Topics in Reinforcement Learning: Lessons from AlphaZero for (Sub)Optimal Control and Discrete Optimization

Arizona State University
Course CSE 691, Spring 2022

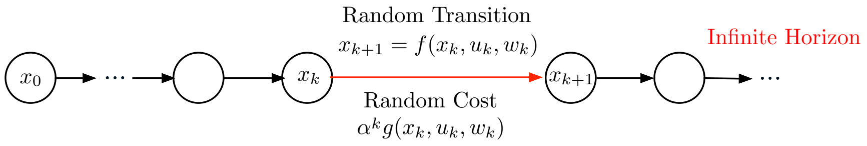
Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

Dimitri P. Bertsekas
dbertsek@asu.edu

Lecture 3
Approximation in Value Space and Linear Quadratic Problems
Problem Formulations, Reformulations, and Examples

Outline

Review of Infinite Horizon Problems



Bellman operators: Abstract notation, convenient for visualization and analysis

The min-Bellman operator T that transforms a function $J(\cdot)$ into a function $(TJ)(\cdot)$

$$(TJ)(x) = \min_{u \in U(x)} E\Big\{g(x,u,w) + \alpha J\big(f(x,u,w)\big)\Big\}, \qquad \text{for all } x$$

The μ-Bellman operator T_μ for any stationary policy $\{\mu, \mu, \ldots\}$

$$(T_\mu J)(x) = E\Big\{g\big(x,\mu(x),w\big) + \alpha J\big(f(x,\mu(x),w)\big)\Big\}, \qquad \text{for all } x$$

Theory and Algorithms using Bellman operators (with some exceptions)

- J^* satisfies $J^* = TJ^*$ (the min-Bellman equation). If $T_\mu J^* = TJ^*$, μ is optimal
- J_μ satisfies $J_\mu = T_\mu J_\mu$ (the μ-Bellman equation).
- VI: $J_{k+1} = TJ_k$; converges to J^*. Also $J_{k+1} = T_\mu J_k$ converges to J_μ
- PI: $J_{\mu^k} = T_{\mu^k} J_{\mu^k}$ (policy evaluation) and $T_{\mu^{k+1}} J_{\mu^k} = TJ_{\mu^k}$ (policy improvement)

Deterministic Linear Quadratic Problem - Riccati Operators

- System $x_{k+1} = ax_k + bu_k$ and cost function $\lim_{N\to\infty} \sum_{k=0}^{N-1} (qx_k^2 + ru_k^2)$
- The min-Bellman eq. is $J^*(x) = \min_u \left[qx^2 + ru^2 + J^*(ax + bu)\right]$
- For linear $\mu(x) = Lx$, the μ-Bellman eq. is $J_\mu(x) = (q + rL^2)x^2 + J_\mu\big((a + bL)x\big)$
- The Bellman eqs. admit quadratic solutions $J^*(x) = K^* x^2$ and $J_\mu(x) = K_L x^2$, where K^* and K_L solve the Riccati eqs. (restrictions of Bellman eqs. to quadratics)

$$K = F(K) = \frac{a^2 rK}{r + b^2 K} + q, \qquad K = F_L(K) = (a + bL)^2 K + q + rL^2$$

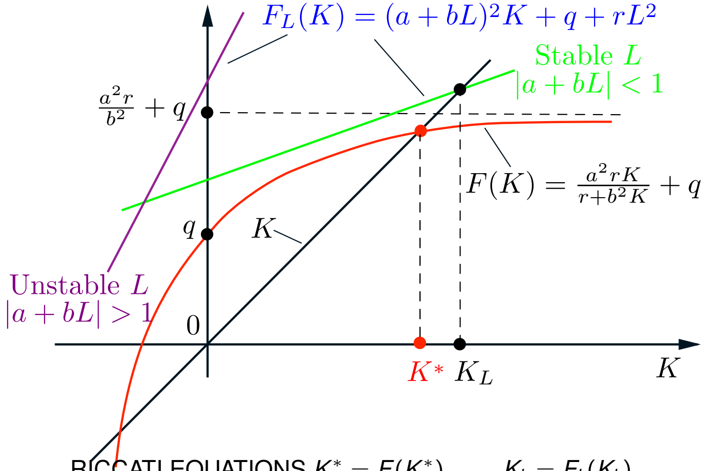
- The optimal policy is a linear function of x, $\mu^*(x) = L^* x$, and is obtained from

$$\mu^*(x) = \arg\min_u \left[qx^2 + ru^2 + qK^*(ax + bu)^2\right], \qquad L^* = -\frac{abK^*}{r + b^2 K^*}$$

- The VI algorithm is $J_{k+1}(x) = \min_u \left[qx^2 + ru^2 + J_k(ax + bu)\right]$
- Starting with $J_0(x) = K_0 x^2$, the value iterates J_k are quadratic: $J_k(x) = K_k x^2$, where $\{K_k\}$ is generated by

$$K_0 \geq 0, \qquad K_{k+1} = \frac{\alpha^2 rK_k}{r + b^2 K_k} + q$$

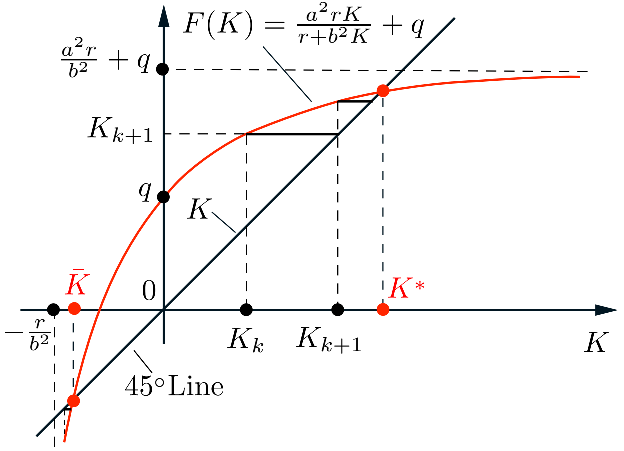
Graphical Solution of Min-Riccati and L-Riccati Equations



RICCATI EQUATIONS $K^* = F(K^*)$, $\quad K_L = F_L(K_L)$

$J^*(x) = K^* x^2$, $\quad J_\mu(x) = K_L x^2$ for a stable linear policy $\mu(x) = Lx$ ($|a + bL| < 1$)

Algorithmic Solution by VI

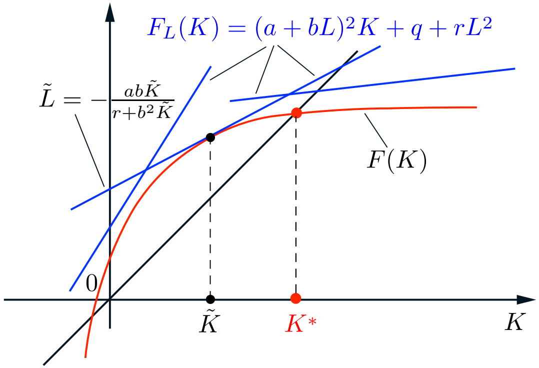


Value Iteration: $K_{k+1} = F(K_k)$

from

$$J_{k+1}(x) = K_{k+1} x^2 = F(K_k) x^2 = J_k(x)$$

Min-Riccati Operator as Lower Envelope of L-Riccati Operators



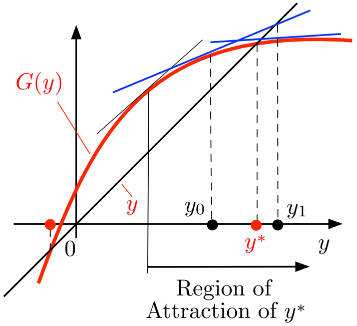
$F_L(K) = (a + bL)^2 K + q + rL^2$
$\tilde{L} = -\frac{ab\tilde{K}}{r+b^2\tilde{K}}$
$F(K)$
0
$\tilde{K}$
K^*
K

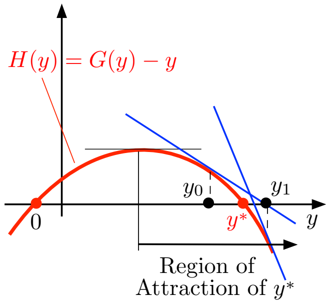
$$F(K)x^2 = \min_{u\in\Re}\left\{qx^2 + ru^2 + K(ax + bu)^2\right\}$$
$$= \min_{L\in\Re}\min_{u=Lx}\left\{qx^2 + ru^2 + K(ax + bu)^2\right\}$$
$$= \min_{L\in\Re}\left\{q + bL + K(a + bL)^2\right\}x^2$$

or

$$F(K) = \min_{L\in\Re} F_L(K), \quad \text{with} \quad F_L(K) = (a + bL)^2 K + q + bL$$

Newton's Method for Solving the Fixed Point Problem $y = G(y)$





At the typical iteration k

- We linearize the problem at the current iterate y_k using a first order Taylor series expansion of G,

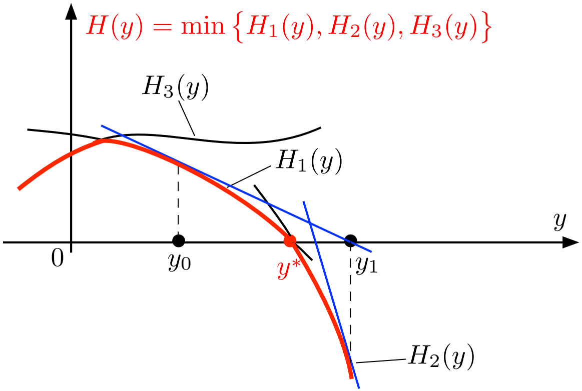
$$G(y) \approx G(y_k) + \nabla G(y_k)(y - y_k),$$

where $\nabla G(y_k)$ is the gradient of G at y_k

- We solve the linearized problem to obtain y_{k+1}:

$$y = G(y_k) + \nabla G(y_k)(y - y_k)$$

Newton's Method for Solving Nondifferentiable Equations

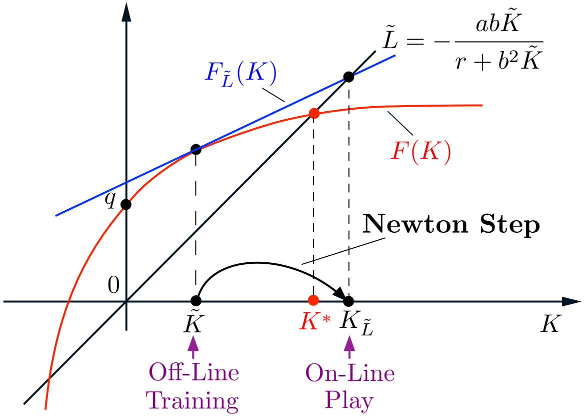


H consists of the minimum of multiple differentiable functions H_i, $i = 1, \ldots, m$

At the typical iteration k

- We linearize the problem at the current iterate y_k using a first order Taylor series expansion of any one of the active components of H at y_k
- We solve the linearized problem to obtain y_{k+1}
- Can also be used for the fixed point problem $y = \min\{G_1(y), G_2(y), G_3(y)\}$

Visualization of Approximation in Value Space - One-Step Lookahead



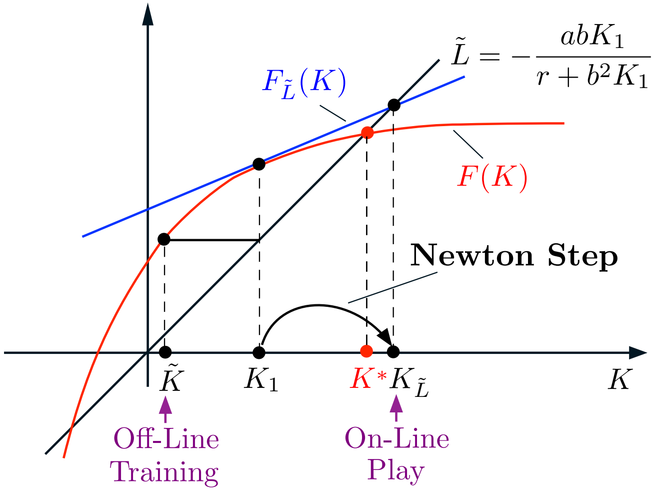
Given quadratic cost approximation $\tilde{J}(x) = \tilde{K}x^2$, we find

$$\tilde{\mu}(x) = \arg\min_{\mu}(T_\mu \tilde{J})(x) \quad \text{or} \quad \tilde{L} = \arg\min_{L} F_L(\tilde{K})$$

to construct the one-step lookahead policy $\tilde{\mu}(x) = \tilde{L}x$

and its cost function $J_{\tilde{\mu}}(x) = K_{\tilde{L}}x^2$

Visualization of Approximation in Value Space - Two-Step Lookahead

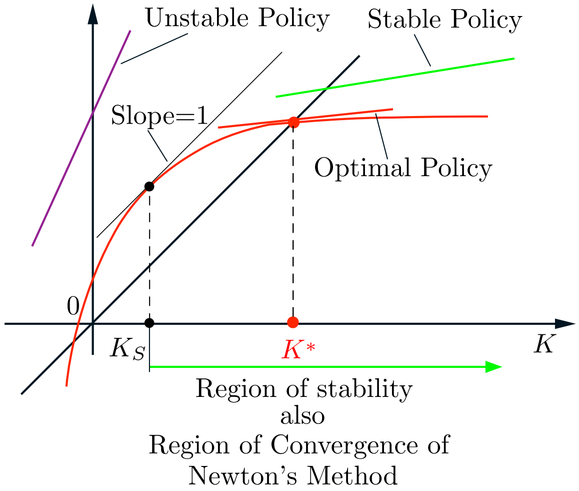


Multistep lookahead moves

the starting point of the Newton step closer to K^*

The longer the lookahead the better

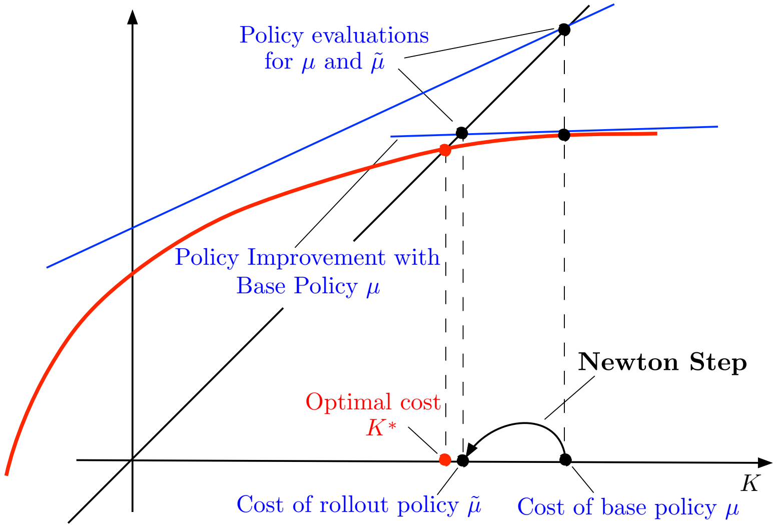
Visualization of Region of Stability



The start of the Newton step must be within the region of stability

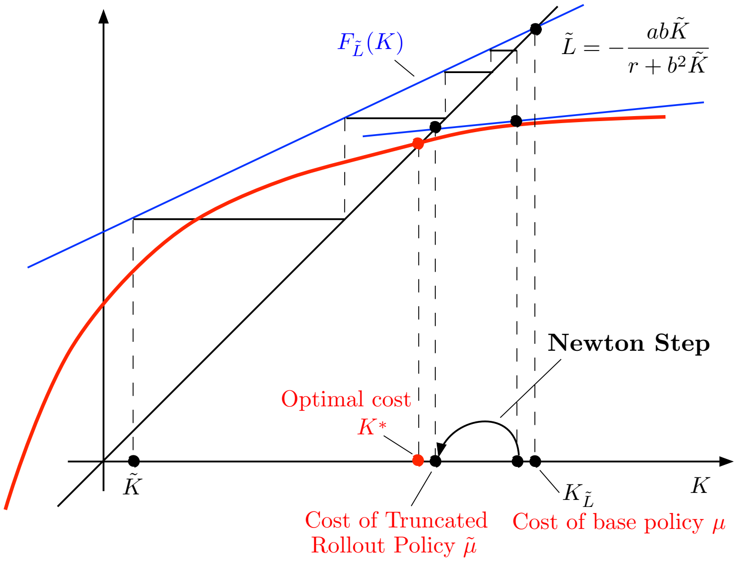
Longer lookahead promotes stability of the multistep lookahead policy

Visualization of Rollout with Stable Linear Base Policy



Policy evaluations for μ and $\tilde{\mu}$
Policy Improvement with Base Policy μ
Newton Step
Optimal cost K^*
K
Cost of rollout policy $\tilde{\mu}$
Cost of base policy μ

Visualization of Truncated Rollout



$F_{\tilde{L}}(K)$
$\tilde{L} = -\dfrac{ab\tilde{K}}{r + b^2\tilde{K}}$
Newton Step
Optimal cost
K^*
$\tilde{K}$
$K_{\tilde{L}}$
K
Cost of Truncated Rollout Policy $\tilde{\mu}$
Cost of base policy μ

Policy Iteration for the Linear Quadratic Problem

Starts with linear policy $\mu^0(x) = L_0 x$, generates sequence of linear policies $\mu^k(x) = L_k x$ (see class notes for details)

- Policy evaluation:

$$J_{\mu^k}(x) = K_k x^2$$

where

$$K_k = \frac{q + rL_k^2}{1 - (a + bL_k)^2}$$

- Policy improvement:

$$\mu^{k+1}(x) = L_{k+1} x$$

where

$$L_{k+1} = -\frac{abK_k}{r + b^2 K_k}$$

- Rollout is a single Newton iteration
- PI is a full-fledged Newton method for solving the Riccati equation $K = F(K)$
- An important variant, Optimistic PI, consists of repeated truncated rollout iterations
- Can be viewed as a Newton-SOR method (repeated application of a Newton step, preceded by first order VIs)

Let’s Take a 15-min Working Break: Catch your Breath, Collect your Questions, and Consider the Following Challenge Question

How long should the length of the truncated rollout be?

Consider issues of performance and stability of the lookahead policy

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 visit order, or 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.

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 $J_N^*(x_N) = g_N(x_N)$

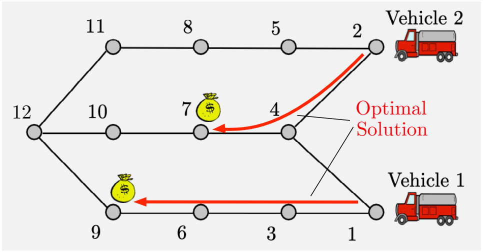
$$J_k^*(x_k, x_{k-1}, u_{k-1}) = \min_{u_k \in U_k(x_k)} E_{w_k}\Big\{g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, x_{k-1}, u_k, u_{k-1}, w_k), x_k, u_k\big)\Big\}$$

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

See the class notes for other examples of state augmentation

Problems Where There is a Cost-Free and Absorbing Terminal State

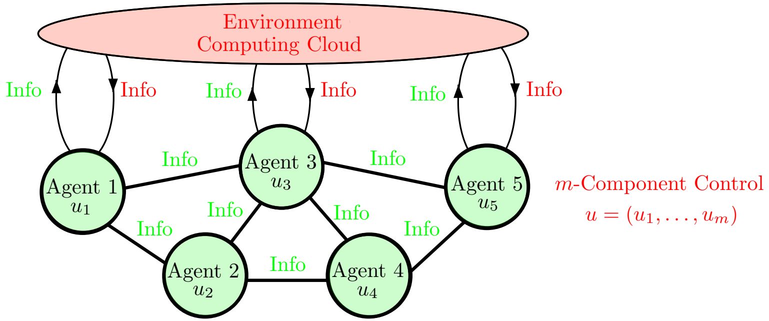
- Generally, we can view them as infinite horizon problems
- Another possibility is to convert to a finite horizon problem: Introduce as horizon an upper bound to the optimal number of stages (assuming such a bound is known)
- Add BIG penalty for not terminating before the end of the horizon



Example: Multi-vehicle routing; vehicles move one step at a time

- Minimize the number of vehicle moves to perform all tasks
- How to formulate the problem by DP problem? States? Controls?
- Astronomical numbers, even for modest number of tasks and vehicles
- A good candidate for the multiagent framework that we will introduce shortly

Multiagent Problems (1960s $\rightarrow$)

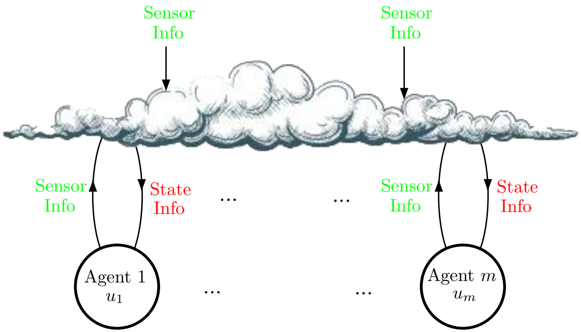


- Multiple agents collecting and sharing information selectively with each other and with an environment/computing cloud
- Agent i applies decision u_i sequentially in discrete time based on info received

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)

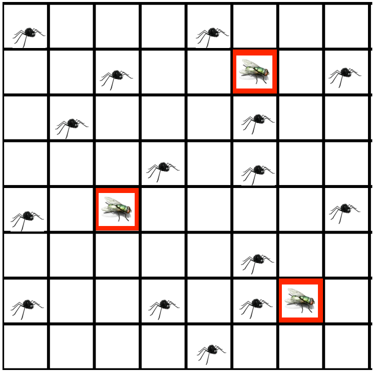


At each time: Agents have exact state info; choose their controls as function of state

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

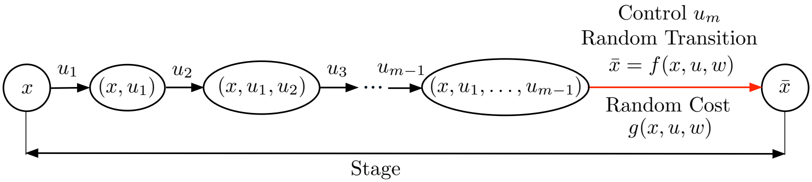
3 blind flies move randomly

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 choices) $\cdot$ (15 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)



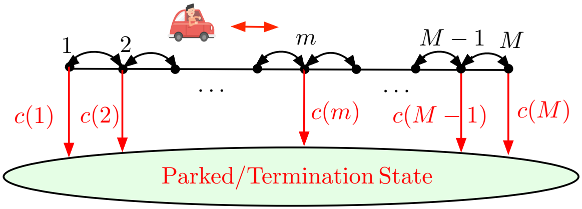
An equivalent reformulation - “Unfolding" the control action

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

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

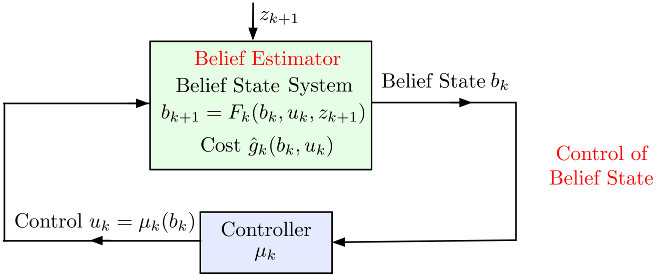
- Allows far more efficient rollout (one-agent-at-a-time). This is just standard rollout for the reformulated problem
- The increase in size of the state space does not adversely affect rollout (only one state per stage is looked at during on-line play)
- Complexity reduction: The one-step lookahead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

Parking with a Deadline: An Example of Partial State Observation



- At each time step, move one spot in either direction. Decide to park or not at spot m (if free) at cost $c(m)$. If we have not parked by time N there is a large cost C
- We observe the free/taken status of only the spot we are in. Parking spots may change status at the next time step with some probability.
- The free/taken status 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: $p(1), p(2), \ldots, p(M)$, conditioned on all the observations so far

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

About the Next Lecture

- We will discuss partial state observation problems, adaptive, and model predictive control
- 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): EXERCISE 1.2 OF CLASS NOTES

WATCH 2ND HALF OF VIDEOLECTURE 3 AND 1ST HALF OF VIDEOLECTURE 4 OF THE 2021 OFFERING OF THE COURSE