MULTIAGENT REINFORCEMENT LEARNING:
ROLLOUT AND POLICY ITERATION

Dimitri P. Bertsekas
Arizona State University

October 15, 2020
Based on material from my research monograph
Rollout, Policy Iteration, and Distributed Reinforcement Learning, Athena Scientific, 2020

Related research can be found at my website including:

- An overview paper to be published in IEEE/CAA J. of Automatica Sinica
- Several research papers and multiagent policy iteration, value iteration, and discrete deterministic optimization (2019-2020, ArXiv)
Outline

1. Multiagent Problems in General
2. Dynamic Programming Formulation
3. Agent-by-Agent Policy Improvement
4. Approximate Policy Iteration - Use of Value and/or Policy Networks
5. Autonomous Multiagent Rollout with Signaling Policies
6. Multirobot Repair - A Large-Scale Multiagent POMDP Problem
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 Dynamic Programming (DP).
- The nonclassical information pattern: Agents are partially sharing information, and may be antagonistic. HARD because it cannot be treated by DP.
Our Starting Point: A Classical Information Pattern ... but we will Generalize

At each time, the agents have exact state info, and choose their controls as functions of the 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. Our initial aim is 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)
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 this to \( 5 \cdot 15 = 75 \) choices (while maintaining good properties).

Idea: Break down the control into a sequence of one-spider-at-a-time moves

We will introduce “precomputed signaling/coordination” between the spiders, so the 15 spiders will choose moves in parallel (an extra speedup factor of 15)
Team theory/Decentralized control (also decentralized MDP and POMDP)

- Agents have common goals but do not fully share information; a **nonclassical information pattern**
- For example, some of the spiders can see some of the flies, but others cannot
- Notoriously difficult problems. Theory/algorithms here often try to exploit weak couplings between some of the agents

A RL/policy gradient approach for nonclassical information patterns

- Forget about DP. Parametrize the agent policies in a way that is consistent with the information pattern
- Tune the parameters using neural networks and gradient descent (**policy gradient methods**)
- **Advantage**: Can deal with a nonclassical information pattern
- **Drawback**: Strictly off-line (and difficult) training (cannot adapt to on-line changes of problem data)
- **No solid theory**, lack of performance guarantees.
For this Talk we Focus on Finite-State Infinite Horizon Problems

Random Transition
\[ x_{k+1} = f(x_k, u_k, w_k) \]

Random Cost
\[ \alpha^k g(x_k, u_k, w_k) \]

Infinite Horizon

**Stationary system and cost accumulated over an infinite number of stages**

- System \( x_{k+1} = f(x_k, u_k, w_k) \) with state \( x_k \), \( m \)-component control \( u_k \) (\( w_k \): random)
- Policies \( \mu = (\mu_1, \ldots, \mu_m) \) that map states \( x \) to control components \( \mu_i(x) \in U_i(x) \) for all \( x \) and \( i = 1, \ldots, m \)
- Cost of stage \( k \): \( \alpha^k g(x_k, \mu_1(x_k), \ldots, \mu_m(x), w_k) \); \( \alpha \in (0, 1] \) is the discount factor
- Cost of policy \( \mu \)

\[
J_\mu(x_0) = \lim_{N \to \infty} E_{w_k} \left\{ \sum_{k=0}^{N-1} \alpha^k g(x_k, \mu_1(x_k), \ldots, \mu_m(x), w_k) \right\}
\]

- Optimal cost function \( J^*(x_0) = \min_{\mu} J_\mu(x_0) \)
- Optimality condition: Minimize the RHS of Bellman’s equation

\[
\mu^*(x) \in \arg \min_{(u_1, \ldots, u_m)} E_w \left\{ g(x, u_1, \ldots, u_m, w) + \alpha J^*(f(x, u_1, \ldots, u_m, w)) \right\}
\]
Policy Iteration Algorithm

Base Policy $\mu$

Policy Evaluation $J_\mu$

Policy Improvement
Bellman Eq. with $J_\mu$ instead of $J^*$

Rollout Policy $\tilde{\mu}$

$\tilde{\mu}(x) \in \arg \min_{(u_1,\ldots,u_m)} E_w\{g(x,u_1,\ldots,u_m,w) + \alpha J_\mu(f(x,u_1,\ldots,u_m,w))\}$

Fundamental policy improvement property

$J_{\tilde{\mu}}(x) \leq J_\mu(x)$, for all $x$

The rollout algorithm is a one-time policy iteration

It can be implemented on-line if values of $J_\mu$ are (approximately) available, through simulation (possibly in conjunction with a trained neural net)

A second major advantage of the rollout algorithm: Robustness

If implemented on-line, it can adapt to variations of the problem data through on-line replanning
Outline of Our Approach for Multiagent Problems

- We propose a policy iteration (PI) method with a new form of policy improvement, namely one-agent-at-a-time policy improvement.
- Rollout is a single-iteration version of PI; but can be implemented on-line.

Extension to a nonclassical information pattern

- We use “guesses" to make up for missing information.
- The “guesses" are precomputed, possibly through neural network training.
- Subject of ongoing research.
A New Form of Policy Improvement Operation When $u = (u_1, \ldots, u_m)$

The standard rollout algorithm

$$(\tilde{\mu}_1(x), \ldots, \tilde{\mu}_m(x)) \in \arg \min_{(u_1, \ldots, u_m)} E_w \left\{ g(x, u_1, \ldots, u_m, w) + \alpha J_\mu (f(x, u_1, \ldots, u_m, w)) \right\}$$

has a search space with size that is exponential in $m$

Proposed alternative: Multiagent rollout algorithm

Perform a sequence of $m$ successive minimizations, one-agent-at-a-time

$$\tilde{\mu}_1(x) \in \arg \min_{u_1} E_w \left\{ g(x, u_1, \mu_2(x), \ldots, \mu_m(x), w) + \alpha J_\mu (f(x, u_1, \mu_2(x), \ldots, \mu_m(x), w)) \right\}$$

$$\tilde{\mu}_2(x) \in \arg \min_{u_2} E_w \left\{ g(x, \tilde{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x), w) + \alpha J_\mu (f(x, \tilde{\mu}_1(x), u_2, \mu_3(x), \ldots, \mu_m(x), w)) \right\}$$

$$\vdots$$

$$\tilde{\mu}_m(x) \in \arg \min_{u_m} E_w \left\{ g(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), \ldots, \tilde{\mu}_{m-1}(x), u_m, w) + \alpha J_\mu (f(x, \tilde{\mu}_1(x), \tilde{\mu}_2(x), \ldots, \tilde{\mu}_{m-1}(x), u_m, w)) \right\}$$

- Has a search space with size that is linear in $m$
- ENORMOUS SPEEDUP!

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

\[ J^1(x_k, u_k^1), J^2(x_k, u_k^1, u_k^2), \ldots, J^{m-1}(x_k, u_k^1, \ldots, u_k^{m-1}) \]

- Multiagent (one-agent-at-a-time) rollout is just standard rollout for the reformulated problem

- The increase in size of the state space does not adversely affect rollout

- Key theoretical fact: The cost improvement property is maintained

- Complexity reduction: The one-step lookahead branching factor is reduced from $n^m$ to $nm$, where $n$ is the number of possible choices for each component $u_k^i$
### Base policy:
Move along the shortest path to the closest surviving fly (in the Manhattan distance metric).

### Time to catch the flies:
- **Base policy** (each spider follows the shortest path): Capture time = 85
- **Standard rollout** (all spiders move at once, $4^5 = 625$ move choices): Capture time = 34
- **Agent-by-Agent rollout** (spiders move one at a time, $4 \cdot 5 = 20$ move choices): Capture time = 34
Comparing Standard with Multiagent Rollout/Policy Iteration

Comparison of agent-by-agent and standard (all-agents-at-once) rollout

- They produce different rollout policies
- One may be better than the other
- ... BUT standard rollout requires intractable computation for even a modest number of agents
- We speculate that agent-by-agent rollout will often perform about as well as standard rollout

Agent-by-agent PI: Uses agent-by-agent policy improvement

- Agent-by-agent PI stops at an agent-by-agent optimal policy ... which may not be optimal
- **Convergence result:** Agent-by-agent PI converges finitely to an agent-by-agent optimal policy
- Rate of convergence seems comparable to standard PI
Approximate policy improvement property: With approximations, policy improvement holds approximately:

\[ J_{\tilde{\mu}}(x) \leq J_{\mu}(x) + \epsilon, \quad \text{for all } x \]

- If a single policy iteration is done (rollout), no need to train value and policy networks
- Multiple policy iterations can be done only with off-line training
- Many RL algorithms, including Alphazero, use schemes of this type (off-line PI plus on-line rollout)
Parallelization of Agent-by-Agent Policy Improvement

One-agent-at-a-time policy improvement is an inherently serial computation. How can we parallelize it?

Precomputed signaling

- **Obstacle to parallelization**: To compute the $k$th agent rollout control we need the rollout controls of the preceding agents $i < k$
- **Signaling remedy**: Use precomputed substitute “guesses” $\hat{\mu}_i(x)$ in place of the preceding rollout controls $\tilde{\mu}_i(x)$

Signaling possibilities

- **Use the base policy controls for signaling** $\hat{\mu}_i(x) = \mu_i(x), i = 1, \ldots, k - 1$ (this may work poorly)
- **Use a neural net representation of the rollout policy controls for signaling** $\hat{\mu}_i(x) \approx \tilde{\mu}_i(x), i = 1, \ldots, k - 1$ (this requires off-line computation)
- **Other, problem-specific possibilities**
Two spiders trying to catch two stationary flies in minimum time

- The spiders have perfect vision/perfect information
- Base policy for each fly: Move one step towards the closest surviving fly

Performance of various algorithms

- Optimal policy: Splits the spiders towards their closest flies
- Standard rollout is optimal for all initial states
- Agent-by-agent rollout is also optimal for all initial states
- Agent-by-agent rollout with base policy signaling is optimal for most initial states, with A SIGNIFICANT EXCEPTION
- When the spiders start at the same location, the spiders oscillate and never catch the flies
Damage level of each site is unknown, except when inspected. It deteriorates according to a known Markov chain unless the site is repaired.

Control choice of each robot: Inspect and repair (which takes one unit time), or inspect and move to a neighboring site.

State of the system: The set of robot locations, plus the belief state of the site damages (the joint probability distribution of the damage levels of the sites).

Stage cost at each unrepaired site: Depends on the level of its damage.
Multirobot Repair in a Network of Damaged Sites
Agents Start from the Same Location

Cost comparisons

- Base policy cost: 5294 (30 steps)
- Multiagent rollout: 1124 (9 steps)
- Approx. Multiagent Rollout with base policy: 31109 (Never stops)
- Approx. Multiagent Rollout with neural network policy: 2763 (15 steps)
Cost comparisons

- Base policy cost: 12945 (62 steps)
- Multiagent rollout: 5294 (19 steps)
- Approx. Multiagent Rollout with base policy: 6920 (25 steps)
- Approx. Multiagent Rollout with neural network policy: 7241 (21 steps)
Recall that a policy network must be used to represent a policy generated by PI. As a result, the PI training must be done off-line. Typical performance: Large cost improvement at first few iterations, which tails off and ends up in an oscillation as the number of generated policies increases.
Concluding Remarks

- We have focused on multiagent rollout and policy iteration.
- These are approximation in value space methods that can be applied to very complex RL problems with multi-component controls.
- We have introduced a simplified form of policy improvement for multiagent problems.
- They offer a solid performance guarantee (performance improvement property).
- They have interesting theoretical properties (PI convergence to an agent-by-agent optimal policy).
- Our methods require a classical information pattern (key assumption is the sharing of perfect state info).
- They admit extensions to nonclassical information pattern problems through precomputed signaling policies.
- Our methods compare very favorably on the multi-robot repair problem with existing methods (POMCP, DESPOT, MADDPG).
- **Important research question:** Can perfect state info be replaced by state estimates?
- A broad range of very challenging analytical and algorithmic questions lie ahead.
Thank you!