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 4
A Closer Look at Approximations in DP/RL

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 $J_1^*, \ldots, J_{N-1}^*$

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

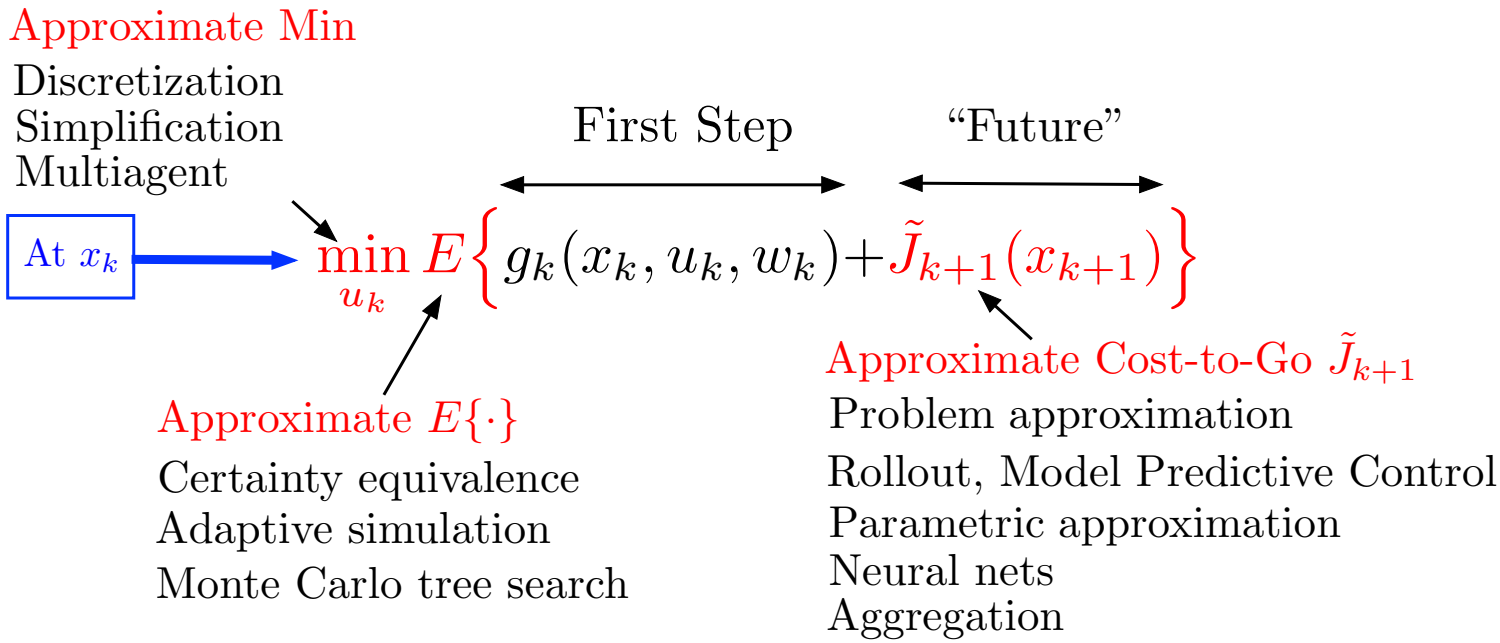
$$u_k^* \in \arg \min_{u_k \in U_k(x_k)} E\Big\{ g_k(x_k, u_k, w_k) + J_{k+1}^*\big(f_k(x_k, u_k, w_k)\big) \Big\}.$$

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

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 Value Space: One-Step Lookahead



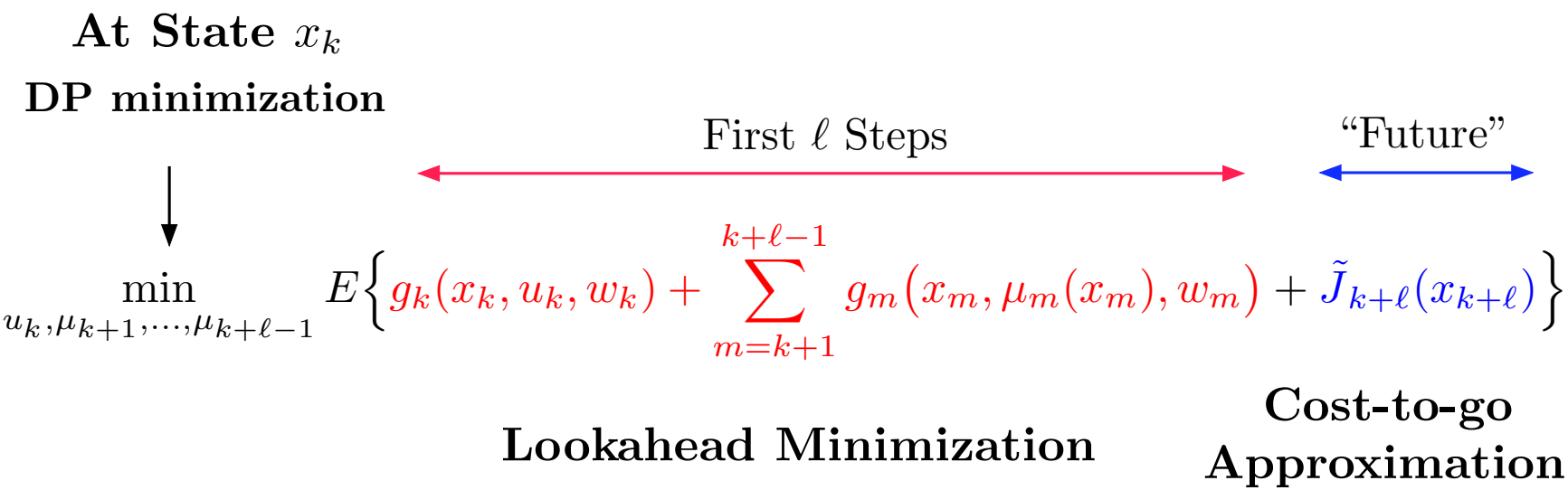
Approximation in value space and one-step lookahead minimization defines:

- The control to use at state x_k for on-line play
- A suboptimal control law $\tilde{\mu}_k$ that can be approximated by off-line training

The three approximations; they can be addressed separately

- How to construct the cost function approximations $\tilde{J}_{k+1}$.
- How to simplify $E\{\cdot\}$ operation.
- How to simplify min operation.

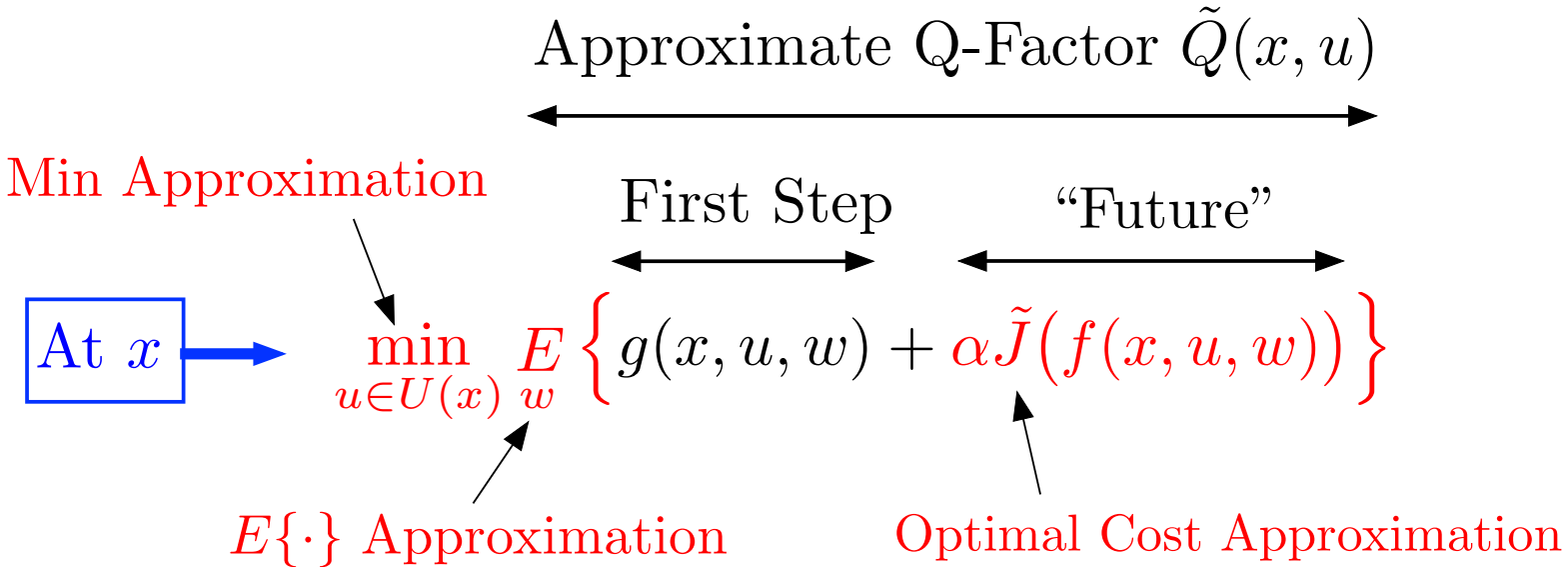
Approximation in Value Space: Multistep Lookahead



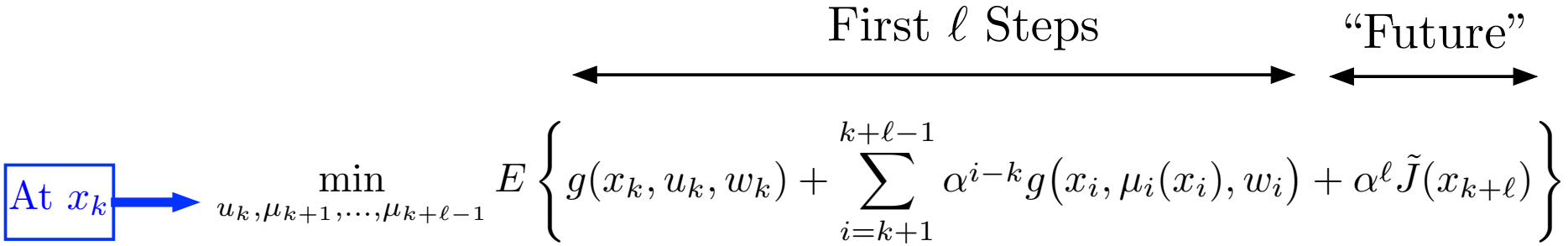
- At state x_k, we solve an ℓ-stage version of the DP problem with x_k as the initial state and $\tilde{J}_{k+\ell}$ as the terminal cost function
- Use the first control of the ℓ-stage policy thus obtained, discard the others

We can view ℓ-step lookahead as a special case of one-step lookahead:

The "effective" one-step lookahead approximate cost function is the optimal cost function of an $(\ell-1)$-stage DP problem with terminal cost $\tilde{J}_{k+\ell}$

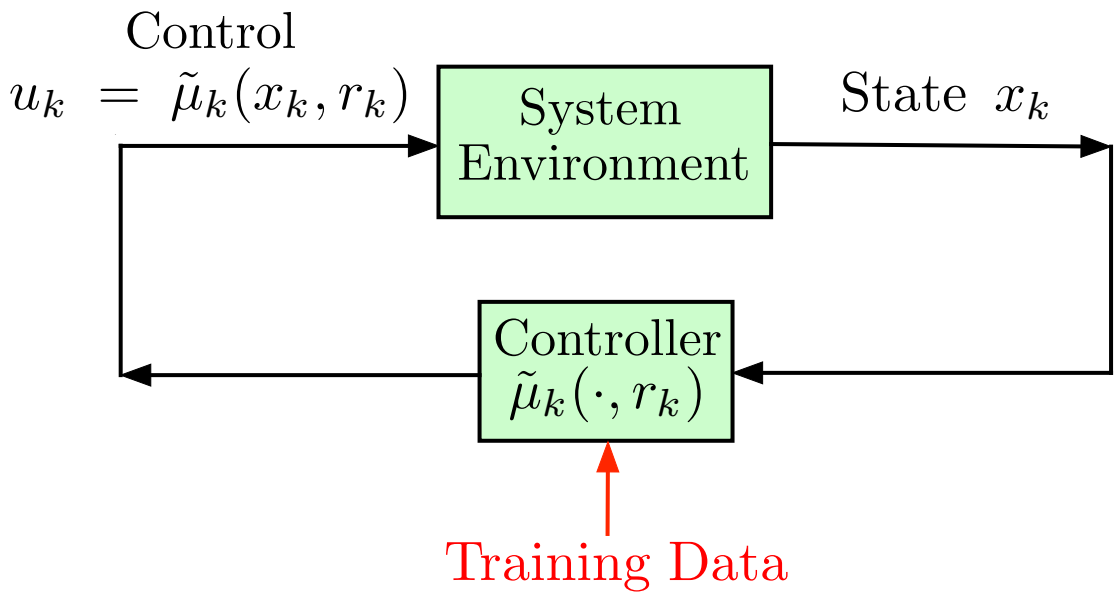


One-Step Lookahead



Multistep Lookahead

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 $\tilde{\mu}_k(x_k, r_k)$, $k = 0, \ldots, N-1$, of some form, where r_k is a parameter (e.g., a neural net)
- Important advantage once the parameters r_k are computed: The on-line computation of controls is often much faster ... at state x_k apply $u_k = \tilde{\mu}_k(x_k, r_k)$
- Important disadvantage: It does not allow for on-line replanning

From Values to Policies: 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 or multistep lookahead minimization

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

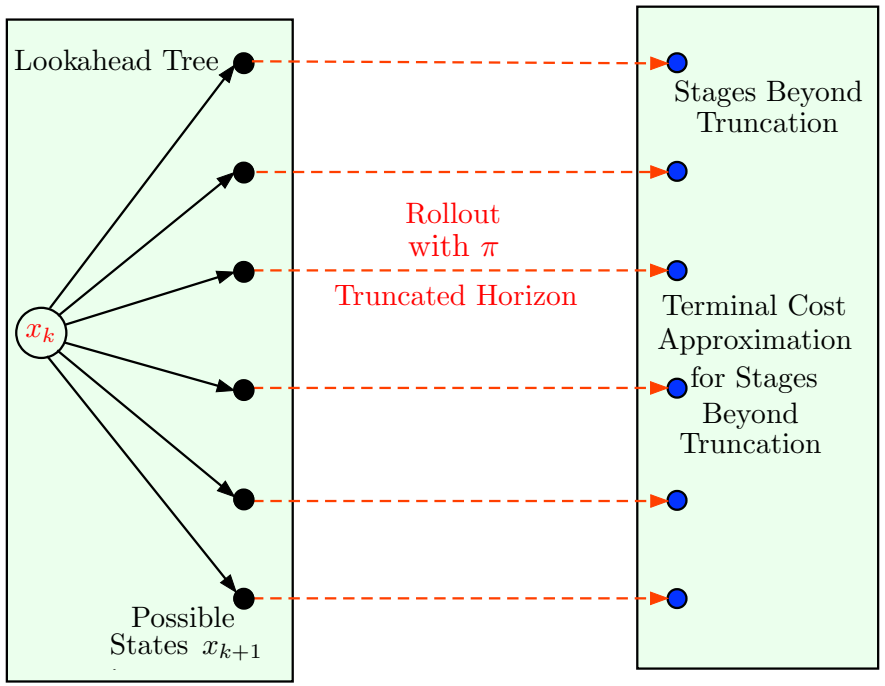
$$u_k^s \in \arg \min_{u \in U_k(x_k)} E\Big\{ g_k(x_k^s, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k^s, u, w_k)\big) \Big\}$$

  Similarly for multistep lookahead.
- Example (for finite number of controls): 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 least squares fit:

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

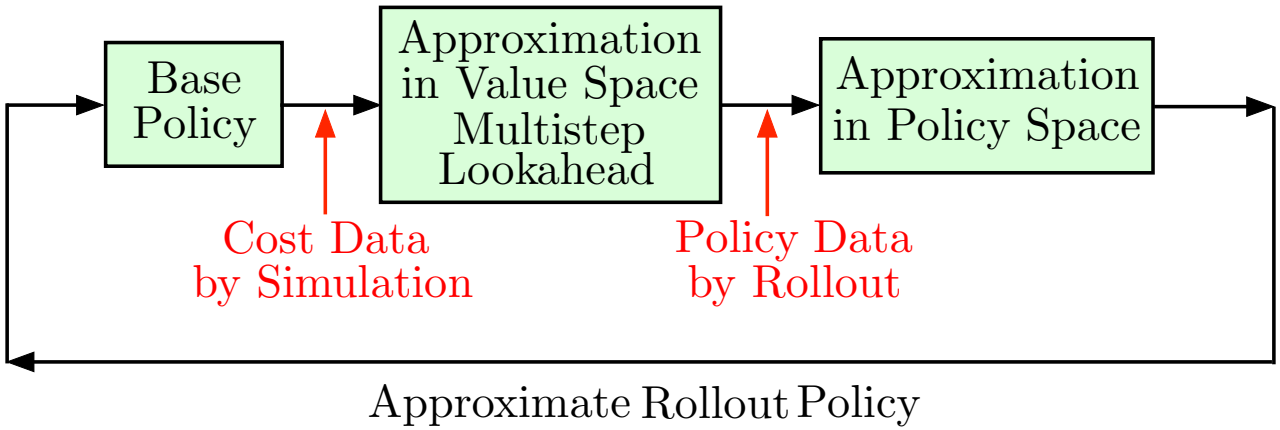
- For this, the parametrization $\mu_k(x_k^s, r)$ must take continuous values.
- To deal with this, u_k^s is coded to take values 0 or 1 and $\mu_k(x_k, r)$ is a randomized policy (relation to classification ... policy <–> classifier; more on this later).

From Policies to Values to New Policies by Rollout



- Start with some policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$, a base policy, possibly obtained through approximation in policy space
- Use one-step or multistep lookahead rollout where $\tilde{J}_{k+1}(x_{k+1}) \approx J_{k+1,\pi}(x_{k+1})$
- The policy $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \mu_{N-1}\}$ thus obtained is the truncated rollout policy
- Important 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

Combined Approximation in Value and Policy Space



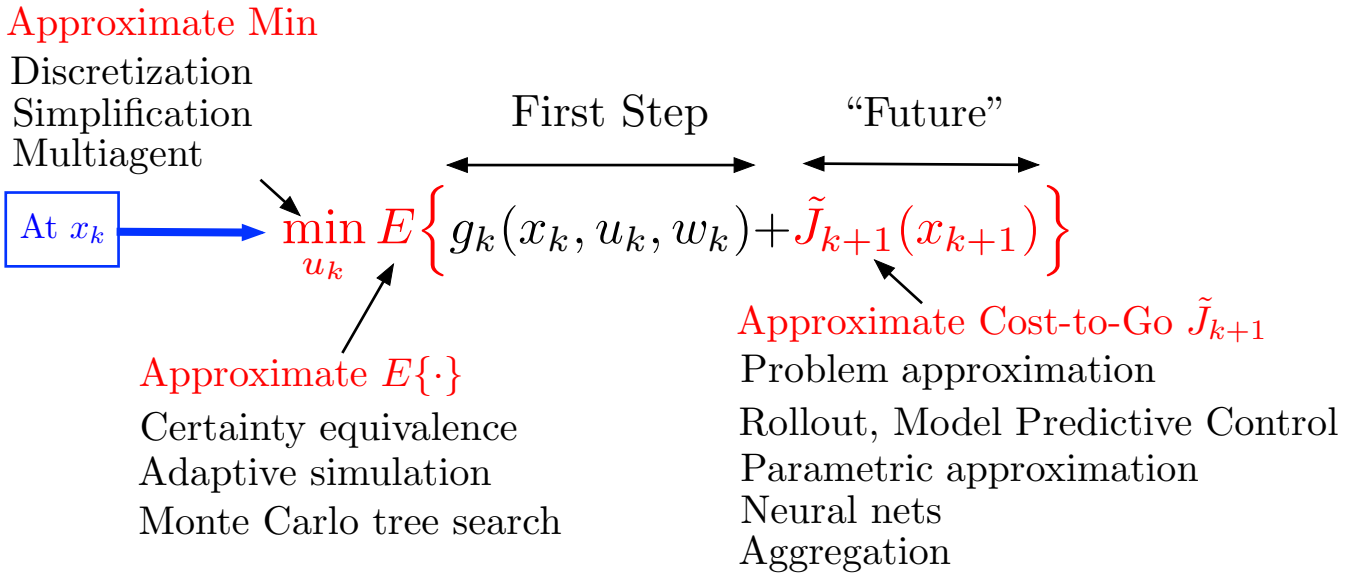
Perpetual rollout and policy improvement

- 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 (no external training data is needed)
- Its natural extension to infinite horizon problems is the policy iteration (PI) 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 PI, Q-learning, temporal differences, etc. They involve challenging implementation issues

Balance Between Off-Line and On-Line Computations



- Off-line methods (primarily): 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 in the context of PI-like methods; also aggregation.
- On-line methods (primarily): 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 MPC.

Simplifying the Expected Value Calculation: Probabilistic Approximation

Modify the probability distributions $P(w_k \mid x_k, u_k)$ to simplify the lookahead minimization and/or the calculation/training of $\tilde{J}_{k+\ell}$.

Assume certainty equivalence (inspired by linear-quadratic control problems)

- Replace uncertain quantities with deterministic nominal values.
- Then the lookahead and tail problems are deterministic, so they could be solvable by DP or by special deterministic methods on-line.
- Use expected values or forecasts to determine nominal values; update policy when forecasts change (on-line replanning).
- A major possibility for POMDP: Use state estimates instead of belief states.
- A variant: Partial certainty equivalence. Fix only some uncertain quantities to nominal values.
- A generalization: Approximate $E\{\cdot\}$ by limited simulation.

Model-Based vs Model-Free

What does model-free mean? Is it good or bad? There is no free lunch in RL

We will not deal with interactions with the environment for combined model identification and control (this is hard)

For us it's all model-based (but the model may be a computer/simulation model)

Monte Carlo simulation is useful when:

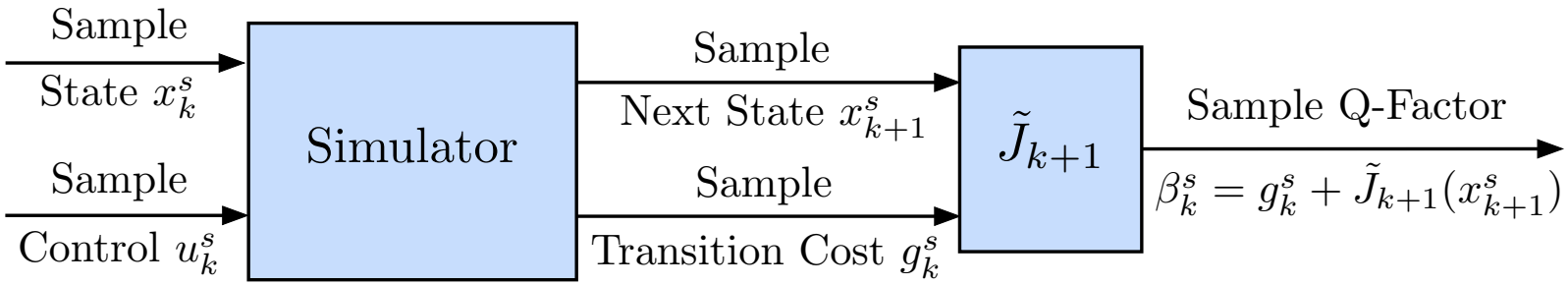
- A mathematical model of the probabilities $p_k(w_k \mid x_k, u_k)$ is not available but a computer model/simulator is. It simulates sample probabilistic transitions to a successor state x_{k+1}
- When for reasons of computational efficiency we prefer to compute expected values by using sampling and Monte Carlo simulation; e.g., approximate an integral or a huge sum of numbers by a Monte Carlo estimate

A common example: Simulation-based calculations of approximate Q-factors

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

See the next slide

Monte Carlo-Based Q-Factor Approximation for Stochastic Problems (for Given $\tilde{J}_{k+1}$)



- Use the simulator to collect a large number of "representative" samples of state-control-successor states-stage cost quadruplets $(x_k^s, u_k^s, x_{k+1}^s, g_k^s)$, and corresponding sample Q-factors

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

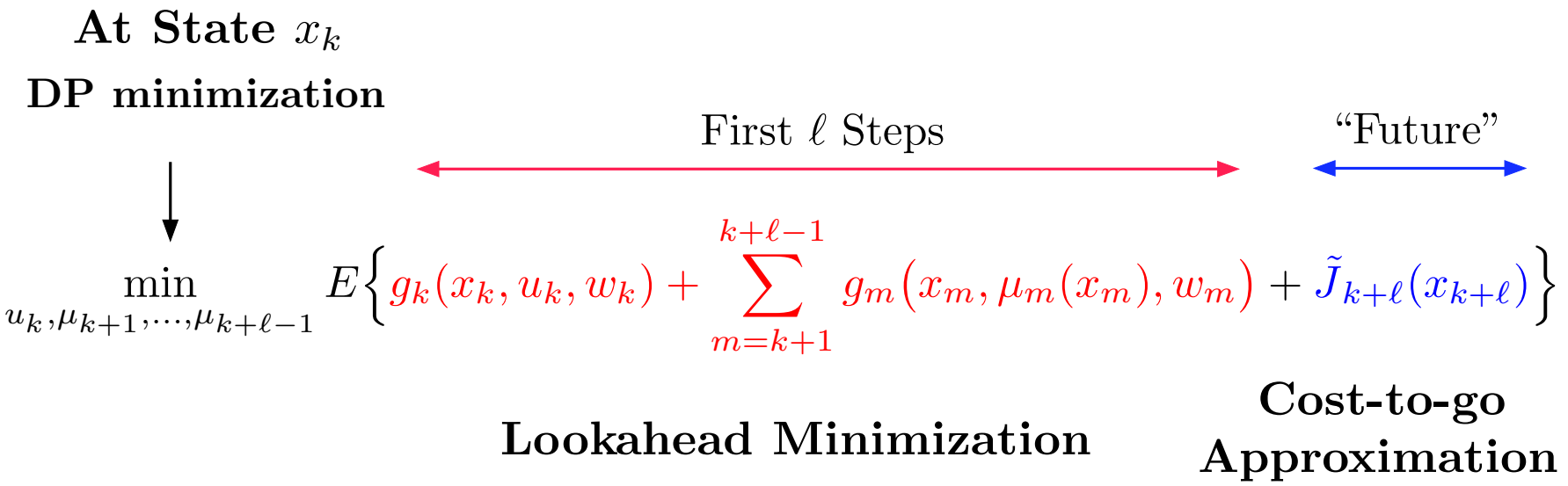
- Introduce a parametric family of Q-factors $\tilde{Q}_k(x_k, u_k, r_k)$.
- Determine the parameter vector $\bar{r}_k$ by the least-squares fit

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

- Use the policy

$$\tilde{\mu}_k(x_k) \in \arg\min_{u_k \in U_k(x_k)} \tilde{Q}_k(x_k, u_k, \bar{r}_k)$$

Multistep Lookahead Issues



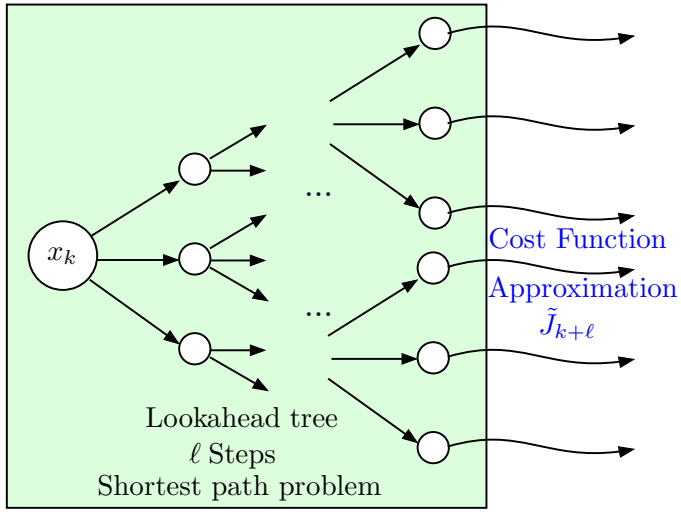
Main hope:

- Lookahead over many steps will work better than over few steps
- Intuition: With long lookahead we act optimally over more stages; with long enough lookahead we are optimal
- Bottom line: By using a long-step lookahead, we can afford a simpler/less accurate cost-to-go approximation.

Main difficulty:

Minimization over many stages is costly; stochastic problems are harder because of a larger branching factor of the lookahead tree.

Multistep Lookahead and Deterministic Problems



If the problem is deterministic and finite-state, the lookahead minimization is a shortest path problem and may be solved on-line

If the problem has continuous-state/control

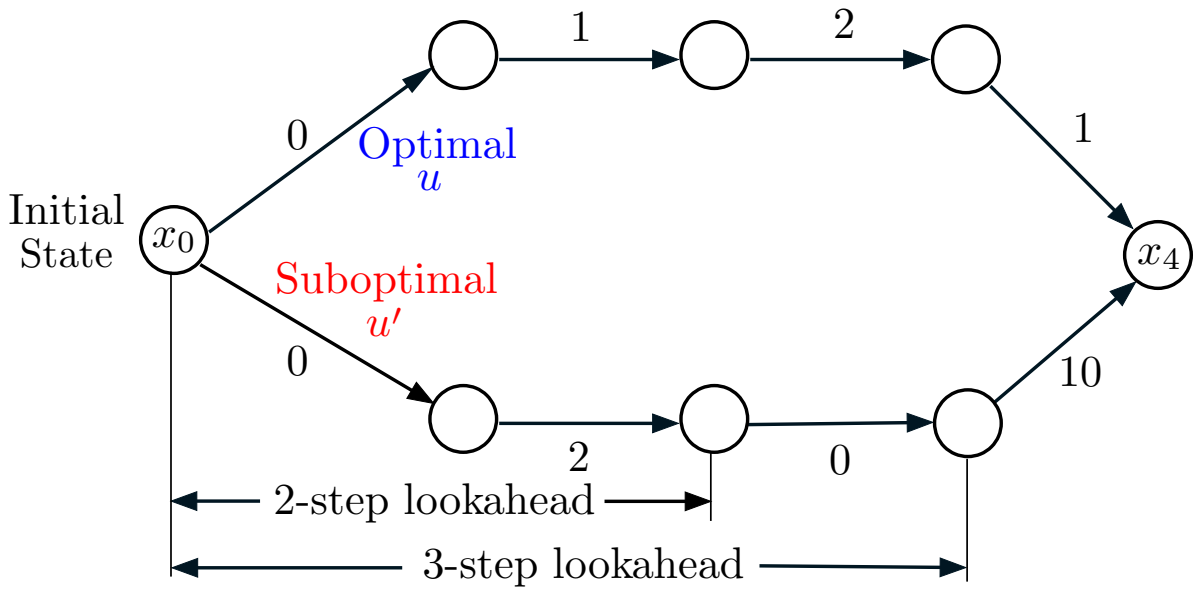
- If the problem is deterministic, the lookahead minimization may be quickly solvable by nonlinear programming (MPC case)
- If the problem is stochastic, the lookahead minimization may be solvable by stochastic programming (to be discussed later)

If the problem is stochastic and finite-state, the lookahead minimization can be split into a first stochastic step and a deterministic remainder; i.e., use a deterministic shortest path problem approximation for the remaining steps

Let's Take a Break; Consider the Following Challenge Question

Will longer lookahead produce a better policy than shorter lookahead?

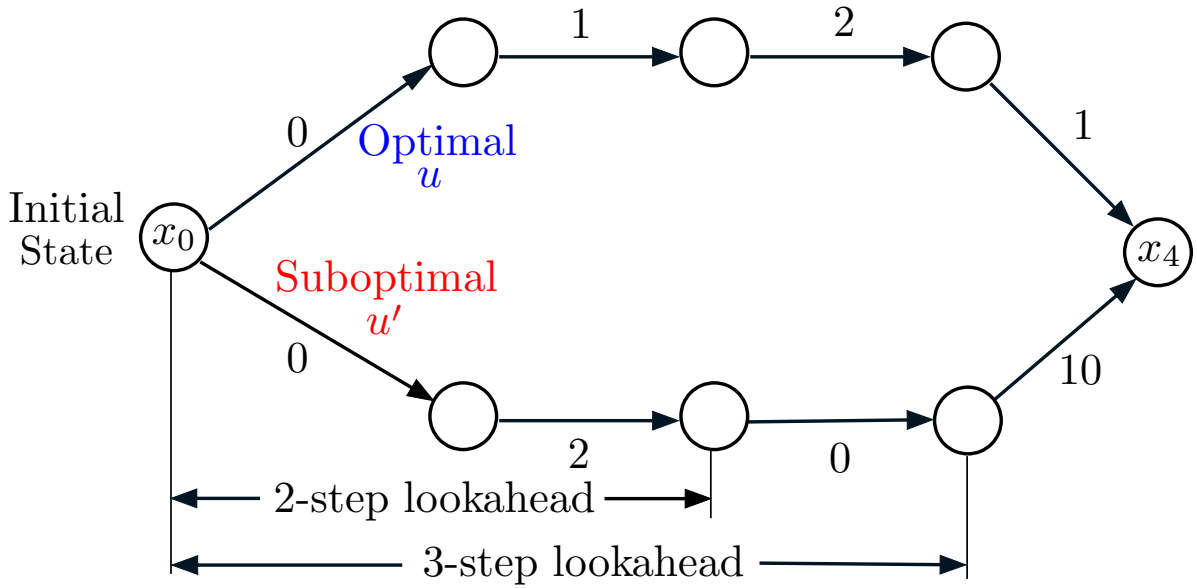
Consider the following example



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 that “Usually" Longer is Better, 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

Rollout will be Important for this Course

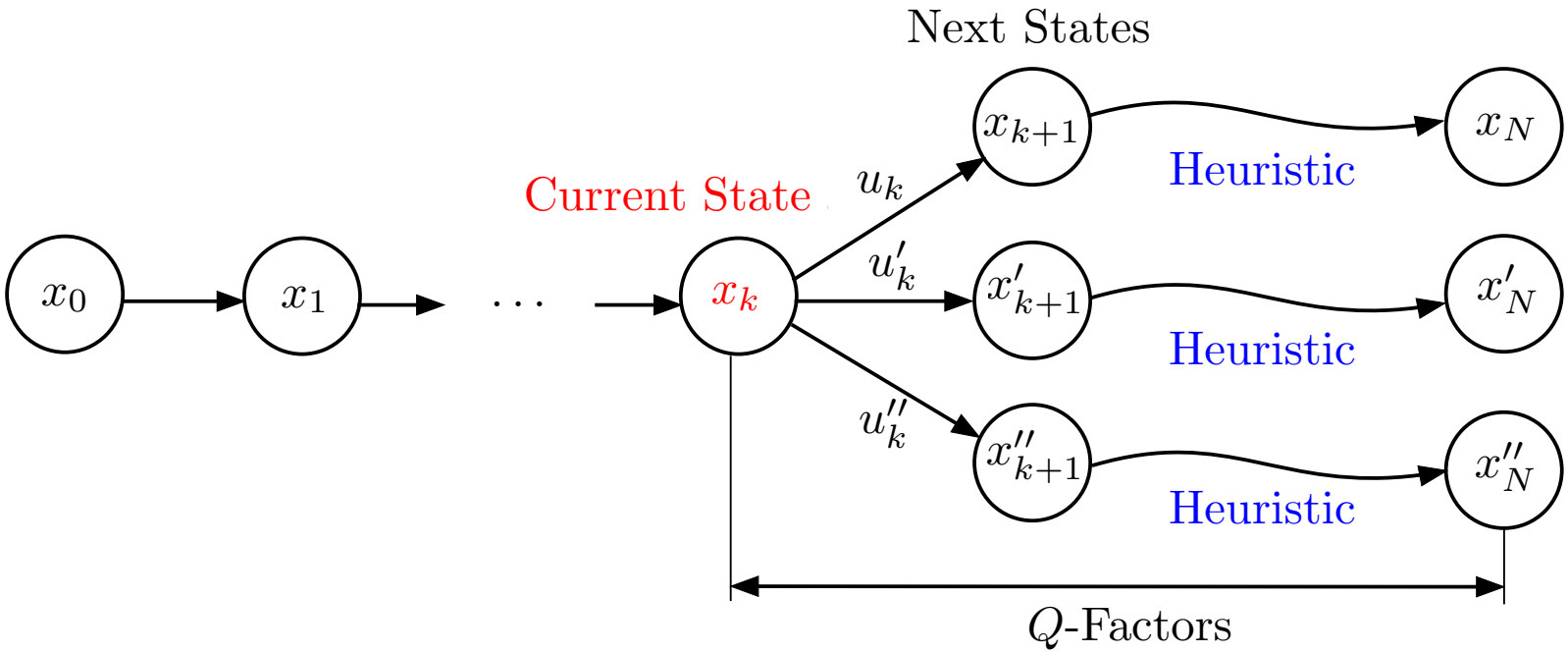
Aim of rollout: Start with a policy, get a better policy

It is the basic building block of the fundamental DP algorithm of policy iteration

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 many of the RL methods used in practice (including approximate policy iteration, Q-learning, temporal differences, etc)

General Structure of Deterministic Rollout with Some Base Heuristic

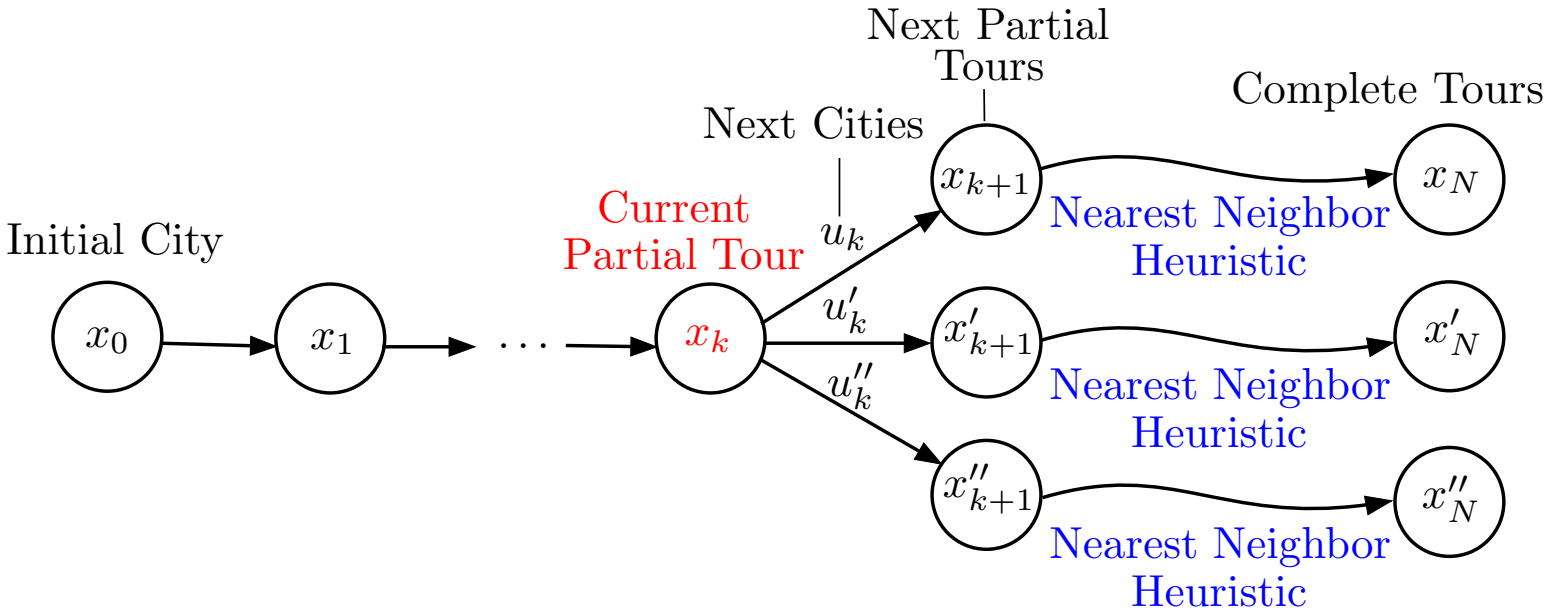


- At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, we generate a Q-factor

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

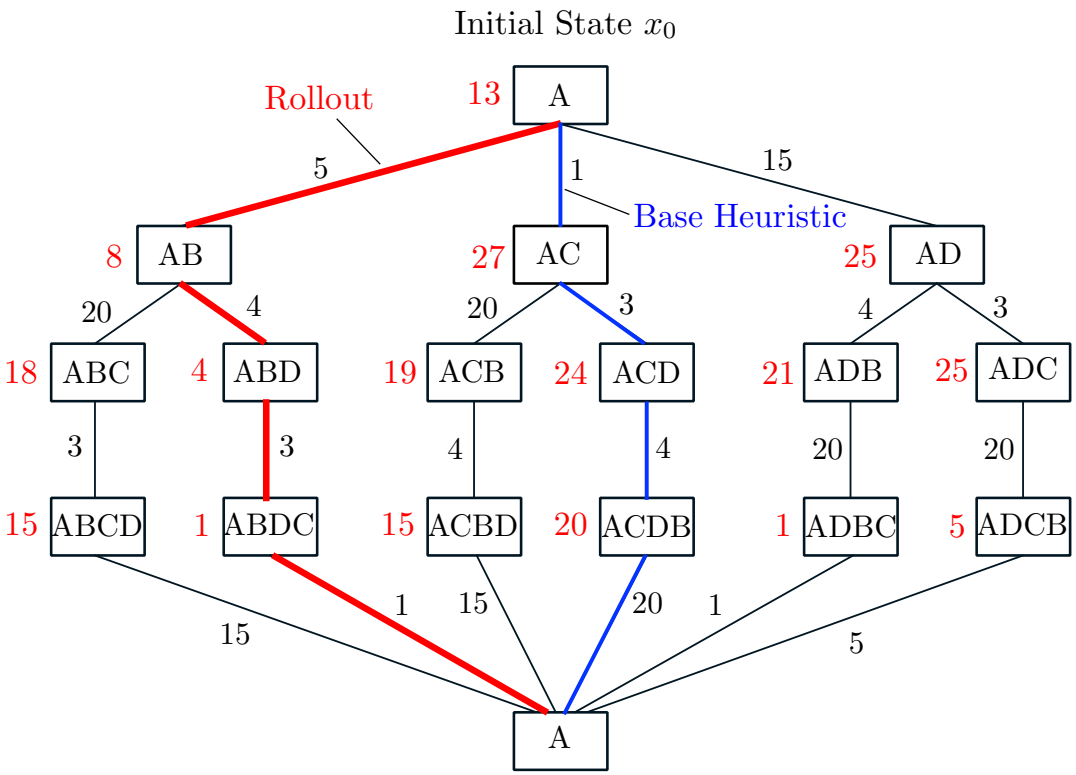
  using the base heuristic [$H_{k+1}(x_{k+1})$ is the heuristic cost starting from x_{k+1}]
- We select the control u_k with minimal Q-factor
- We move to next state x_{k+1}, and continue
- Multistep lookahead versions (length of lookahead limited by the branching factor of the lookahead tree)

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
- 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

Traveling Salesman Example: Rollout with a Nearest Neighbor Heuristic



Matrix of Intercity Travel Costs

	5	1	15
20		20	4
1	20		3
15	4	3	

Base heuristic: Nearest neighbor

The rollout algorithm has “long range vision" that the base heuristic lacks

Criteria for Cost Improvement of a Rollout Algorithm - Sequential Consistency

- Cost improvement is not automatic: 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 (see next lecture)

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)

Sequentially Improvement Criterion - Holds in MPC

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

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

where $H_k(x_k)$ is the cost of the trajectory generated by the heuristic starting from x_k
Holds under sequential consistency [$H_k(x_k)$ is the heuristic's Q-factor 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:

- Extensions of deterministic rollout
- Continuous time deterministic rollout
- Rollout for stochastic problems
- Monte Carlo tree search
- Continuous space deterministic rollout

Homework to be announced

Watch videolecture 4 from 2019 ASU course offering