Last Time:

- ☐ Intro to embodied AI safety

Lecture 1

EAIS SP'25

Andrea Bajcsy

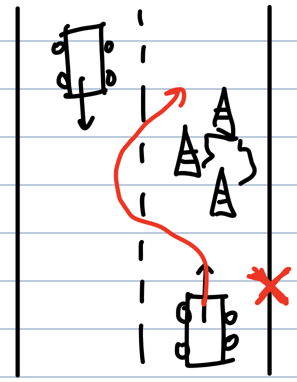
This Time:

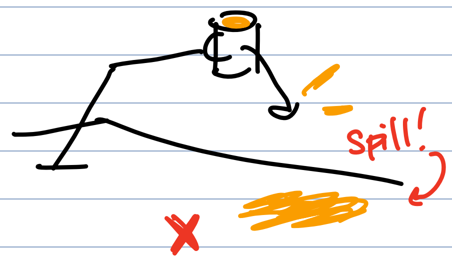
- ☐ sequential decision-making
- ☐ Dynamic Programming
- ☐ MPC

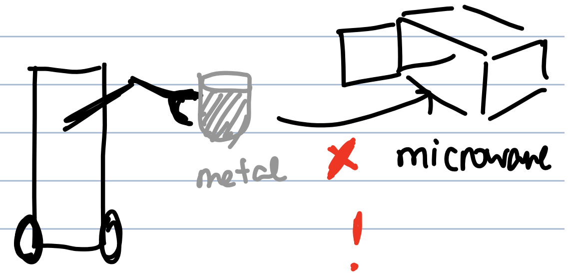
⊛ make sure to fill out Ken's OH when2meet!

<u>Sequential Decision-Making</u> - b/c the first half of this class is all about safety w.r.t. <u>decisions</u>.

<u>e.g.</u> I want an embodied AI (EAI) system to make "good" decisions and prevent "bad" outcomes.







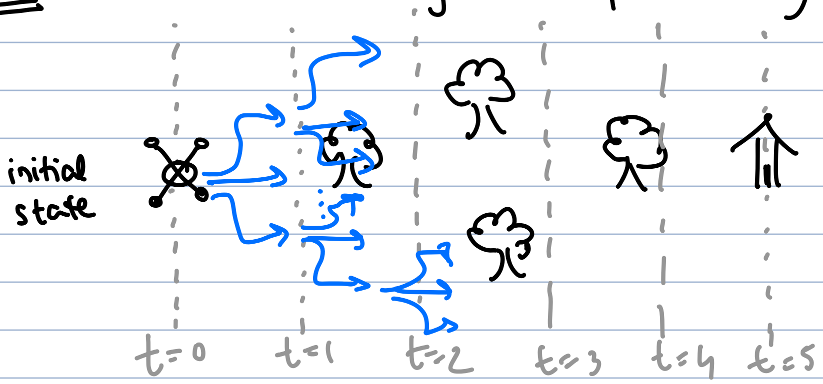
Our goal is to

- mathematically <u>model</u> decision-making
- <u>quantify</u> the "goodness" of decisions
- <u>compute</u> these good decisions

This is where sequential decision-making (i.e. control theory) will provide us with a framework for answering these!

Q What makes sequential decision-making <u>hard</u>?

A naive solution grows exponentially in time horizon



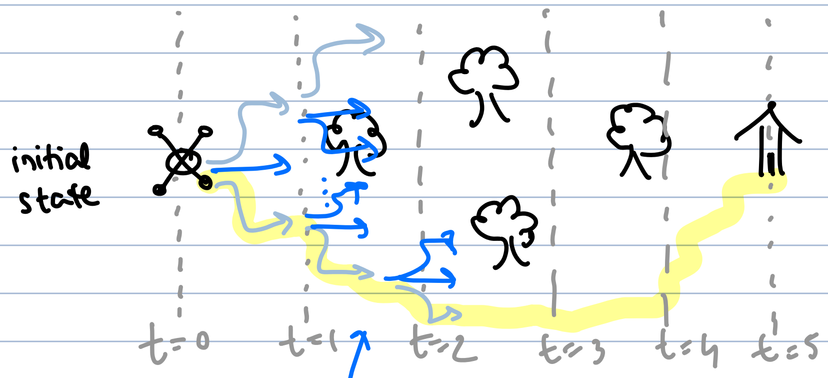
$|A| = 3$

starting just from one state

$|A|^{T+1}$

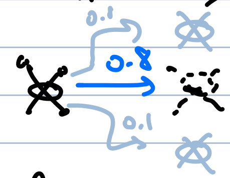
possible sequences of decisions

$\boxed{A2}$ outcomes of taking actions can be stochastic (or unknown)



some paths / states have lower probability!

$|A| = 3$

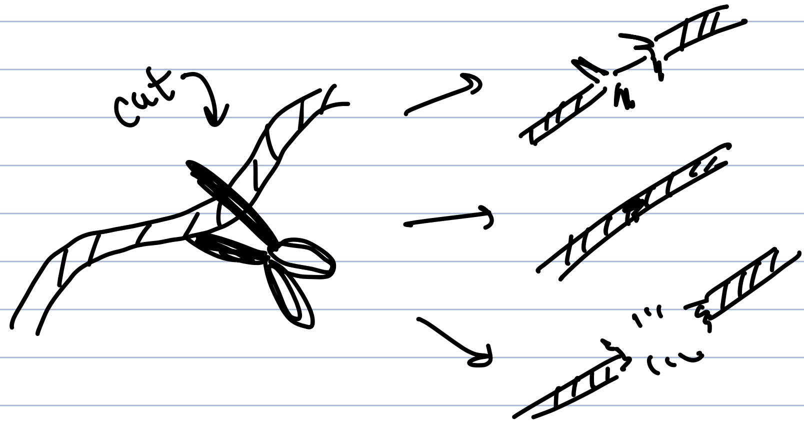


starting just from one state

$|A|^{T+1}$

possible sequences of decisions

$\boxed{A3}$ Some aspects of the world are hard to represent + predict



One framework that can help us describe these phenomenona are

state space reprepresentations.

state : $x(t) \in \mathbb{R}^n$ ← continuous time

$x_t \in \mathbb{R}^n$ ↖ discrete time (often written compactly as x) / s in the AI lit.

describes the minimal necessary characteristics of a system

e.g. positions of a 2D system, speed, orientation.

control / action : $u(t) \in \mathbb{R}^m$ / a in the AI lit.

$u_t \in \mathbb{R}^m$

inputs that we choose @ each instance in time.

e.g. joint torques, acceleration

output/observation: $y(t) \in \mathbb{R}^L$
$y_t \in \mathbb{R}^L$ / o in AI lit.

outputs that are actually measurable (typically through some sensor. ex. position through GPS, image observations via RGB camera)

! initially, we assume $y \equiv x$ but this is often not the case and this is a safety challenge we will return to.

in ctrl. ↘ ↙ in AI

dynamics/transition: how the system evolves over time

	continuous-time ($t \in \mathbb{R}$)	discrete time ($t \in \mathbb{Z}$)
deterministic	$\dot{x}(t) = f(x(t), u(t))$	$x_{t+1} = f(x_t, u_t)$
stochastic	$dX = f(X, u)dt + \underbrace{dW(t)}_{\text{wiener process}}$	$x_{t+1} \sim P(x_{t+1} \mid x_t, u_t)$

Formalizing Optimal Decision-Making

minimize Objective → "AI Alignment"
decisions

s.t. system dynamics
other constraints

ctrl bounds
state constraints

Uncertainty Quant.

Safe ctrl

An optimization problem but it has a temporal aspect to it and we want a sequence of decisions that are optimal.

Discrete-time:

total cost starting from x_0 & applying $u_{0:T-1}$ — running cost — terminal cost

$$\min_{u_{0:T-1}} J(x_0, u_{0:T-1}) := \sum_{t=0}^{T-1} L(x_t, u_t) + \ell(x_T)$$

s.t. $x_{t+1} = f(x_t, u_t)$

$\underline{u} \le u_t \le \bar{u}, \quad \forall t \in \{0, \dots T-1\}$

Continuous-time:

total cost starting from x_0 & applying $u_{0:T-1}$ — running cost — terminal cost

$$\min_{u(\cdot)} J(x(0), u(\cdot)) := \int_{t=0}^{T} L(x(t), u(t)) + \ell(x(T))$$

ctrl signals

s.t. $\dot{x}(t) = f(x(t), u(t))$

$\underline{u} \le u(t) \le \bar{u}, \quad \forall t \in [0, T]$

$\boxed{\text{BIG Q: How to solve?}}$

Each method listed below has its pros / cons -- BUT there are MANY solution strategies we can try! :)

1) Calculus of variations { "convert constrained opt. → unconstrained via Lagrange multipliers

2) Model Predictive Control (MPC) { like (1) but w/ replanning

3) Dynamic Programming ← NOW :)

4) Reinforcement Learning { shares foundation w/ (3) but often studies unknown dyns. & uses modern sim., func. approx, tool..

Dynamic Programming

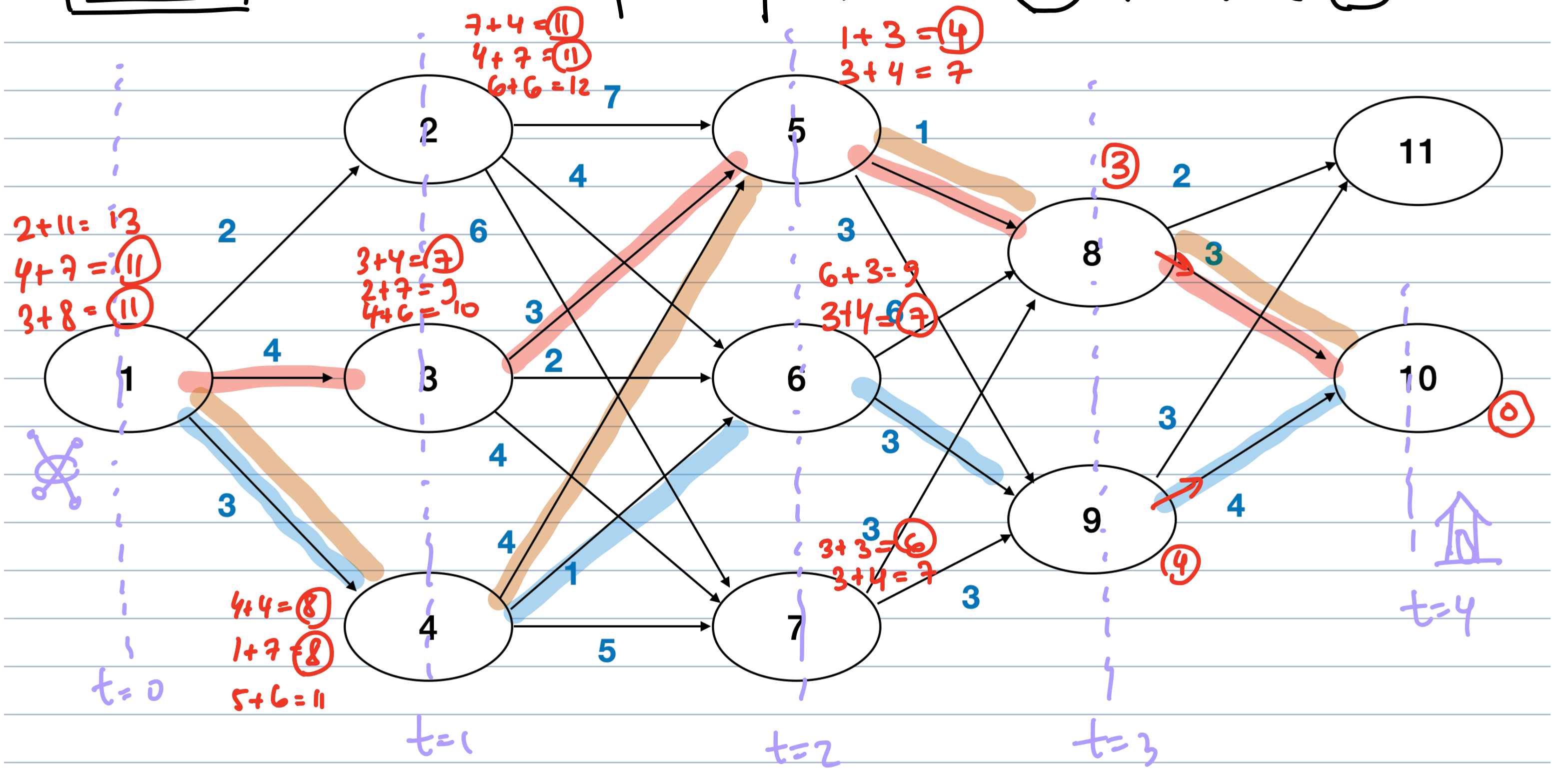
We will study the method of dynamic programming (D.P.) to solve optimal ctrl / sequential decision-making. DP relies on the principle of optimality developed by Richard Bellman around 1958 when he was working @ the RAND corporation. Since then, it has been used in C.S, operations research, controls, robotics, and more!

> **Bellman explains the name**
>
> The 1950s were not good years for mathematical research. We had a very interesting gentleman in Washington named Wilson. He was Secretary of Defense, and he actually had a pathological fear and hatred of the word, research... His face would suffuse, he would turn red, and he would get violent if people used the term, research, in his presence. You can imagine how he felt, then, about the term, mathematical... I thought dynamic programming was a good name. It was something not even a Congressman could object to.

But first, lets intuite dynamic programming by looking @ a toy example & thinking through its solution. How did you solve it "naively"? What was your strategy for a "better" solution? You may have intuited DP....

(exercise below)

Exercise Find shortest path from node 1 to node 10



3 optimal paths from 1 → 10!

A few key properties of D.P.:

- DP gives you optimal path from all nodes to 10 so you get intermediate sol^{n} "for free".
- Because it explores all intermediate nodes, it gives you globally optimal solution.
- DP gives you significant computational advantages over forward simulation.

From Sutton & Barto, "Reinforcement Learning: An Intro"

4.7 Efficiency of Dynamic Programming

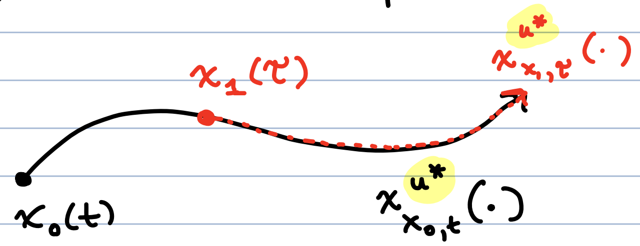
DP may not be practical for very large problems, but compared with other methods for solving MDPs, DP methods are actually quite efficient. If we ignore a few technical details, then the (worst case) time DP methods take to find an optimal policy is polynomial in the number of states and actions. If n and m denote the number of states and actions, this means that a DP method takes a number of computational operations that is less than some polynomial function of n and m. A DP method is guaranteed to find an optimal policy in polynomial time even though the total number of (deterministic) policies is m^n. In this sense, DP is exponentially faster than any direct search in policy space could be, because direct search would have to exhaustively examine each policy to provide the same guarantee. Linear programming methods can also be used to solve MDPs, and in some cases their worst-case convergence guarantees are better than those of DP methods. But linear programming methods become impractical at a much smaller number of states than do DP methods (by a factor of about 100). For the largest problems, only DP methods are feasible.

DP is sometimes thought to be of limited applicability because of the *curse of dimensionality* (Bellman, 1957a), the fact that the number of states often grows exponentially with the number of state variables. Large state sets do create difficulties, but these are inherent difficulties of the problem, not of DP as a solution method. In fact, DP is comparatively better suited to handling large state spaces than competing methods such as direct search and linear

Let's understand underlying mathematical principle. DP relies on

$\boxed{\text{Principle of optimality: "In an optimal sequence of decisions or choices, each subsequence must also be optimal. Thus, if we take any state along the opt. state trajectory, then the remaining subtrajectory is also optimal"}}$

In the example earlier, if we take any intermediate node along the optimal route, we still take optimal route to destination.



Let's write this principle down mathematically:

We want to <u>solve</u>:

$$\underbrace{V_t(x_t)}_{} := \min_{u_{t:T-1}} J_t(x_t, u_{t:T-1})$$

← rest of ctrls

← opt. ctrl. problem from before $= \sum_{\tau=t}^{T-1} L(x_\tau, u_\tau) + \ell(x_T)$

let's define V_t as storing the best cost value "cost-to-go" from x @ time t
⇒ "value function"

Let's expand out the RHS over time w/ our cost function:

depends on (x_t, u_t)

$$V_t(x_t) := \min_{u_{t:T-1}} \Big\{ L(x_t,u_t) + \underbrace{L(x_{t+1},u_{t+1}) + \cdots L_{T-1}(x_{T-1},u_{T-1}) + \ell(x_T)}_{\text{by def this is } J_{t+1}(x_{t+1},\, u_{t+1:T-1})} \Big\}$$

$$= \min_{u_{t:T-1}} \Big\{ L(x_t,u_t) + J_{t+1}(x_{t+1},\, u_{t+1:T-1}) \Big\}$$

principle of optimality!

$$= \min_{u_t} \Big\{ L(x_t,u_t) + V_{t+1}(x_{t+1}) \Big\}$$

← Just restrict ourselves to optimal trajectories starting from next state x_{t+1}

↖ NO LONGER optimize over <u>sequence</u>, just over <u>current action</u>!

Bellman Equation (discrete time, determ. dyns)

$$V_t(x_t) = \min_{u_t} \big[L(x_t,u_t) + V_{t+1}(x_{t+1}) \big]$$

$$V_T(x_T) = \ell(x_T)$$