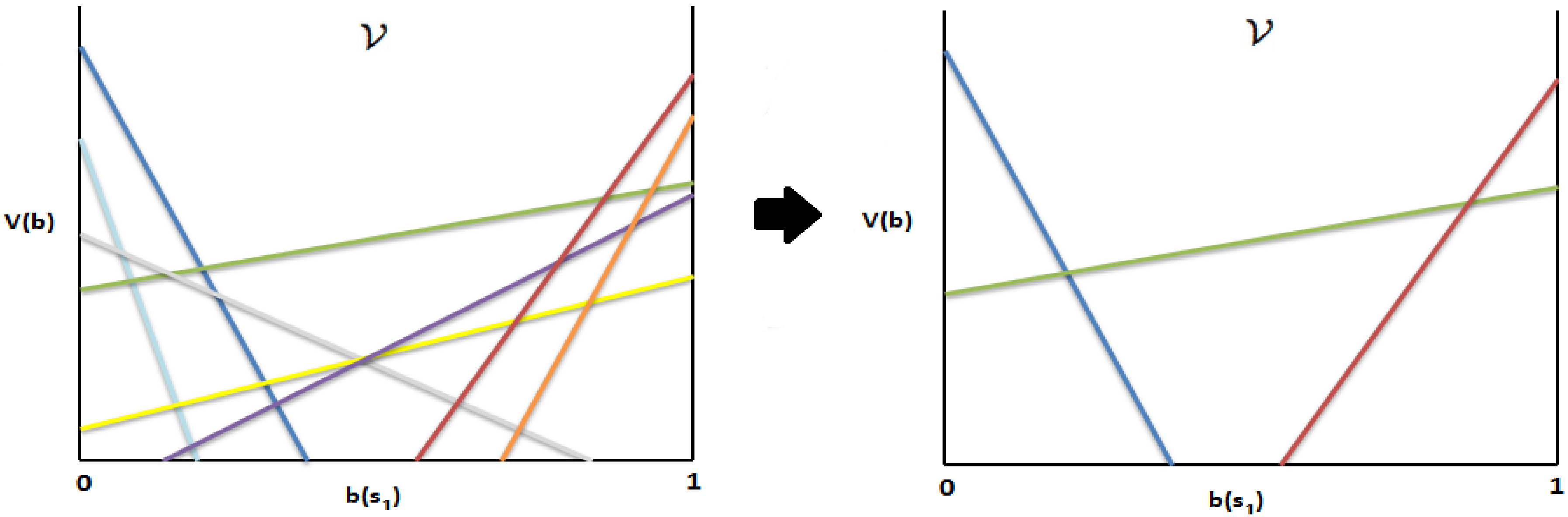
Value Functions

- The most naive way to construct $\mathcal{V}_{t+1}$ is by enumerating all possible actions and observation mappings from $\mathcal{V}_t$.
- Thus $|\mathcal{V}_{t+1}| = |A||\mathcal{V}_t|^{|O|}$
- Curse of history and dimesionality problem make POMDPs computationally intractable
- But many α-vectors in $\mathcal{V}_t$ may be dominated by others



Value Functions

- Pruning



- Improves computational speeds

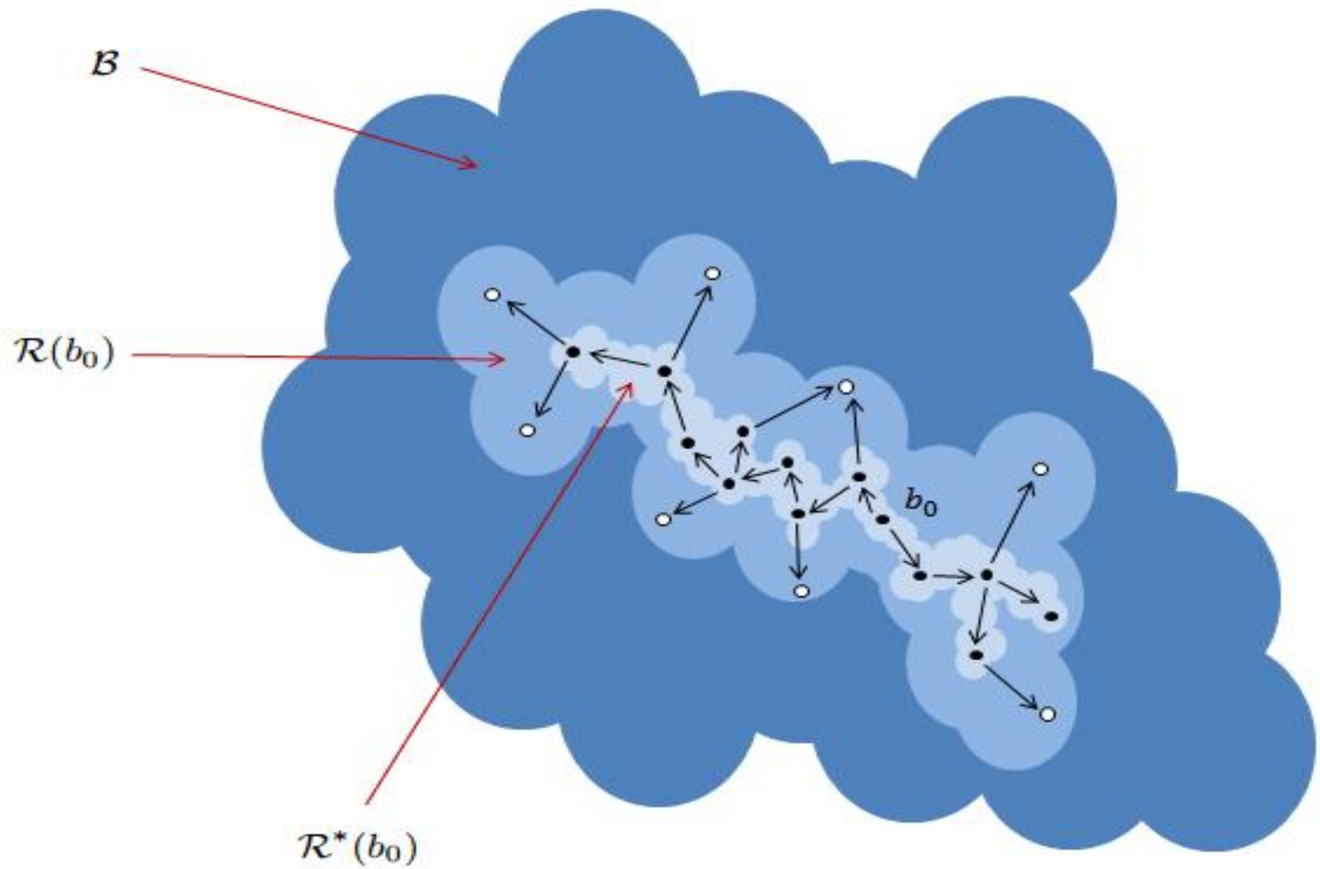
Point-based POMDPs

- Early Algorithms
  - Sample a set of beliefs B from $\mathcal{B}$ to approximate the belief space and compute an approximately optimal value function over those sampled points

- Later Algorithms
  - focus on reachable beliefs $R(b_0)$ from an initial belief point b_0

- SARSOP
  - focuses on the optimally reachable beliefs $R^*(b_0)$ from an initial belief point b_0

SARSOP Algorithm

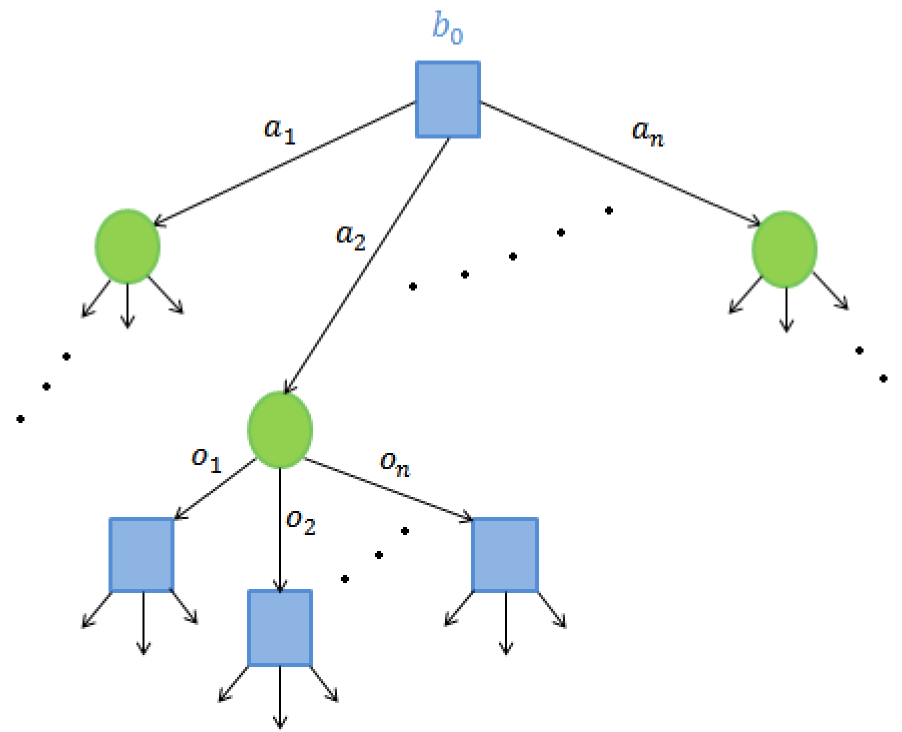


Algorithm Overview

- Successively build a tree T_R through sampling from b_0

SARSOP BASICS

```
repeat
        SAMPLE(T_R, Γ)
        α-vector BACKUP(T_R, Γ, b)
        PRUNE(T_R, Γ)
until terminate
```



Algorithm Overview

- Successively build a tree T_R through sampling from b_0

1. Sample new belief points with bias towards R^*

2. Backup the information of the children of (similarly to Bellman backup)

3. Prune dominated α-vectors and not needed nodes

4. Repeat until convergence

SARSOP BASICS

repeat
 SAMPLE(T_R, Γ)
 α-vector BACKUP(T_R, Γ, b)
 PRUNE(T_R, Γ)
until terminate

Sampling

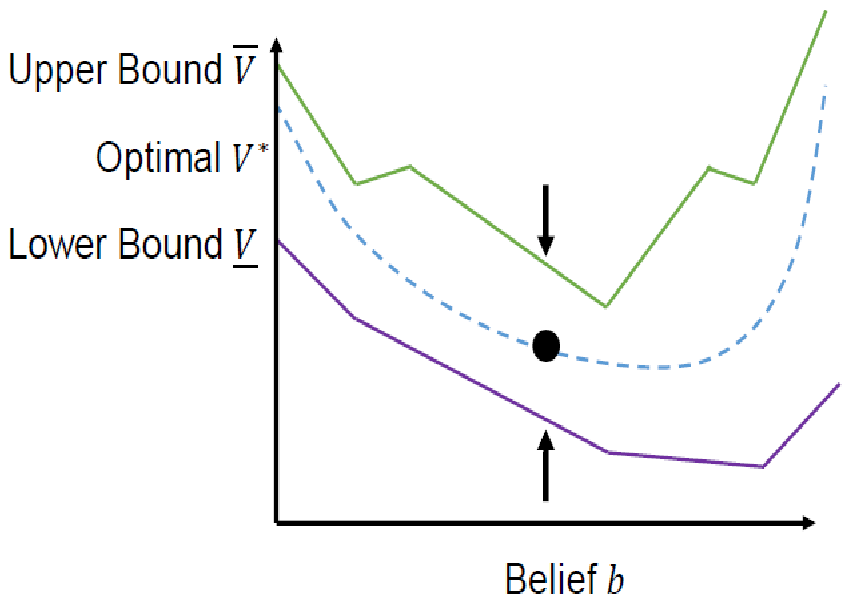
- **Upperbound** $\overline{V}$

  initialised using the MDP, FIB or Sawtooth Approximation

- **Lowerbound** $\underline{V}$

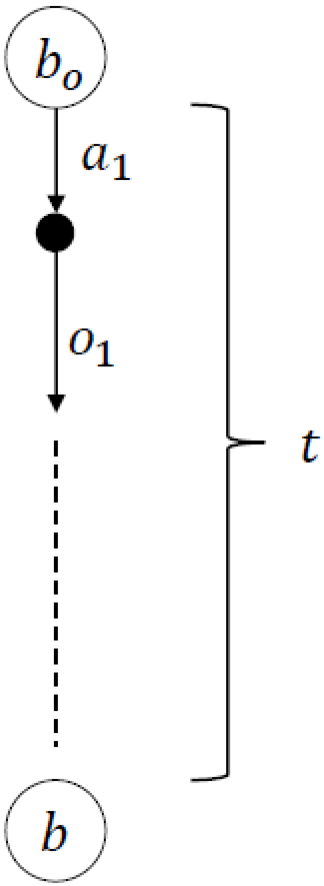
  initialised using Fixed Action Policy or Blind Policy

- Goal is too minimise the the gap between $\overline{V}$ and $\underline{V}$ at b



Sampling Strategy

- Traverse down the tree using
  - action with the highest upper bound $V(b)$
  - the observation that makes the largest contribution to the gap at the root
- Note
  - The algorithm keeps a sampling threshold of $\gamma^{-t}\epsilon$, where the *target gap size* at b_0 is ϵ



Selective Deep Sampling

- We may sometimes want to go deeper past the $\gamma^{-t}\epsilon$ threshold

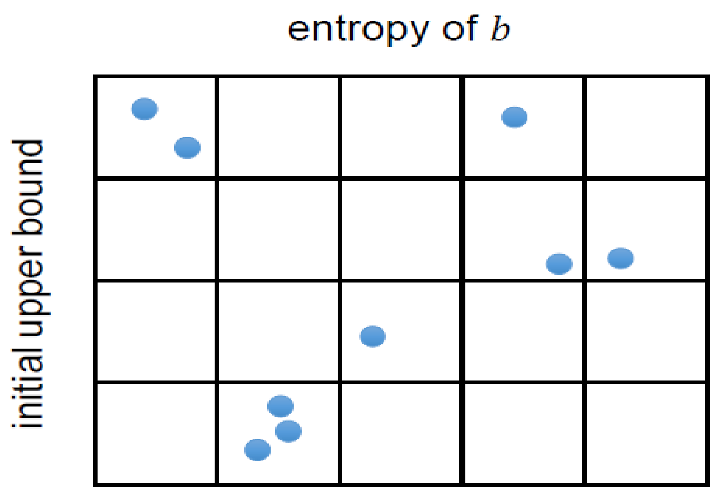
- Predict $V^*(b)$ and see if knowing it will improve the bounds at the root

  if yes **then** go deeper

  if no **then** stop

Selective Deep Sampling

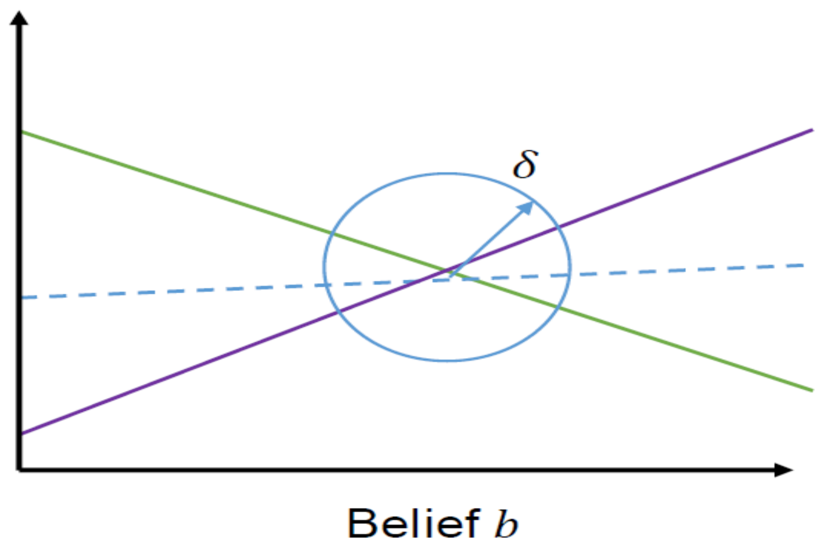
- Prediction
  - features: initial upper bound and entropy of b



  - use the average of the beliefs or the initial upper bound if bin is empty

Pruning

- Prunes α-vector only if it is dominated over R^* (SARSOP uses beliefs $B \in T_R$ as a proxy for R^*)
- If $\overline{Q}(b,a) < \underline{Q}(b,a)$, prunes all sampled points in the subtree after taking action a at b
- If $a_i \cdot b' \le a_{\neg i} \cdot b'$ for all $a_{\neg i}$ at every point b' within δ of b prune a_i



Research Project

- To show how action priors can accelerate the workings of the SARSOP algorithm through the implementation of action priors

- Gather Action Priors using Reinforcement learning and the SARSOP algorithm

- Action Priors from SARSOP
  1. Gathered from Sampling and
  2. Simulation

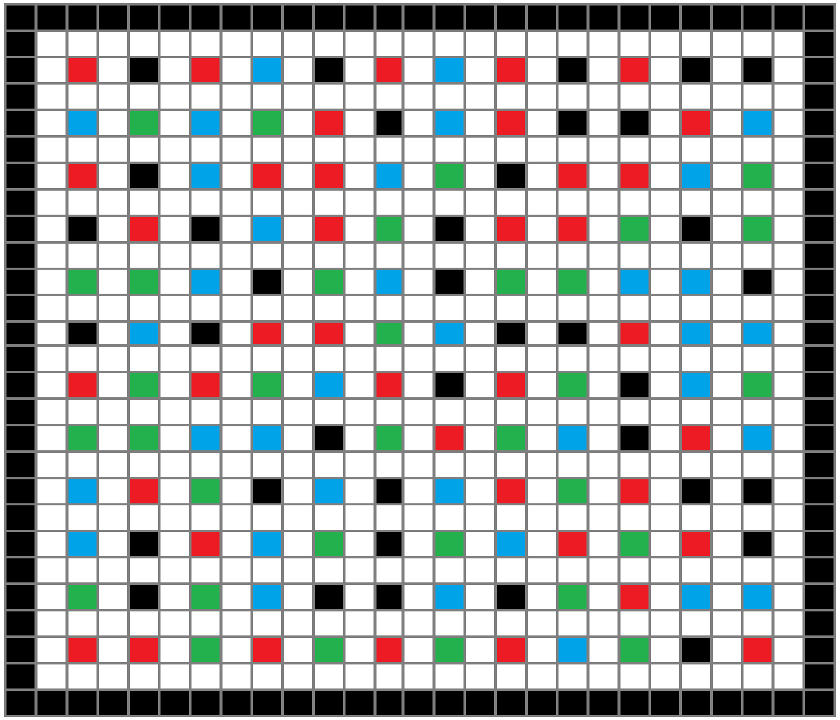
- Using the priors

  - Will be used as an add on to prune away the alpha vectors of the SARSOP algorithm

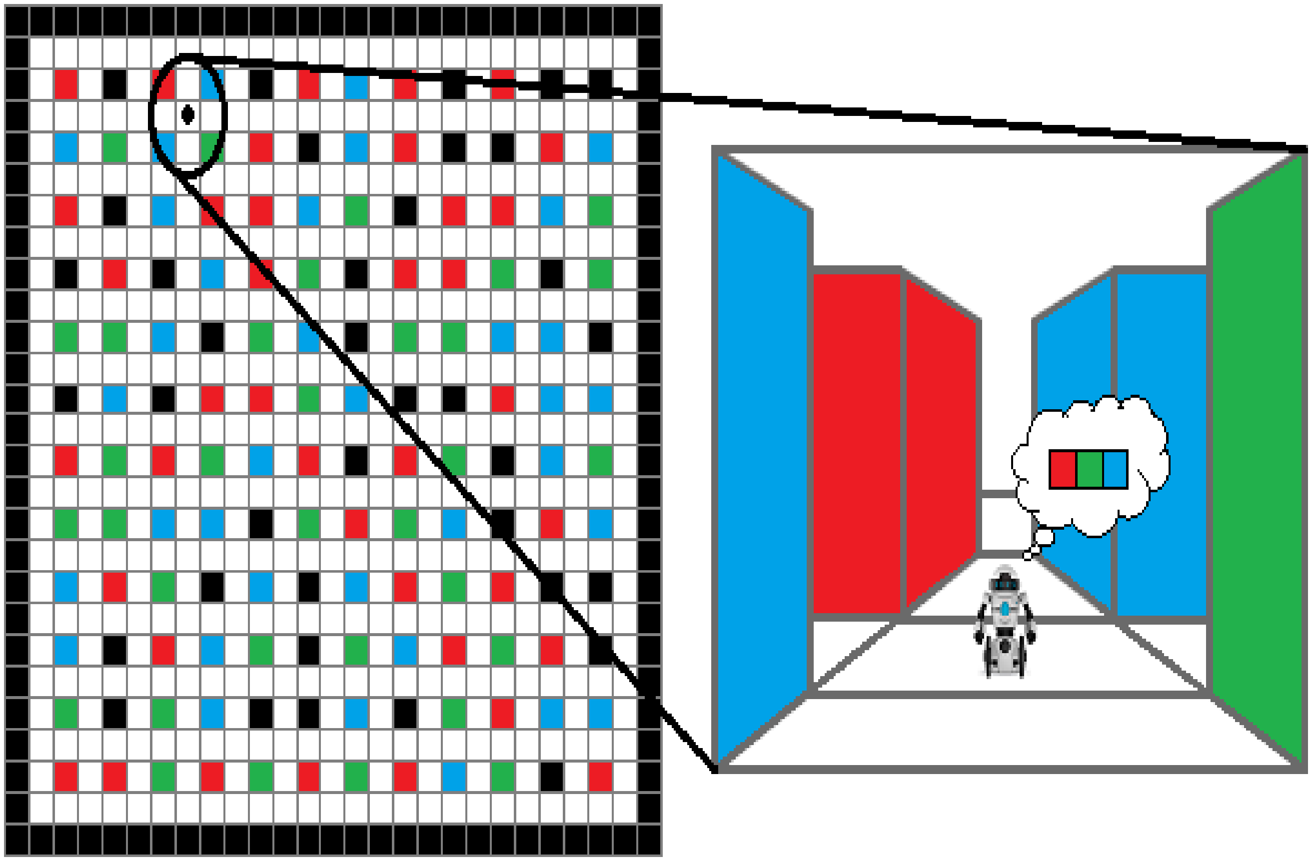
  - Will be used to choose actions in the sampling technique of SARSOP

Domain

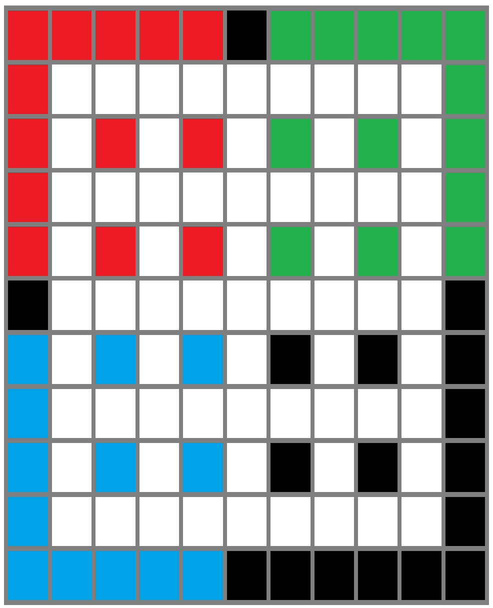
- Maze domain in which we allow a robot agent to travel from
- Tasks will be to travel from an initial location to some goal location



- Robot Perception Capability



Recent Progress



Domain

Observation	Up	Down	Left	Right
	0.01	0.49	0.01	0.49
	0.01	0.57	0.01	0.41
	0.01	0.41	0.01	0.57
	0.01	0.49	0.01	0.49
	0.02	0.70	0.02	0.25
	0.02	0.25	0.02	0.70
	0.04	0.46	0.04	0.46
	0.25	0.25	0.25	0.25
	0.01	0.42	0.01	0.55
	0.01	0.55	0.01	0.42
	0.25	0.25	0.25	0.25
	0.04	0.46	0.04	0.46
	0.04	0.46	0.04	0.46
	0.25	0.25	0.25	0.25
	0.04	0.46	0.04	0.46

Reinforcement Learning Priors

Observation	Up	Down	Left	Right
	0.00	0.52	0.00	0.48
	0.00	0.86	0.00	0.14
	0.00	0.29	0.00	0.71
	0.00	0.35	0.00	0.65
	0.00	1.00	0.00	0.00
	0.00	0.32	0.00	0.68
	0.00	0.60	0.00	0.40
	0.25	0.25	0.25	0.25
	0.00	0.85	0.00	0.15
	0.00	0.45	0.00	0.55
	0.25	0.25	0.25	0.25
	0.00	1.00	0.00	0.00
	0.00	1.00	0.00	0.00
	0.25	0.25	0.25	0.25
	0.00	1.00	0.00	0.00

SARSOP Simulation Priors