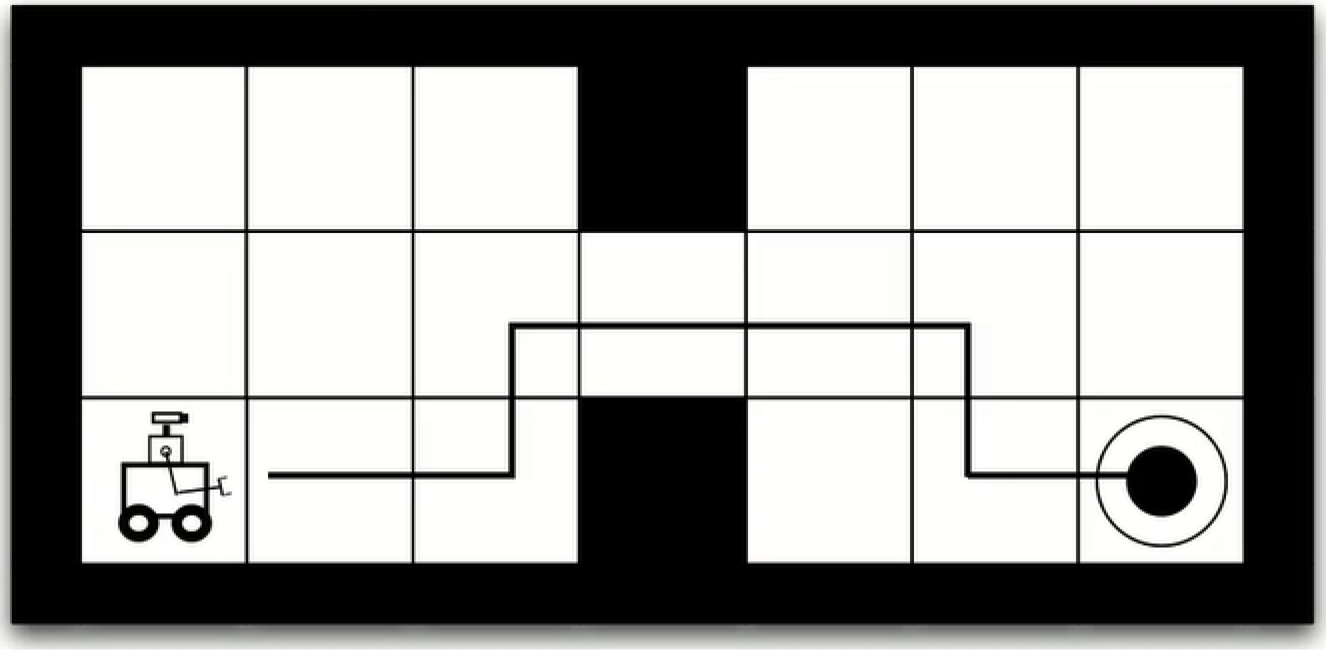
Artificial Intelligence

Steve James

Probabilistic Planning

The planning problem

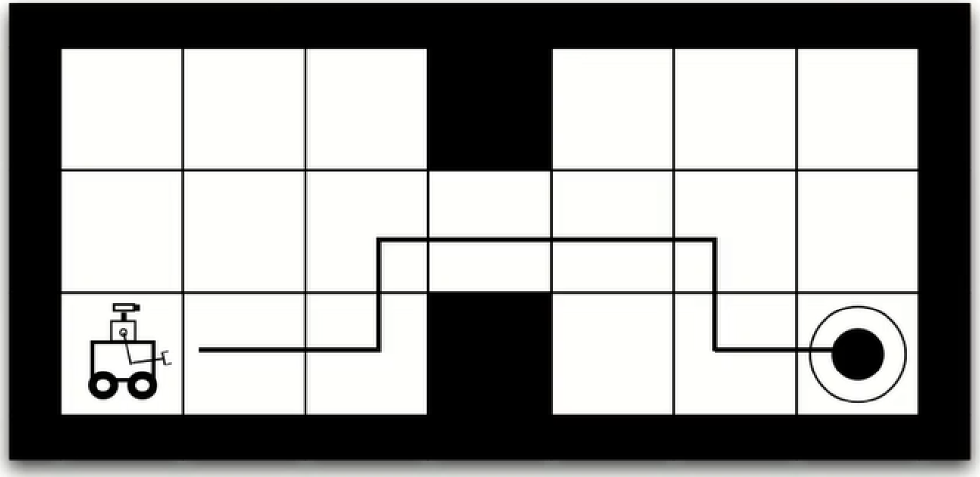
- Find a sequence of actions to achieve some goal



Plans

- It’s great when a plan just works
  - But the real world isn’t like that

- To plan effectively, we must take uncertainty seriously

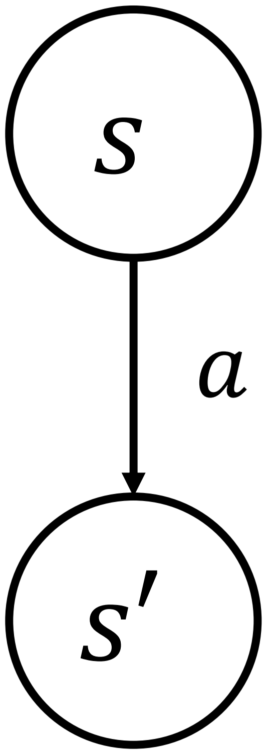


Probabilistic planning

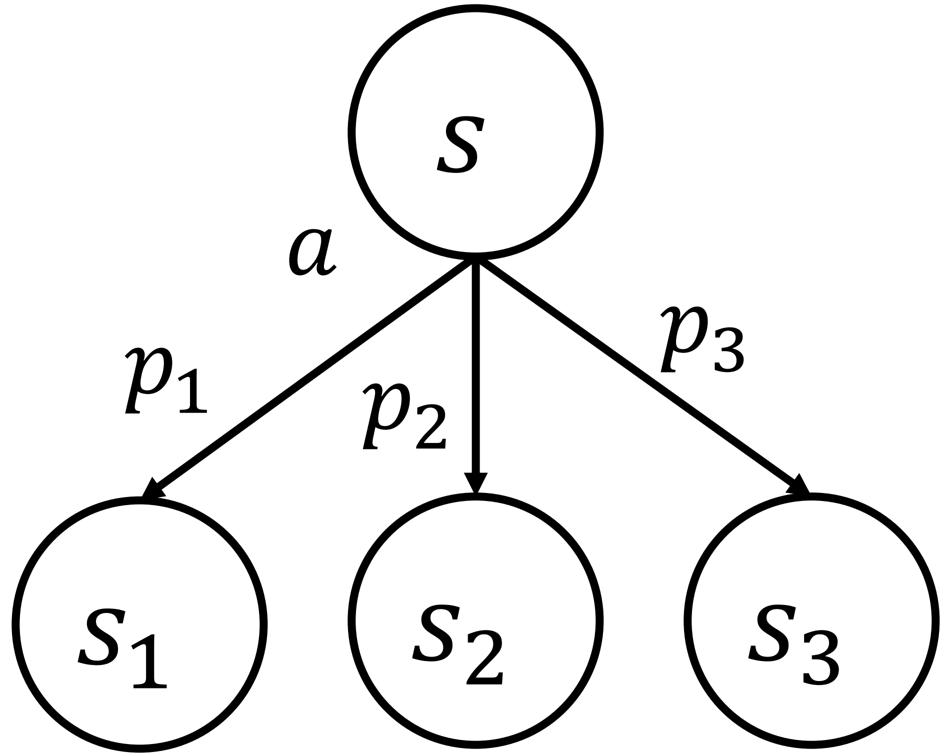
- As before:
  - Generalise deterministic logic to probabilities
  - Generalise deterministic planning to probabilistic planning

- This results in a harder planning problem
- In particular:
  - Must model stochasticity
  - Plans can fail
  - Can no longer compute straight-line plans

Stochastic outcomes



$s' = T(s,a)$
$C(s,a,s')$



Probability distribution over transitions $T(s'|s,a)$
$R(s,a,s')$

Probabilistic planning

- Recall – systems that change over time
  - Problem has a state
  - State has the Markov property

$$P(S_t|S_{t-1}, A_{t-1}, S_{t-2}, A_{t-2}, \ldots, S_0, A_0) = P(S_t|S_{t-1}, A_{t-1})$$

Only previous state

But also previous action (**controlled** **process**)

The Markov Property

- Needs to be extended for planning
  - s_{t+1} depends only on s_t and a_t Agent chooses this
  - r_t depends only on s_t, a_t and s_{t+1}

- Current state is a sufficient statistic of agent's history
- This means that:
  - Decision-making depends only on current state
  - The agent does not need to remember its history

Probabilistic planning

- Markov Decision Processes (MDPs):
  - The canonical decision-making formulation.
  - Problem has a set of states.
  - Agent has available actions.

  - Actions cause stochastic transitions.
  - Transitions have costs/rewards.
    - Transitions, costs depend only on previous state.

  - Agent must choose actions to maximise expected reward (minimise costs) summed over time.

Markov decision processes

$\langle S, A, R, T, \gamma \rangle$

- S: set of states
- A: set of actions
- γ: discount factor $\in [0,1]$
- R: reward function
  - $R(s, a, s')$ is the reward received taking action a from state s and transitioning to state s'
- T: transition function
  - $T(s'|s, a)$ is the probability of transitioning to state s' after taking action a in state s

MDPs

- Goal: choose actions to maximises return: expected sum of discounted rewards
  - (equivalent to min sum of costs)

$$R^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

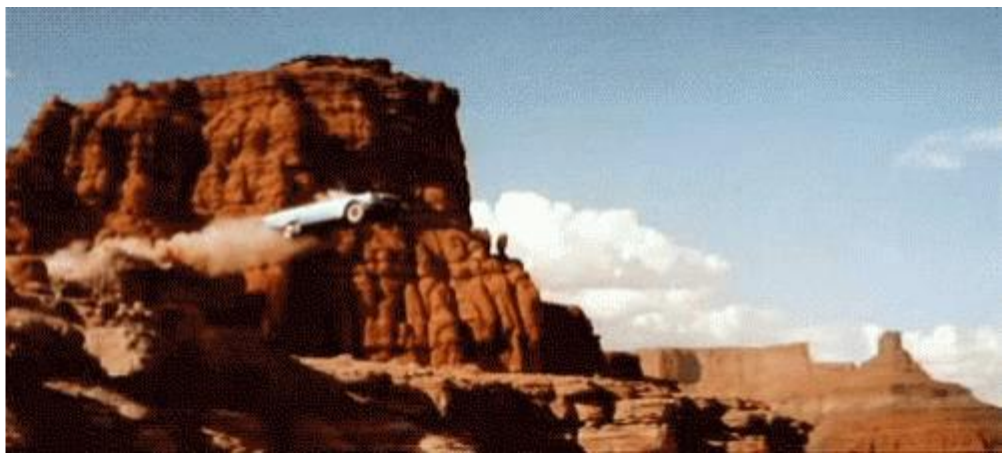
Due to stochasticity

All future rewards

Now matters more

Rewards summed

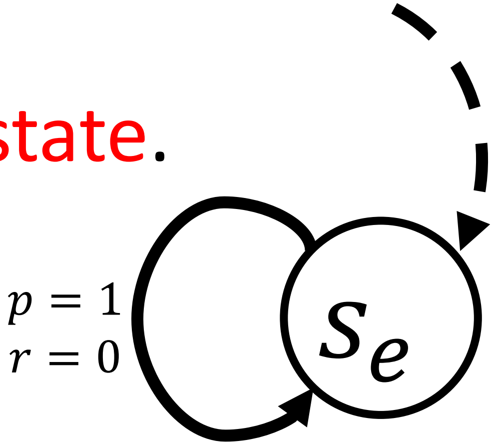
Why summed rewards?



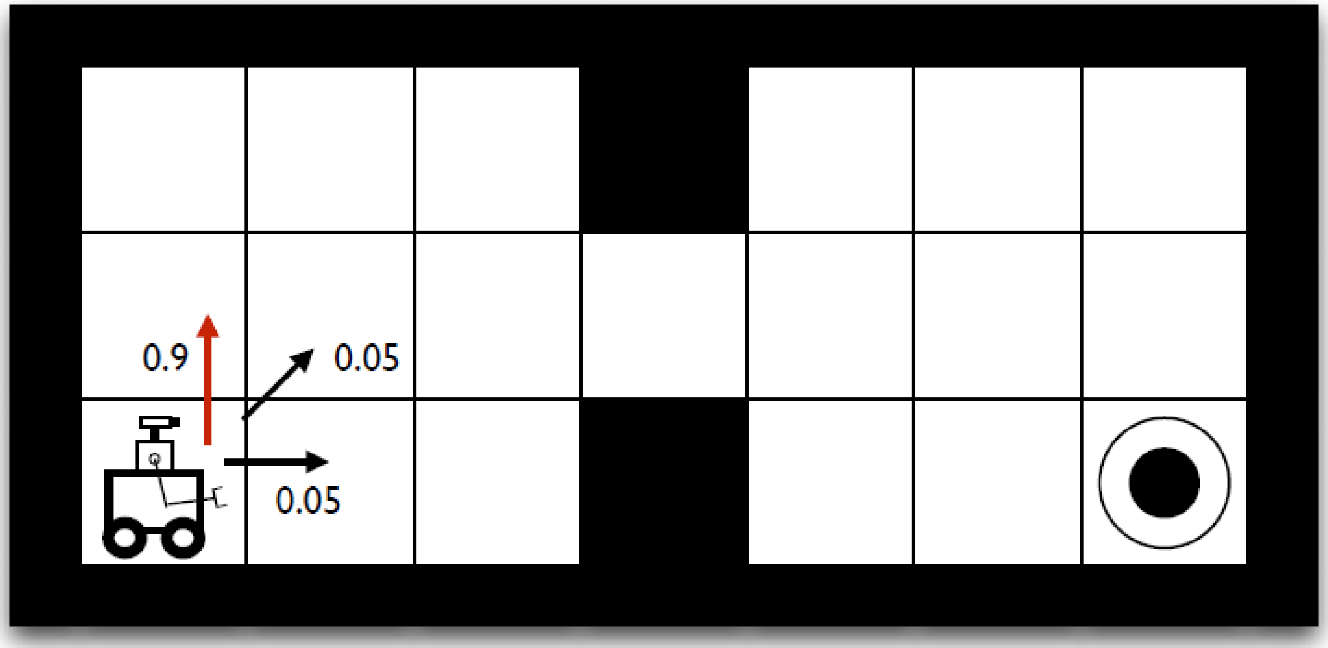
Episodic problems

- Some problems end when you hit a particular state

- Model: transition to absorbing state.
- In practice: reset the problem.



Example



- States: set of grid locations
- Actions: up, down, left, right
- Transition function: move in direction of action with $p = 0.9$
- Reward function: -1 for every step, 1000 for (absorbing) goal

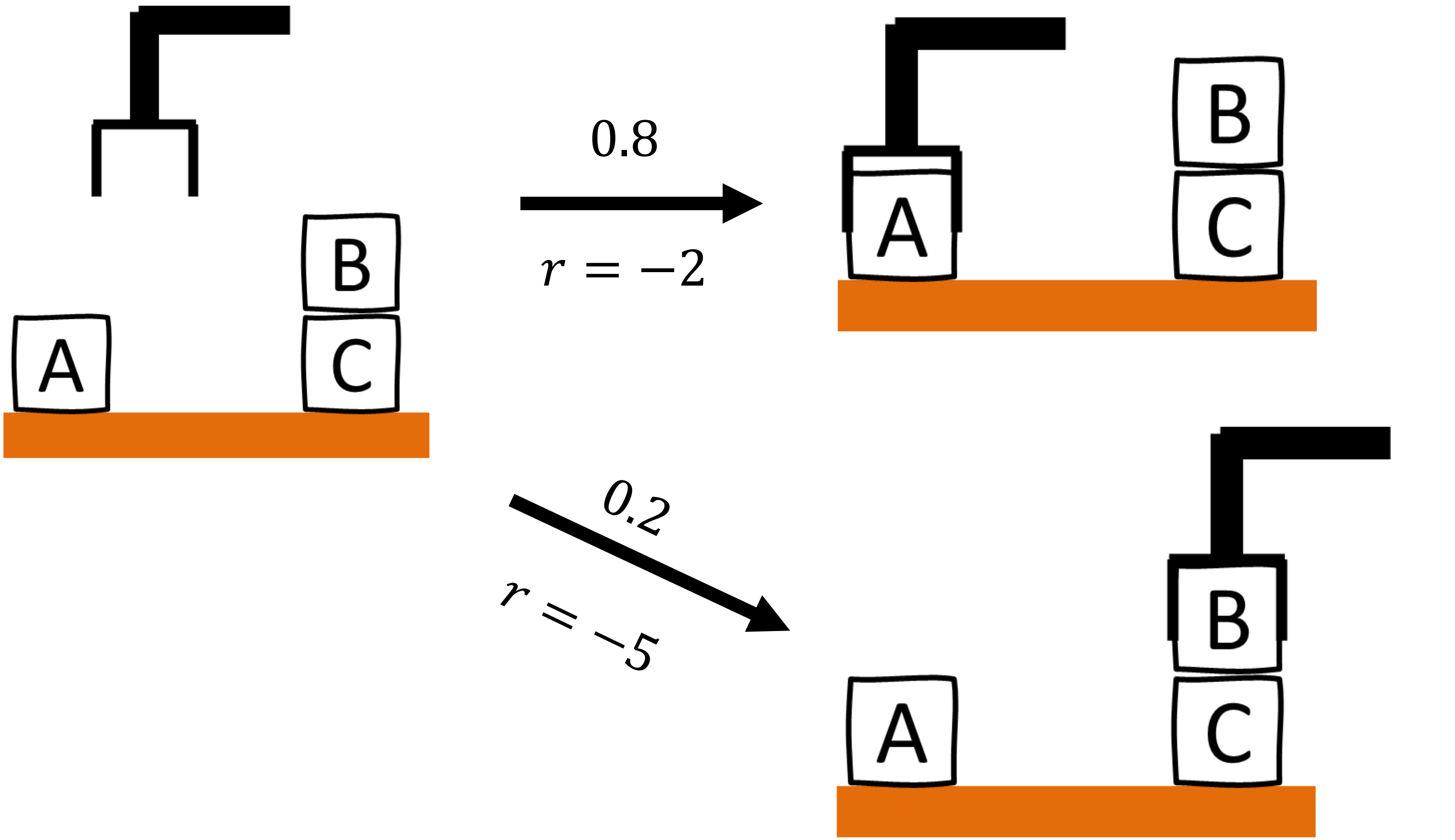
Back to PDDL

- MDPs do not contain the structure of PDDL
  - PPDDL: probabilistic planning domain definition language

- Now operators have probabilistic outcomes:

```
(:action move_left
 :parameters (x, y)
 :precondition (not (wall(x-1, y))
 :effect (probabilistic
    0.8 (and (at(x-1)) (not at(x)) (decrease (reward) 1))
    0.2 (and (at(x+1)) (not(at(x))(decrease (reward) 1))
    )
)
```

Example



MDPs

- Our goal is to find a policy

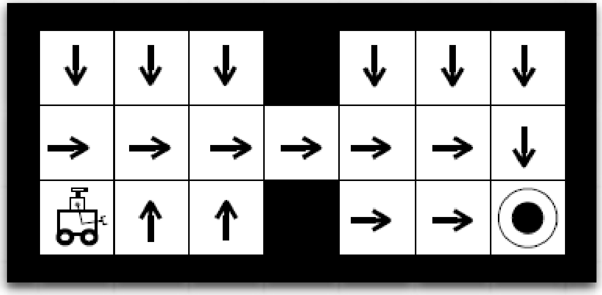
$$\pi : S \rightarrow A$$

- … that maximises return: expected sum of rewards: (equiv min sum of costs)

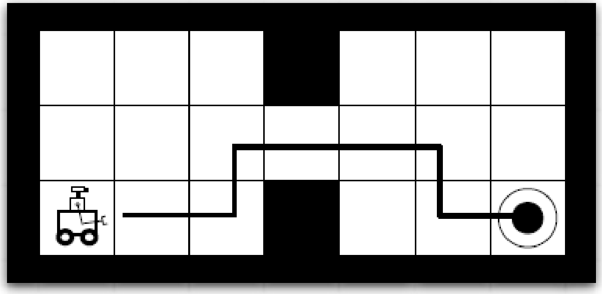
$$R^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t}\right]$$

Policies and plans

- Compare a policy:
  - An action for every state

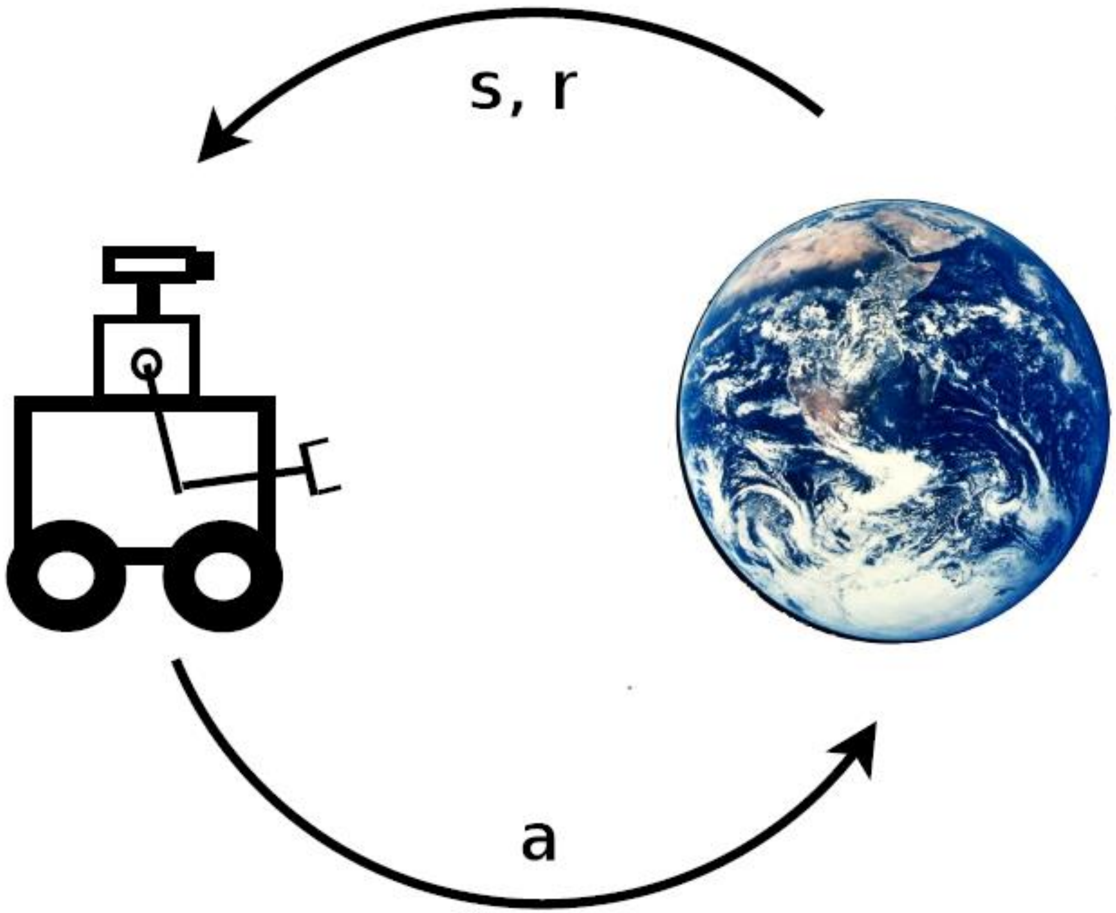


- With a plan
  - A sequence of actions



- Why the difference?

Policies



Planning

- So our goal is to produce an optimal policy:

$$\pi^*(s) := \underset{\pi}{\operatorname{argmax}} R^{\pi}(s)$$

- Note: we know T and R
- Useful fact: such a policy always exists
  - But might be more than one

Planning

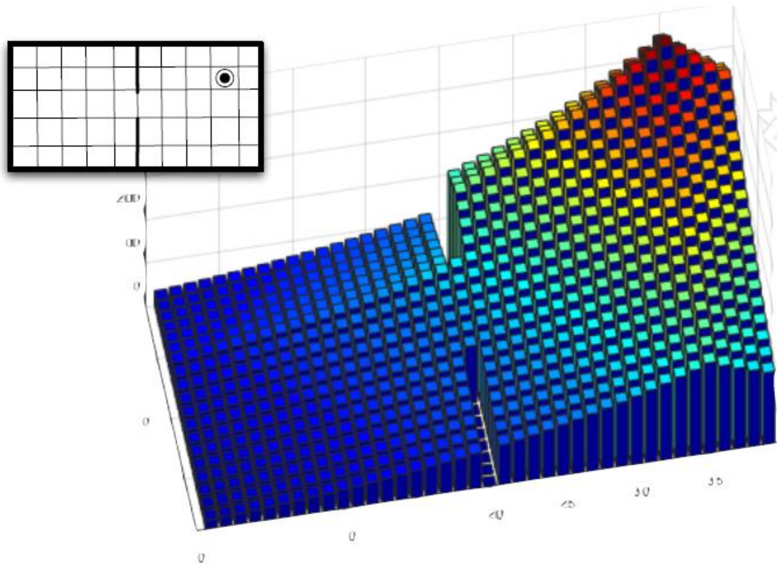
- Key quantity is the return given by a policy from a state:

$$R^{\pi}(s)$$

- Define the value function to estimate this quantity:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^{t} r_{t}\right]$$

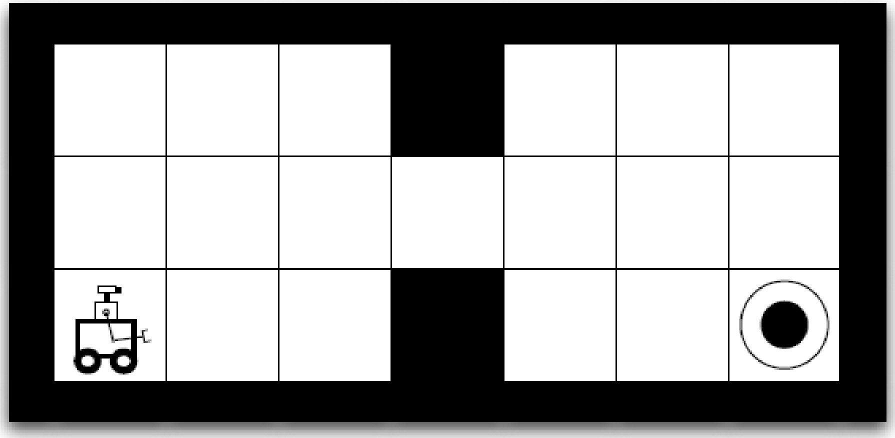
Value functions



- V is a useful thing to know
  - Maybe we can use it to improve π?
  - How to find V?

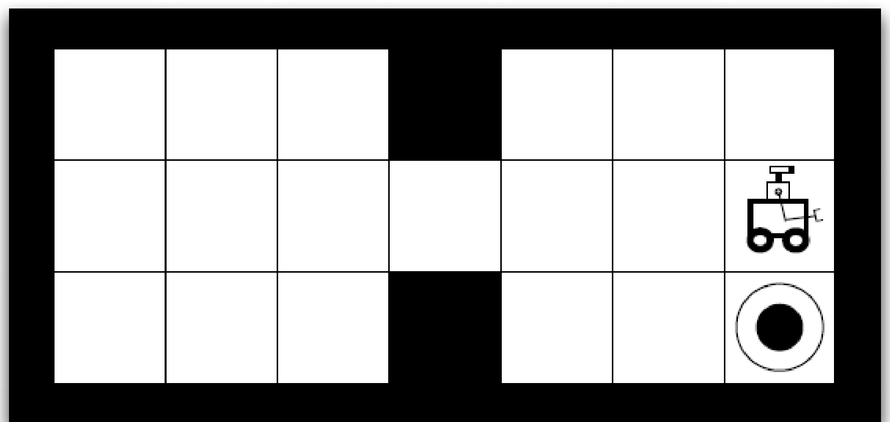
Monte Carlo

- Simplest thing you can do: sample $R(s)$



Monte Carlo

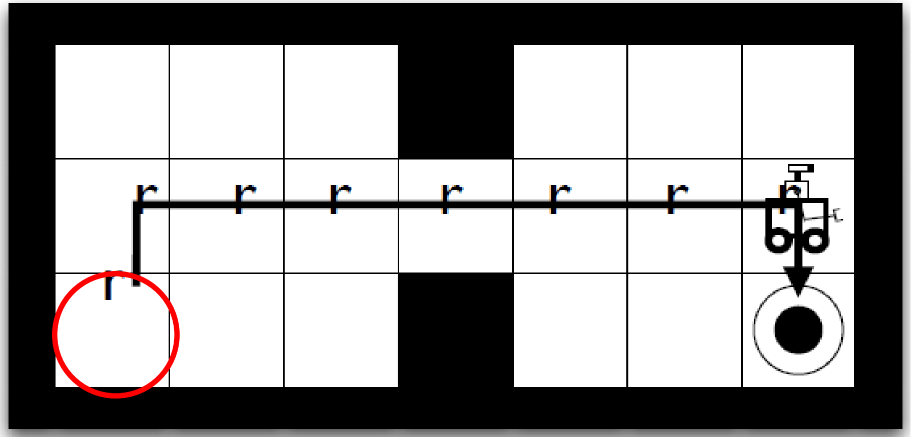
- Simplest thing you can do: sample $R(s)$



Monte Carlo

- Simplest thing you can do: sample $R(s)$

$R = \sum \gamma^t r_t$



- Do this repeatedly and average:

$$V^{\pi}(s) = \frac{R_1(s) + R_2(s) + \ \ldots + R_n(s)}{n}$$

Monte Carlo estimation

- For each state s
  - Repeat many times:
    - Start at s
    - Run policy forward until absorbing state (or $\gamma^t < \epsilon$)
    - Write down discount sum of rewards received
    - This is a sample of $V(s)$
  - Average these samples
- This always works!
  - But very high variance. Why?

Monte Carlo estimation

$$R = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots + \gamma^n r_n$$

Random variable

Random variable

Random variable

Random variable

Random variable

Doing better

- We need estimate of R that doesn't grow in variance as episode length increases
- Might there be some relationship between values that we can use as an extra source of information?

$$R(s_0) = r_0 + \gamma r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots + \gamma^n r_n$$

$$R(s_1) = \gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + \cdots + \gamma^{n-1} r_n$$

Bellman

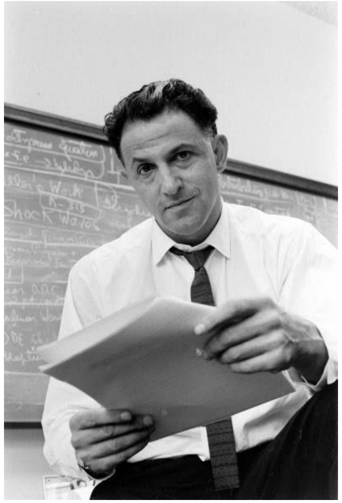
- Bellman's equation is a condition that must hold for V:

Value of next state

$$V^{\pi}(s) = \mathbb{E}_{s'}[r(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

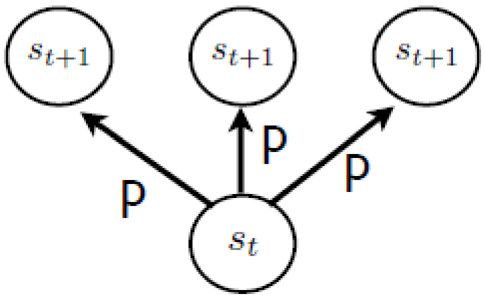
Value of this state

Reward



Dynamic programming

- We can use this expression to update V



$$V^{\pi}(s) \leftarrow \sum_{s'} [T(s'|s,\pi(s)) \times (r(s,\pi(s),s') + \gamma V^{\pi}(s'))]$$

- This algorithm is called dynamic programming

Value iteration

- This gives us an algorithm for computing the value function for a specific given fixed policy

- Repeat:
  - Make a copy of the VF
  - For each state in VF, assign value using BE
  - Replace old VF

Value iteration

- $V(s) = 0, \forall s$

- Do:
  - $V_{old} = copy(V)$
  - For each state s:
    - $V(s) = \sum_{s'} [T(s, \pi(s), s') \times (r(s, \pi(s), s') + \gamma V_{old}(s'))]$
- Until V converges

- Notes:
  - Fixed policy π
  - $V(s') = 0$, definitionally, if s is absorbing

Policy iteration

- Recall that we seek the policy that maximises $V^{\pi}(s), \forall s$
- Therefore we know that for the optimal policy π^*

$$V^{\pi^*}(s) \geq V^{\pi}(s), \forall \pi, s$$

- This means that any change to V^{π} that increases V^{π} anywhere obtains a better policy

Policy iteration

- Leads to a general policy improvement framework:
  - Start with policy π
  - Estimate V^{π}
  - Improve π
    - $\pi(s) = \underset{a}{\operatorname{argmax}}\, \mathbb{E}[r + \gamma V^{\pi}(s')]\,, \forall s$

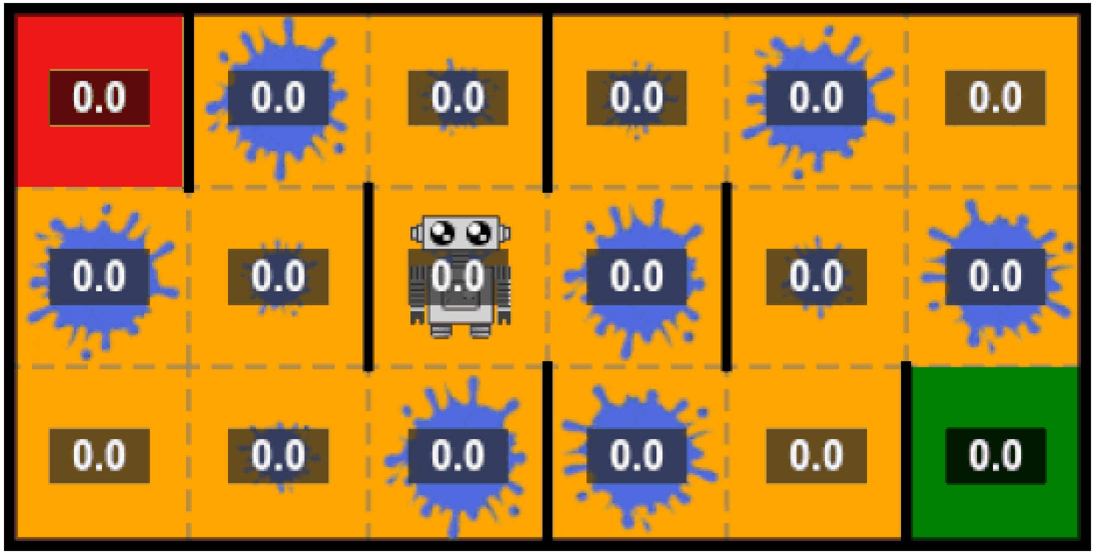
Repeat

- This is policy iteration
  - Guaranteed to converge to optimal policy
  - Steps 2,3 can be interleaved as rapidly as you like

Policy iteration

- $V(s) = 0, \forall s$
- Do:
  - $V_{old} = copy(V)$
  - For each state s:
    - $V(s) = \sum_{s'}[T(s, \pi(s), s') \times (r(s, \pi(s), s') + \gamma V_{old}(s'))]$
  - For each state s:
    - $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s'}[T(s, a\ s') \times (r(s, a, s') + \gamma V_{old}(s'))]$
- While π changes

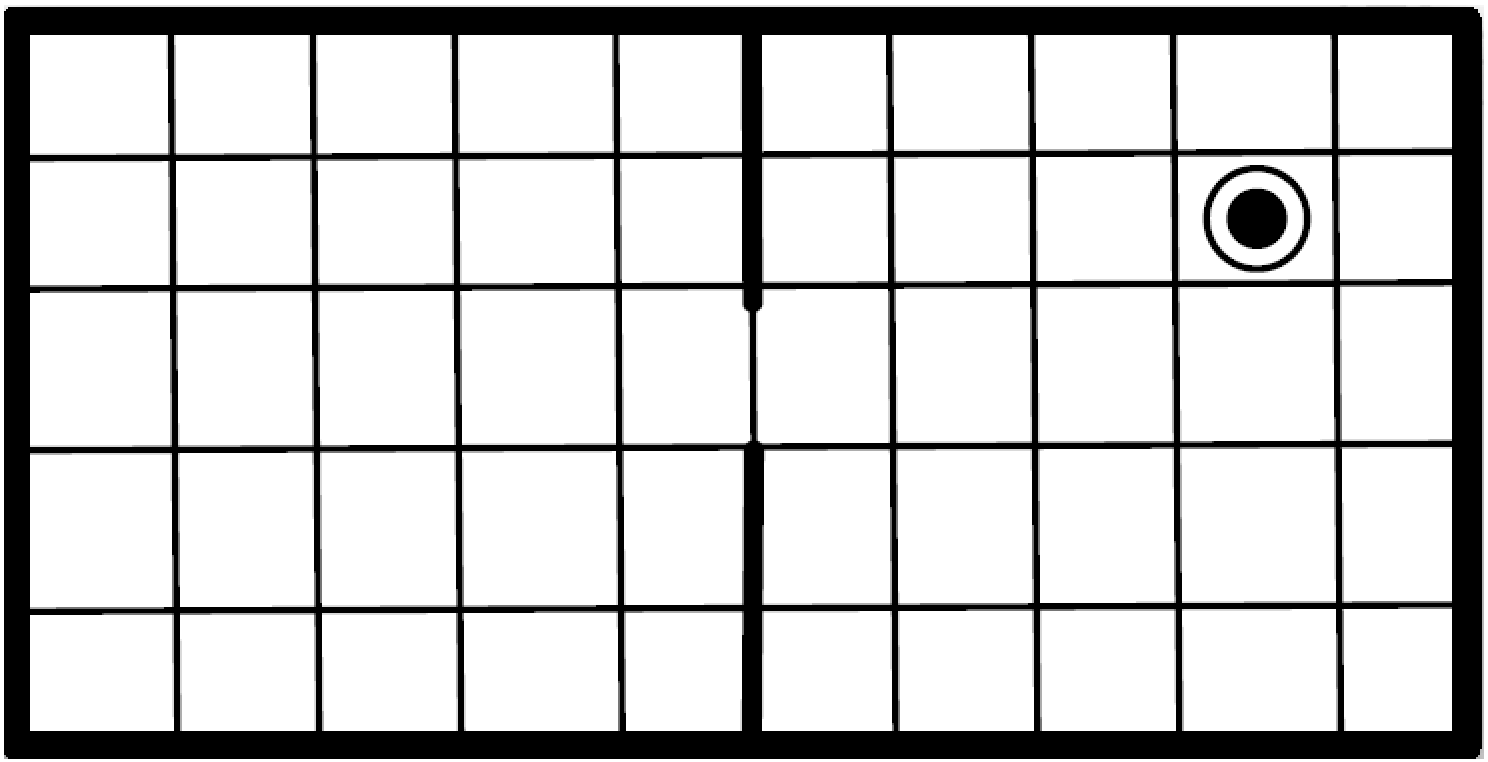
- Finds an optimal policy in time polynomial in $|S|$ and $|A|$
  - There are $|A|^{|S|}$ possible policies


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
0.0
0.0

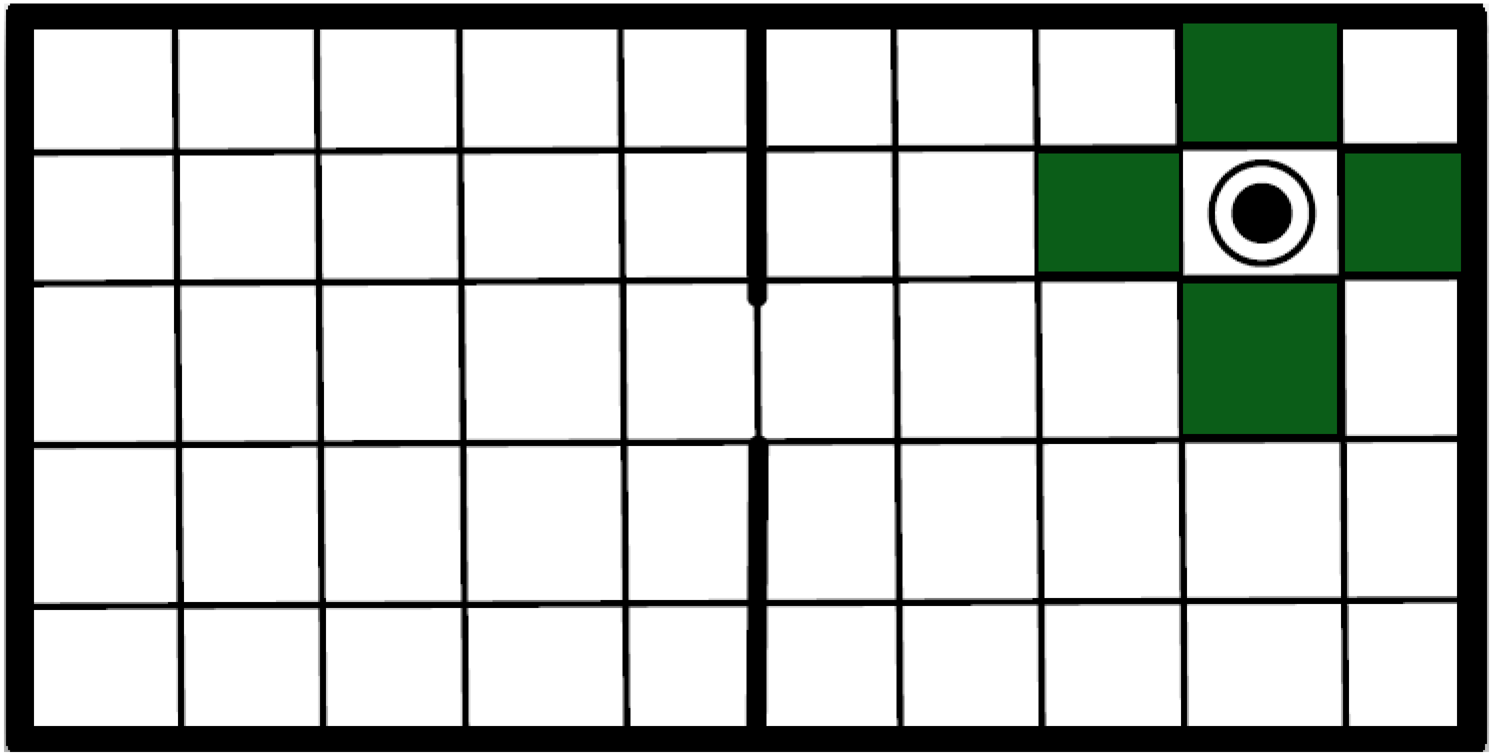
Improvements

- Extensions to the basic algorithm largely deal with controlling the size of the state sweeps
  - Not all states are reachable
  - Not all states need to be updated at each iteration
  - Not all states are likely to be encountered from a start state

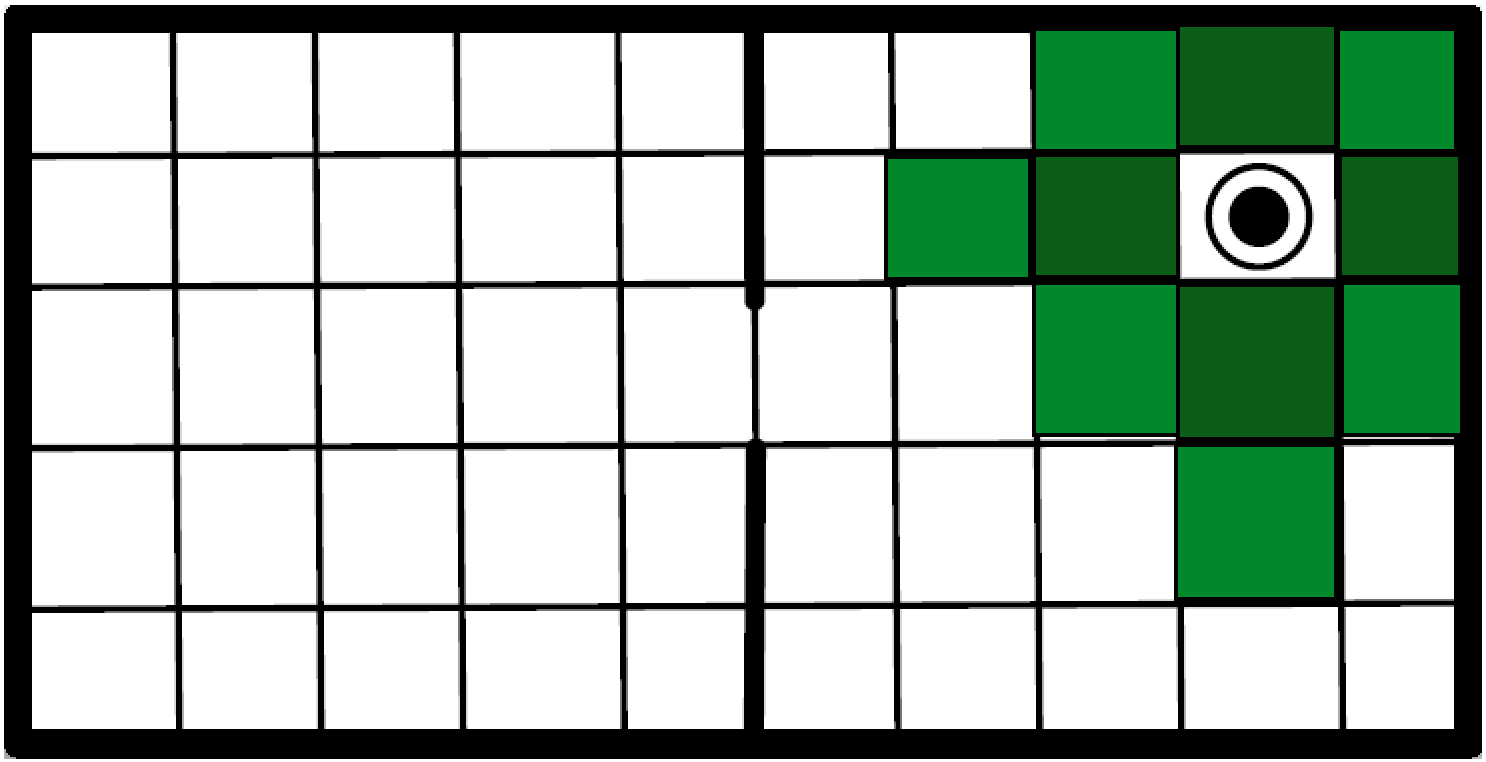
Prioritised sweeping



Prioritised sweeping



Prioritised sweeping



Prioritised sweeping

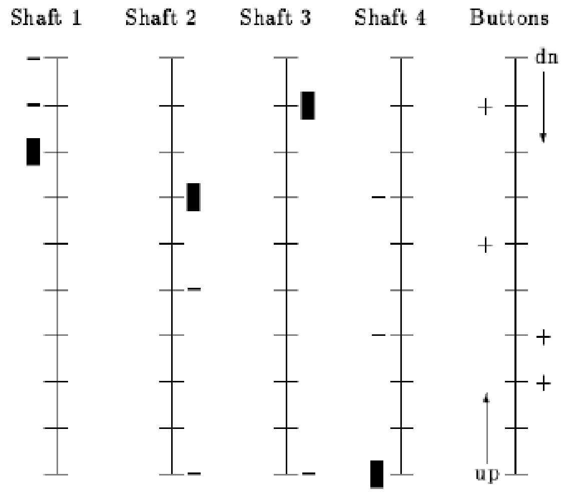
- $V(s) = 0, \forall s$
- vQueue.insert(s, 0) $\forall s$

- While π changes
  - s ←vQueue.pop()
  - $V(s) = \sum_{s'} [T(s, \pi(s), s') \times (r(s, \pi(s), s') + \gamma V_{old}(s'))]$
  - $\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} [T(s, a\ s') \times (r(s, a, s') + \gamma V_{old}(s'))]$
  - For all s_p such that $T\left(s_p, \pi(s), s\right) > 0$:
    - vQueue.insert(s_p, $\Delta V(s)$)

- DP algorithms can solve problems with millions of states.

Elevator scheduling

- Crites and Barto (1985)
  - System with 4 lifts, 10 floors
  - Realistic simulator
  - 46 dimensional state space



Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
SECTOR	30.3	1643	59.5	13.50
HUFF	22.8	884	55.3	5.10
DLB	22.6	880	55.8	5.18
LQF	23.5	877	53.5	4.92
BASIC HUFF	23.2	875	54.7	4.94
FIM	20.8	685	53.4	3.10
ESA	20.1	667	52.3	3.12
RLd	18.8	593	45.4	2.40
RLp	18.6	585	45.7	2.49