Topics in Reinforcement Learning: Rollout and Approximate Policy Iteration

ASU, CSE 691, Spring 2020

Dimitri P. Bertsekas
dbertsek@asu.edu

Lecture 3

Outline

Recall 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

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

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

Online implementation of the optimal policy, given $J_1^*, \ldots, J_N^*$

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

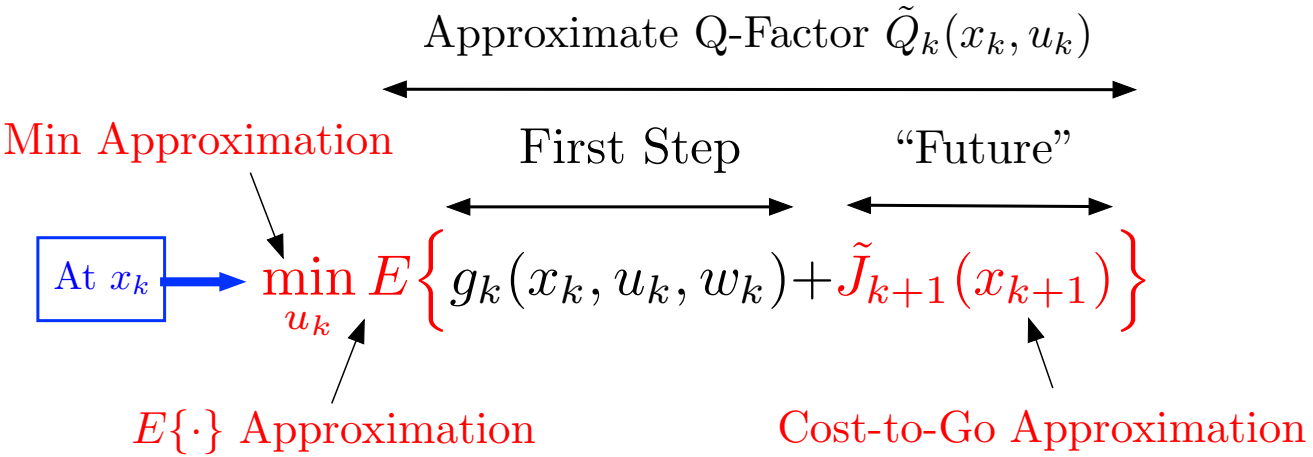
$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big) \Big\}.$$

The main difficulties: Too much computation, too much memory storage.

We will outline the main conceptual RL framework to deal with these difficulties

- Approximation in value space: Use $\tilde{J}_{k+1}$ in place of J_{k+1}^*; possibly approximate $E\{\cdot\}$ and $\min_{u_k}$
- Approximation in policy space: Directly approximate the optimal policies

Approximation in Value Space: One-Step Lookahead



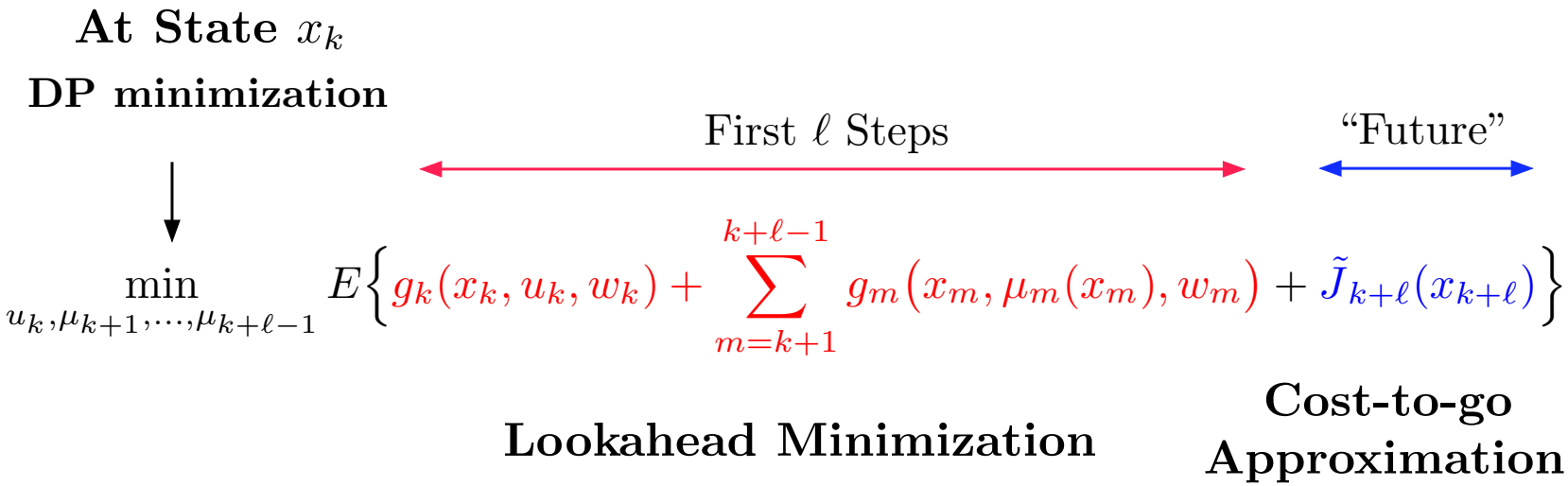
At state x_k, approximation in value space uses $\tilde{J}_{k+1}$ (in place of J^*_{k+1}) and lookahead minimization to obtain a suboptimal control $\tilde{u}_k = \tilde{\mu}_k(x_k)$.

THE THREE APPROXIMATIONS:

- How to construct $\tilde{J}_{k+1}$, $k = 0, \ldots, N-1$.
- How to simplify $E\{\cdot\}$ operation.
- How to simplify min operation.

Each of the three approximations can be designed almost independently of the others, leading to a large variety of methods.

Approximation in Value Space: Multistep Lookahead

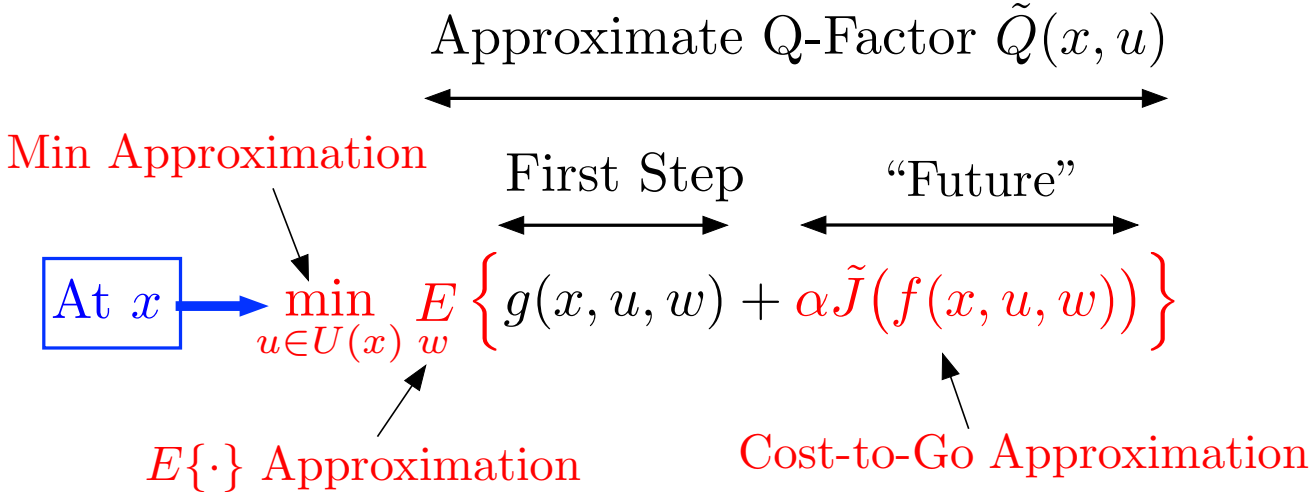


- At state x_k, we solve an ℓ-stage version of the DP problem with x_k as the initial state and $\tilde{J}_{k+\ell}$ as the terminal cost function.
- Use the first control of the ℓ-stage policy thus obtained, while discarding the others.

Hoped benefits from using the more costly multistep optimization:

- Minimization over many steps will work better than minimization over few steps (with long enough lookahead we are optimal).
- By using a long-step lookahead, we can afford a simpler/less accurate cost-to-go approximation $\tilde{J}_{k+\ell}$.

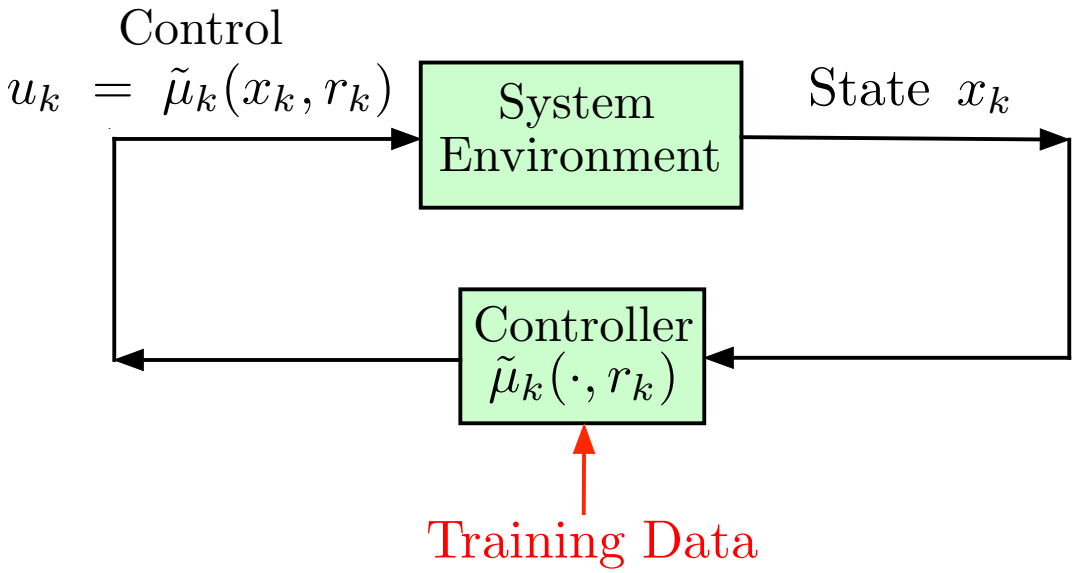
Approximation in Value Space - Infinite Horizon



Major advantages of the infinite horizon context

- Only one approximate cost function $\tilde{J}$ is needed, rather than the N functions $\tilde{J}_1, \ldots, \tilde{J}_N$ of the N-step horizon case.
- Additional important algorithms are available for infinite horizon approximation in value space. Approximate policy iteration, Q-learning, temporal difference methods, and their variants are some of these.
- Many of the finite horizon RL ideas generalize to infinite horizon ... so it is convenient to develop them first within the simpler framework of finite horizon.

Approximation in Policy Space: The Major Alternative to Approximation in Value Space



- Idea: Select the policy by optimization over a suitably restricted class of policies.
- The restricted class is usually a parametric family of policies $\mu_k(x_k, r_k)$, $k = 0, \ldots, N-1$, of some form, where r_k is a parameter (e.g., a neural net).
- Important advantage once the parameters r_k are computed: The computation of controls during on-line operation of the system is often much easier ... at state x_k apply $u_k = \mu_k(x_k, r_k)$.
- Often $\tilde{\mu}_k(x_k, r_k)$ is computed as a randomized policy, i.e., a set of probabilities of applying each of the available controls at x_k. It is implemented by applying at state x_k the control of maximum probability.

Approximation in Policy Space on Top of Approximation in Value Space

The approximate cost-to-go functions $\tilde{J}_{k+1}$ define a suboptimal policy $\tilde{\mu}_k$ through one-step lookahead.

- Given functions $\tilde{J}_{k+1}$, how do we simplify computation of the lookahead policy?
- Idea: Approximate $\tilde{\mu}_k$ using some form of regression and a training set consisting of a large number q of sample pairs $\left(x_k^s, u_k^s\right)$, $s = 1, \ldots, q$, where $u_k^s = \tilde{\mu}_k(x_k^s)$, i.e.,

$$u_k^s \in \arg \min_{u \in U_k(x_k)} E\Big\{ g_k(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k)\big) \Big\}$$

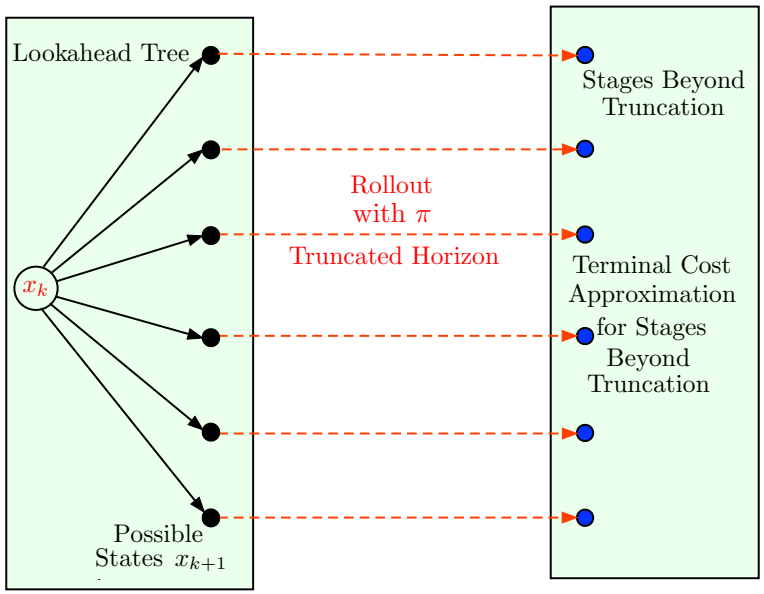
  Similarly for multistep lookahead.

- Example: Introduce a parametric family of randomized policies $\mu_k(x_k, r_k)$, $k = 0, \ldots, N-1$, of some form (e.g., a neural net), where r_k is a parameter. Then estimate the parameters r_k by

$$r_k \in \arg \min_r \sum_{s=1}^{q} \left\| u_k^s - \mu_k(x_k^s, r) \right\|^2$$

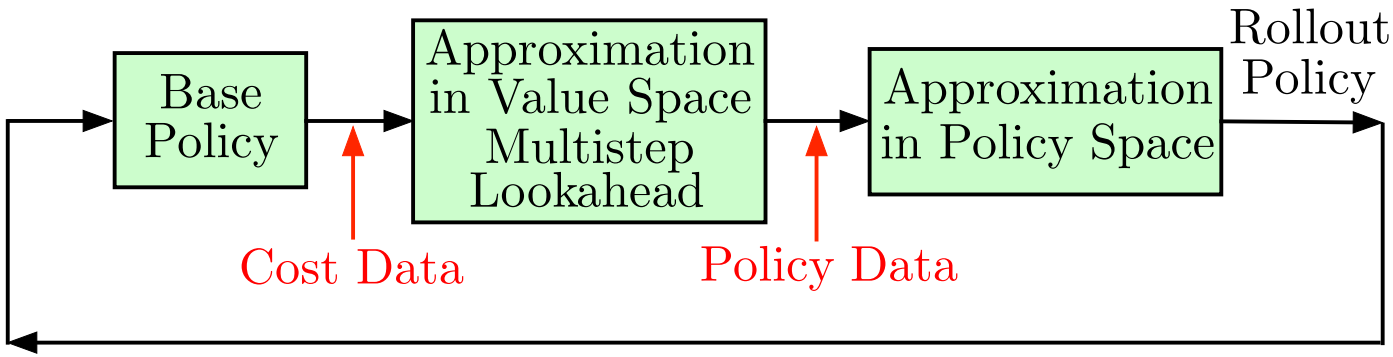
- Note that to apply regression the parametrization $\mu_k(x_k^s, r)$ must take continuous values. Often, u_k^s takes values 0 or 1 and $\mu_k(x_k, r)$ is a randomized policy.

Approximation in Value Space on Top of Approximation in Policy Space



- Start with some policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, called base policy, possibly obtained through approximation in policy space.
- Use one-step or multistep lookahead where $\tilde{J}_{k+1}(x_{k+1})$ is equal to the tail problem cost $J_{k+1,\pi}(x_{k+1})$ starting from x_{k+1} and using policy π.
- The policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \mu_{N-1}\}$ thus obtained is called the rollout policy.
- Major issue: How to compute $J_{k+1,\pi}(x_{k+1})$?
  - For deterministic problems: Run π from x_{k+1} once and accumulate stage costs.
  - For stochastic problems: Run π from x_{k+1} many times and Monte Carlo average.
  - Simulate π for a limited number of stages, and neglect the costs of the remaining stages or add some heuristic cost approximation at the end to compensate. This is called truncated rollout.

Combined Approximation in Value and Policy Space



Perpetual rollout and policy improvement

- A fundamental property: In its idealized form (no approximations) each new policy has no worse cost function than the preceding one, i.e., for all x_k and k,

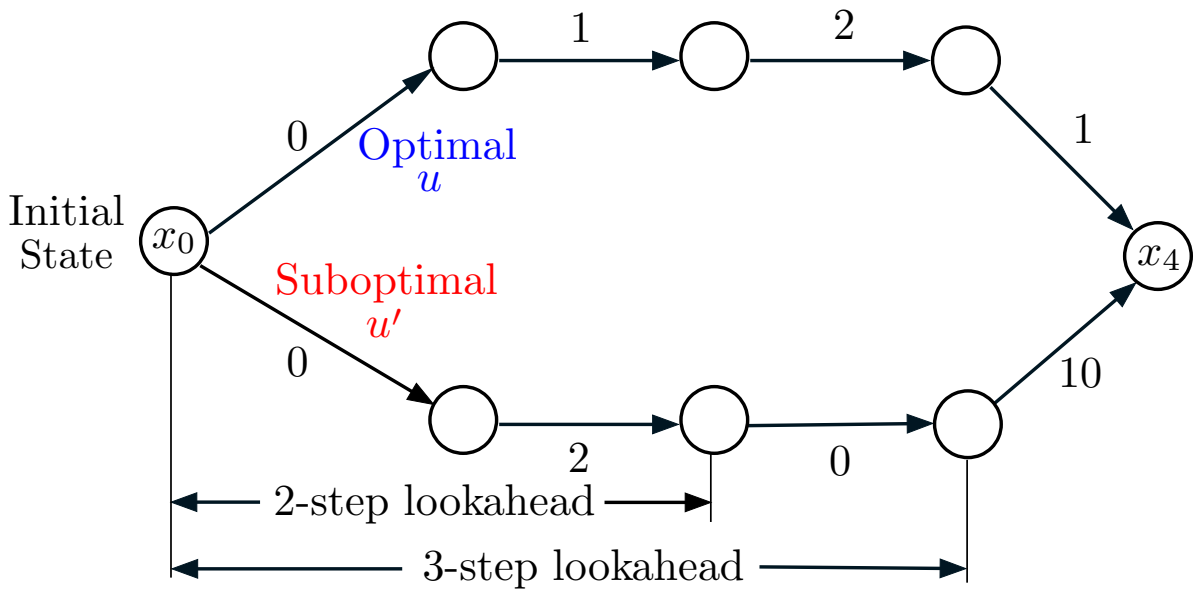
$$J_{k,\tilde{\pi}}(x_k) \leq J_{k,\pi}(x_k)$$

- Thus the algorithm is capable of self-improvement or self-learning.
- Its natural extension to infinite horizon problems is the policy iteration algorithm, and its foundation is the policy improvement property.
- With approximations, self-improvement is approximate (to within an error bound).
- There are many variations of this scheme: Optimistic policy iteration, Q-learning, temporal differences, etc. They involve challenging implementation issues.
- Most RL algorithms, including AlphaGo and Alphazero, use variants of the above scheme.

Let's Take a Working Break to Consider the Following Challenge Question

Will longer lookahead produce a better policy than shorter lookahead?

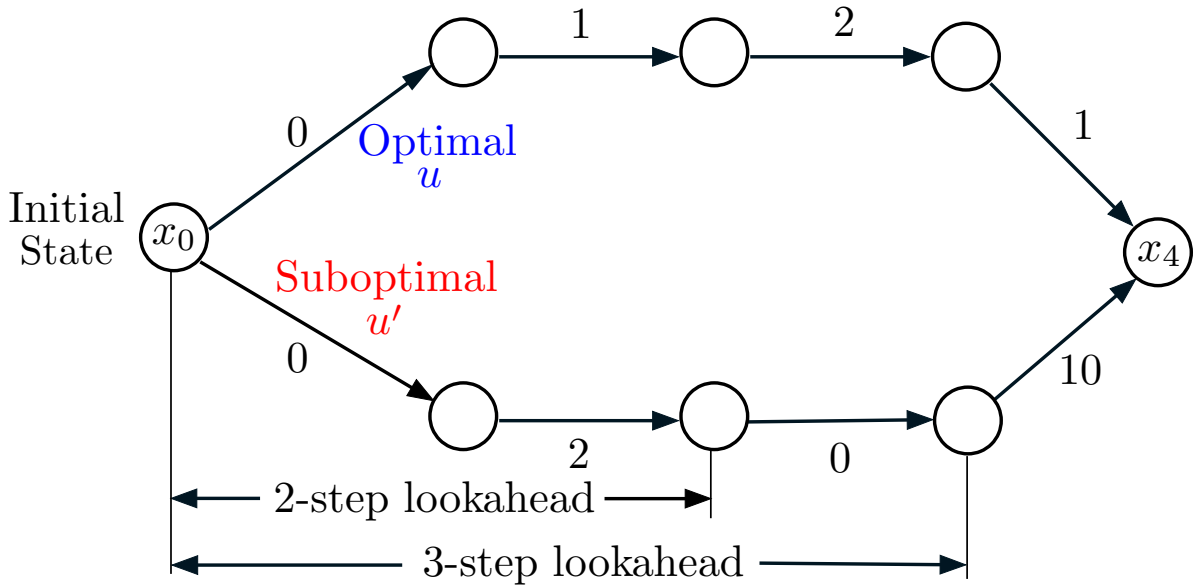
Consider the following example



Two controls, u, u', and cost function approximation $\tilde{J}_k(x_k) \equiv 0$.

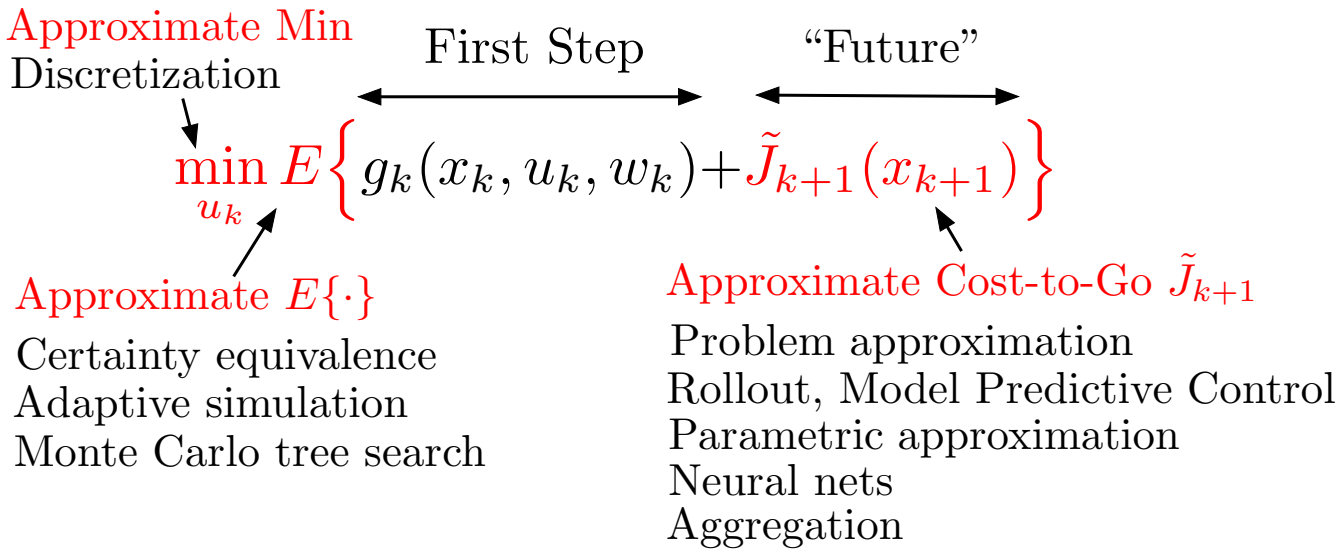
There is a choice only at x_0.

The Answer is “Usually", but NOT for this Example



Problem with “edge effects": u will be preferred based on 2-step lookahead. u' will be preferred based on 3-step lookahead.

On-Line and Off-Line Lookahead Implementations



- For many-state problems, the minimizing controls $\tilde{\mu}_k(x_k)$ are computed on-line (storage issue).
- Off-line methods: All the functions $\tilde{J}_{k+1}$ are computed for every k, before the control process begins.
- Examples of off-line methods: Neural network and other parametric approximations; also aggregation.
- On-line methods: The values $\tilde{J}_{k+1}(x_{k+1})$ are computed only at the relevant next states x_{k+1}, and are used to compute the control to be applied at the N time steps.
- Examples of on-line methods: Rollout and model predictive control.
- Rollout is well-suited for on-line replanning ... involves lots of on-line computation.

Simplifying the Minimization in Lookahead Schemes

$$\min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, w_k)\big) \Big\}$$

- If $U_k(x_k)$ is a finite set, the minimization can be done by brute force.
- If $U_k(x_k)$ is an infinite set, it may be replaced by a finite set through discretization.
- For deterministic problems and continuous control spaces, a more efficient alternative may be to use nonlinear programming techniques.
- For stochastic problems and continuous control spaces, we may use stochastic programming. Lookahead must be short because of the high branching factor of the lookahead tree when the problem is stochastic.

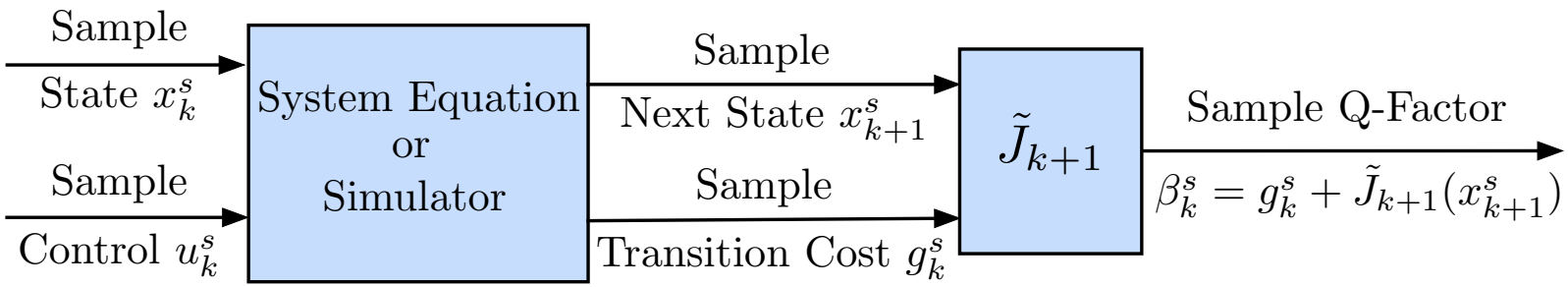
One simplification possibility is to simplify the $E\{\cdot\}$:

Assumed certainty equivalence, i.e., choose a typical value $\tilde{w}_k$ of w_k, and use the control $\tilde{\mu}_k(x_k)$ that solves the deterministic problem

$$\min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k, \tilde{w}_k) + \tilde{J}_{k+1}\big(f_k(x_k, u_k, \tilde{w}_k)\big) \Big]$$

However, this may degrade performance significantly.

Another Approach to Simplifying the Minimization: Policy Space Approximation



- Collect (off-line) a large number of "representative" samples $(x_k^s, u_k^s, x_{k+1}^s, g_k^s)$ and corresponding sample Q-factors

$$\beta_k^s = g_k^s + \tilde{J}_{k+1}(x_{k+1}^s), \qquad s = 1, \ldots, q$$

- Introduce a parametric family of Q-factors $\tilde{Q}_k(x_k, u_k, r_k)$.
- Determine the parameter vector $\bar{r}_k$ by the least-squares regression

$$\bar{r}_k \in \arg\min_{r_k} \sum_{s=1}^{q} \left(\tilde{Q}_k(x_k^s, u_k^s, r_k) - \beta_k^s\right)^2$$

- Use (on-line) the policy $\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k, \bar{r}_k)$

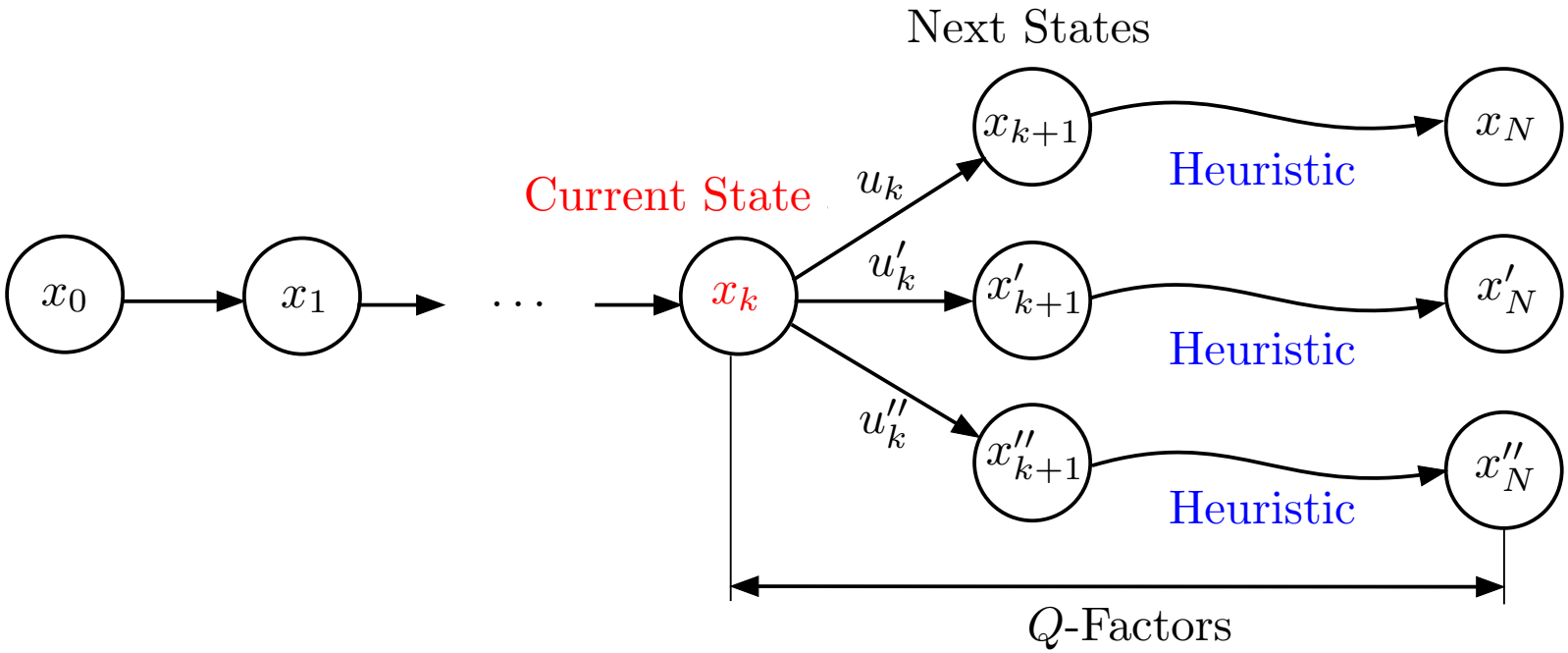
Rollout will be Important for this Course

Aim of rollout: Start with a policy, get a better policy.

Reasons why it will be important:

- Rollout is the RL method that is easiest to understand and apply
- Rollout is the not the most ambitious RL method, but it is the most reliably successful
- It is very general: Applies to deterministic and stochastic, to finite horizon and infinite horizon
- It contains as a special case model predictive control, one of the most important control system design methods
- It forms a building block for most of RL methods used in practice (including approximate policy iteration, Q-learning, temporal differences, etc)
- We will go fairly deeply into the subject and cover new research

General Structure of Deterministic Rollout with Some Base Heuristic

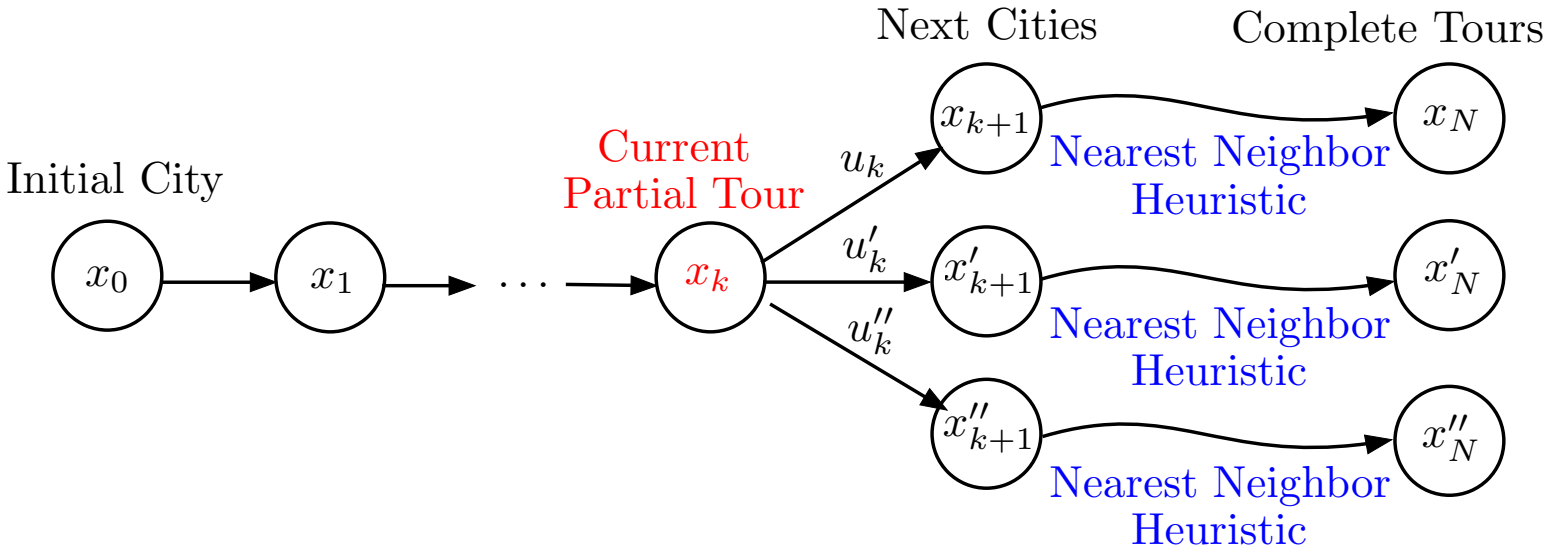


- At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, we generate a Q-factor

$$\tilde{Q}_k(x_k, u_k) = g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big)$$

  using the base heuristic [$H_{k+1}(x_{k+1})$ is the heuristic cost starting from x_{k+1}].
- We select the control u_k with minimal Q-factor.
- We move to next state x_{k+1}, and continue.
- Multistep lookahead versions (length of lookahead limited by the branching factor of the lookahead tree).

Traveling Salesman Example of Rollout with a Greedy Heuristic



- N cities $c = 0, \ldots, N-1$; each pair of distinct cities c, c', has traversal cost $g(c, c')$.
- Find a minimum cost tour that visits each city once and returns to the initial city.
- Recall that it can be viewed as a shortest path/deterministic DP problem. States are the partial tours, i.e., the sequences of ordered collections of distinct cities exponentially growing size of state space.
- Nearest neighbor heuristic; chooses the best one-hop extension of a partial tour.
- Rollout algorithm: Start at some city; given a partial tour $\{c_0, \ldots, c_k\}$ of distinct cities, select as next city c_{k+1} the one that yielded the minimum cost tour under the nearest neighbor heuristic.

Criteria for Cost Improvement of a Rollout Algorithm - Sequential Consistency

- Special conditions must hold to guarantee that the rollout policy has no worse performance than the base heuristic.
- Two such conditions are sequential consistency and sequential improvement.
- A sequentially consistent heuristic is also sequentially improving.
- Any heuristic can be modified to become sequentially improving.

The base heuristic is sequentially consistent if it "stays the course"

- If the heuristic generates the sequence

$$\{x_k, x_{k+1}, \ldots, x_N\}$$

starting from state x_k, it also generates the sequence

$$\{x_{k+1}, \ldots, x_N\}$$

starting from state x_{k+1}.

- The base heuristic is sequentially consistent if and only if it can be implemented with a legitimate DP policy $\{\mu_0, \ldots, \mu_{N-1}\}$.
- Greedy heuristics are sequentially consistent (e.g., nearest neighbor for TS).

Policy Improvement for Sequentially Improving Heuristics

Sequential improvement holds if for all x_k (Best heuristic Q-factor $\leq$ Heuristic cost):

$$\min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big) \Big] \leq H_k(x_k),$$

where $H_k(x_k)$ is the cost of the trajectory generated by the heuristic starting from x_k.
True for a sequentially consistent heuristic [$H_k(x_k)$ is the Q-factor of the heuristic at x_k].

Cost improvement property for a sequentially improving heuristic

Let the rollout policy be $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$, and let $J_{k,\tilde{\pi}}(x_k)$ denote its cost starting from x_k. Then for all x_k and k, $J_{k,\tilde{\pi}}(x_k) \leq H_k(x_k)$.

Proof by induction: It holds for $k = N$, since $J_{N,\tilde{\pi}} = H_N = g_N$. Assume that it holds for index $k + 1$.

$$\begin{aligned} J_{k,\tilde{\pi}}(x_k) &= g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k)\big)\Big) \\ &\leq g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + H_{k+1}\big(f_k(x_k, \tilde{\mu}_k(x_k))\big) \\ &= \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k, u_k)\big) \Big] \\ &\leq H_k(x_k) \end{aligned}$$

About the Next Lecture

We will cover:

- Rollout for deterministic and stochastic problems
- Monte Carlo tree search
- Model predictive control

PLEASE READ AS MUCH OF SECTIONS 2.3, 2.4, 2.5 AS YOU CAN

PLEASE DOWNLOAD THE LATEST VERSION OF NOTES FROM CANVAS