AA203 Optimal and Learning-based Control

Lecture 5

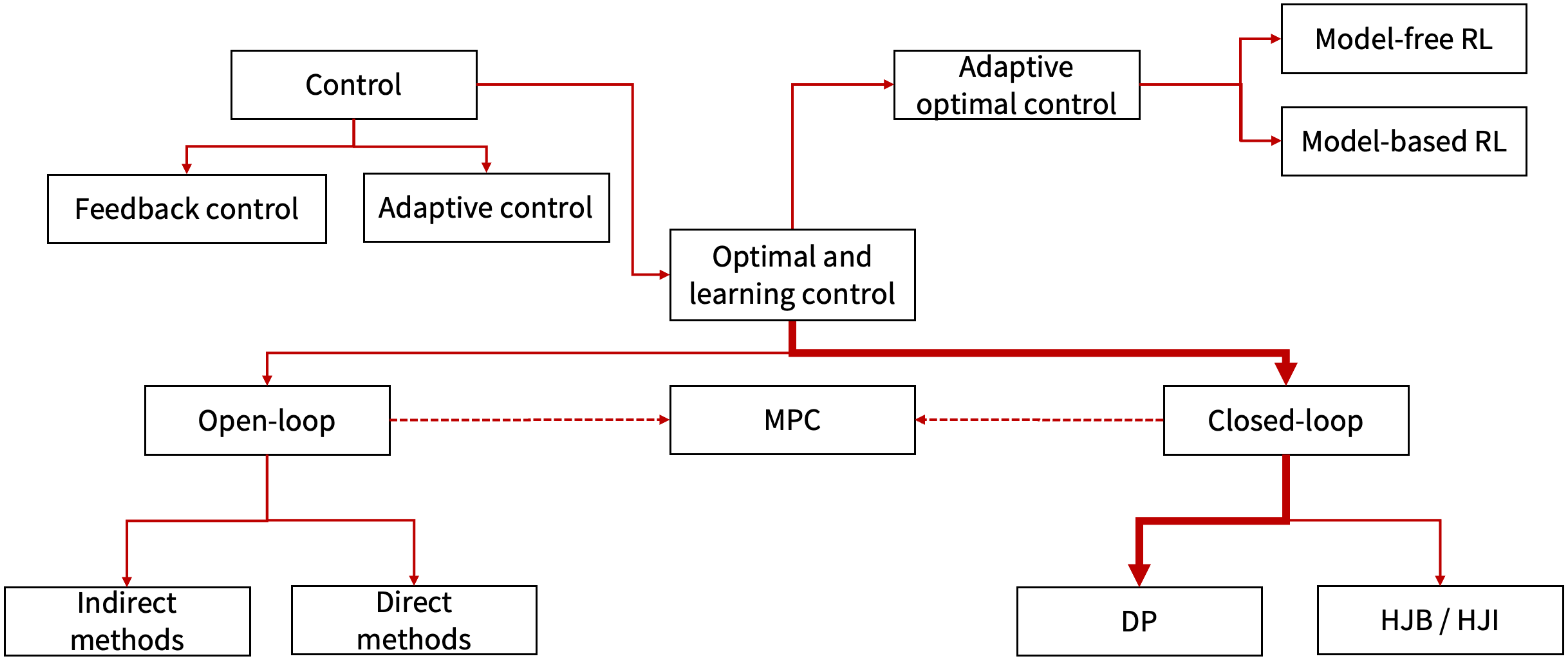
Dynamic Programming

Autonomous Systems Laboratory
Daniele Gammelli

Stanford University

Autonomous Systems Laboratory
Stanford Aeronautics & Astronautics

Roadmap



Control
Feedback control
Adaptive control
Optimal and learning control
Adaptive optimal control
Model-free RL
Model-based RL
Open-loop
MPC
Closed-loop
Indirect methods
Direct methods
DP
HJB / HJI

Outline of the next two lectures

Intro to dynamic programming (DP) and *principle of optimality*

The dynamic programming algorithm

Dynamic programming in control:

- Discrete LQR

This lecture

- Stochastic Optimal Control Problem / Markov Decision Process (MDP): Stochastic LQR
- Policy Iteration and Value Iteration

Dynamic Programming

A method for solving complex problems, by:

- Breaking them down into subproblems
- Combining solutions to subproblems

Dynamic Programming is a very general solution method for problems which have two properties:

- Optimal substructure (*Principle of optimality* applies)
  - Optimal solution can be decomposed into subproblems, e.g., shortest path

- Overlapping subproblems
  - Subproblems recur many times
  - Solutions can be cached and reused

Other applications of Dynamic Programming

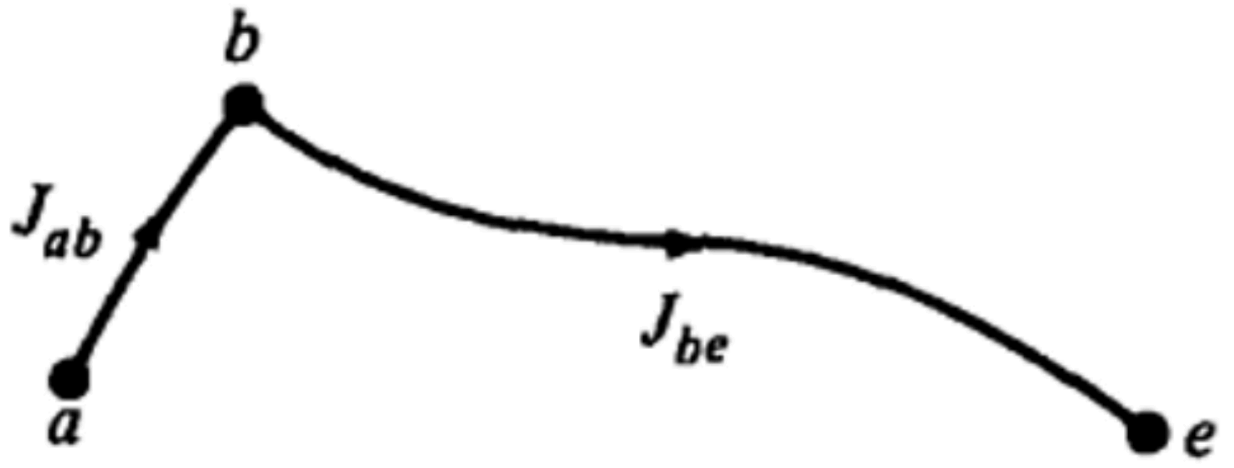
Dynamic programming is used across a wide variety of domains, e.g.

- Scheduling algorithms
- Graph algorithms (e.g., shortest path algorithms)
- Graphical models in ML (e.g., Viterbi algorithm)
- Etc.

Principle of optimality

The *key* concept behind the dynamic programming approach is the *principle of optimality*

Suppose the optimal path for a multi-stage decision-making problem with additive cost structure is



Multi-stage:

- First decision yields segment a-b with cost J_{ab}
- Remaining decisions yield segments b-e with cost J_{be}

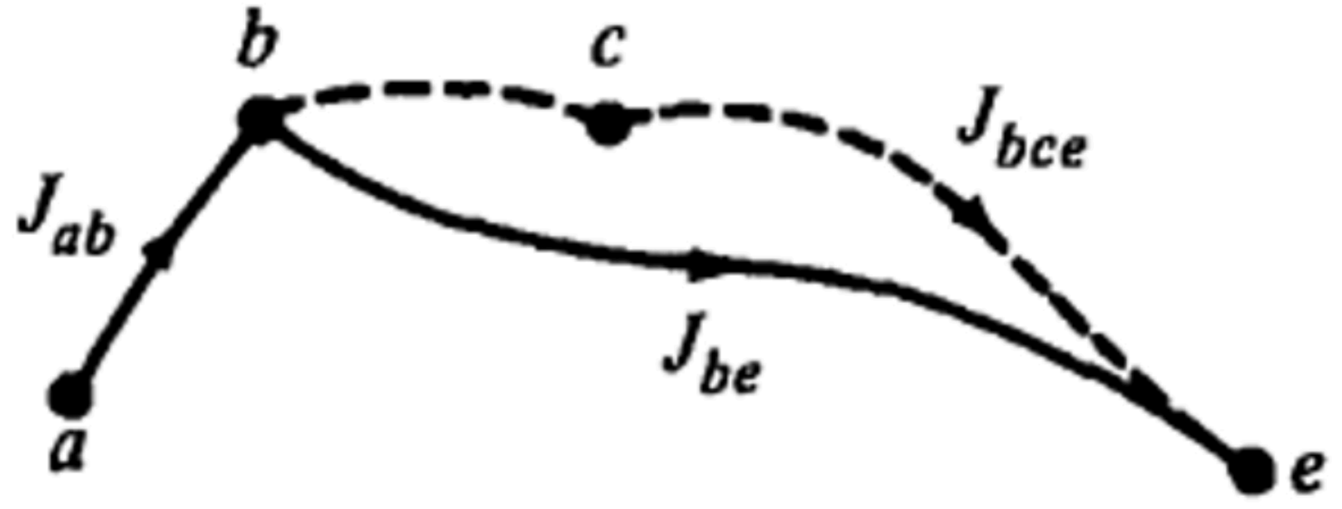
Additive cost:

- The optimal cost is then $J^*_{ab} = J_{ab} + J_{be}$

Principle of optimality

Claim: if a - b - e is the optimal path from a to e, then b - e is the optimal path from b to e

Proof: suppose b-c-e is the optimal path from b to e. Then



$$J_{bce} < J_{be}$$

$$J_{ab} + J_{bce} < J_{ab} + J_{be} = J^*_{ae}$$

Contradiction!

Principle of optimality

Principle of optimality (for discrete-time systems):

Let $\pi^* := \left\{\pi_0^*, \pi_1^*, \ldots, \pi_{N-1}^*\right\}$ be an optimal policy.

Assume state x_k is reachable.

Consider the subproblem whereby we are at x_k at time k and we wish to minimize the cost-to-go from time k to time N.

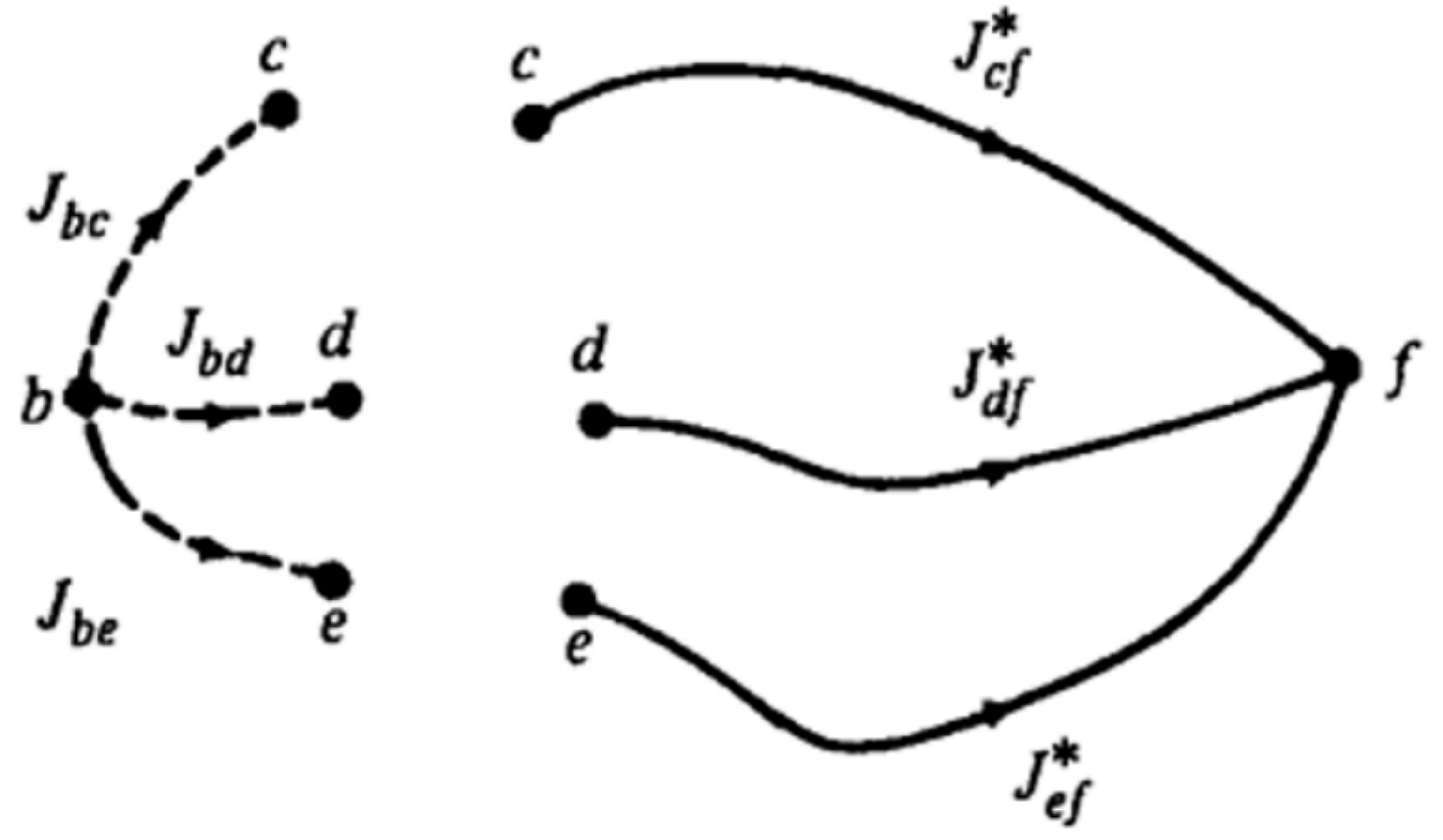
Then the truncated policy $\left\{\pi_k^*, \pi_{k+1}^*, \ldots, \pi_{N-1}^*\right\}$ is optimal for the subproblem.

Tail policies are optimal for **tail** subproblems

Notation: for brevity $\pi_k^*\left(\mathbf{x}_k\right) = \pi^*\left(\mathbf{x}_k, k\right)$

Applying the principle of optimality

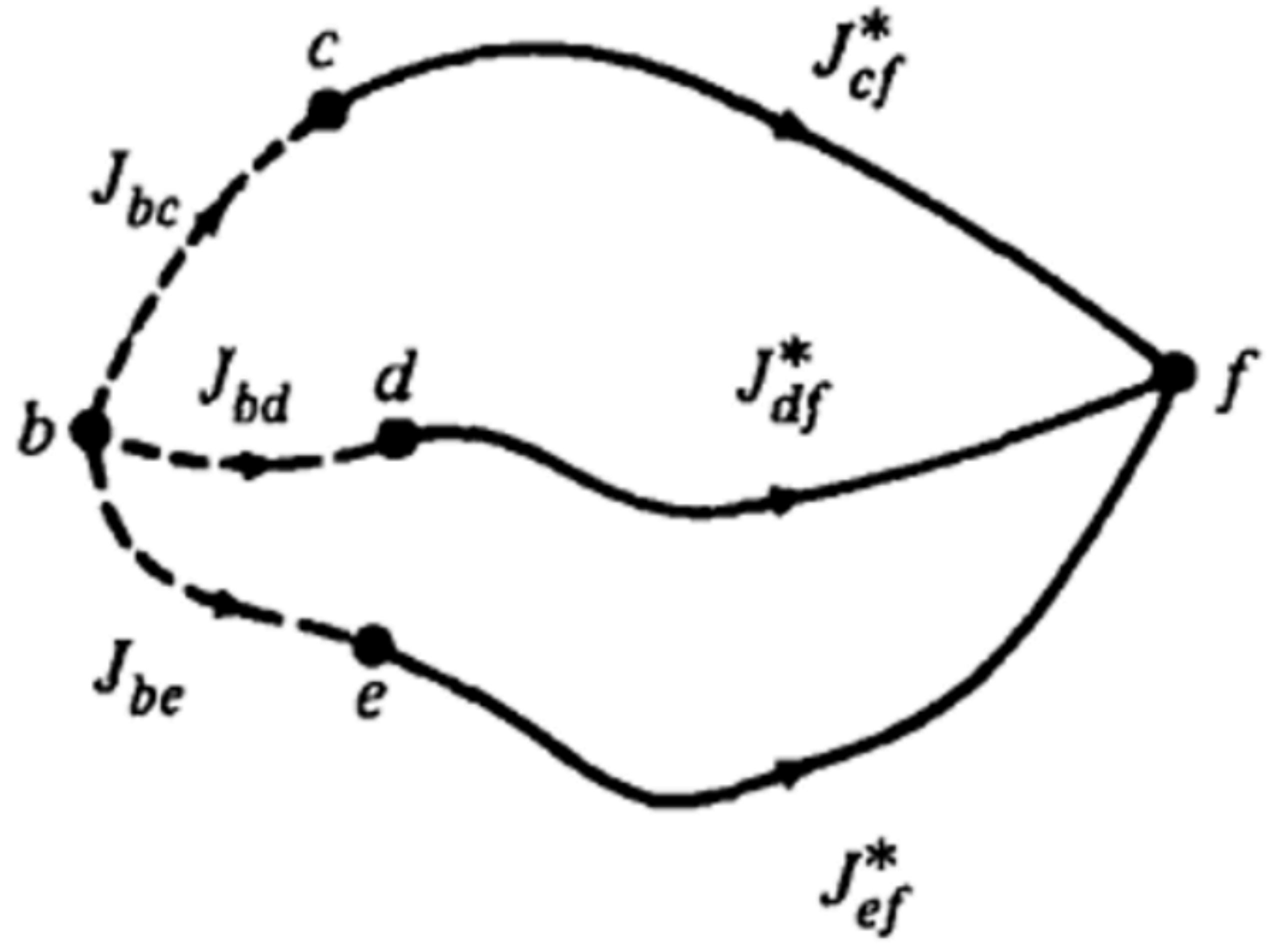
Consider the case where we want to find the optimal path from b to f, and that we know the cost of the optimal path from {c, d, e} to f.



The principle of optimality tells us that the optimal policy is comprised of optimal sub-policies

Hence, the optimal trajectory is found by comparing:

$$C_{bcf} = J_{bc} + J^*_{cf}$$
$$C_{bdf} = J_{bd} + J^*_{df}$$
$$C_{bef} = J_{be} + J^*_{ef}$$

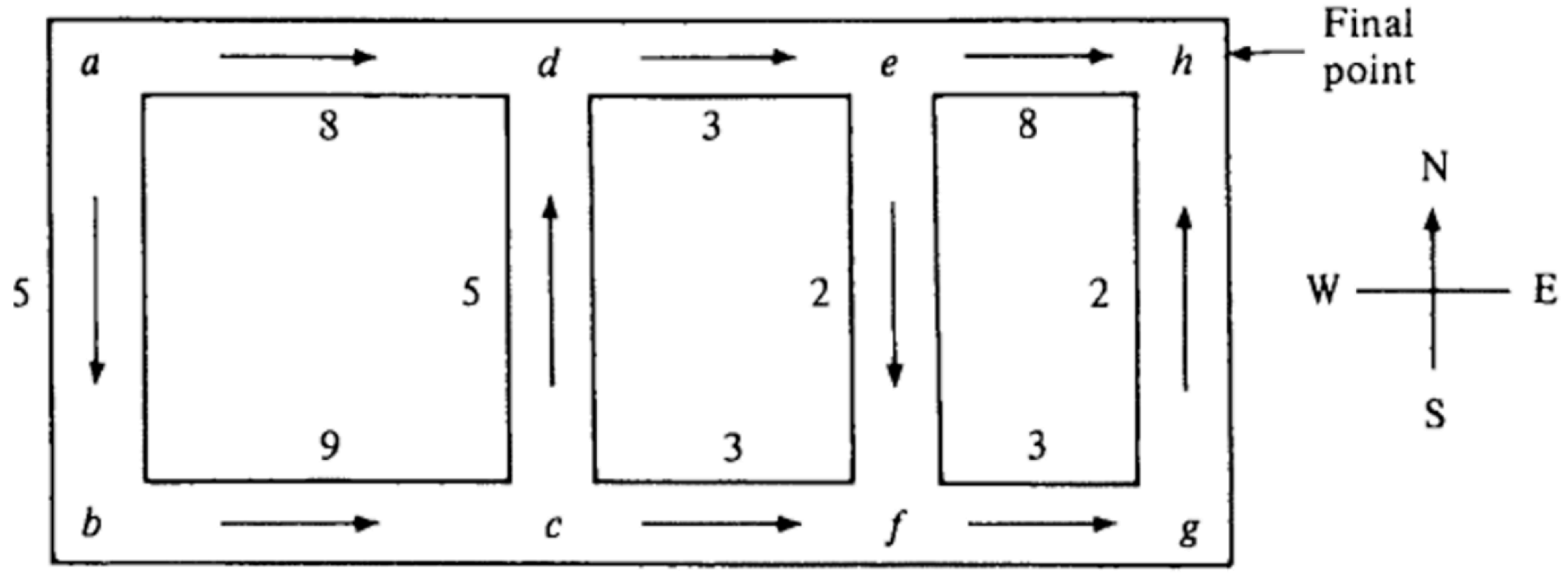


The “cost-to-go” allows us to only compute one-step look-ahead

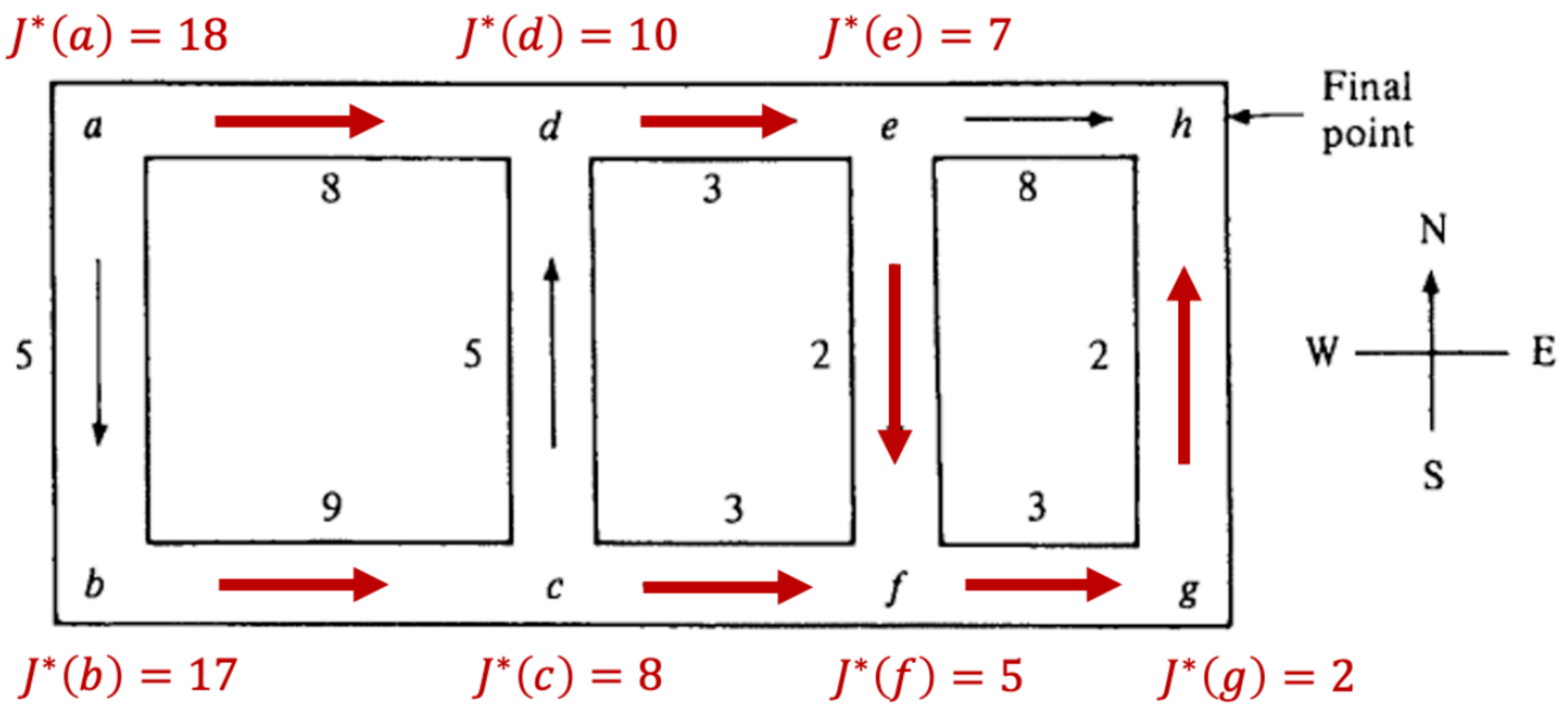
Applying the principle of optimality

- Need only to compare the concatenations of immediate decisions and optimal decisions → significant decrease in computation/possibilities
- In practice: carry out this procedure **backward** in time

Example



Example



$J^*(a) = 18$
$J^*(d) = 10$
$J^*(e) = 7$
a
d
e
h
Final point
8
3
8
5
5
2
2
9
3
3
b
c
f
g
N
W
E
S
$J^*(b) = 17$
$J^*(c) = 8$
$J^*(f) = 5$
$J^*(g) = 2$

DP Algorithm

Model: $\mathbf{x}_{k+1} = f\left(\mathbf{x}_k, \mathbf{u}_k, k\right), \quad \mathbf{u}_k \in U\left(\mathbf{x}_k\right)$

Cost: $J\left(\mathbf{x}_0\right) = h_N\left(\mathbf{x}_N\right) + \sum_{k=0}^{N-1} g\left(\mathbf{x}_k, \pi_k\left(\mathbf{x}_k\right), k\right)$

DP Algorithm:

For every initial state $\mathbf{x}_0$, the optimal cost $J^*(\mathbf{x}_0)$ is equal to $J_0^*(\mathbf{x}_0)$, given by the last step of the following algorithm, which proceeds backward in time from stage $N-1$ to stage 0:

$$J_N^*(\mathbf{x}_N) = h_N(\mathbf{x}_N)$$
$$J_k^*(\mathbf{x}_k) = \min_{\mathbf{u}_k \in U(\mathbf{x}_k)} g(\mathbf{x}_k, \mathbf{u}_k, k) + J_{k+1}^*(f(\mathbf{x}_k, \mathbf{u}_k, k)), \quad k = 0, \ldots, N-1$$

Furthermore, if $\mathbf{u}_k^* = \pi_k^*\left(\mathbf{x}_k\right)$ minimizes the right hand side of the above equation for each $\mathbf{x}_k$ and k, the policy $\left\{\pi_0^*, \pi_1^*, \ldots, \pi_{N-1}^*\right\}$ is optimal

Comments

- Discretization (from differential equations to difference equations)
- Quantization (from continuous to discrete state variables / controls)
- Guaranteed to converge to a global minimum
- Constraints, in general, simplify the numerical procedure
- Optimal control in **closed-loop** form
- Curse of dimensionality (both computationally and w.r.t. memory)
- Typically involves
  - *offline* computation of optimal costs (backward)
  - *online* planning through (forward) construction of solution

Outline

Intro to dynamic programming (DP) and *principle of optimality*

The dynamic programming algorithm

Dynamic programming in control:

- Discrete LQR
- Stochastic Optimal Control Problem / Markov Decision Process (MDP): Stochastic LQR
- Policy Iteration and Value Iteration

Discrete LQR

- Canonical application of dynamic programming for control
- One case where DP can be solved analytically (in general, DP algorithm must be performed numerically)

Discrete (Deterministic) LQR : Select control inputs to minimize

$$J_0\left(\mathbf{x}_0\right) = \frac{1}{2}\mathbf{x}_N^T Q_N \mathbf{x}_N + \frac{1}{2}\sum_{k=0}^{N-1}\left(\mathbf{x}_k^T Q_k \mathbf{x}_k + \mathbf{u}_k^T R_k \mathbf{u}_k + 2\mathbf{x}_k^T S_k \mathbf{u}_k\right)$$

Subject to dynamics

$$\mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k, \quad k \in \{0,1,\ldots,N-1\}$$

Assuming

$$Q_k = Q_k^T \succeq 0, \quad R_k = R_k^T \succ 0, \quad \begin{bmatrix} Q_k & S_k \\ S_k^T & R_k \end{bmatrix} \succeq 0 \quad \forall k$$

Extensions

Many important extensions, some of which we'll cover later in this class

- Cost with linear terms, affine dynamics: can consider today's analysis with augmented dynamics

$$\mathbf{y}_{k+1} = \begin{bmatrix} \mathbf{x}_{k+1} \\ 1 \end{bmatrix} = \begin{bmatrix} A_k & c_k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x}_k \\ 1 \end{bmatrix} + \begin{bmatrix} B \\ 0 \end{bmatrix} \mathbf{u}_k = \tilde{A}\mathbf{y}_k + \tilde{B}\mathbf{u}_k$$

- Tracking LQR: $\mathbf{x}_k$, $\mathbf{u}_k$ represent small deviations ("errors") from a nominal trajectory (possibly with nonlinear dynamics)

- Stochastic systems $$\boldsymbol{x}_{k+1} = A_k\boldsymbol{x}_k + B_k\boldsymbol{u}_k + \boldsymbol{w}_k, \quad \boldsymbol{w}_k \sim \mathcal{N}(0, \Sigma_{\boldsymbol{w}_k})$$

Discrete LQR - Trajectory Optimization

- We could approach the LQR problem as a trajectory optimization problem, where we rewrite

$$J_0\left(\mathbf{x}_0\right) = \frac{1}{2}\mathbf{x}_N^T Q_N \mathbf{x}_N + \frac{1}{2}\sum_{k=0}^{N-1}\left(\mathbf{x}_k^T Q_k \mathbf{x}_k + \mathbf{u}_k^T R_k \mathbf{u}_k + 2\mathbf{x}_k^T S_k \mathbf{u}_k\right)$$

Subject to dynamics

$$\mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k, \quad k \in \{0,1,\ldots,N-1\}$$

↓

$$\begin{aligned} \min_{\mathbf{z}} \quad & \frac{1}{2}\mathbf{z}^T W \mathbf{z} \\ \text{s.t.} \quad & C\mathbf{z} + \mathbf{d} = \mathbf{0} \end{aligned}$$

$$\min_{\mathbf{x}_k,\mathbf{u}_k} \quad \frac{1}{2} \underbrace{\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{u}_0 \\ \mathbf{x}_1 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \\ \mathbf{x}_N \end{bmatrix}^T}_{\mathbf{z}} \underbrace{\begin{bmatrix} Q_0 & S_0 & & & & & & \\ S_0^T & R_0 & & & & & & \\ & & Q_1 & S_1 & & & & \\ & & S_1^T & R_1 & & & & \\ & & & & \ddots & & & \\ & & & & & Q_{N-1} & S_{N-1} & \\ & & & & & S_{N-1}^T & R_{N-1} & \\ & & & & & & & Q_N \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{u}_0 \\ \mathbf{x}_1 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \\ \mathbf{x}_N \end{bmatrix}}_{\mathbf{z}}$$

$$\text{s.t.} \quad \underbrace{\begin{bmatrix} -I & & & & & & & \\ A_0 & B_0 & -I & & & & & \\ & & A_1 & B_1 & -I & & & \\ & & & & \ddots & & & \\ & & & & & A_{N-1} & B_{N-1} & -I \end{bmatrix}}_{C} \underbrace{\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{u}_0 \\ \mathbf{x}_1 \\ \mathbf{u}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \\ \mathbf{x}_N \end{bmatrix}}_{\mathbf{z}} + \underbrace{\begin{bmatrix} \mathbf{x}_0 \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}}_{\mathbf{d}} = \mathbf{0}$$

Discrete LQR - Trajectory Optimization

We can then solve this problem by applying the NOC (which, due to the problem's convexity, are also SOC)

$$\begin{aligned} \min_{\mathbf{z}} \quad & \frac{1}{2}\mathbf{z}^T W \mathbf{z} \\ \text{s.t.} \quad & C\mathbf{z} + \mathbf{d} = \mathbf{0} \end{aligned}$$

Specifically:

$$L(z, \lambda) = \frac{1}{2} z^\top W z + \lambda^\top (Cz + d)$$

$$\nabla_z L = \frac{1}{2} W z + \frac{1}{2} W^\top z + C^\top \lambda = W z + C^\top \lambda = 0$$

$$\nabla_\lambda L = Cz + d = 0$$

Compactly, $\begin{bmatrix} \mathbf{z}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} W & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{0} \\ -\mathbf{d} \end{bmatrix}$ Solving this requires $\mathcal{O}[(N(m+n))^3]$

Discrete LQR - Dynamic programming

Solving through DP allows us to

(1) Solve in $\mathcal{O}[N(m+n)^3]$ vs $\mathcal{O}[(N(m+n))^3]$

(2) Obtain a *closed-loop* policy $\boldsymbol{\pi(\mathbf{x_t})}$

First step: $J_N^*(\mathbf{x}_N) = \frac{1}{2} x_N^T Q_N x_N = \frac{1}{2} x_N^T P_N x_N$

Proceeding backward in time:

$$J_{N-1}^*(\mathbf{x}_{N-1}) = \min_{\mathbf{u}_{N-1}} \frac{1}{2} \left(\begin{bmatrix} \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \end{bmatrix}^T \begin{bmatrix} Q_{N-1} & S_{N-1} \\ S_{N-1}^T & R_{N-1} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \end{bmatrix} + \mathbf{x}_N^T P_N \mathbf{x}_N \right)$$

$$= \min_{\mathbf{u}_{N-1}} \frac{1}{2} \left(\begin{bmatrix} \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \end{bmatrix}^T \begin{bmatrix} Q_{N-1} & S_{N-1} \\ S_{N-1}^T & R_{N-1} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{N-1} \\ \mathbf{u}_{N-1} \end{bmatrix} + \right.$$

$$\left. (A_{N-1}\mathbf{x}_{N-1} + B_{N-1}\mathbf{u}_{N-1})^T P_N (A_{N-1}\mathbf{x}_{N-1} + B_{N-1}\mathbf{u}_{N-1}) \right)$$

Discrete LQR - Dynamic programming

Unconstrained NOC:

$$\nabla_{u_{N-1}} J_{N-1}(\mathbf{x}_{N-1}) = R_{N-1}\mathbf{u}_{N-1} + S_{N-1}^T\mathbf{x}_{N-1} + B_{N-1}^T P_N (A_{N-1}\mathbf{x}_{N-1} + B_{N-1}\mathbf{u}_{N-1}) = \mathbf{0}$$

$$\implies \mathbf{u}_{N-1}^* = -(R_{N-1} + B_{N-1}^T \underbrace{P_N B_{N-1})^{-1}(B_{N-1}^T P_N}_{\text{does not depend on time}} A_{N-1} + S_{N-1}^T)\mathbf{x}_{N-1} := F_{N-1}x_{N-1}$$

Note also that SOC hold: $\nabla^2_{u_{N-1}} J_{N-1}(\mathbf{x}_{N-1}) = R_{N-1} + B_{N-1}^T P_N B_{N-1} \succ 0$

To obtain the optimal cost-to-go, we plug in the optimal policy to obtain:

$$J_{N-1}^*(\mathbf{x}_{N-1}) = \frac{1}{2}\mathbf{x}_{N-1}^T \big(Q_{N-1} + A_{N-1}^T P_N A_{N-1} - (A_{N-1}^T P_N B_{N-1} + S_{N-1})(R_{N-1} + B_{N-1}^T P_N B_{N-1})^{-1}(B_{N-1}^T P_N A_{N-1} + S_{N-1}^T)\big)\mathbf{x}_{N-1} := \frac{1}{2}\mathbf{x}_{N-1}^T P_{N-1}\mathbf{x}_{N-1}$$

Notice that:

- The optimal policy is a time-varying linear feedback policy (i.e., we can just store the matrices F_k)
- The cost-to-go is a quadratic function of the state at each step (!)

Additionally:

- In the infinite horizon case, this is guaranteed to converge to the optimal policy (as long as there exist a policy that can drive the system to zero)
- Often most convenient to use steady state F_∞

Discrete LQR - Dynamic programming

Proceeding by induction, we derive the Riccati recursion:

1. $P_N = Q_N$
2. $F_k = -(R_k + B_k^T P_{k+1} B_k)^{-1}(B_k^T P_{k+1} A_k + S_k^T)$
3. $P_k = Q_k + A_k^T P_{k+1} A_k -$
   $(A_k^T P_{k+1} B_k + S_k)(R_k + B_k^T P_{k+1} B_k)^{-1}(B_k^T P_{k+1} A_k + S_k^T)$
4. $\pi_k^*(\mathbf{x}_k) = F_k \mathbf{x}_k$
5. $J_k^*(\mathbf{x}_k) = \frac{1}{2}\mathbf{x}_k^T P_k \mathbf{x}_k$

Which enables us to

- Compute the policy backwards in time (and store it)
- Apply the policy forward in time

Outline

Intro to dynamic programming (DP) and *principle of optimality*

The dynamic programming algorithm

Dynamic programming in control:

- Discrete LQR
- Stochastic Optimal Control Problem / Markov Decision Process (MDP): Stochastic LQR
- Policy Iteration and Value Iteration

Stochastic Optimal Control Problem: Markov Decision Problem (MDP)

- **System**: $\boldsymbol{x}_{k+1} = f_k\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right), k = 0,\ldots,N-1$
- **Probability distribution**: $\boldsymbol{w}_k \sim P_k\left(\cdot \mid \boldsymbol{x}_k, \boldsymbol{u}_k\right)$
- **Control constraints:** $\boldsymbol{u}_k \in U\left(\boldsymbol{x}_k\right)$
- **Policies:** $\pi = \left\{\pi_0 \ldots, \pi_{N-1}\right\}$, where $\boldsymbol{u}_k = \pi_k\left(\boldsymbol{x}_k\right)$
- **Expected Cost**:

$$J_\pi\left(\boldsymbol{x}_0\right) = \mathbb{E}_{\boldsymbol{w}_k, k=0,\ldots,N-1}\left[g_N\left(\boldsymbol{x}_N\right) + \sum_{k=0}^{N-1} g_k\left(\boldsymbol{x}_k, \pi_k\left(\boldsymbol{x}_k\right), \boldsymbol{w}_k\right)\right]$$

Stochastic Optimal Control Problem:

$$J^*\left(x_0\right) = \min_\pi J_\pi\left(x_0\right)$$

Key points

- Discrete-time model
- Markovian model
- Objective: find optimal **closed-loop** policy
- Additive cost (central assumption in DP)
- Risk-neutral formulation

Other communities use different notation:
[Powell, W. B. *AI, OR and control theory: A Rosetta Stone for stochastic optimization.* Princeton University, 2012.]

The DP algorithm (stochastic case)

Principle of optimality:

- Let $\pi^* := \left\{\pi_0^*, \pi_1^*, \ldots, \pi_{N-1}^*\right\}$ be an optimal policy
- Consider the tail subproblem

$$\mathbb{E}_{w_k}\left[g_N\left(\boldsymbol{x}_N\right) + \sum_{k=i}^{N-1} g_k\left(\boldsymbol{x}_k, \pi_k\left(\boldsymbol{x}_k\right), \boldsymbol{w}_k\right)\right]$$

the tail policy $\left\{\pi_i^*, \ldots, \pi_{N-1}^*\right\}$ is optimal for the tail subproblem

Intuition:

- DP first solves ALL tail subproblems at the final stage
- At the generic step, it solves ALL tail subproblems of a given time length, using solution of tail subproblems of shorter length

DP Algorithm (stochastic case)

Like in the deterministic case, start with:

$$J_N\left(x_N\right) = g_N\left(x_N\right)$$

and iterate backwards in time using

$$J_k\left(\boldsymbol{x}_k\right) = \min_{\boldsymbol{u}_k \in U(\boldsymbol{x}_k)} \mathbb{E}_{w_k}\left[g_k\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) + J_{k+1}\left(f\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right)\right)\right], \quad k = 0, \ldots, N-1$$

for which the optimal cost $J^*(\mathbf{x}_0)$ is equal to $J_0(\mathbf{x}_0)$ and the optimal policy is constructed by setting

$$\pi_k^*\left(\boldsymbol{x}_k\right) = \underset{\boldsymbol{u}_k \in U(\boldsymbol{x}_k)}{\operatorname{argmin}} \mathbb{E}_{w_k}\left[g_k\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right) + J_{k+1}\left(f\left(\boldsymbol{x}_k, \boldsymbol{u}_k, \boldsymbol{w}_k\right)\right)\right]$$

Next time

- Stochastic Dynamic Programming
- Infinite-Horizon MDPs
- Value Iteration
- Policy Iteration