Reinforcement Learning and Optimal Control

ASU, CSE 691, Winter 2019

Dimitri P. Bertsekas
dimitrib@mit.edu

Lecture 4

Outline

1. Approximation in Value Space and Rollout
2. On-Line Rollout for Deterministic Finite-State Problems
3. Stochastic Rollout and Monte Carlo Tree Search

Approximate Min
Discretization

First Step "Future"

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

Approximate $E\{\cdot\}$

Certainty equivalence
Adaptive simulation
Monte Carlo tree search

Approximate Cost-to-Go $\tilde{J}_{k+1}$

Problem approximation
Rollout, Model Predictive Control
Parametric approximation
Neural nets
Aggregation

ONE-STEP LOOKAHEAD

At State x_k
DP minimization

First ℓ Steps "Future"

$$\min_{u_k, \mu_{k+1}, \ldots, \mu_{k+\ell-1}} E\left\{ g_k(x_k, u_k, w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m, \mu_m(x_m), w_m\big) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}$$

Lookahead Minimization

Cost-to-go Approximation

MULTISTEP LOOKAHEAD

The Pure Form of Rollout

At State x_k

DP minimization

First ℓ Steps

"Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde J_{k+\ell}(x_{k+\ell}) \right\}$$

Lookahead Minimization

Heuristic Cost

Run the Base Policy

Use a suboptimal/heuristic policy at the end of limited lookahead

- The heuristic is called base policy (or default policy).
- The lookahead policy is called rollout policy.
- The aim of rollout is policy improvement (i.e., rollout policy performs better than the base policy); true under some assumptions. In practice: good performance, very reliable, very simple to implement.
- Rollout in its "standard" forms involves simulation and on-line implementation.
- The simulation can be prohibitively expensive (so further approximations may be needed); particularly for stochastic problems and multistep lookahead.

Connection/Overlap with Other Methods

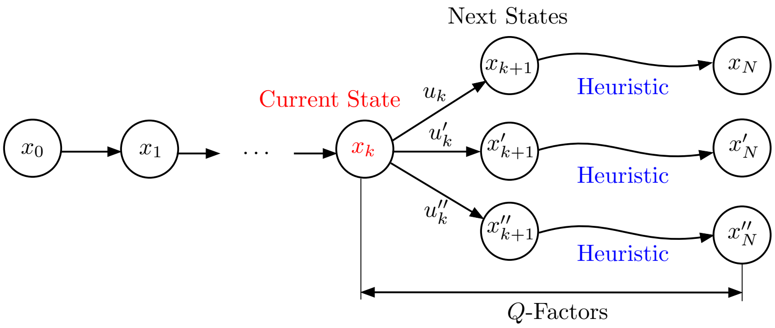
Connection with problem approximation

- Suppose the base heuristic is an optimal policy for the approximating problem.
- Then rollout is lookahead with problem approximation: the optimal cost of the approximating problem is used as lookahead function.
- True for both one-step and multistep lookahead.

Connection with policy iteration/self learning - Infinite horizon problems

- Rollout can be viewed as one-step policy iteration (more on this later).
- Cost improvement property of rollout is based on the fundamental cost improvement property of policy iteration (more on this later).
- Policy iteration can be viewed as “perpetual" rollout, i.e., every so often replace the base policy with the current rollout policy (or an approximation thereoff).

General Structure of Deterministic Rollout



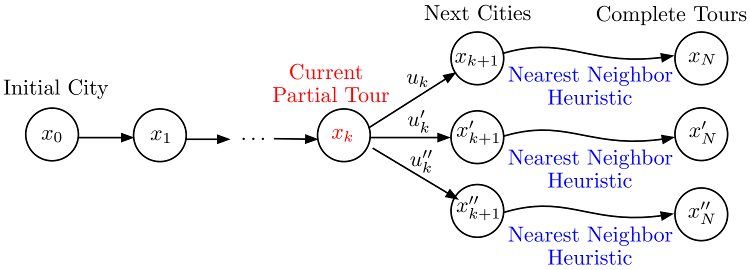
- At state x_k, for every pair (x_k, u_k), $u_k \in U_k(x_k)$, we generate a Q-factor

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

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 $c = 0, \dots, N-1$; 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 $\{c_0, \dots, c_k\}$ 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.

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].

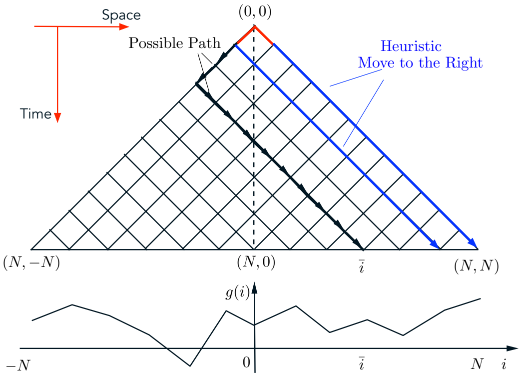
Cost improvement property for a sequentially improving heuristic

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 $k+1$.

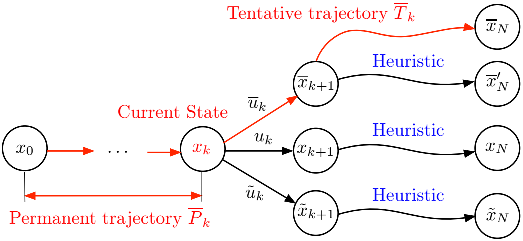
$$\begin{aligned}
J_{k,\tilde{\pi}}(x_k) &= g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + J_{k+1,\tilde{\pi}}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k)\big)\Big) \\
&\leq g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + H_{k+1}\big(f_k(x_k, \tilde{\mu}_k(x_k))\big) \\
&= \min_{u_k \in U_k(x_k)} \Big[g_k(x_k, u_k) + H_{k+1}\big(f_k(x_k,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

A Working Break: Challenge Question



- Walk on a line of length $2N$ starting at position 0. At each of N steps, move one unit to the left or one unit to the right.
- Objective is to land at a position i of small cost $g(i)$ after N steps.
- Question: Consider a base heuristic that takes steps to the right only. How will the rollout perform compared to the base heuristic?
- Compare with a superheuristic/combination of two heuristics: 1) Move only to the right, and 2) Move only to the left. Base heuristic chooses the path of best cost.

Fortified Rollout: Restores Cost Improvement for Base Heuristics that are not Sequentially Consistent



- Upon reaching state x_k it stores the permanent trajectory

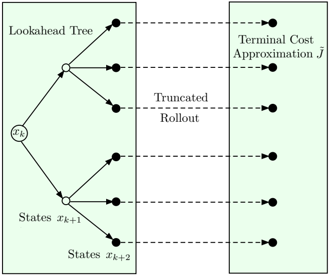
$$\overline{P}_k = \{x_0, u_0, \ldots, u_{k-1}, x_k\}$$

that has been constructed up to stage k, called, and it also stores a tentative trajectory

$$\overline{T}_k = \{x_k, \overline{u}_k, \overline{x}_{k+1}, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N\}$$

- The tentative trajectory is such that $\overline{P}_k \cup \overline{T}_k$ is the best end-to-end trajectory computed up to stage k of the algorithm.
- At each step follow the best trajectory.

Multistep Rollout with Terminal Cost Approximation



- Saves computation but the cost improvement property is lost.
- We can prove cost improvement, assuming sequential consistency and a special property of the terminal cost function approximation that resembles sequential improvement (more on this when we discuss infinite horizon rollout).
- It is not necessarily true that longer lookahead leads to improved performance; but usually true (similar counterexamples as in the last lecture).
- It is not necessarily true that increasing the length of the rollout leads to improved performance (some examples indicate this). Moreover, long rollout is costly.
- Experimentation with length of rollout and terminal cost function approximation are recommended.

Stochastic Rollout - Cost Improvement

At State x_k

DP minimization

First ℓ Steps

"Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{m=k+1}^{k+\ell-1} g_k\big(x_m,\mu_m(x_m),w_m\big) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}$$

Lookahead Minimization

Heuristic Cost

Run the Base Policy

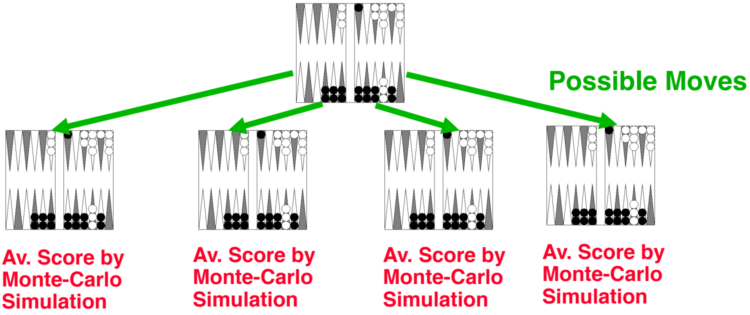
Consider the pure case (no truncation, no terminal cost approximation)

- Assume that the base heuristic is a legitimate policy $\pi = \{\mu_0, \ldots, \mu_{N-1}\}$ (i.e., is sequentially consistent, in the context of deterministic problems).
- Let $\tilde{\pi} = \{\mu_0, \ldots, \mu_{N-1}\}$ be the rollout policy. Then cost improvement is obtained

$$J_{k,\tilde{\pi}}(x_k) \le J_{k,\pi}(x_k), \qquad \text{for all } x_k \text{ and } k.$$

- Essentially identical induction proof as for the sequentially improving case (see the text).

Backgammon Example



- Announced by Tesauro in 1996.
- Truncated rollout with cost function approximation provided by TD-Gammon (earlier program involving a neural network trained by a form of policy iteration).
- Plays better than TD-Gammon, and better than any human.
- Too slow for real-time play (without parallel hardware), due to excessive simulation time.

Monte Carlo Tree Search - Motivation

We assumed equal effort for evaluation of Q-factors of all controls at a state x_k

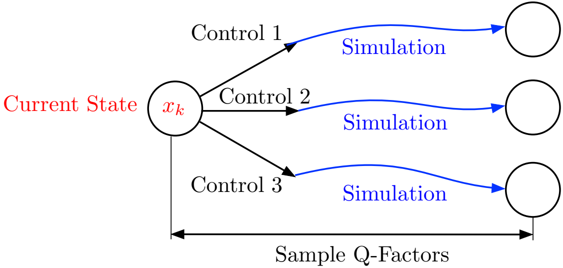
Drawbacks:

- The trajectories may be too long because the horizon length N is large (or infinite, in an infinite horizon context).
- Some of the controls u_k may be clearly inferior to others, and may not be worth as much sampling effort.
- Some of the controls u_k that appear to be promising, may be worth exploring better through multistep lookahead.

Monte Carlo tree search (MCTS) is a “randomized" form of lookahead

- MCTS aims to trade off computational economy with a hopefully small risk of degradation in performance.
- It involves adaptive simulation (simulation effort adapted to the perceived quality of different controls).
- Aims to balance exploitation (extra simulation effort on controls that look promising) and exploration (adequate exploration of the potential of all controls).

Monte Carlo Tree Search - Adaptive Simulation



Find a control $\tilde{u}_k$ that minimizes the approximate Q-factor

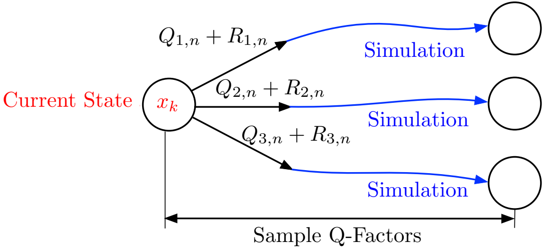
$$\tilde{Q}_k(x_k, u_k) = E\Big\{g_k(x_k, u, w_k) + \tilde{J}_{k+1}\big(f_k(x_k, u, w_k)\big)\Big\}$$

over $u_k \in U_k(x_k)$, by averaging samples of $\tilde{Q}_k(x_k, u_k)$.

Assume that $U_k(x_k)$ contains m elements, denoted $1, \ldots, m$

- After the nth sampling period we have $Q_{i,n}$, the empirical mean of the Q-factor of control i (total sample value divided by total number of samples).
- How do we use the estimates $Q_{i,n}$ to select the control to sample next?

MCTS Based on Statistical Tests



A good sampling policy balances exploitation (sample controls that seem most promising, i.e., a small $Q_{i,n}$) and exploration (sample controls with small sample count).

- A popular strategy: Sample next the control i that minimizes the sum $Q_{i,n} + R_{i,n}$ where $R_{i,n}$ is an exploration index.
- $R_{i,n}$ is based on a confidence interval formula and depends on the sample count s_i of control i (which comes from analysis of multiarmed bandit problems).
- The UCB rule (upper confidence bound) sets $R_{i,n} = -c\sqrt{\log n/s_i}$, where c is a positive constant, selected empirically (values $c \approx \sqrt{2}$ are suggested, assuming that $Q_{i,n}$ is normalized to take values in the range $[-1, 0]$).
- MCTS with UCB rule has been extended to multistep lookahead.

About the Next Lecture

We will cover:

- Model predictive control
- Approximation architectures
- Training approximation architectures

PLEASE READ AS MUCH OF SECTIONS 2.5, 3.1 AS YOU CAN

PLEASE DOWNLOAD THE LATEST VERSIONS FROM MY WEBSITE