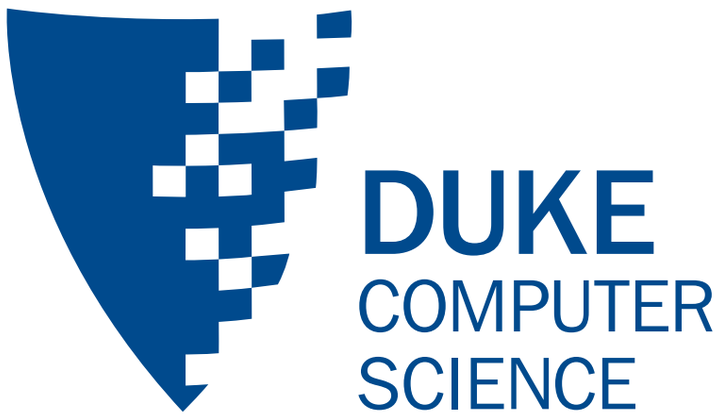
Probabilistic Planning

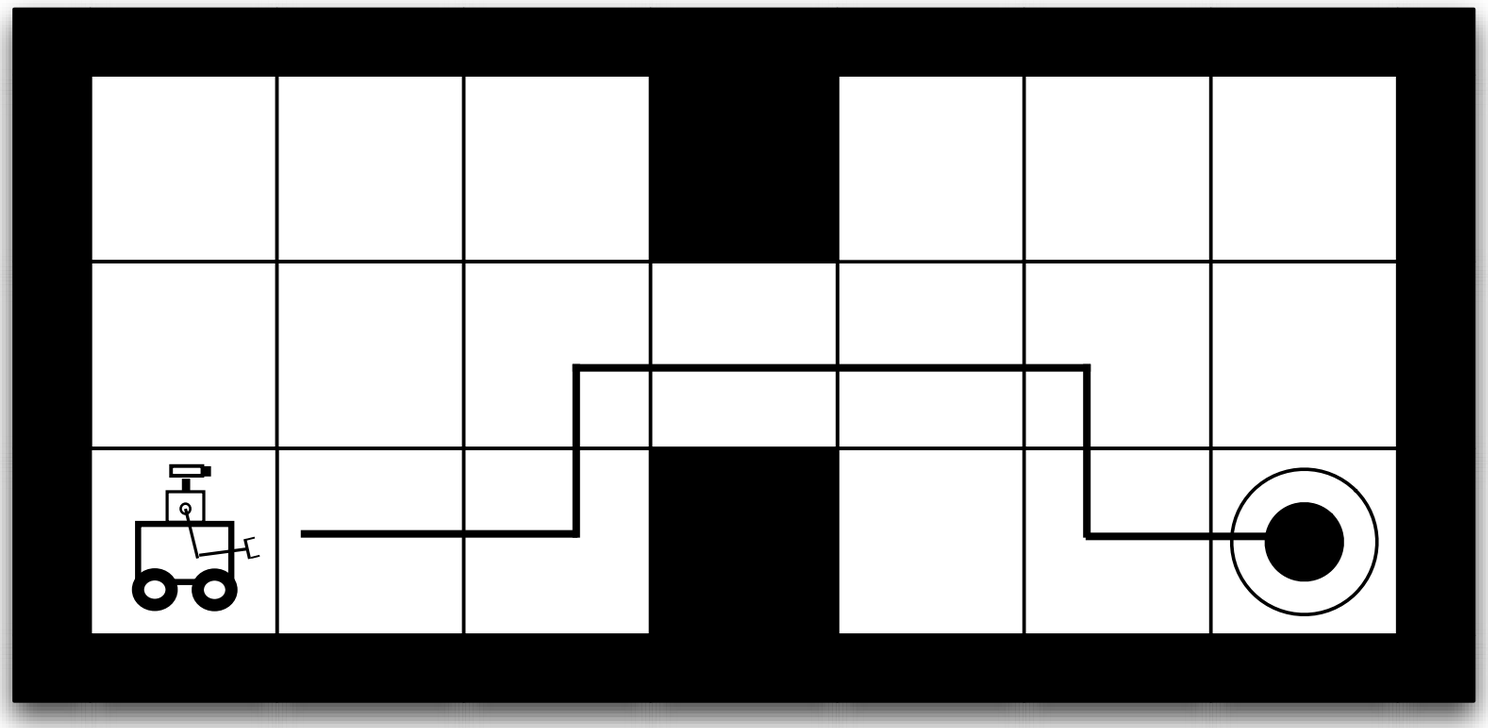
George Konidaris
gdk@cs.duke.edu



Spring 2016

The Planning Problem

Finding a sequence of actions to achieve some goal.



Plans

It's great when a plan works …

… but the world doesn't work like that.

To plan effectively we need to take uncertainty seriously.

Probabilistic Planning

As before:

- Generalize logic to probabilities.
- Probabilistic planning.

This results in a harder planning problem.

In particular:

- **Plans can fail.**
- **Can no longer compute straight-line plans.**

Probabilistic Planning

Recall:

- Problem has a state.
- State has the Markov property.

$$P(S_t|S_{t-1}, a_{t-1}, S_{t-2}, a_{t-2}, ..., S_0, a_0) = P(S_t|S_{t-1}, a_{t-1})$$

Markov Decision Processes (MDPs):

- *The* canonical decision making formulation.
- Problem has a set of states.
- Agent has available actions.
- Actions cause stochastic *state transitions*.
- Transitions have costs/rewards.

Markov Decision Processes

S : set of *states*
A : set of actions
γ : discount factor

$$< S, A, \gamma, R, T >$$

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.

(some states are absorbing - execution stops)

The Markov Property

Needs to be extended for MDPs:

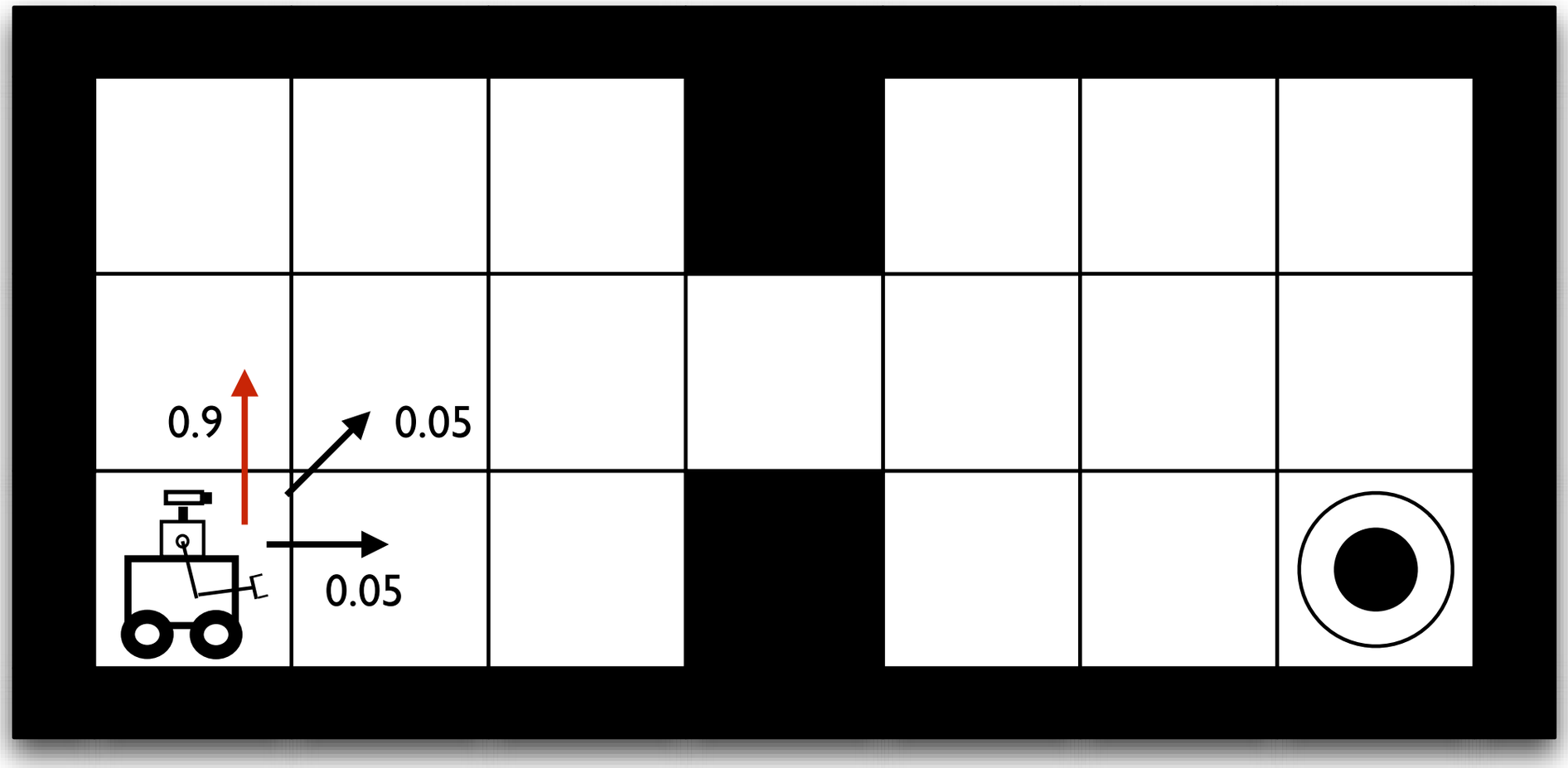
- s_{t+1} depends only on s_t and a_t
- r_t depends only on s_t and a_t

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

Example



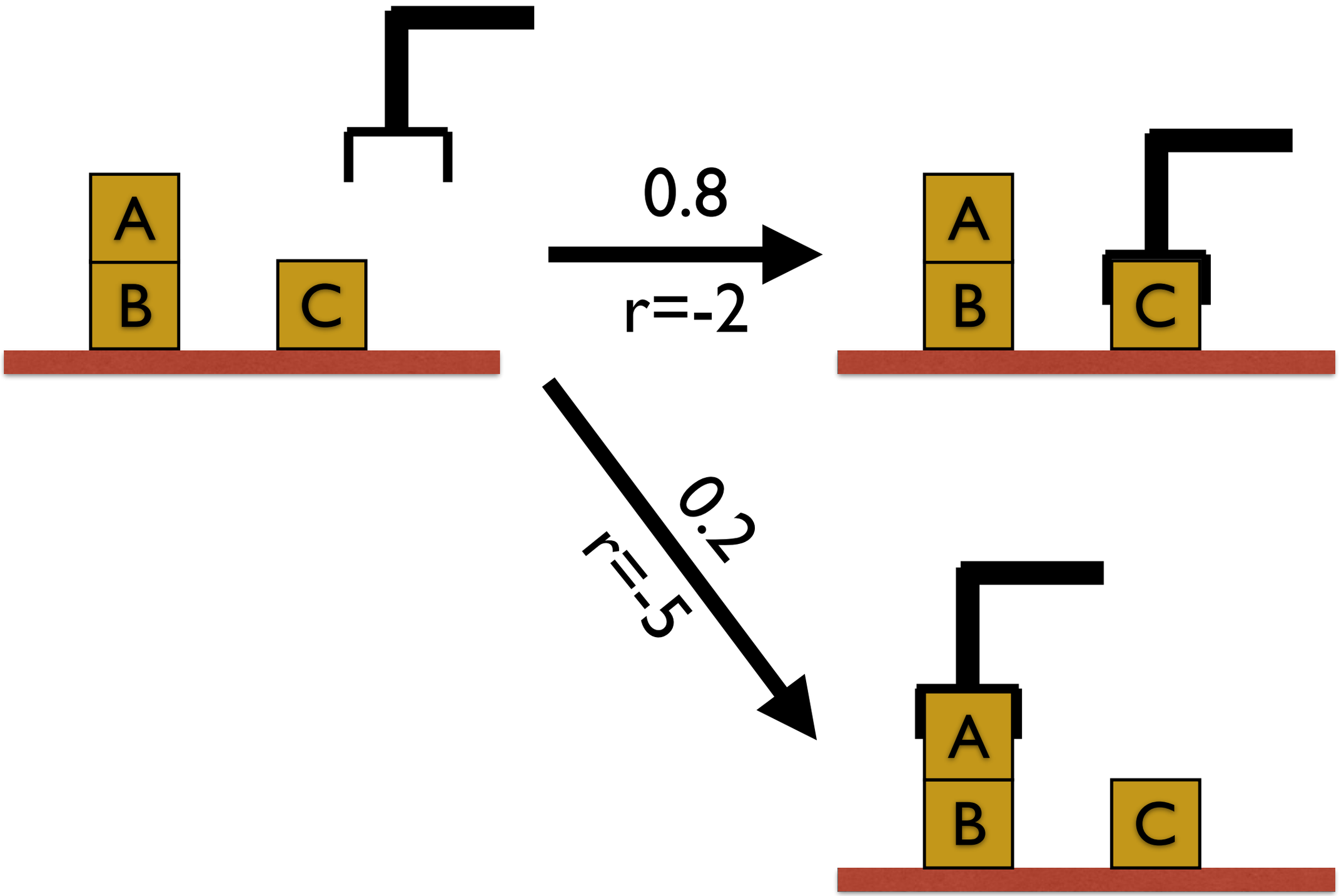
States: set of grid locations

Actions: up, down, left, right

Transition function: move in direction of action with p=0.9

Reward function: -1 for every step, 1000 for finding the goal

Example



MDPs

Our goal is to find a policy:

$$\pi : S \rightarrow A$$

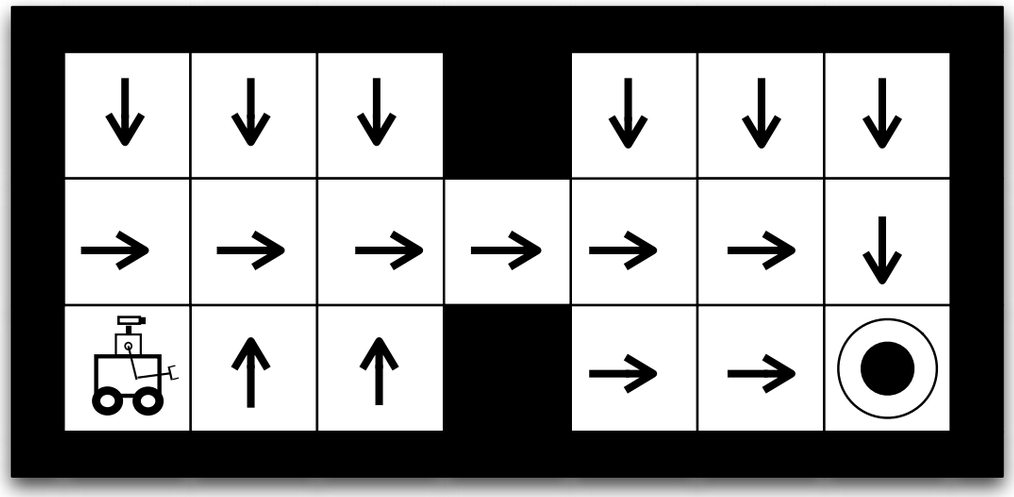
That **maximizes *return*: expected sum of rewards.**
(equiv: min sum of costs)

$$\sum_{i=1}^{\infty} \mathbb{E}[\gamma^i r_i]$$

Policies and Plans

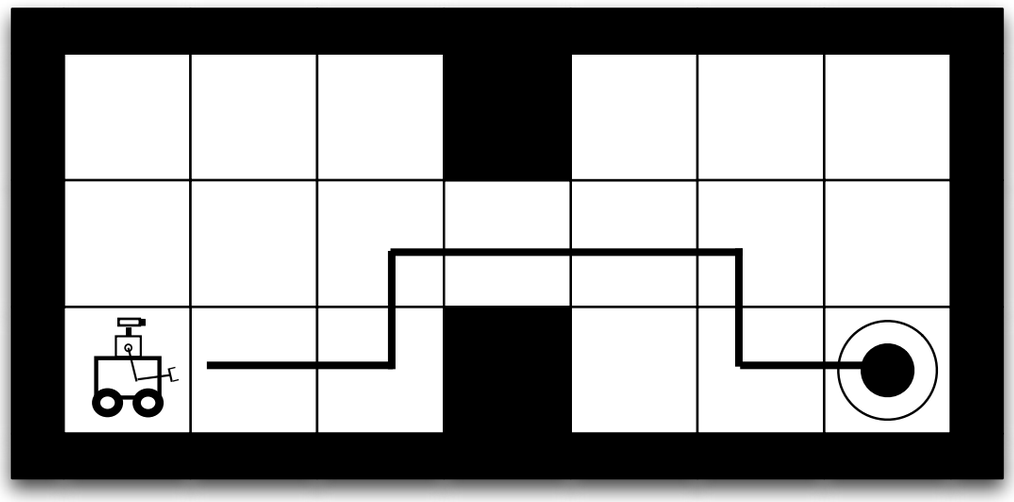
Compare a policy:

- An action for every state.



… with a plan:

- A sequence of actions.



Why the difference?

Planning

So our goal is to produce optimal policy.

$$\pi^*(s) = \max_\pi R^\pi(s)$$

Planning: we know *T* and *R*.

Define the *value function* to estimate this quantity:

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

***V* is a useful thing to know.**

How to find *V*?

Monte Carlo Estimation

One approach:

- For each state *s*
- *Repeat many times:*
  - Start at *s*
  - Run policy forward until absorbing state (or γ^t small)
  - 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?

Bellman

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

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \max_{a} \sum_{s'} T(s'|s, a) V\pi(s')$$

value of this state

reward

expected value of next state

Value Iteration

This gives us an algorithm for ***learning 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.

This is known as ***value iteration***.
(In practice, only adjust "reachable" states.)

Why do we can so much about VF?

Policy Iteration

We can adjust the policy given the VF:

$$\pi(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} T(s'|s,a) V(s') \right]$$

Adjust policy to be *greedy w.r.t VF*.

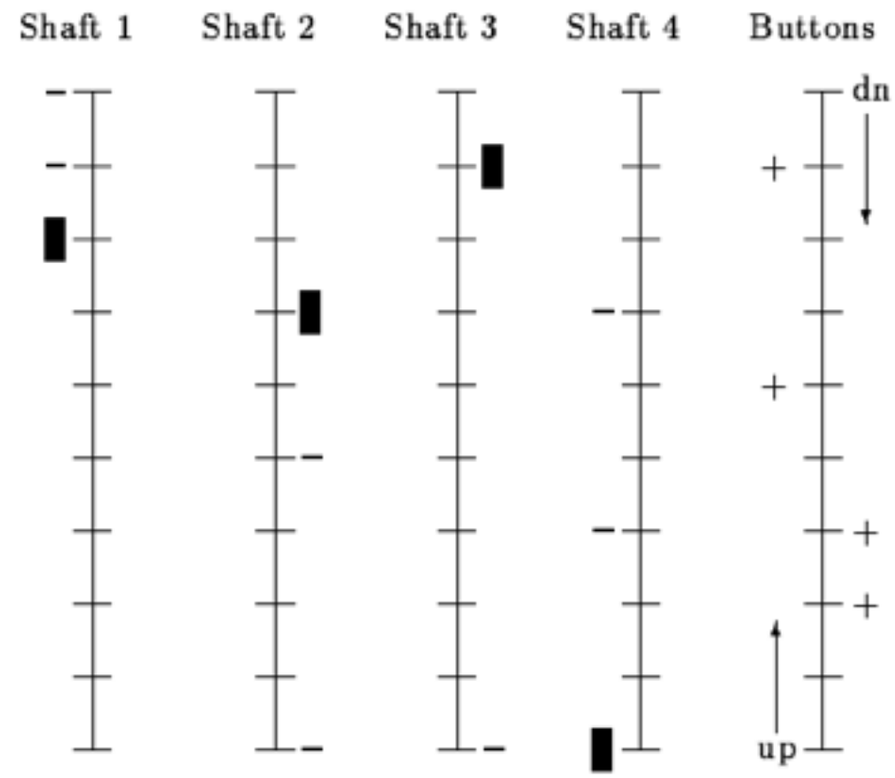
We can alternate value and policy iteration.
Surprising results:

- This converges even if alternate every step.
- Converges to optimal policy.
- Converges in polynomial time.

Elevator Scheduling

Crites and Barto (1985):

- System with 4 elevators, 10 floors.
- Realistic simulator.
- 46 dimensional state space.



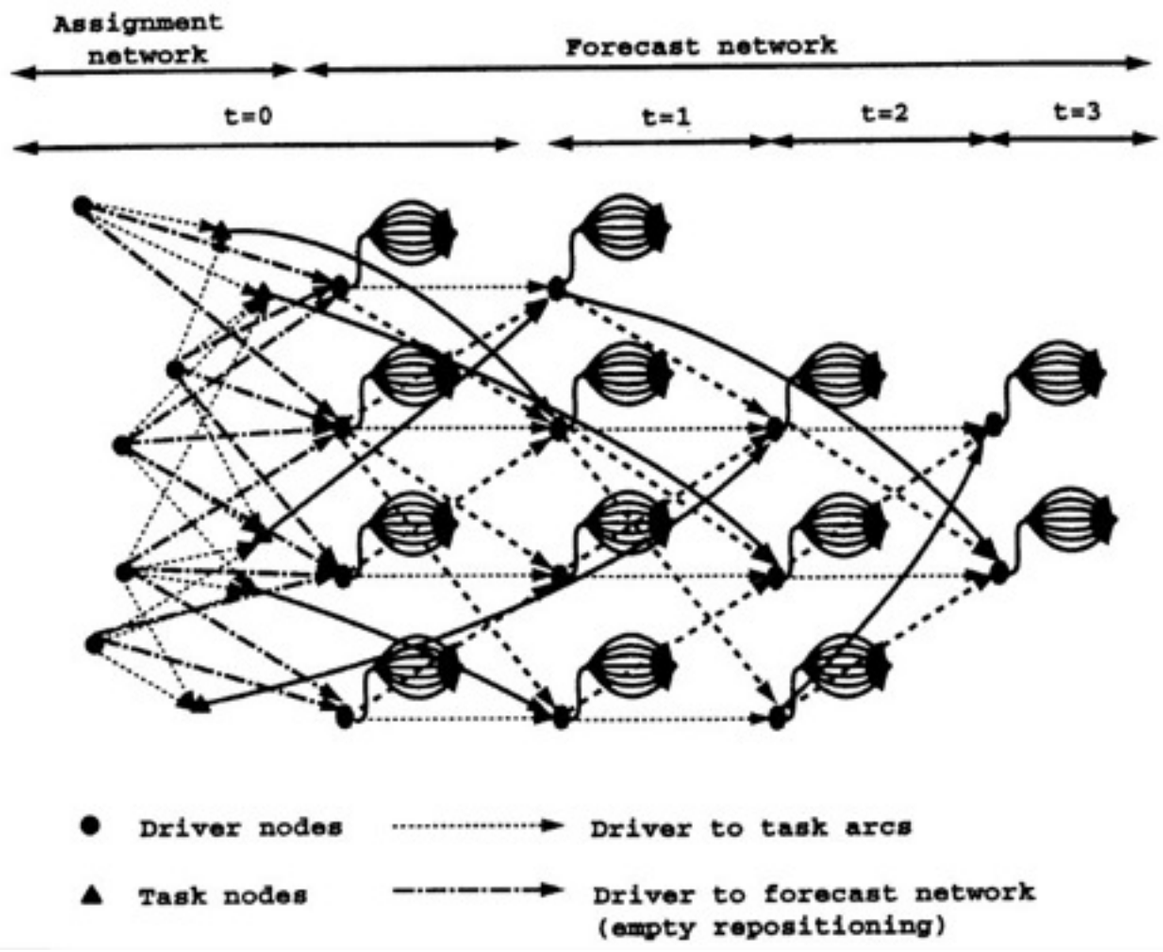
Algorithm	AvgWait	SquaredWait	SystemTime	Percent>60 secs
SECTOR	30.3	1643	59.5	13.50
HUFF	22.8	884	55.3	5.10
DLB	22.6	880	55.8	5.18
LQF	23.5	877	53.5	4.92
BASIC HUFF	23.2	875	54.7	4.94
FIM	20.8	685	53.4	3.10
ESA	20.1	667	52.3	3.12
RLd	18.8	593	45.4	2.40
RLp	18.6	585	45.7	2.49

MicroMAP

DUKE COMPUTER SCIENCE

"Drivers and Loads" (trucking), CASTLE lab at Princeton

"the model was used by 20 of the largest truckload carriers to dispatch over 66,000 drivers"



Back to PDDL

Note that an MDP does not contain the structure of PDDL.

If we wish to combine that structure and probabilistic planning, we can use a related language called PPDDL - *probabilistic* problem domain definition language.

PPDDL Operators

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))
           )
)
```

PPDDL

Instead of computing a *plan*, we again need a *policy*.

Most planners:

- Take as input PPDDL.
- Use a simulator.
- Compute policy for states reachable from start.
- Are evaluated jointly with simulator.

Robot Planning

One more common type of planning left!