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 5
Rollout for Deterministic and Stochastic Problems

Outline

Rollout: A Special Case of Approximation in Value Space

At State x_k
DP minimization

First ℓ Steps | "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{i=k+1}^{k+\ell-1} g_i\big(x_i,\mu_i(x_i),w_i\big) + \tilde J_{k+\ell}(x_{k+\ell}) \right\}$$

Rollout Control $\tilde u_k$
Rollout Policy $\tilde \mu_k$

Lookahead Minimization

Base Policy Cost

$\tilde J_{k+\ell}(x_{k+\ell})$ is the Cost Function of Some Policy or Heuristic

- The policy used for rollout is called base policy
- The policy obtained by lookahead minimization is called rollout policy

Approximate variant

- $\tilde J_{k+\ell}(x_{k+\ell})$ may also approximate the cost function of the base policy
- Possibility of truncated rollout

Rollout is Important for this Course

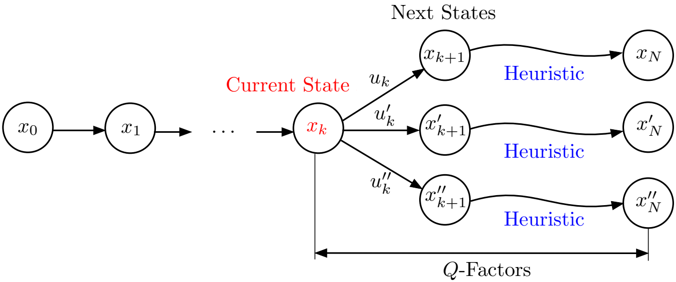
Role of Rollout

- It provides important options for cost function approximation in the context of value space methods
- It is the basic building block of the fundamental PI algorithm (and approximate variants)

Reasons why it will be important:

- Rollout, in its pure form, is the RL method that is easiest to understand and apply
- Rollout is the most reliably successful (with “correct" implementation)
- It is very general: Applies to deterministic and stochastic problems, to finite horizon and infinite horizon
- As a special case of approximation in value space, it relates to Newton’s method
- It provides a useful alternative to reoptimization in indirect adaptive control
- It relates to model predictive control, one of the most important control system design methods (it is used to bring $\tilde{J}$ within the region of stability)
- It forms a building block for many of the RL methods used in practice [including Q-learning, self-learning (approximate PI), and others]

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
- Is rollout cost improving? (Performs no worse than the base heuristic, from x_0)

Criteria for Cost Improvement of a Rollout Algorithm

- 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.

The base heuristic is sequentially consistent if at a given state it chooses control that depends only on that state (and not on how we got to that state)

- If the heuristic generates the sequence

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

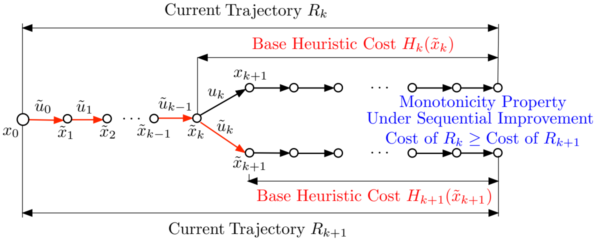
starting from state x_k, it also generates the sequence

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

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 $\{\mu_0, \dots, \mu_{N-1}\}$
- "Greedy" heuristics are sequentially consistent (e.g., nearest neighbor for TS)
- We will focus on a less restrictive condition: sequential improvement

Sequential Improvement Condition



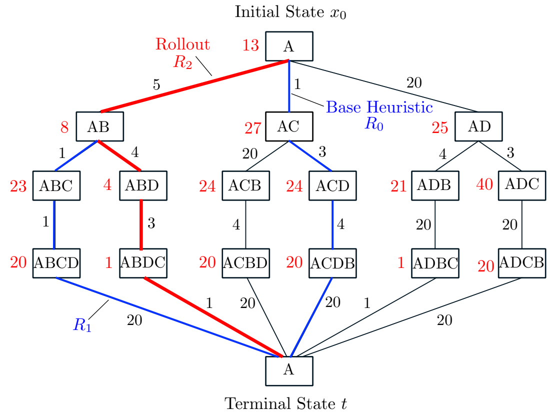
Implies cost improvement: (Cost of Rollout Policy) $\leq$ (Cost of Base Heuristic)

- Definition: Best heuristic Q-factor $\leq$ Heuristic cost, i.e.,

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

  where $H_k(x_k)$: cost of the trajectory generated by the heuristic starting from x_k
- Justification: Rollout, upon reaching $\tilde{x}_k$, has obtained a “current" trajectory R_k. Sequential improvement implies monotonicity: Cost of $R_k \geq$ Cost of R_{k+1}
- R_0 is the cost of the base heuristic, R_N is the cost of the rollout, so $R_0 \geq R_N$
- Note that Sequential consistency (i.e., heuristic is a DP policy) –> Sequential improvement

Traveling Salesman Example: Rollout with a Nearest Neighbor Heuristic



Matrix of Intercity Travel Costs

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

Base heuristic: Nearest neighbor (sequentially consistent and sequentially improving)

Cost of $R_0 \geq$ Cost of $R_1 \geq$ Cost of R_2

Simplified Rollout Algorithm - Assuming Sequential Improvement

Simplified algorithm: Instead of control w/ minimal Q-factor, use any control with Q-factor $\leq$ heuristic cost $H_k(x_k)$

- At any x_k, choose as rollout control any $\tilde{\mu}_k(x_k)$ such that

$$g_k\big(x_k, \tilde{\mu}_k(x_k)\big) + H_{k+1}\Big(f_k\big(x_k, \tilde{\mu}_k(x_k)\big)\Big) \leq H_k(x_k),$$

  where $H_k(x_k)$ is the cost of the trajectory generated by the heuristic from x_k.
- May save lots of computation (case of multiagent rollout, where u_k has multiple components)

Cost improvement for the simplified algorithm:

Let the rollout policy under the simplified algorithm 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: The monotonicity property

$$H_0(x_0) = \text{Cost of } R_0 \geq \cdots \geq \text{Cost of } R_k \geq \text{Cost of } R_{k+1} \geq \cdots \geq \text{Cost of } R_N = J_{0,\tilde{\pi}}(x_0)$$

is maintained

Rollout with Superheuristic/Multiple Heuristics

Consider combining several heuristics in the context of rollout

- The idea is to construct a superheuristic, which runs all the heuristics at each state encountered, and selects the best out of the trajectories produced
- The superheuristic can be viewed as the base heuristic for a rollout algorithm
- It can be verified using the definitions, that if all the heuristics are sequentially improving, the same is true for the superheuristic

Proof: Write the sequential improvement condition for each of the M heuristics

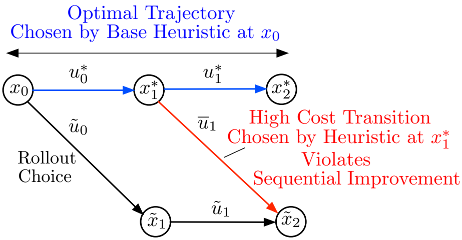
$$\min_{u_k \in U_k(x_k)} \tilde{Q}_k^m(x_k, u_k) \leq H_k^m(x_k), \qquad m = 1, \ldots, M,$$

and all x_k and k, where $\tilde{Q}_k^m(x_k, u_k)$ and $H_k^m(x_k)$ are Q-factors and heuristic costs that correspond to the mth heuristic. By taking minimum over m, and interchanging the order of the minimization $\min_{m=1,\ldots,M} \min_{u_k \in U_k(x_k)}$,

$$\min_{u_k \in U_k(x_k)} \underbrace{\min_{m=1,\ldots,M} \tilde{Q}_k^m(x_k, u_k)}_{\text{Superheuristic Q-factor}} \leq \underbrace{\min_{m=1,\ldots,M} H_k^m(x_k)}_{\text{Superheuristic cost}},$$

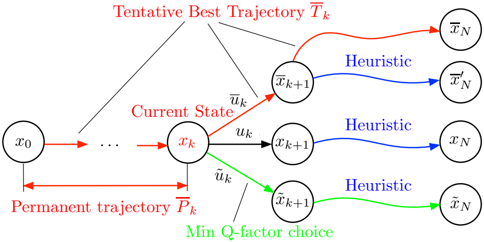
which is the sequential improvement condition for the superheuristic.

A Counterexample to Cost Improvement (w/out Sequential Improvement Condition)



- Suppose at x_0 there is a unique optimal trajectory $(x_0, u_0^*, x_1^*, u_1^*, x_2^*)$.
- Suppose the base heuristic produces this optimal trajectory starting at x_0.
- Rollout uses the base heuristic to construct a trajectory starting from x_1^* and $\tilde{x}_1$.
- Suppose the heuristic's trajectory starting from x_1^* is “bad" (has high cost).
- Then (Q-factor of u_0^*)>(Q-factor of $\tilde{u}_0$). So the rollout algorithm selects $\tilde{u}_0$, and moves to a nonoptimal next state $\tilde{x}_1 = f_0(x_0, \tilde{u}_0)$.
- So in the absence of sequential improvement, the rollout can deviate from an already available good “current" trajectory.
- This suggests a possible remedy: Follow the best “current" trajectory found even if rollout suggests following a different (but inferior) trajectory.

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



Idea: At each step, follow the best trajectory computed thus far

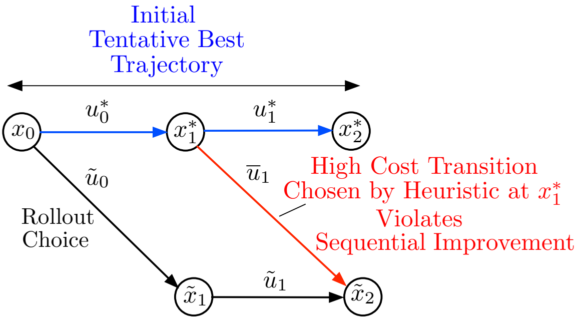
- At state x_k: In addition to the permanent rollout trajectory $\overline{P}_k = \{x_0, u_0, \ldots, u_{k-1}, x_k\}$, also store a tentative best trajectory

$$\overline{T}_k = \{x_0, \ldots, x_k, \overline{u}_k, \overline{x}_{k+1}, \overline{u}_{k+1}, \ldots, \overline{u}_{N-1}, \overline{x}_N\}$$

  $\overline{T}_k$ is the best end-to-end trajectory computed up to stage k

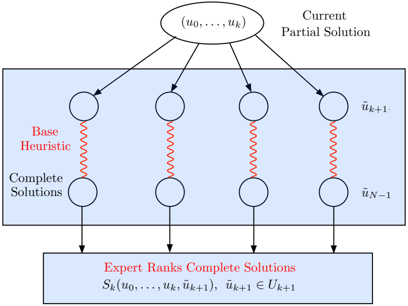
- We reject the minimum Q-factor choice $\tilde{u}_k$ if its complete trajectory is more costly than the current tentative best; otherwise we accept $\tilde{u}_k$, and update the tentative best trajectory.

Illustration of Fortified Algorithm



- At x_0, the fortified rollout stores as initial tentative best trajectory the unique optimal trajectory $(x_0, u_0^*, x_1^*, u_1^*, x_2^*)$ generated by the base heuristic.
- In the first rollout step, it computes the Q-factors of u_0^* and $\tilde{u}_0$ by running the heuristic from x_1^* and $\tilde{x}_1$.
- Even though the rollout prefers $\tilde{u}_0$ to u_0^*, it discards $\tilde{u}_0$ in favor of u_0^*, which is dictated by the tentative best trajectory.
- It then sets the permanent trajectory to (x_0, u_0^*, x_1^*) and keeps the tentative best trajectory unchanged to $(x_0, u_0^*, x_1^*, u_1^*, x_2^*)$.

Model-Free Rollout with an Expert for the General Discrete Optimization $\min_{u_0 \in U_0, \ldots, u_{N-1} \in U_{N-1}} G(u_0, \ldots, u_{N-1})$

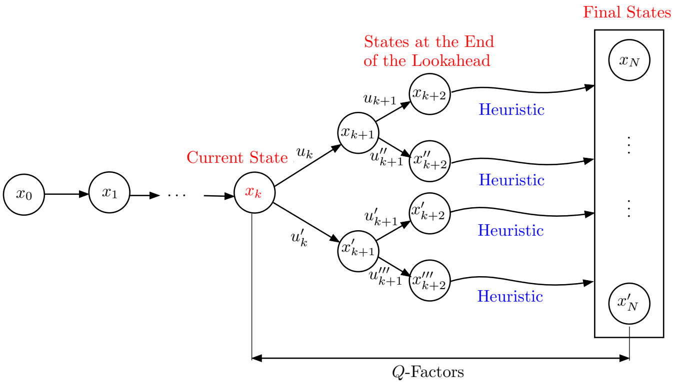


- Assume we do not know G, and/or the constraint sets U_k
- Instead we have a base heuristic, which given a partial solution $(u_0, \ldots, u_k)$, outputs all next controls $\tilde{u}_{k+1}$, and generates from each a complete solution

$$S_k(u_0, \ldots, u_k, \tilde{u}_{k+1}) = (u_0, \ldots, u_k, \tilde{u}_{k+1}, \ldots, \tilde{u}_{N-1})$$

- Also, we have a human or software "expert" that can rank any two complete solutions without assigning numerical values to them.
- Deterministic rollout can be applied to this problem; we have all we need.

A Working Break with a Challenge Question



Consider deterministic rollout with multistep lookahead

- How would the rollout algorithm work?
- What is the main computational difficulty in applying multistep rollout?

Stochastic Rollout with MC Simulation: Multistep Approximation in Value Space with $\tilde{J}_{k+\ell}(x_{k+\ell})$ the Cost Function of Some Policy

At State x_k
DP minimization

First ℓ Steps — "Future"

$$\min_{u_k,\mu_{k+1},\ldots,\mu_{k+\ell-1}} E\left\{ g_k(x_k,u_k,w_k) + \sum_{i=k+1}^{k+\ell-1} g_i\big(x_i,\mu_i(x_i),w_i\big) + \tilde{J}_{k+\ell}(x_{k+\ell}) \right\}$$

Rollout Control $\tilde{u}_k$
Rollout Policy $\tilde{\mu}_k$

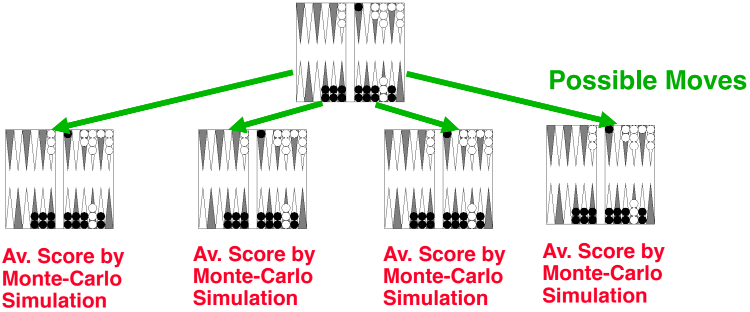
Lookahead Minimization — **Base Policy Cost**

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 $\tilde{\pi} = \{\tilde{\mu}_0, \ldots, \tilde{\mu}_{N-1}\}$ be the rollout policy. Then cost improvement is obtained

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

- A simple induction proof
- The big issue: How do we save in simulation effort?



- Truncated rollout with cost function approximation provided by TD-Gammon (a 1992 program, involving a neural network trained by a form of approximate policy iteration that uses “Temporal Differences").
- The truncated rollout program (1996) plays better than TD-Gammon, and better than any human.
- It is slow due to excessive Monte Carlo simulation time.

Monte Carlo Tree Search - Motivation: Save Simulation Effort

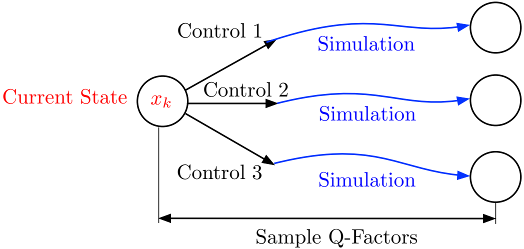
We assumed equal effort for evaluation of Q-factors of all controls at a state x_k

Drawbacks:

- Some controls may be clearly inferior to others and may not be worth as much sampling effort.
- Some controls 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 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).
- MCTS does not directly improve performance; it just tries to save in simulation effort.

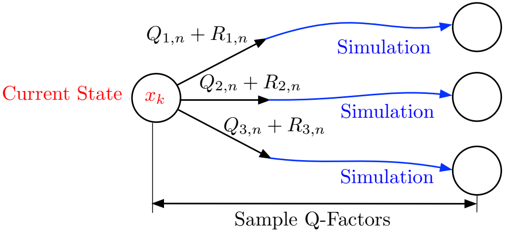


MCTS provides an economical sampling policy to estimate the Q-factors

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

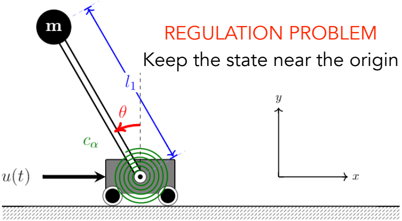
Assume that $U_k(x_k)$ contains a finite number of elements, say $u = 1, \ldots, m$

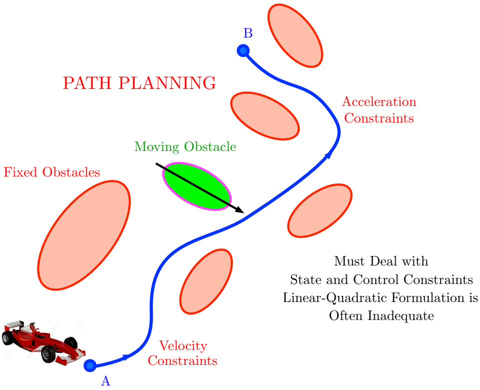
- After the nth sampling period we have $Q_{u,n}$, the empirical mean of the Q-factor of each control u (total sample value divided by total number of samples corresponding to u). We view $Q_{u,n}$ as an exploitation index.
- How do we use the estimates $Q_{u,n}$ to select the control to sample next?



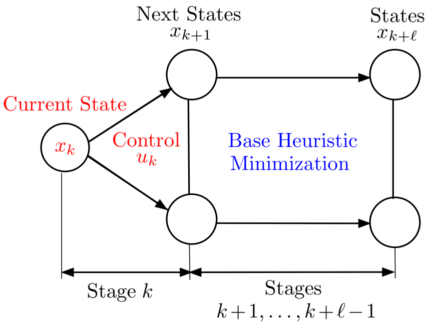
MCTS balances exploitation (sample controls that seem most promising, i.e., a small $Q_{u,n}$) and exploration (sample controls with small sample count).

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


m
REGULATION PROBLEM
Keep the state near the origin
l_1
θ
c_α
$u(t)$
y
x


B
PATH PLANNING
Acceleration Constraints
Moving Obstacle
Fixed Obstacles
Must Deal with
State and Control Constraints
Linear-Quadratic Formulation is
Often Inadequate
Velocity Constraints
A

On-Line Rollout for Deterministic Infinite-Spaces Problems



Suppose the control space is infinite (so the number of Q-factors is infinite)

- One possibility is discretization of $U_k(x_k)$; but excessive number of Q-factors.
- Another possibility is to use optimization heuristics that look $(\ell - 1)$ steps ahead.
- Seemlessly combine the kth stage minimization and the optimization heuristic into a single ℓ-stage deterministic optimization.
- Can solve it by nonlinear programming/optimal control methods (e.g., quadratic programming, gradient-based). Constraints can be readily accommodated.
- This is the idea underlying model predictive control (MPC).

About the Next Lecture

We will cover:

- Model predictive control; relation to rollout
- Rollout for multiagent problems

Homework to be announced next week

Watch videolecture 6 from the 2021 ASU course offerings