MDPs: overview

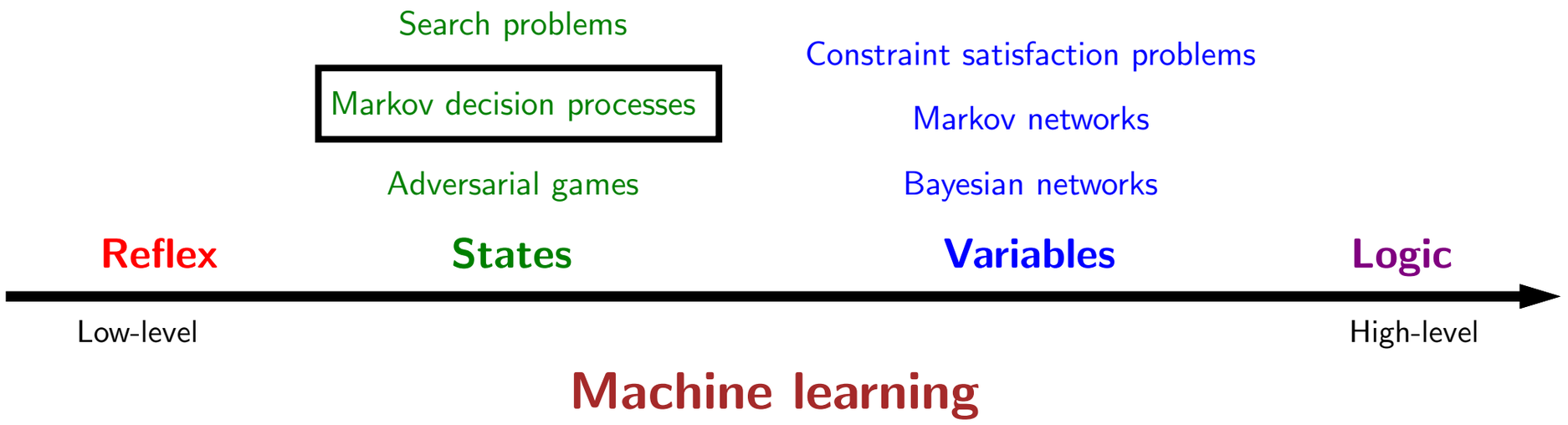
answer in chat

Question

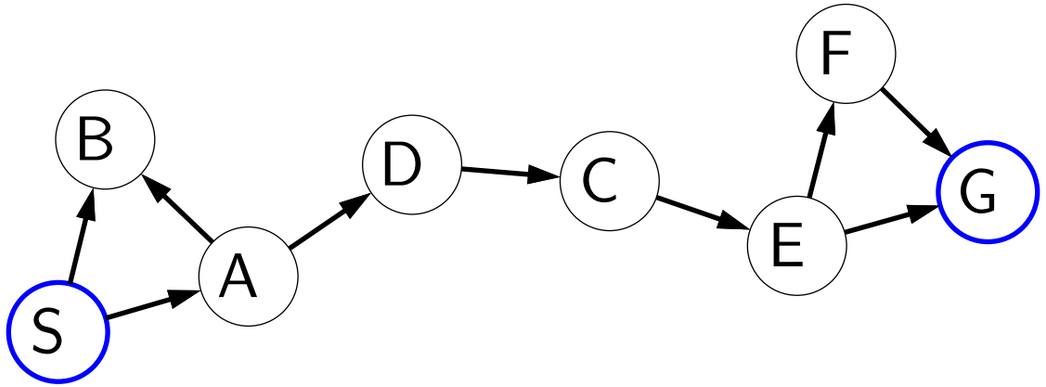
How would you get groceries on a Saturday afternoon in the least amount of time?

- order grocery delivery
- bike to the store
- drive to the store
- Uber/Lyft to the store
- fly to the store

Course plan

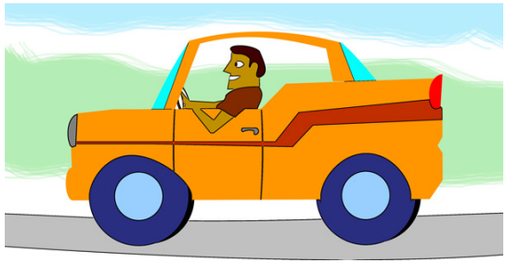

Search problems
Markov decision processes
Adversarial games
Constraint satisfaction problems
Markov networks
Bayesian networks
Reflex
States
Variables
Logic
Low-level
High-level
Machine learning

So far: search problems



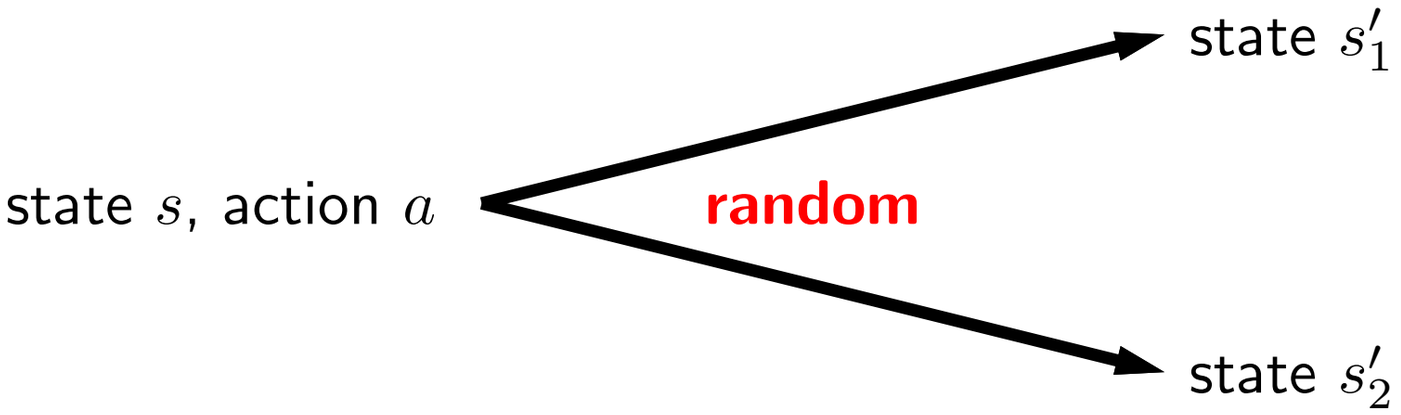
deterministic

state s, action a ⟶ state Succ(s, a)



- Last week, we looked at search problems, a powerful paradigm that can be used to solve a diverse range of problems ranging from word segmentation to package delivery to route finding. The key was to cast whatever problem we were interested in solving into the problem of finding the minimum cost path in a graph.
- However, search problems assume that taking an action a from a state s results **deterministically** in a unique successor state Succ(s, a).

Uncertainty in the real world



- In the real world, the deterministic successor assumption is often unrealistic, for there is **randomness**: taking an action might lead to any one of many possible states.
- One deep question here is how we can even hope to act optimally in the face of randomness? Certainly we can't just have a single deterministic plan, and talking about a minimum cost path doesn't make sense.
- Today, we will develop tools to tackle this more challenging setting. We will fortunately still be able to reuse many of the intuitions about search problems, in particular the notion of a state.

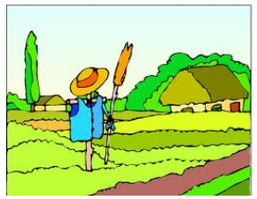
History

- MDPs: Mathematical Model for decision making under uncertainty.
- MDPs were first introduces in 1950s-60s.
- Ronald Howard's book on Dynamic Programming and Markov Processes
- The term 'Markov' refers to Andrey Makov as MDPs are extensions of Markov Chains, and they allow making decisions (taking actions or having choice).

Applications

Robotics: decide where to move, but actuators can fail, hit unseen obstacles, etc.

Resource allocation: decide what to produce, don't know the customer demand for various products



Agriculture: decide what to plant, but don't know weather and thus crop yield

- Randomness shows up in many places. They could be caused by limitations of the sensors and actuators of the robot (which we can control to some extent). Or they could be caused by market forces or nature, which we have no control over.
- We'll see that all of these sources of randomness can be handled in the same mathematical framework.

Volcano crossing

Run (or press ctrl-enter)

		-50	20
		-50	
2			

- Let us consider an example. You are exploring a South Pacific island, which is modeled as a 3x4 grid of states. From each state, you can take one of four actions to move to an adjacent state: north (N), east (E), south (S), or west (W). If you try to move off the grid, you remain in the same state. You start at (2,1). If you end up in either of the green or red squares, your journey ends, either in a lava lake (reward of -50) or in a safe area with either no view (2) or a fabulous view of the island (20). What do you do?
- If we have a deterministic search problem, then the obvious thing will be to go for the fabulous view, which yields a reward of 20. You can set `numIters` to 10 and press `Run`. Each state is labeled with the maximum expected utility (sum of rewards) one can get from that state (analogue of FutureCost in a search problem). We will define this quantity formally later. For now, look at the arrows, which represent the best action to take from each cell. Note that in some cases, there is a tie for the best, where some of the actions seem to be moving in the wrong direction. This is because there is no penalty for moving around indefinitely. If you change `moveReward` to -0.1, then you'll see the arrows point in the right direction.
- In reality, we are dealing with treacherous terrain, and there is on each action a probability `slipProb` of slipping, which results in moving in a random direction. Try setting `slipProb` to various values. For small values (e.g., 0.1), the optimal action is to still go for the fabulous view. For large values (e.g., 0.3), then it's better to go for the safe and boring 2. Play around with the other reward values to get intuition for the problem.
- Important: note that we are only specifying the dynamics of the world, not directly specifying the best action to take. The best actions are computed automatically from the algorithms we'll see shortly.

Roadmap

Modeling

Modeling MDP Problems

Algorithms

Policy Evaluation

Value Iteration

Learning

Intro to Reinforcement Learning

Model-Based Monte Carlo

Model-Free Monte Carlo

SARSA

Q-learning

Epsilon Greedy

Function Approximation