Topics in Reinforcement Learning: Rollout and Approximate Policy Iteration

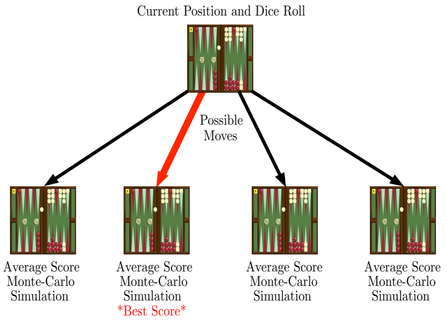
ASU, CSE 691, Spring 2021

Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

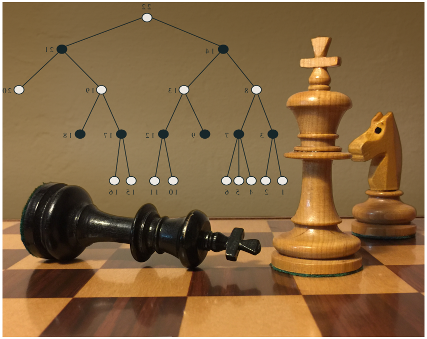
Dimitri P. Bertsekas
dbertsek@asu.edu

Lecture 3
Problem Formulations and Examples

Backgammon and Chess - Off-Line Training and On-Line Play



Current Position and Dice Roll
Possible Moves
Average Score Monte-Carlo Simulation
Average Score Monte-Carlo Simulation *Best Score*
Average Score Monte-Carlo Simulation
Average Score Monte-Carlo Simulation

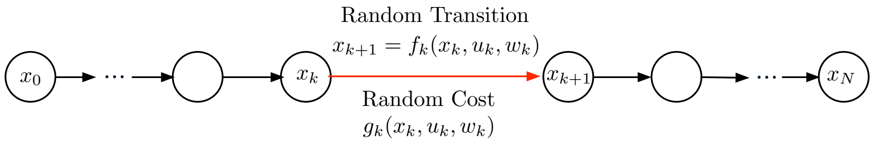


Strong connections to DP, policy iteration, and approximation in value space

- Off-line training of value and/or policy network approximations
- On-line play by multistep lookahead, rollout, and cost function approximations

We are aiming to develop this methodology, so it applies far more generally

Stochastic DP Problems - Perfect State Observation (We Know x_k)



- System $x_{k+1} = f_k(x_k, u_k, w_k)$ with random “disturbance" w_k (e.g., physical noise, market uncertainties, demand for inventory, unpredictable breakdowns, etc)
- Cost function:
$$E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\right\}$$
- Policies $\pi = \{\mu_0, \dots, \mu_{N-1}\}$, where μ_k is a “closed-loop control law" or “feedback policy"/a function of x_k. A “lookup table" for the control $u_k = \mu_k(x_k)$ to apply at x_k.
- For given initial state x_0, minimize over all $\pi = \{\mu_0, \dots, \mu_{N-1}\}$ the cost
$$J_\pi(x_0) = E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k\big(x_k, \mu_k(x_k), w_k\big)\right\}$$
- Optimal cost function: $J^*(x_0) = \min_\pi J_\pi(x_0)$. Optimal policy: $J_{\pi^*}(x_0) = J^*(x_0)$

The Stochastic DP Algorithm

Produces the optimal costs $J_k^*(x_k)$ of the tail subproblems that start at x_k

Start with $J_N^*(x_N) = g_N(x_N)$, and for $k = 0, \ldots, N-1$, let

$$J_k^*(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}, \qquad \text{for all } x_k.$$

- The optimal cost $J^*(x_0)$ is obtained at the last step: $J_0^*(x_0) = J^*(x_0)$.

On-line implementation of the optimal policy, given $J_1^*, \ldots, J_{N-1}^*$

Sequentially, going forward, for $k = 0, 1, \ldots, N-1$, observe x_k and apply

$$u_k^* \in \arg\min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big)\Big\}.$$

Issues: Need to know J_{k+1}^*, compute $E_{w_k}\{\cdot\}$ for each u_k, minimize over all u_k

Approximation in value space: Use $\tilde{J}_k$ in place of J_k^*; approximate $E_{w_k}\{\cdot\}$ and $\min_{u_k}$ (the three approximations)

Note the division in precomputation phase (off-line training) and real-time implementation phase (on-line play)

Outline

How do we Formulate DP Problems?

An informal recipe: First define the controls, then the stages (and info available at each stage), and then the states

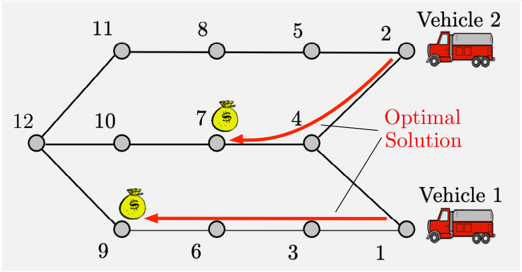
- Define as state x_k something that “summarizes" the past for purposes of future optimization, i.e., as long as we know x_k, all past information is irrelevant.
- Rationale: The controller applies action that depends on the state. So the state must subsume all info that is useful for decision/control.

Some examples

- In the traveling salesman problem, we need to include all the relevant info in the state (e.g., the past cities visited). Other info, such as the costs incurred so far, need not be included in the state.
- In partial or imperfect information problems, we use “noisy" measurements for control of some quantity of interest y_k that evolves over time (e.g., the position/velocity vector of a moving object). If I_k is the collection of all measurements up to time k, it is correct to use I_k as state.
- It may also be correct to use alternative states; e.g., the conditional probability distribution $P_k(y_k \mid I_k)$. This is called belief state, and subsumes all the information that is useful for the purposes of control choice.

Cost-Free and Absorbing Terminal State: Finite or Infinite Horizon?

- One possibility is to convert to a finite horizon problem: Introduce as horizon an upper bound to the optimal number of stages (assuming such a bound is known)
- Add BIG penalty for not terminating before the end of the horizon

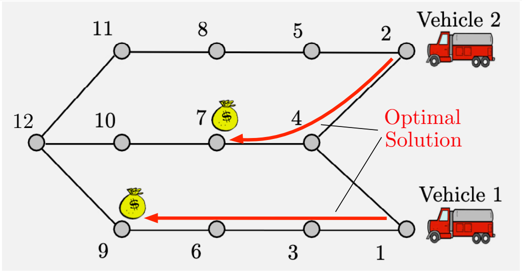


Typical discrete optimization problem: Multi-vehicle routing; all vehicles move one step at a time

- Minimize the number of vehicle moves to perform all tasks
- How do we formulate the problem as a DP problem? What is the state?
- Astronomical number of states, even for modest number of tasks and vehicles
- Use rollout. Base heuristic: Move vehicles, one-at-a-time, one step to nearest task

Multi-Vehicle Routing Example: Solution by Rollout

- At the current state consider all possible joint vehicle moves, and from each of the next states, run the base heuristic to termination
- Use the joint move that results in min cost. Repeat at the next state, etc



At state/position pair (1,2), consider possible moves to (3,5), (3,4), (4,5), (4,4)

- (3,5): (1) 3->6, (2) 5->2, (1) 6->9 (performs task), (2) 2->4, (1) 9->12, (2) 4->7 (performs task)
- (3,4): (1) 3->6, (2) 4->7 (performs task), (1) 6->9 (performs task)
- Repeat for (4,5) and (4,4); winner is move (3,4). Repeat starting from (3,4), etc
- Rollout algorithm performs optimally (5 moves). Base heuristic needs 9 moves

State Augmentation: Delays

$$x_{k+1} = f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), \qquad x_1 = f_0(x_0, u_0, w_0)$$

- Introduce additional state variables y_k and s_k, where $y_k = x_{k-1}$, $s_k = u_{k-1}$. Then

$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ s_{k+1} \end{pmatrix} = \begin{pmatrix} f_k(x_k, y_k, u_k, s_k, w_k) \\ x_k \\ u_k \end{pmatrix}$$

- Define $\tilde{x}_k = (x_k, y_k, s_k)$ as the new state, we have

$$\tilde{x}_{k+1} = \tilde{f}_k(\tilde{x}_k, u_k, w_k)$$

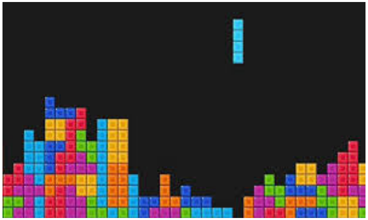
- Reformulated DP algorithm: Start with $J_N^*(x_N) = g_N(x_N)$

$$J_k^*(x_k, x_{k-1}, u_{k-1}) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), x_k, u_k\big)\Big\}$$

$$J_0^*(x_0) = \min_{u_0 \in U_0(x_0)} E_{w_0}\Big\{ g_0(x_0, u_0, w_0) + J_1^*\big(f_0(x_0, u_0, w_0), x_0, u_0\big)\Big\}$$

Deal similarly with delays in the cost function

Simplification for Uncontrollable State Components: Tetris



TETRIS

An Infinite Horizon Stochastic Shortest Path Problem

State = (Board position x, Shape of falling block y); Control = Apply translation and rotation on y

y is an "uncontrollable" component of the state, evolving according to $y_{k+1} = w_k$

Simplified/Averaged Bellman Equation [involves functions of x and not (x, y)]

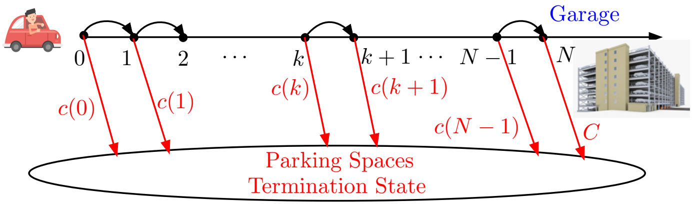
It involves $\hat{J}(x)$, the optimal expected score at position x (averaged over shapes y):

$$\hat{J}(x) = \sum_y p(y) \max_u \Big[g(x,y,u) + \hat{J}\big(f(x,y,u)\big)\Big], \qquad \text{for all } x,$$

where

- $g(x, y, u)$ is the number of points scored (rows removed),
- $f(x, y, u)$ is the next board position (or termination state).

Parking: Uncontrollable State Components, Terminal State



- Start at spot 0; either park at spot k with cost $c(k)$ (if free) or continue; park at garage at cost C if not earlier
- Spot k is free with known probability $p(k)$; status is observed upon reaching it
- How do we formulate the problem as a DP problem?

States: F: current spot is free, $\overline{F}$: current spot is taken, Parked (terminal state)

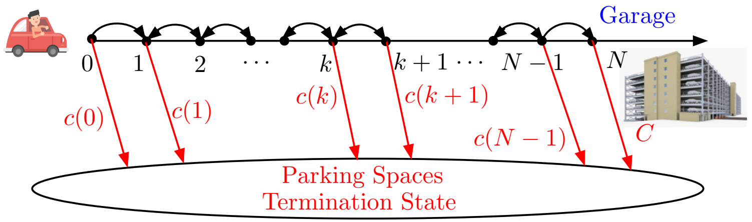
Averaged DP algorithm ($\hat{J}_k$ is the expected cost upon arrival at position k):

$$\hat{J}_{N-1} = p(N-1)\min\big[c(N-1),\ C\big] + \big(1-p(N-1)\big)C,$$

$$\hat{J}_k = p(k)\min\big[c(k),\ \hat{J}_{k+1}\big] + \big(1-p(k)\big)\hat{J}_{k+1}, \qquad k = 0,\ldots,N-2$$

Optimal policy: Park at the first free spot within m of the garage (m depends on data)

More Complex Parking Problems

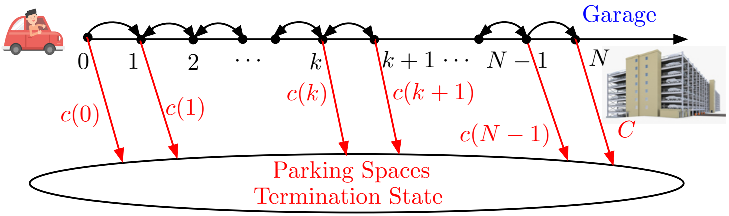


- Bidirectional parking: We can go back to parking spots we have visited at a cost
- More complicated parking lot topologies
- Multiagent versions: Multiple drivers/autonomous vehicles, “parkers" and “searchers", etc

A major distinction

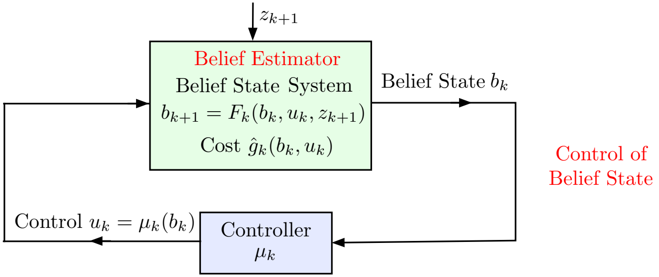
- “Relatively easy" case: The status of already seen spots stays unchanged
- More complex case: The status of already seen spots changes probabilistically

Bidirectional Parking: Partial State Observation Problems



- Consider a complex type of parking example, where free or taken parking spots may get taken or free up, at the next time step with some probability
- The free/taken state of the spots is “estimated" in a “probabilistic sense" based on the observations (the free/taken status of the spots visited ... when visited)
- What should the “state" be? It should summarize all the info needed for the purpose of future optimization
- First candidate for state: The set of all observations so far.
- Another candidate: The “belief state", i.e., the conditional probabilities of the free/taken status of all the spots: $p(0), p(1), \ldots, p(N-1)$, conditioned on all the observations so far
- Generally, partial observation problems (POMDP) can be “solved" by DP with state being the belief state: $b_k = P(x_k \mid \text{set of observations up to time } k)$

Partial State Observation Problems: Reformulation via Belief State

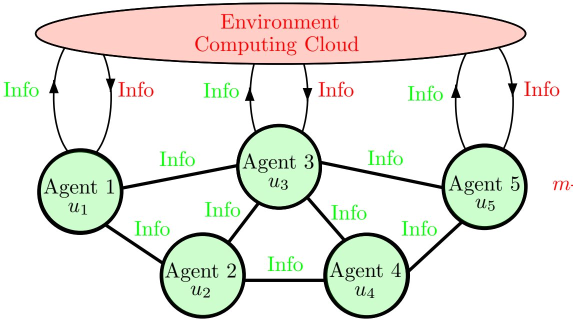


The reformulated DP algorithm has the form

$$J_k^*(b_k) = \min_{u_k \in U_k} \left[\hat{g}_k(b_k, u_k) + E_{z_{k+1}} \left\{ J_{k+1}^*\big(F_k(b_k, u_k, z_{k+1})\big) \right\} \right]$$

- $J_k^*(b_k)$ denotes the optimal cost-to-go starting from belief state b_k at stage k.
- U_k is the control constraint set at time k
- $\hat{g}_k(b_k, u_k)$ denotes expected cost of stage k: expected stage cost $g_k(x_k, u_k, w_k)$, with distribution of (x_k, w_k) determined by b_k and the distribution of w_k
- Belief estimator: $F_k(b_k, u_k, z_{k+1})$ is the next belief state, given current belief state b_k, u_k is applied, and observation z_{k+1} is obtained

Multiagent Problems (1960s →)



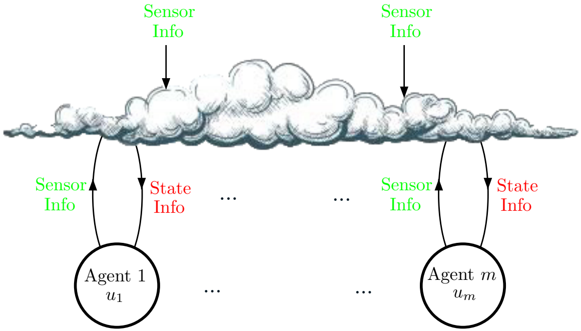
m-Component Control
$u = (u_1, \ldots, u_m)$

- Multiple agents collecting and sharing information selectively with each other and with an environment/computing cloud
- Agent i applies decision u_i sequentially in discrete time based on info received

The major mathematical distinction between problem structures

- The classical information pattern: Agents are fully cooperative, fully sharing and never forgetting information. Can be treated by DP
- The nonclassical information pattern: Agents are partially sharing information, and may be antagonistic. HARD because it cannot be treated by DP

Starting Point: A Classical Information Pattern (We Generalize Later)

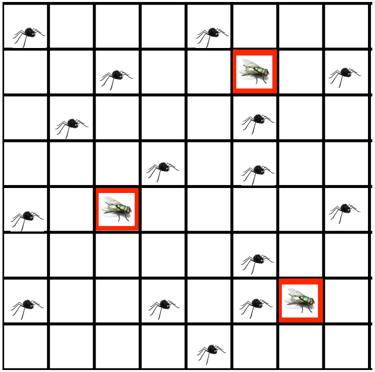


At each time: Agents have exact state info; choose their controls as function of state

Model: A discrete-time (possibly stochastic) system with state x and control u

- Decision/control has m components $u = (u_1, \ldots, u_m)$ corresponding to m "agents"
- "Agents" is just a metaphor - the important math structure is $u = (u_1, \ldots, u_m)$
- The theoretical framework is DP. We will reformulate for faster computation
  - Deal with the exponential size of the search/control space
  - Be able to compute the agent controls in parallel (in the process we will deal in part with nonclassical info pattern issues)

Spiders-and-Flies Example (e.g., Vehicle Routing, Maintenance, Search-and-Rescue, Firefighting)



15 spiders move in 4 directions with perfect vision

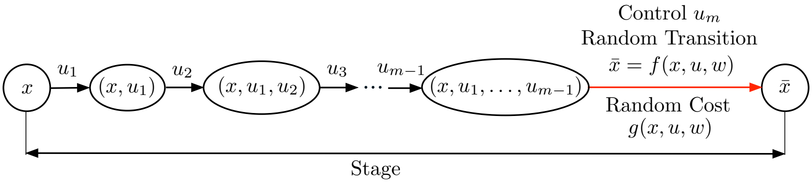
3 blind flies move randomly

Objective is to

Catch the flies in minimum time

- At each time we must select one out of $\approx 5^{15}$ joint move choices
- We will reduce to $(5 \text{ choices}) \cdot (15 \text{ times}) = 75$ (while maintaining good properties)
- Idea: Break down the control into a sequence of one-spider-at-a-time moves
- For more discussion, including illustrative videos of spiders-and-flies problems, see https://www.youtube.com/watch?v=eqbb6vVIN38&t=1654s

Reformulation Idea: Trading off Control and State Complexity (NDP Book, 1996)



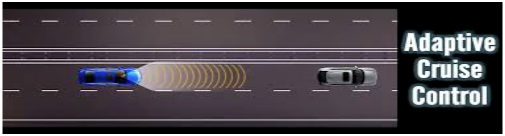
An equivalent reformulation - “Unfolding" the control action

- The control space is simplified at the expense of $m - 1$ additional layers of states, and corresponding $m - 1$ cost functions

$$J^1(x, u_1), J^2(x, u_1, u_2), \ldots, J^{m-1}(x, u_1, \ldots, u_{m-1})$$

- Allows far more efficient rollout (one-agent-at-a-time). This is just standard rollout for the reformulated problem
- The increase in size of the state space does not adversely affect rollout (only one state per stage is looked at during on-line play)
- Complexity reduction: The one-step lookahead branching factor is reduced from n^m to $n \cdot m$, where n is the number of possible choices for each component u_i

Changing Problem Parameters: Adaptive Control (1960s $\rightarrow$)



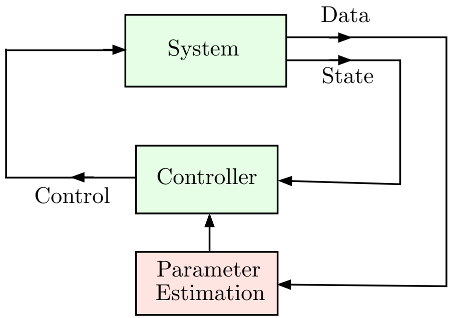
A cruise control-type problem

- Control car velocity: $x_{k+1} = ax_k + bu_k + w_k$ ($a < 1$ models friction, wind drag, etc)
- Cost over N stages: $(x_N - \bar{x}_N)^2 + \sum_{k=0}^{N-1} (x_k - \bar{x}_k)^2 + ru_k^2)$, where $r \geq 0$ is given
- ... but a, b, and $\bar{x}_k$ are changing all the time; they may be measured with error (?)

Adaptive control deals with such situations. Some possibilities:

- Ignore the changes in parameters; design a controller that is robust (works for a broad range of parameters). PID control is a time-honored relevant methodology
- Try to estimate the parameters, and use the estimates to modify the controller
  - On-line replanning by optimization; modify the controller to make it optimal for the current set of estimates. This is sophisticated/time consuming: needs on-line system identification, and optimal control computation. Has other pitfalls (identifiability; see notes)...
  - On-line replanning by rollout with a base policy whose cost function is computed using the current parameter estimates. This is simpler ...

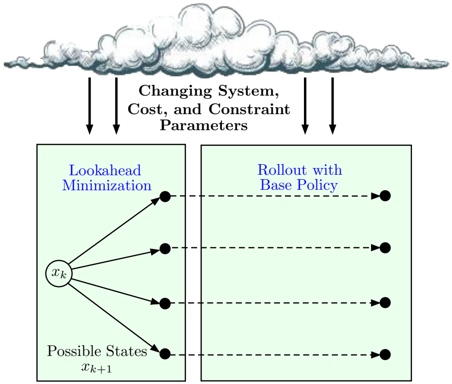
On-Line Replanning by Optimization



Introduce on-line estimation of changing parameters

- Recompute the controller so it is optimal for the new set of parameters
- This can be time-consuming, so a suboptimal controller may be recalculated instead

On-Line Replanning by Rollout

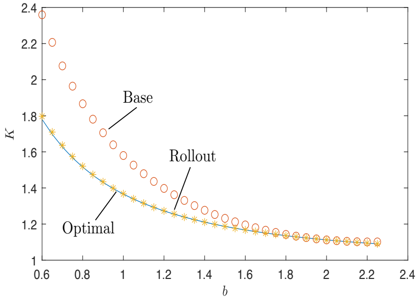


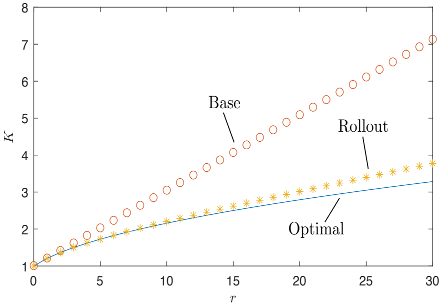
Changing System, Cost, and Constraint Parameters
Lookahead Minimization
Rollout with Base Policy
x_k
Possible States x_{k+1}

Use on-line replanning with rollout instead of controller reoptimization; this is faster

- Introduce new parameter estimates in the lookahead minimization and the rollout
- Continue to use the same base policy
- Possibly recalculate the base policy in the background

A Linear-Quadratic Example of On-Line Replanning





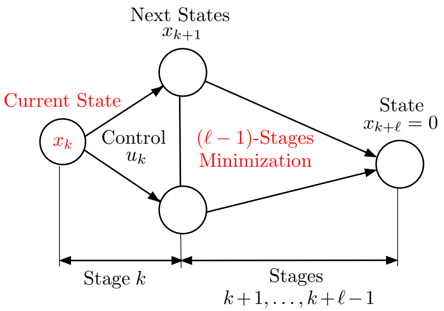
Performance comparison of on-line replanning by rollout and by optimization

One-dimensional linear-quadratic example:

$$x_{k+1} = x_k + bu_k, \qquad \text{Cost} = \lim_{N\to\infty} \sum_{k=0}^{N-1} (x_k^2 + ru_k^2)$$

Quadratic cost coefficient as b and r change. Base policy is optimal for $b = 2$ and $r = 0.5$

Model Predictive Control (MPC) - Relation to Rollout



System: $x_{k+1} = f(x_k, u_k)$

Cost: $g(x_k, u_k) \geq 0$, for all (x_k, u_k)

The system can be kept at the origin at zero cost by some control

Consider undiscounted infinite horizon; we want to keep the system near 0

- We minimize the cost function over the next ℓ stages while requiring $x_{k+\ell} = 0$
- We then apply the first control of the minimizing sequence, discard the other controls
- This is rollout w/ base heuristic the min that drives $x_{k+\ell}$ to 0 in $(\ell - 1)$ steps
- Well-suited for on-line replanning
- A variant that uses a terminal cost approximation instead of $x_{k+\ell} = 0$; can be viewed as rollout/approximation in value space with single or multistep lookahead

About the Next Lecture

- We are done with the overview of the topics of this course. We will now go more deeply
- We will cover general issues of one-step and multistep approximation in value space
- We will start a more in-depth discussion of rollout

HOMEWORK 2 (DUE IN ONE WEEK) TO BE ANNOUNCED

WATCH VIDEOLECTURE 3 OF THE 2019 OFFERING OF THE COURSE