Lec28

Sunday, April 05, 2015 9:29 AM

Dynamic programming: Generic Formulation

Thursday, February 26, 2015 10:36 AM

Discrete-time DP

- System dynamics

$$x_{k+1} = f_k(x_k, u_k, w_k), \quad k = 0, 1, \dots, N-1$$

  - k : discrete time (finite-horizon for now)
  - x_k : state, in the "state space" S
  - u_k : control, in the space C_k
    - u_k may be constrained to take values in a subset $U(x_k) \subset C_k$
  - w_k : random input / disturbance / noise
    - w_k may depend on x_k & u_k
      $P(\cdot \mid x_k, u_k)$

      but not on values of prior disturbance $w_0, w_1, \dots w_{k-1}$

  - "Markov": conditioned on current state x_k and the control u_k, the future states is independent from the past states / controls / noises before k.

  - Alternatively, we can think of w_k as defining a conditional probabilistic distribution
    $P(x_{k+1} \mid x_k, u_k)$.

- Cost function:

- $g_k(x_k, u_k, w_k)$: cost at stage k.
- $g_N(x_N)$: "terminal" cost
- Would like to choose the control $u_0, u_1, \ldots u_N$ to minimize

$$E\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \right]$$

- The expectation is taken w.r.t. the random distribution of $w_0, w_1, \ldots$
- Thanks again to the Markov property, we can define

$$g_k'(x_k, u_k) = E\left[g_k(x_k, u_k, w_k) \mid x_k, u_k \right]$$

Then our objective becomes

$$E\left[g_N(x_N) + \sum_{k=0}^{N-1} g_k'(x_k, u_k) \right]$$

- One of these two versions may be more convenient than the other, depending on whether w_k has a clear physical meaning.

Going back to our pricing problem

Thursday, November 2, 2023 3:57 PM

- The state x_k: the current queue length
  $x_k = 0, 1, 2, \ldots$
- The control u_k: need to discretize the price
  $u_k = 0, 0.1, 0.2, \ldots, 1$
- The randomness ω_k:
  - Arrival is random with prob. $\lambda(u_k)$
  - Service is random with prob. 0.2
- The system dynamics $x_{k+1} = f(x_k, u_k, \omega_k)$
  - $x_{k+1} = x_k + 1$ if $\omega_k = 1$ arrival, 0 service
    - This occurs with prob. $\lambda(u_k)(1 - 0.2)$.
  - $x_{k+1} = x_k - 1$ if $\omega_k = 0$ arrival, 1 service
    - This occurs with prob. $[1 - \lambda(u_k)] \cdot 0.2$
    - Only applies if $x_k \geq 1$
  - $x_{k+1} = x_k$ if $\omega_k = 1$ arrival, 1 service
    or $\omega_k = 0$ arrival, 0 service
    - This occurs with prob.
      $$\lambda(u_k) \times 0.2 + [1 - \lambda(u_k)] \cdot [1 - 0.2]$$
  - In summary, the dynamics are some probability laws of state transition, given u_k & x_k.

laws of state transition, given u_k & x_k.

- The cost, or more correctly, the payoff g:
  - If $x_k < 9$,
    - $g_k(x_k, u_k, w_k) = u_k$ if $w_k = 1$ arrival

      $\uparrow$ revenue from one new arrival
      - This happens with prob. $\lambda(u_k)$.
    - $g_k(x_k, u_k, w_k) = 0$ otherwise
    - This also describes a probabilistic distribution.
    - Note that $E[g_k(x_k, u_k, w_k) | x_k, u_k] = u_k \lambda(u_k)$.
  - If $x_k = 9$
    - $u_k = 1$
    - Then $g_k(x_k, u_k, w_k) = 0$.
  - The terminal payoff can be taken as 0.
- Our goal is then to maximize the total expected payoff.
- In summary, MDP is usually needed when we want to impose more stringent performance requirements, which necessitates the consideration of dynamic policies.

Key Features

① Uncertainty in the system evolution/outcome.

② Decisions are in stages

- At each stage k, only the current/past inputs $\omega_0, \omega_1, \ldots \omega_{k-1}$ are revealed
- Control u_k chosen with knowledge of the current state x_k
- ω_k is "revealed" after u_k is chosen.

③ Current decision affects future evolution/outcomes

④ Cannot reverse past decisions.

- "Causal"
- "Non-anticipatory"

⑤ Need to balance current payoff

$$g_k(x_k, u_k, \omega_k)$$

and future payoffs

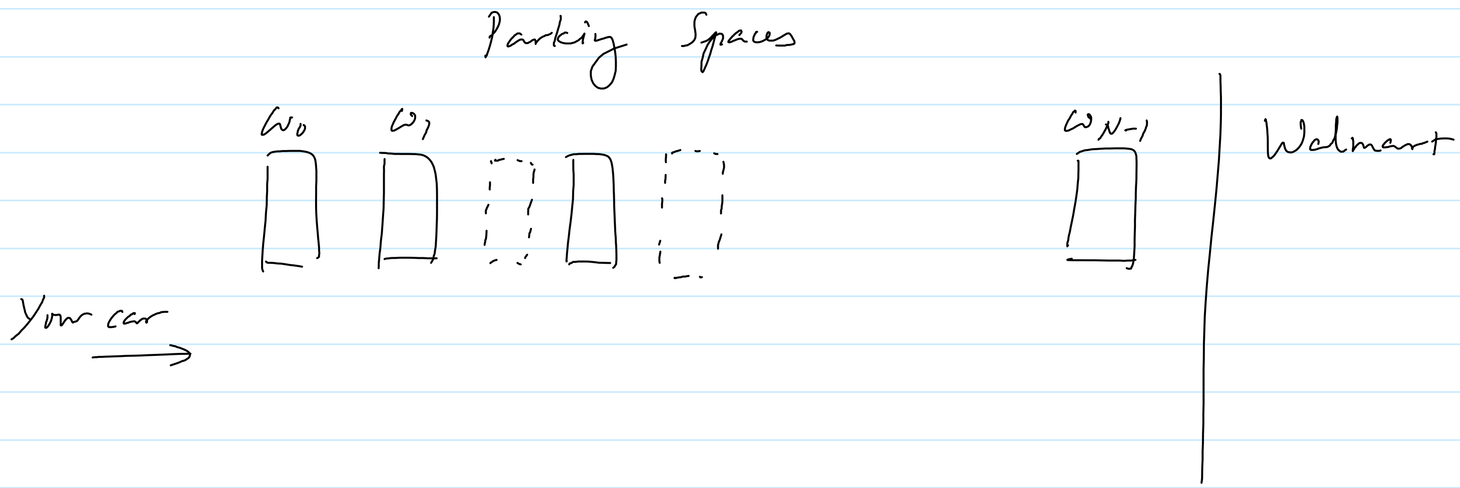
$$\sum_{j=k+1}^{N-1} (x_j, u_j, \omega_j)$$

- How can we solve such MDP?

Another simple example - skip

Thursday, February 26, 2015 10:55 AM

Simplest example



- Suppose you approach the parking lot, and you would like to park as close to the store as possible
- However, you can only see whether the parking spot right next to you is empty or not
  - And you have to make a decision right away whether to park there
- You cannot back off: Someone else may be behind you!
- While this is a simple example, it has the typical features of a DP problem

① Uncertainty of the system?

② Decisions are in stages?

③ Current decision affects future evolution?

④ Cannot reverse past decisions?

⑤ Need to balance current vs future payoffs?
(known) (unknown)

Let us now model it:

- k: You are examining the k-th spot.
- State x_k: 0 – k-th spot empty

  1 – k-th spot occupied

  T – terminated
- Control u_k: P – park here

  S – do not park here (skip)

  - u_k must be "S" if $x_k = 1$

- Input/Disturbance w_k
  - $w_k = 0$ if the next spot is empty
  - $w_k = 1$ if the next spot is occupied
  - Assume there is a distribution

$$P\{w_k = 0\} = 1 - p_k$$

$$P\{w_k = 1\} = p_k$$

- System dynamics

$x_{k+1} = f_k(x_k, u_k, w_k)$

- if $x_k = 1$, then u_k must "S"

  $x_{k+1} = w_{k+1}$

- If $x_k = 0$, then u_k may be "P" or "S"
  - If $u_k =$ "S" (do not park)

    then $x_{k+1} = w_{k+1}$
  - What if $u_k =$ "P"?
    - Need to add a new state "Terminate"

      $x_{k+1} =$ Terminate.
    - All future states are "T" too.

- Cost:
  - I want to be as close to the store as possible
  - If $u_k =$ "P" & $x_k \neq$ terminate, then

    $g_k(x_k, u_k, w_k) = N - k$

    If $u_k =$ "S" or $x_k =$ terminate

    $g_k(x_k, u_k, w_k) = 0$

  - What about x_N?
    - $x_N \neq T$:

      Did not park in any available spots!
    - $g_N(T) =$ large penalty

      "have to turn around"

What would be a reasonable "optimal" solution?

- Park if the available spot is "reasonably" close to the store. Skip otherwise.
- "What is reasonably close" depends on the system parameters
  - p_k
  - May not be easy to estimate in practice.
- Will the optimal policy still be of this structure if the w_k's are not independent?

Important !!!

- Need a probabilistic knowledge of the future uncertainty.
  - May not always easy to describe
- "Markovian" Structure.
  - Random input must be somehow independent.

Can you think of other examples?

- Investment in a stock market

- Inventory management

MDP is a linear program

Thursday, November 2, 2023 4:15 PM

- Somewhat counter-intuitively, MDP can actually be written as a linear program, and hence is a convex problem!

- In the pricing example
  - Consider a finite # of steps N.
  - For any given policy, we can imagine that, at step k, we will land at state $x_k = x$ and use control $u_k = u$ with some probability.
  - Let this prob. be y^k_{xu}.
  - Then this set of variables should satisfy some kind of balance eqn from step k to step $k+1$

$$\sum_u y^{k+1}_{xu} = \sum_{x', u'} y^k_{x'u'} \cdot \underbrace{P\left[\begin{array}{c}\text{The state goes} \\ \text{to } x \text{ at step} \\ k+1\end{array} \middle| \begin{array}{c}\text{The state is} \\ x' \text{ at step } k, \\ \text{\& control is } u'\end{array}\right]}_{P^k_{x', u', x, u}} \qquad (*)$$

    - For example, if $x' = x-1$, then

$$P^k_{x', u', x, u} = P[1 \text{ arrival}, 0 \text{ service}]$$
$$= \underbrace{\lambda(u')}_{\text{just a constant given } u'}.$$

    - Hence, $(*)$ is a linear equality constraint.

- We must also have

- $y^k_{xu} \geq 0$
- $\sum_{xu} y^k_{xu} = 1$ for all k (**).

- Our optimization then becomes

$$\max \sum_{k=1}^{N} \sum_{x,u} y^k_{xu} \cdot u \cdot \lambda(u)$$

$$\text{sub to} \quad (*) \ \& \ (**)$$

  - Which is precisely a linear program!

- However, this linear program can be quite cumbersome!
  - As N increases, the # of variables and constraints will increase
  - It will be increasingly more difficult to solve them!

- Instead, the methods that we will talk about below will be more efficient because they take advantage of a key structure property of MDP

- Nonetheless, the above linear program view can still be quite useful, as it allows us to utilize properties of convex problems when we need.
  - We will see so later when we discuss constrained MDP & index policies.

Basic solution principle

Thursday, February 26, 2015 11:42 AM

Principle of optimality:

- Nearly all DP techniques/solutions are based on a simple "principle of optimality"
- Let μ_i be the "policy" of choosing u_i based on x_i.
- Suppose $\pi^* = \{\mu_0^*, \mu_1^*, \cdots, \mu_{N-1}^*\}$ is the optimal policy,
- Assume that when using π^*, a given state x_i occurs at state i with some positive probability.
- Consider the "tail problem":
  - Starting from x_i at time i, how to minimize the "cost-to-go" from time i to N

$$E\left\{ g_N(x_N) + \sum_{k=i}^{N-1} g_k\left(x_k, \mu_k(x_k), w_k\right) \right\}$$

Principle of Optimality:

- The tail policy $\{\mu_i^*, \mu_{i+1}^*, \cdots \mu_N^*\}$ must also be optimal for the tail-subproblem.
- (Also called the "Dynamic Programming" Principle.)

- The justification is simple:
  - If $\{\mu_i^*, \mu_{i+1}^*, \cdots \mu_N^*\}$ was not optimal for

the tail problem starting from x_i, then we should be able to reduce the overall cost by switching to an optimal policy for the tail problem every time when we reach x_i at time i.

- This will not affect past costs (before time i)
- It for sure reduces the future costs starting from state x_i, since future evolution is indep. of the past conditioned on state x_i.

- Analogy:
  - If the shortest route from LaF to DC is through Indy
  - Then the part of the route from Indy to DC must also be the shortest possible.

Backward Induction

Tuesday, November 7, 2023 9:40 AM

- The principle of optimality suggests that an optimal policy can be constructed "backwards"

- First, solve ALL tail problems at the last stage
  - Each tail problem starts with a different x_{N-1}

- Then, solve ALL tail problems at the second-to-last stage, using the optimal solutions and minimal cost-to-go from the last stage

- And go on until we reach the first stage.

- Known as "Backward Induction".

- This is most easily seen in the "Deterministic" Setting (i.e., no randomness w_k)

Deterministic problem

Monday, March 02, 2015 9:42 AM

Deterministic Shortest Path Problem

- $\{1, 2, \cdots, N\}$: nodes of a graph
  - t is the destination $\in \{1, 2, \cdots N\}$
- a_{ij} >0: cost of moving from node i to j
- Find a shortest path (minimum cost) from each node i to destination t.
- Note that an optimal path should not take more than N moves

  $\rightarrow$ N-stage DP.

---

① Consider the last stage $N-1$

- If starting from node i, the only decision is to go to t directly
- The cost-to-go for stage N is

$$J_{N-1}(i) = a_{it}, \quad i = 1, 2, \cdots, N$$

$$J_{N-1}(t) = 0$$ (Not needed if we assume $a_{ii} = 0$ for all i)

② Now consider the stage $N-2$

- If starting from node i, we want to find the optimal 2-hop path to reach t.
- Suppose that such a path first goes through j,

$j = 1, 2, \cdots, N$

- The cost in the first hop is a_{ij}
- The "optimal" cost for the second hop must be $J_{N-1}(j)$
- Hence, we should pick j that minimizes

$$a_{ij} + J_{N-1}(j)$$

- The "cost-to-go" for the stage $N-2$ is then

$$J_{N-2}(i) = \min_{j=1,2,\cdots N} \{ a_{ij} + J_{N-1}(j) \}$$

③ This procedure then continues. At stage k,

$$J_k(i) = \min_{j=1,2,\cdots,N} \{ a_{ij} + J_{k+1}(j) \}$$

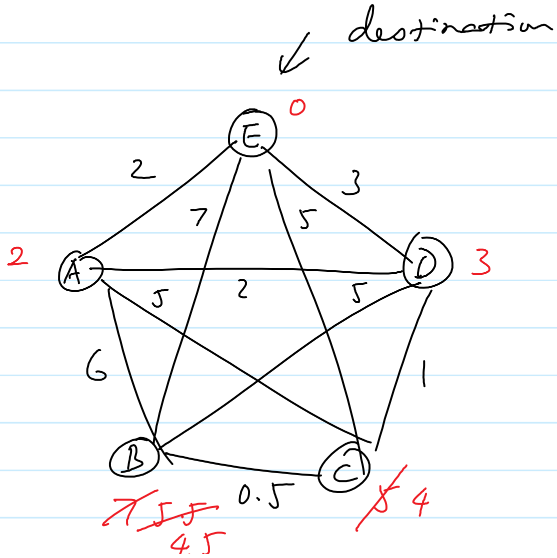
④ The "final" optimal cost is $J_0(i)$, and is equal to the shortest path from $i \to t$.

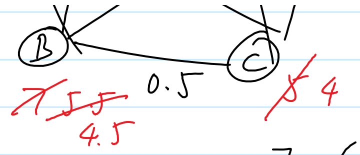
$\Rightarrow$ Bellman-Ford Algorithm

---

skip

Example :





	$J_k(i)$				
k	$i = A$	B	C	D	E
$N-1 = 3$	2	7	5	3	0
2	2	5.5	4	3	0
1	2	4.5	4	3	0
0	2	4.5	4	3	0

What is the path from B → E ?

- At $k=0$, $u_0^*(B) = C$
- At $k=1$, $u_1^*(C) = D$
- At $k=2$, $u_2^*(D) = E$
- At $k=3$, $u_3^*(E) = E$ (stay)

Hence, the path $B - C - D - E - E$ is the shortest.

(Alternately, $B - B - C - D - E$ is also the shortest. Essentially the same.)

For deterministic problems, the entire sequence of decisions $u_0, u_1, \cdots u_{N-1}$ can be calculated before-hand

- "policy" = "decision"

- "policy" = "decision"
- not so for random systems.