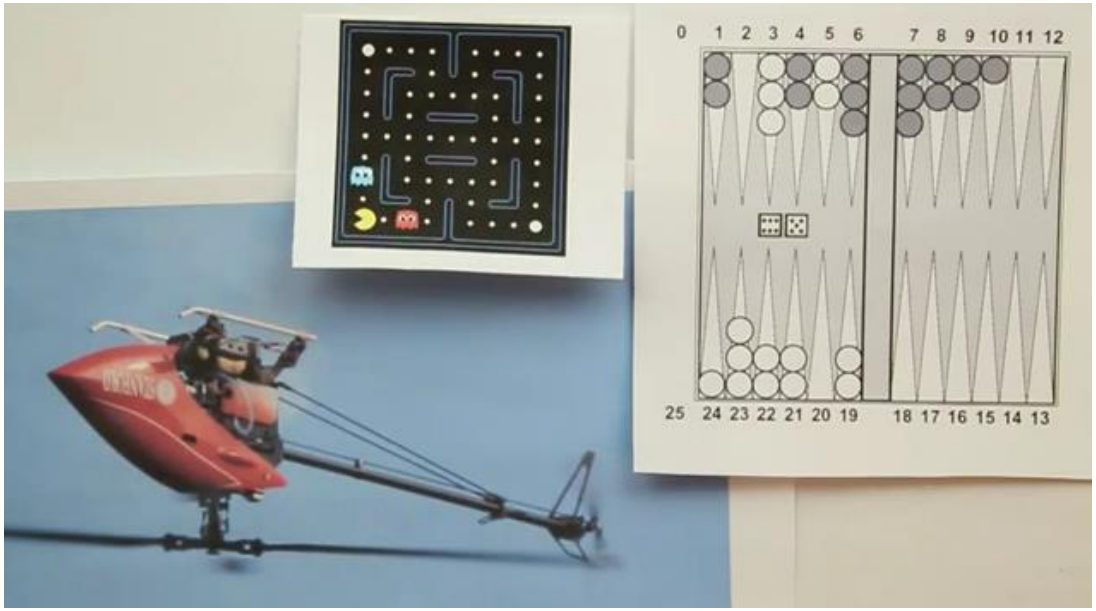
Computational Intelligence
Unit # 11-1

Reinforcement Learning



Acknowledgement

- Several examples of this lecture have been taken from Stanford AI class and Stanford Machine Learning class.

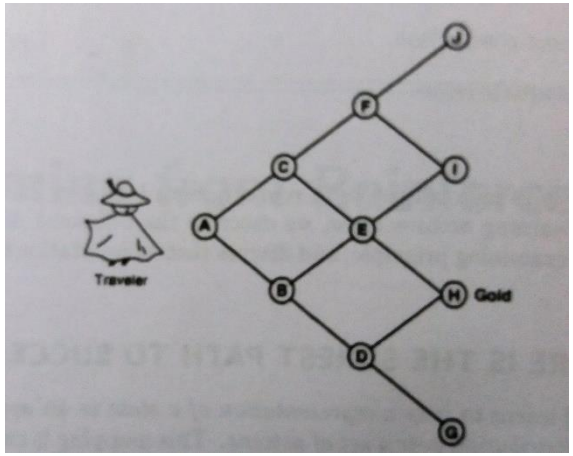
Reinforcement Learning(RL)

- An RL agent learns by interacting with its environment and observing the results of these interactions. This mimics the fundamental way in which humans (and animals alike) learn.

Jackpot Journey Problem

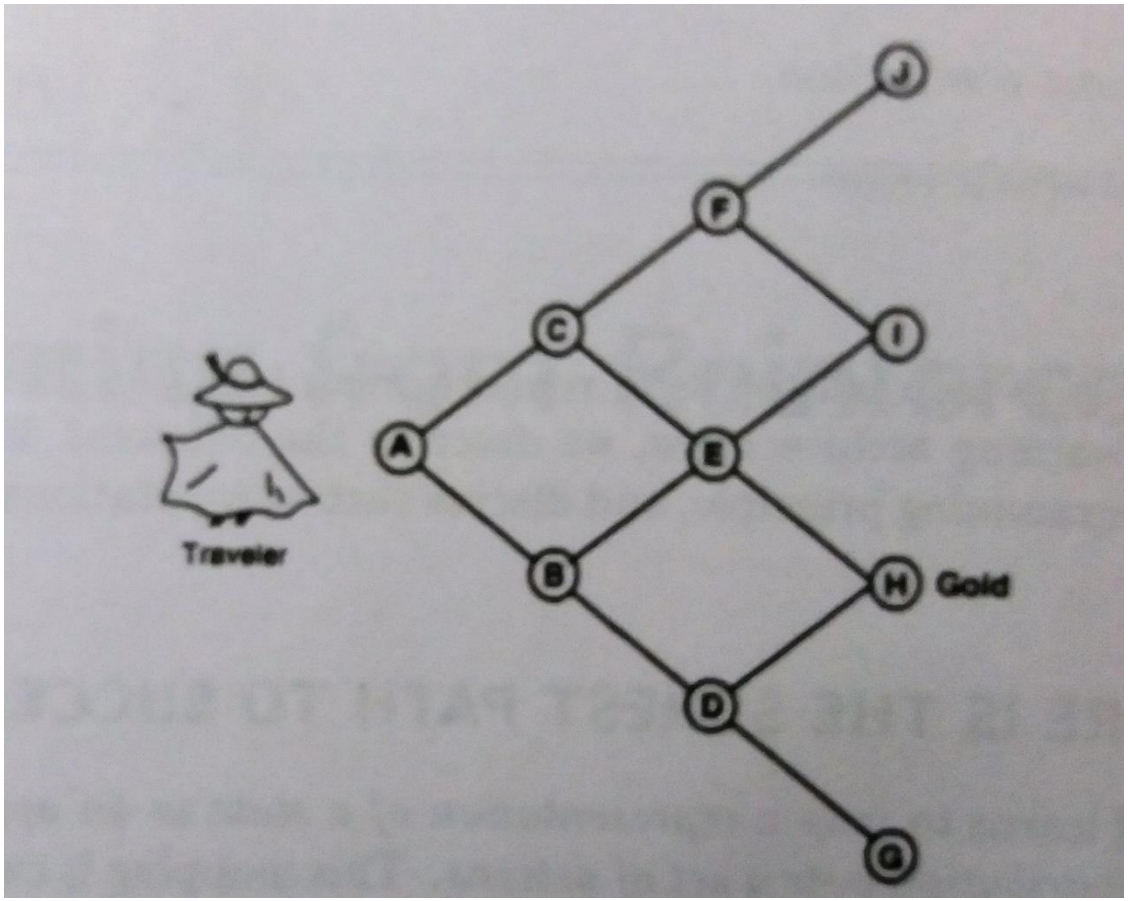
Jackpot Journey Problem

- A traveler is looking for gold which he can found at the end of his journey.
- At each vertex, there is a signpost that has a box with some white and black stones in it. A traveler picks a stone from the signpost box and follows certain instructions;
  - when a white stone is picked, go diagonally upward, denoted by action u.
  - Conversely, when a black stone is chosen, go diagonally downward, denoted by action d”



Jackpot Example

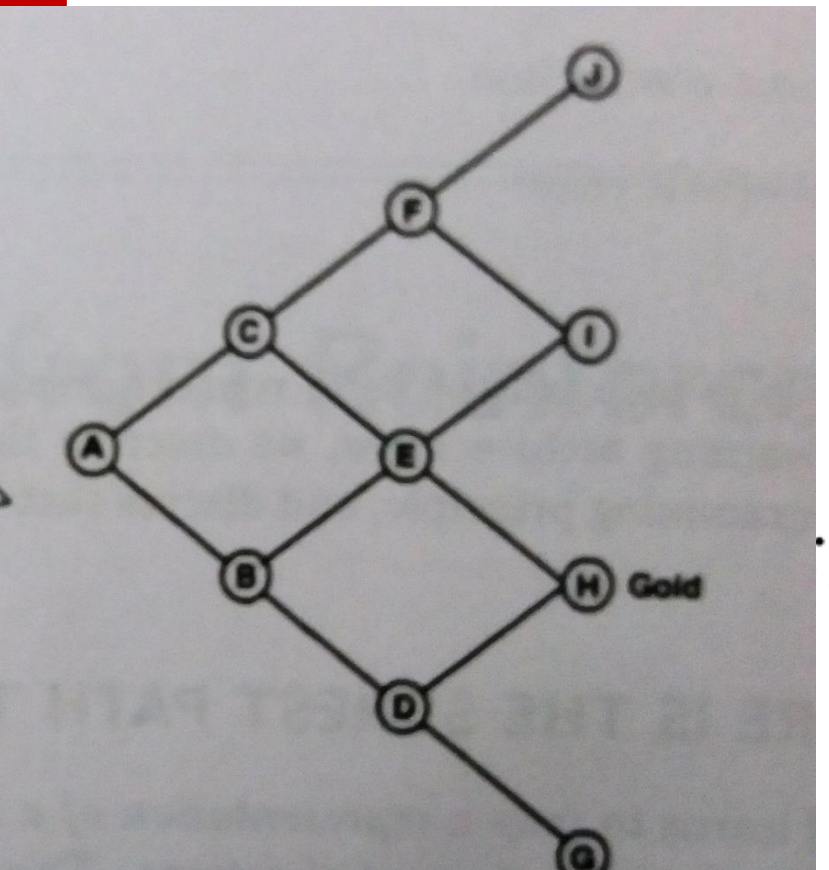
- A traveler is looking for gold which he can found at the end of his journey.

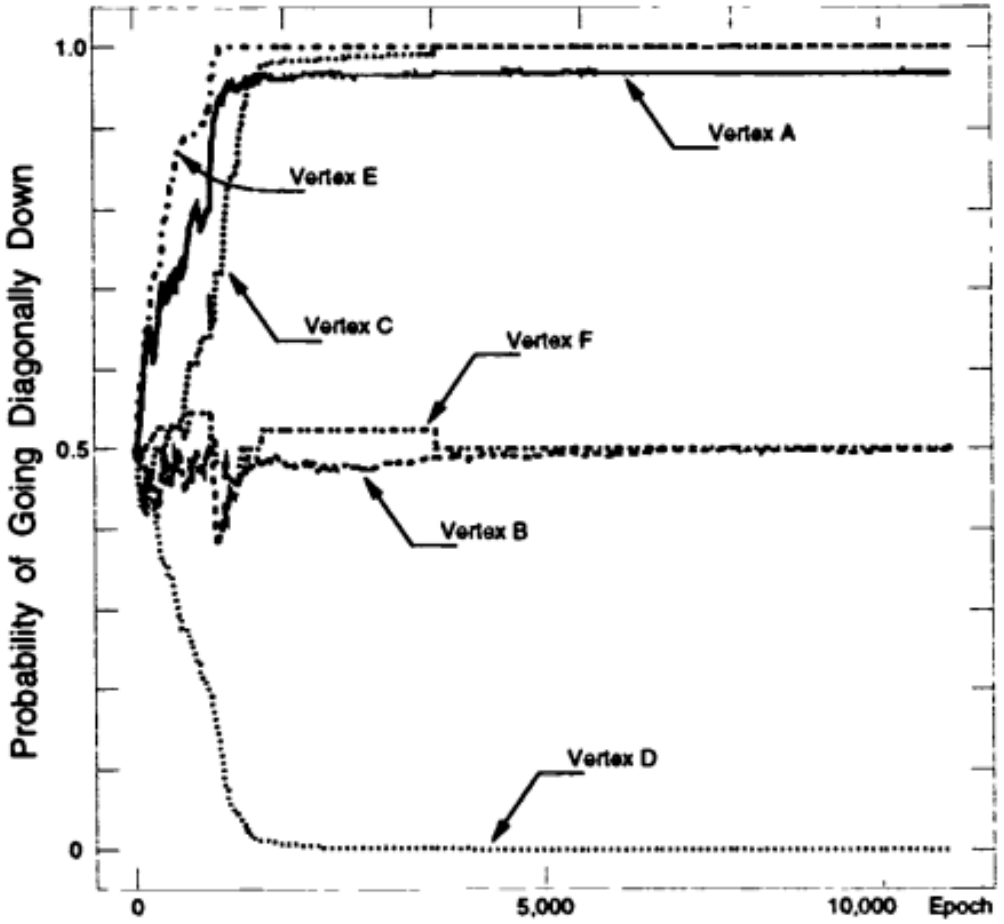


Neuro-fuzzy and Soft Computing – A computational approach to Learning and Machine Intelligence

Jackpot Journey Problem – Reward Scheme

- After concluding the journey, trace back to the starting vertex A; at each visited vertex:
  - put the placed stone back into the signpost with an additional stone of the same color (reward) if the gold was found,
  - or take the placed stone away from the signpost (penalty) if the gold was not found.
- When the traveler returns, the next traveler will have more chances to find gold!”





. *Changing probability of action d, "go diagonally down*
vertex (A–F) as trials progress in the jackpot journey

Reinforcement Learning(RL)

- The key idea can be translated into the following steps for an RL agent:
  - The agent observes an input state
  - An action is determined by a decision making function (policy)
  - The action is performed
  - The agent receives a scalar reward or reinforcement from the environment
  - Information about the reward given for that state / action pair is recorded

- By performing actions, and observing the resulting reward, the policy used to determine the best action for a state can be fine-tuned.
- Eventually, if enough states are observed an optimal decision policy will be generated and we will have an agent that performs perfectly in that particular environment.

Reinforcement Learning

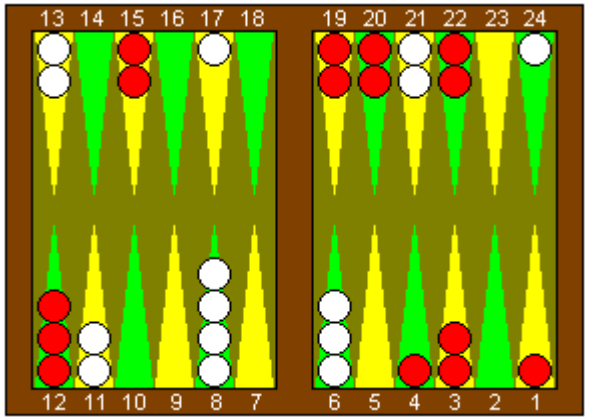
- RL is distinguished from other computational approaches by its emphasis on ***learning by the individual from direct interaction with its environment, without relying on exemplary supervision or complete models of the environment.***

Some RL Examples

- A **master chess player** makes a move. The choice is informed both by planning—anticipating possible replies and counter replies--and by immediate, intuitive judgments of the desirability of particular positions and moves.
- An **adaptive controller** adjusts parameters of a petroleum refinery's operation in real time. The controller optimizes the yield/cost/quality trade-off on the basis of specified marginal costs without sticking strictly to the set points originally suggested by engineers.
- A **gazelle calf** struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.
- A **mobile robot** decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on how quickly and easily it has been able to find the recharger in the past.

Some popular implementation of RL

Stanford autonomous helicopter
http://ai.stanford.edu/~pabbeel/RL-videos.html



TD Gammon – A RL agent of Backgammon
http://www.research.ibm.com/massive/tdl.html

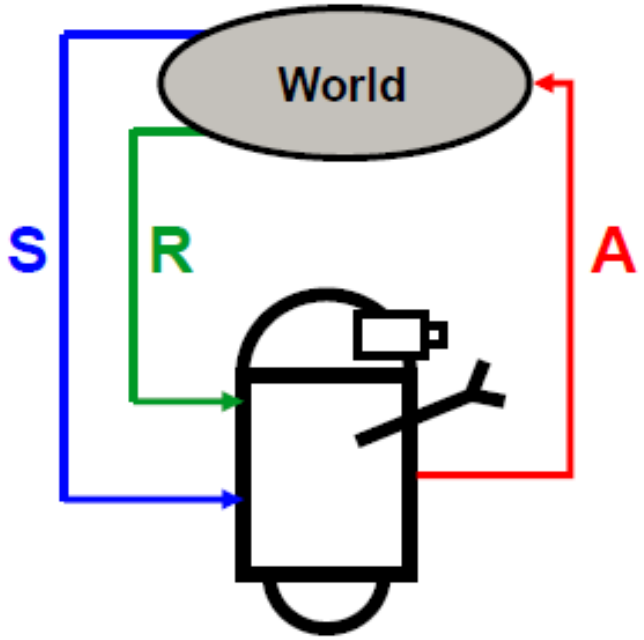
Video

Google's DeepMind AI Just Taught Itself To Walk

https://www.youtube.com/watch?v=gn4nRCC9TwQ

Basic RL Model

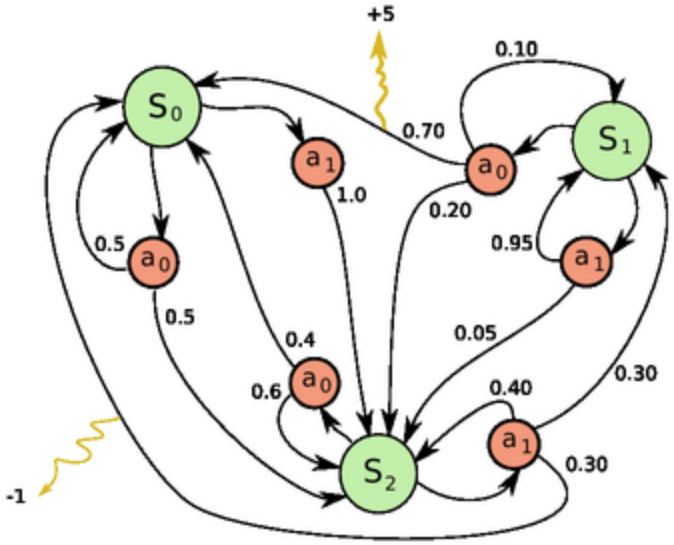
1. Observe state, s_t
2. Decide on an action, a_t
3. Perform action
4. Observe new state, s_{t+1}
5. Observe reward, r_{t+1}
6. Learn from experience
7. Repeat

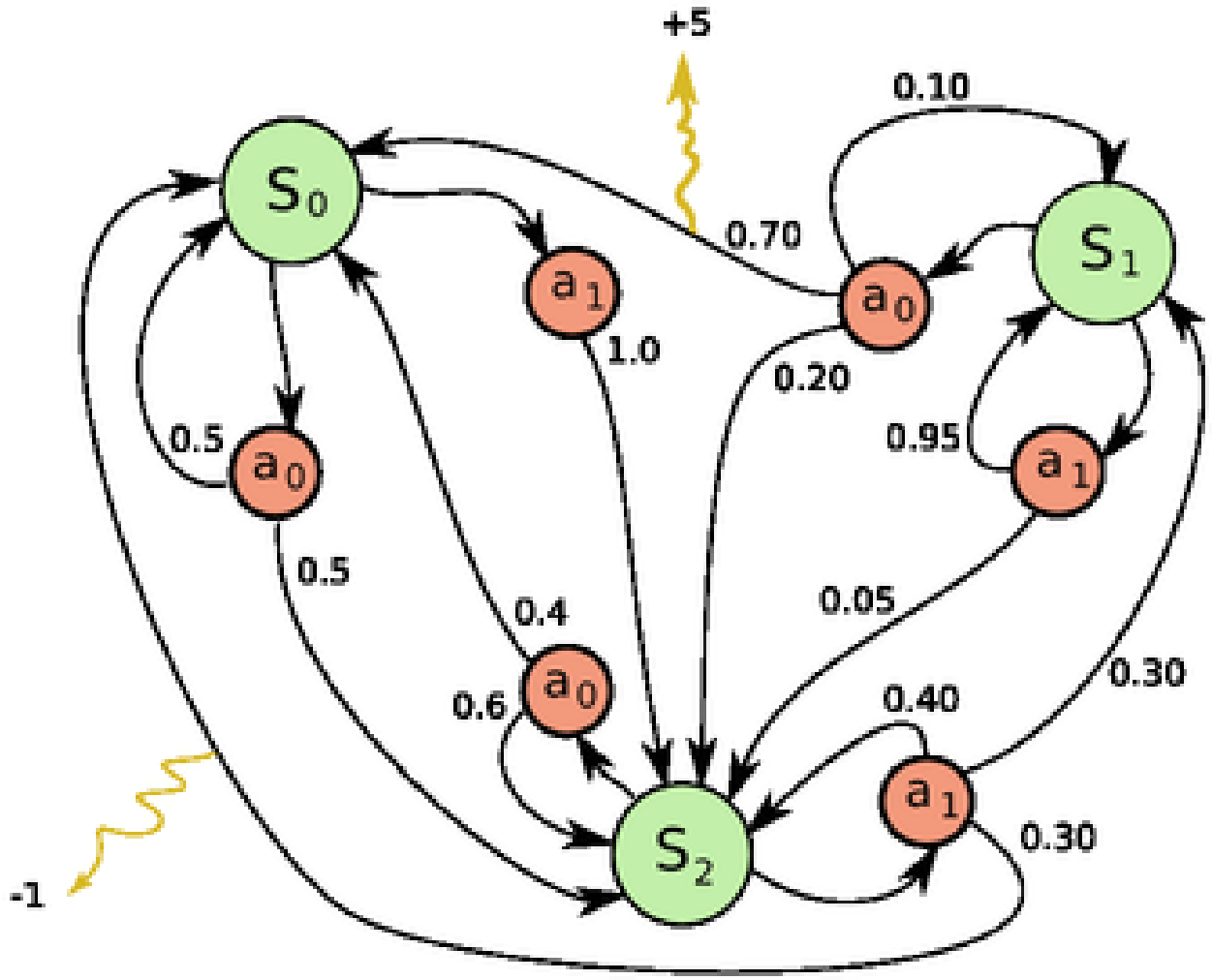


Goal: Find a control policy that will maximize the observed rewards over the lifetime of the agent

Markov Decision Process (MDP)

- RL models the world using Markov Decision Processes (MDPs).
- MDP provides a mathematical framework for modeling decision-making in stochastic situations.





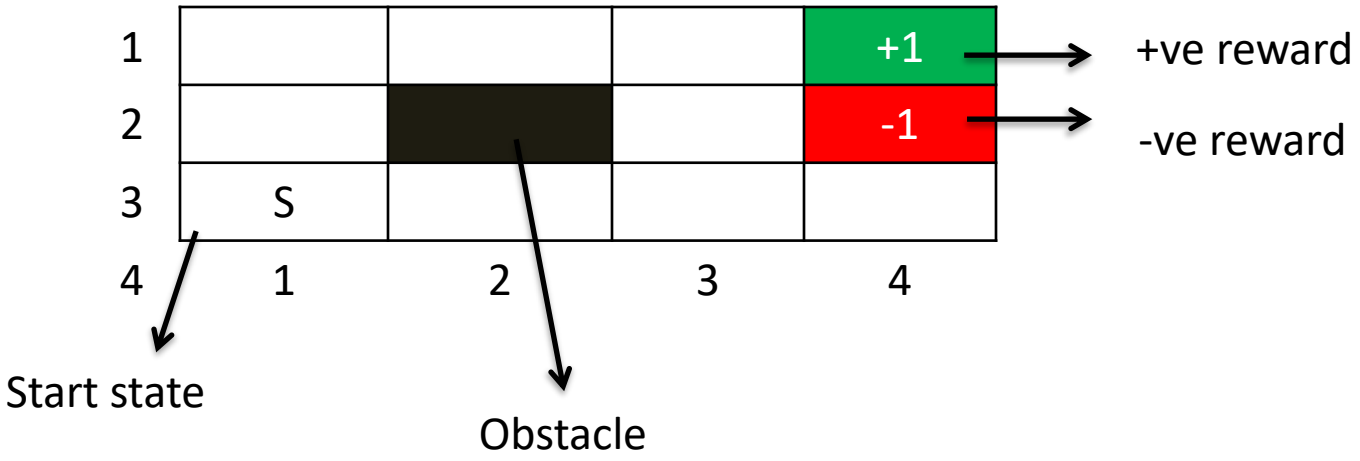
https://becomesentient.com/markove-decision-processes/

Markov Decision Process (MDP)

- A Markov decision process is a 4-tuple (S, A, P(.,.),R(.,.)) , where
  - S is a finite set of states,
  - A is a finite set of actions (alternatively, is the finite set of actions available from state),
  - P_a(s,s') is the probability that action 'a' in state 's' at time 't' will lead to state s' at time t+1
  - R(s') is the expected immediate reward received after transition to state s'.

- **Objective**: The core problem of MDPs is to find a *policy* for the decision maker: a function that specifies the action that the decision maker will choose when in state .

GridWorld Example



The objective is to find a policy that navigates the agent from the start state to the Goal state while resulting in maximum +ve reward.

Elements of RL

- One can identify four main sub elements of a reinforcement learning system:
  - a policy,
  - optionally, a model of the environment,
  - a reward function,
  - a value function

Policy

- A policy is a mapping from perceived states of the environment to actions to be taken when in those states.

Model

- This is something that mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and next reward.
- RL models the environment in the form of MDPs.

Reward Function

- A reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state.

Value Function

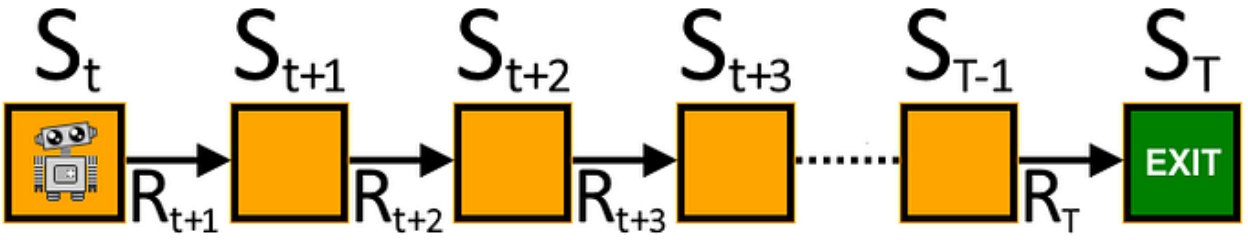
- Roughly speaking, the value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- Whereas a reward function indicates what is good in an immediate sense, a value function specifies what is good in the long run.

Reward vs Values

- Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward.
- Nevertheless, it is values with which we are most concerned when making and evaluating decisions. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run.
- In fact, **the most important component of almost all reinforcement learning algorithms is a method for efficiently estimating values.**

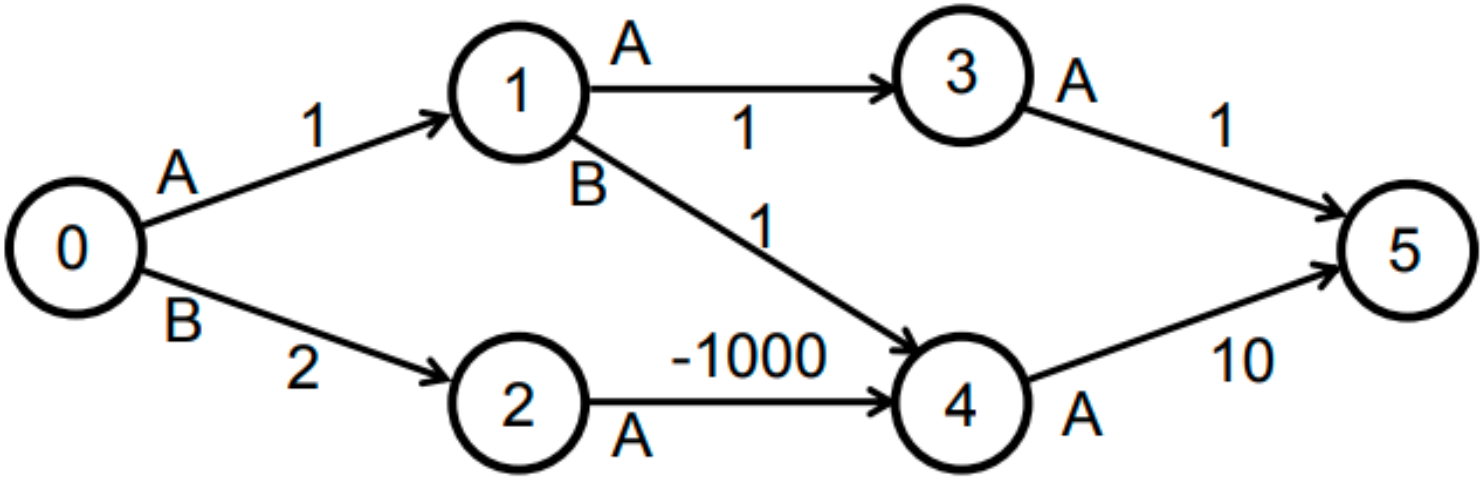
Cumulative Future Reward

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$



$$G_t = \sum_{k=0}^{T} R_{t+k+1}$$

G_t is the total future rewards from time step *t.*

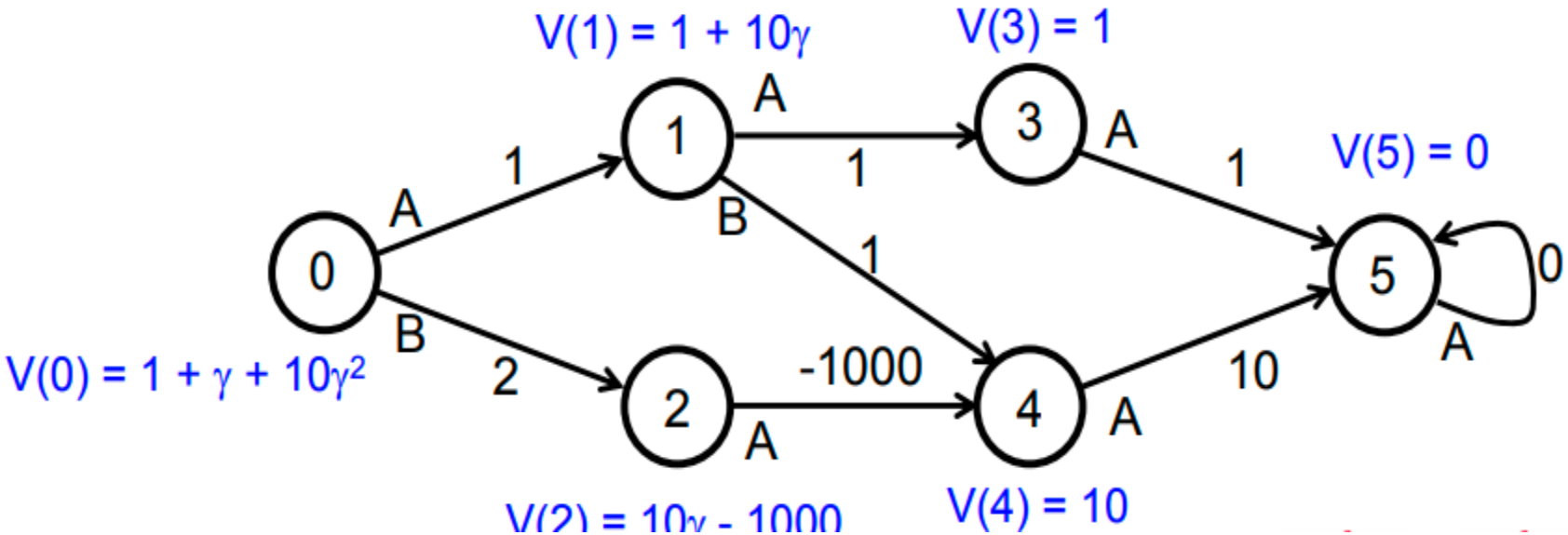

0
1
2
3
4
5
A
1
B
2
A
1
B
1
A
1
-1000
A
A
10

Time value of Reward

- However, in reality, we can't just add the rewards like that. The rewards that come sooner are more probable to happen, since they are more predictable than the long term future reward.

Discounting

- We define a discount rate called gamma. It must be between 0 and 1.
- The larger the gamma, the smaller the discount. This means the learning agent cares more about the long term reward.
- On the other hand, the smaller the gamma, the bigger the discount. This means our agent cares more about the short term reward.


V(1) = 1 + 10γ
V(3) = 1
V(5) = 0
V(0) = 1 + γ + 10γ²
V(2) = 10γ - 1000
V(4) = 10
0
1
2
3
4
5
A
B
1
2
-1000
10
0

Discounted Cumulative Reward

$$R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \ldots$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} where \gamma \in [0, 1)$$

Value Iteration

- Let the path taken by an RL agent to reach the goal state is:

$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow \ldots\ldots$

$\text{Value}(S_0) = R(S_0) + \gamma R(S_1) + \gamma^2 R(S_2) + \gamma^3 R(S_3) + \ldots\ldots$

$\text{Value}(S_0) = R(S_0) + \gamma V(S_1)$

Since state transitions are stochastic, we have

$\text{Value}(S_0) = R(S_0) + \gamma E[\, V(S_1) \,]$

Where E represents 'expected value' .

Value

- *Value* is the prediction of a series of *rewards*, and is calculated as the expected sum of discounted rewards.

Value Based Agent

- The agent will evaluate all the states in the state space, and the policy will be kind of implicit, i.e. the value function tells the agent how good is each action in a particular state and the agent will choose the best one.

Expected Cumulative Future Reward

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

$$v(s) = \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

$$v(s) = \sum_{s',r} p(s', r \mid s)\left[r + \gamma v(s')\right]$$

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma v_\pi(s')\right]$$

Bellman Equation

The Bellman equation is defined recursively as:

$$V(s) = \max_{a \in A(s)} \sum_{s' \in S} P_a(s' \mid s) \left[r(s, a, s') + \gamma V(s') \right]$$

Let's break it down,

$$V(s) = \overbrace{\max_{a \in A(s)}}^{\text{best action from } s} \overbrace{\underbrace{\sum_{s' \in S}}_{\text{for every state}} P_a(s' \mid s) \left[\underbrace{r(s, a, s')}_{\text{immediate reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{V(s')}_{\text{value of } s'} \right]}^{\text{expected reward of executing action } a \text{ in state } s}$$

Value Iteration

Initialize V arbitrarily, e.g., $V(s) = 0$, for all $s \in \mathcal{S}^+$

Repeat
$\Delta \leftarrow 0$
For each $s \in \mathcal{S}$:
$v \leftarrow V(s)$
$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$
$\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)

Output a deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s')\right]$

Bellman Equation

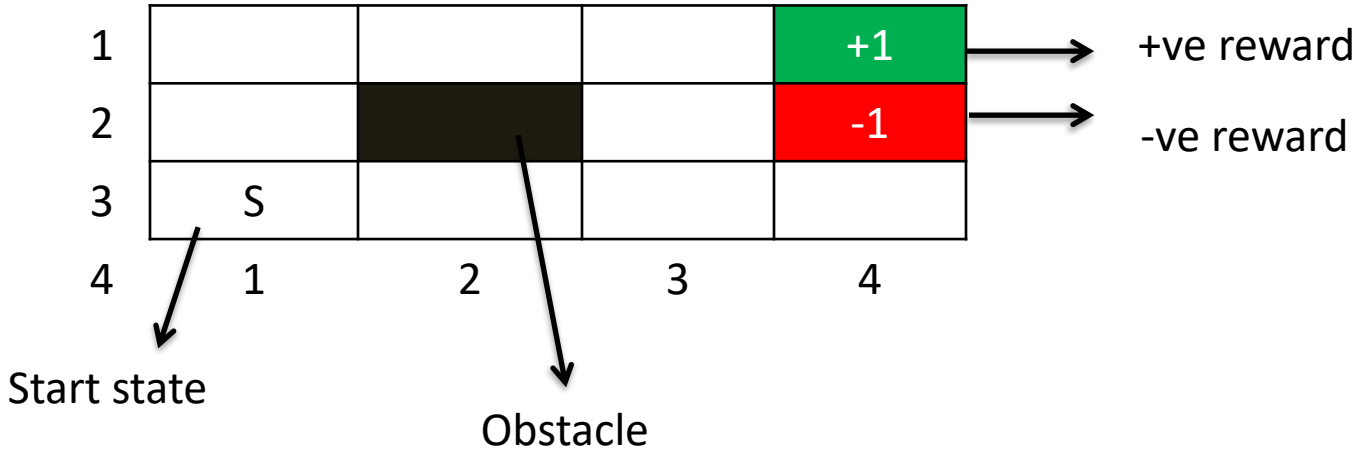
The Bellman equation is defined recursively as:

$$V(s) = \max_{a \in A(s)} \sum_{s' \in S} P_a(s' \mid s) \left[r(s, a, s') + \gamma V(s') \right]$$

Let's break it down,

$$V(s) = \overbrace{\max_{a \in A(s)}}^{\text{best action from } s} \overbrace{\underbrace{\sum_{s' \in S}}_{\text{for every state}} P_a(s' \mid s) \left[\underbrace{r(s, a, s')}_{\text{immediate reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{V(s')}_{\text{value of } s'} \right]}^{\text{expected reward of executing action } a \text{ in state } s}$$

Value Iteration



$$V(s) \leftarrow \max_a \sum_{s'} \mathcal{P}^a_{ss'} \left[\mathcal{R}^a_{ss'} + \gamma V(s') \right]$$

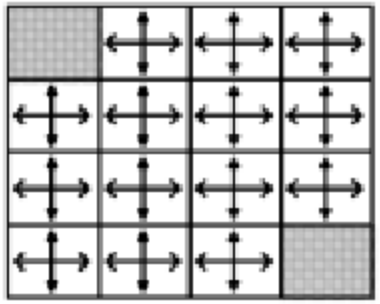
RL Implementation Demo

Value Iteration

V_k for the Random Policy

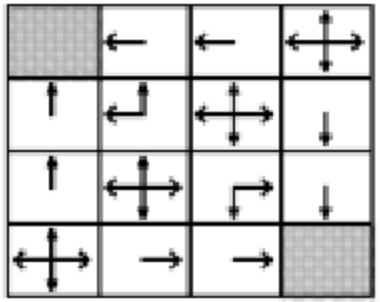
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Greedy Policy w.r.t. V_k

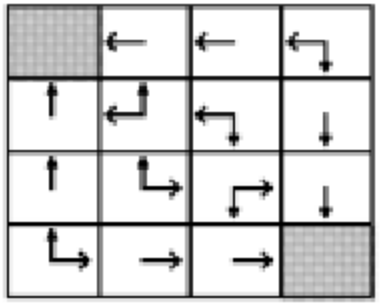


← random policy

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



0.0	-2.4	-2.9	-3.0
-2.4	-2.9	-3.0	-2.9
-2.9	-3.0	-2.9	-2.4
-3.0	-2.9	-2.4	0.0



← optimal policy

Baby Robot Example

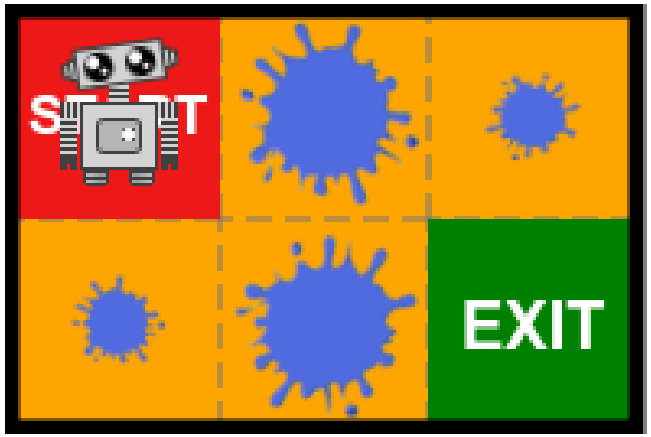
Markov Decision Processes and Bellman Equations | by Steve Roberts | Towards Data Science

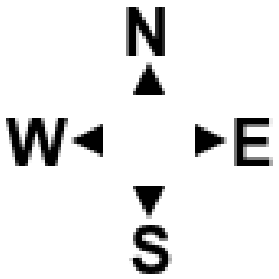
Baby Robot

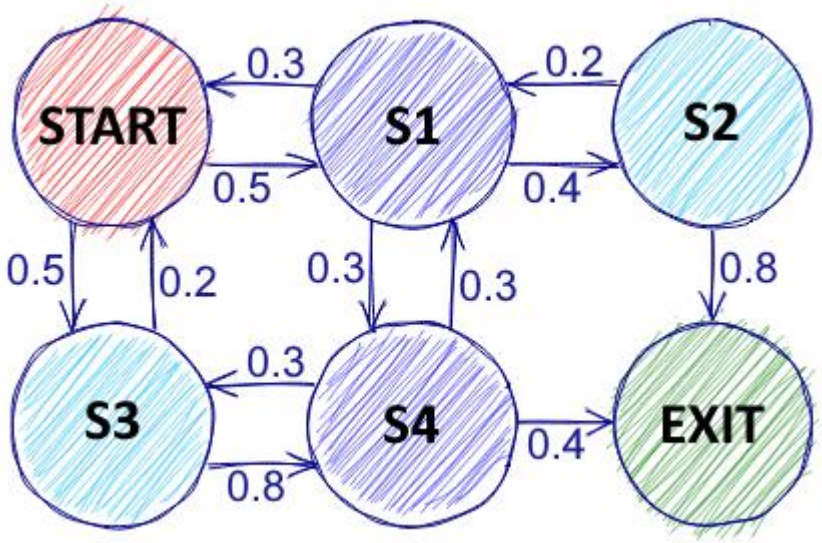
- Baby Robot has found himself in a very small room, so escaping from this should not be too much of a challenge. The only downside is that the roof has been leaking and Baby Robot does not like the wet. Puddles slow him down and potentially cause him to skid.
- 

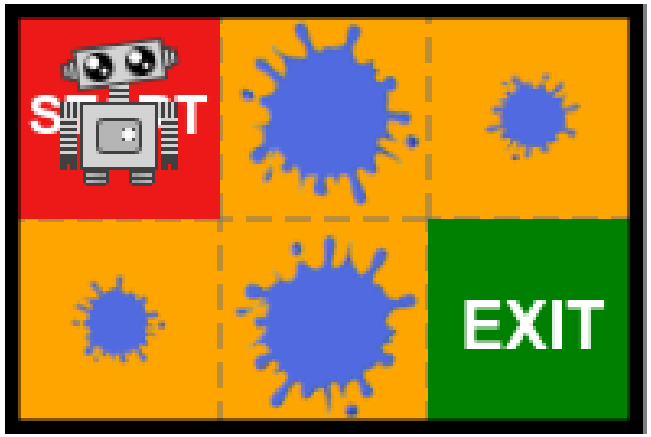
Baby Robot

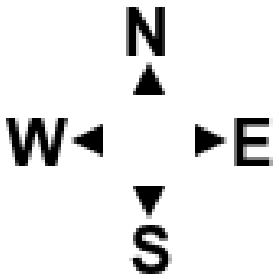
- Small puddles have a 20% chance of skidding, so 80% of the time the chosen direction will be taken.
- Big puddles have a 60% chance of skidding, so the chosen direction will only be taken 40% of the time.

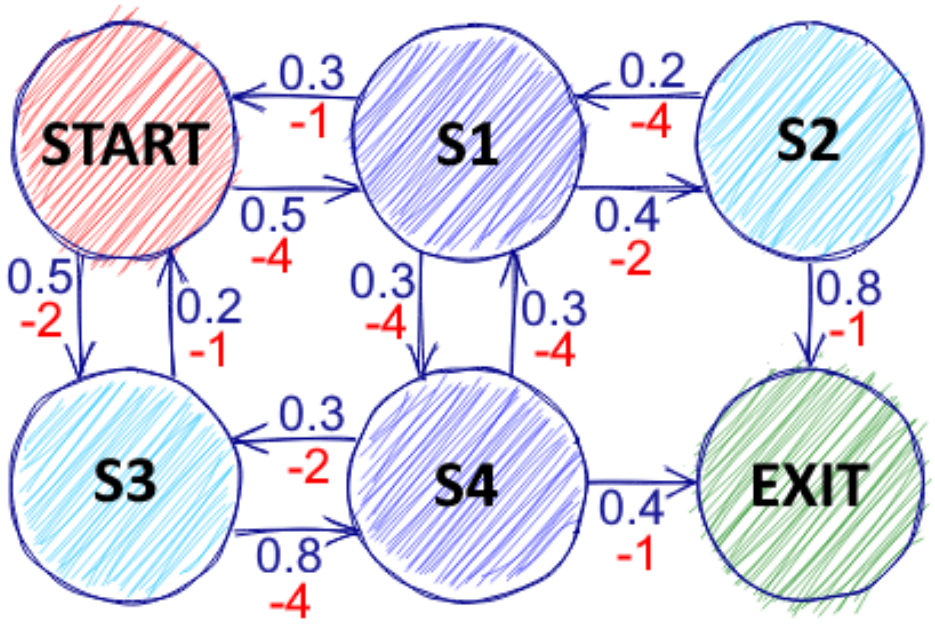

EXIT


N
W
E
S

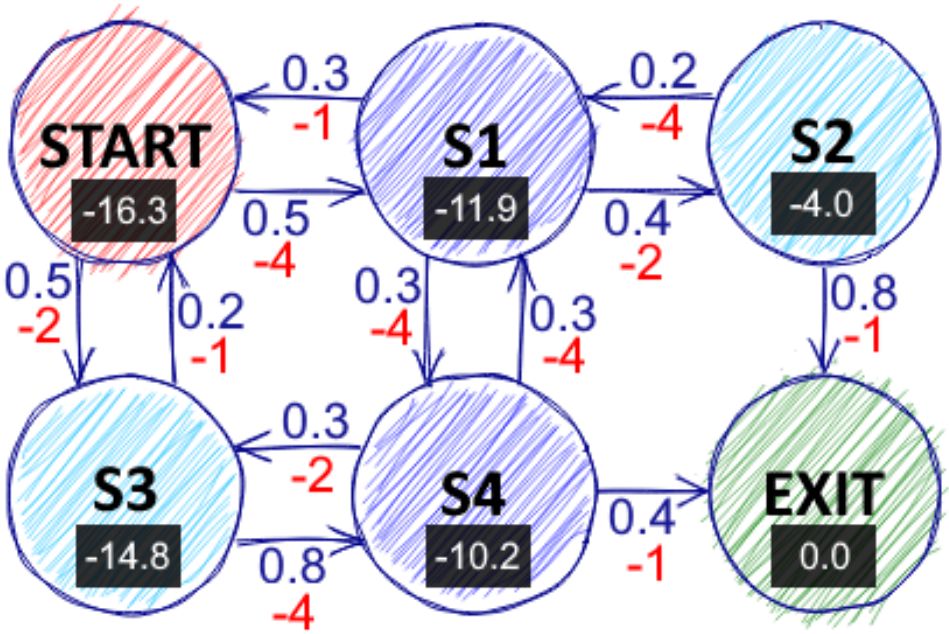

START
S1
S2
S3
S4
EXIT
0.3
0.2
0.5
0.4
0.5
0.2
0.3
0.3
0.8
0.3
0.4
0.8


EXIT


N
W
E
S


START
S1
S2
S3
S4
EXIT
0.3
-1
0.2
-4
0.5
-4
0.4
-2
0.5
-2
0.2
-1
0.3
-4
0.3
-4
0.8
-1
0.3
-2
0.8
-4
0.4
-1

Calculating value of each state



*v(Start) = 0.5 * [-4 -11.9] + 0.5 * [-2 -14.8] = -16.35*
*v(S2) = 0.8 * [-1] + 0.2 * [-4 -11.9] = -0.8 -3.18 = -3.98*

**Discounting Factor is 1.*

Limitations

- It assumes that we have complete knowledge of the dynamics of the MDP (we call this ***model-based RL***). However, this is rarely the case in real-world problems
- For bigger problems, as the number of states and actions increases, the size of the value table grows exponentially (think about trying to define all the possible states of chess). This results in the '***curse of dimensionality***' problem, where the computational and memory requirements escalate rapidly, making it challenging to apply DP to high-dimensional problems.

References

- Reinforcement Learning: an Easy Introduction to Value Iteration | by Carl Bettosi | Towards Data Science
- Markov Decision Processes and Bellman Equations | by Steve Roberts | Towards Data Science

Thanks