Optimization Course Project III:

Markov Decision Processes and Zero-Sum Game

1 Introduction

Reinforcement Learning (RL) and Markov Decision Processes (MDP) provide a mathematical framework for modeling sequential decision-making in situations where outcomes are partly random and partly under the control of a decision-maker. MDPs are useful for studying a wide range of optimization problems solved via Dynamic Programming (DP), known at least as early as the 1950s (cf. Shapley 1953, Bellman 1957). Modern applications include dynamic planning, reinforcement learning, social networking, and almost all other dynamic/sequential decision-game strategy-making problems in Mathematical, Physical, Management, and Social Sciences.

As discussed in class, the MDP problem with m states and total n actions can be formulated as a standard form linear program with m equality constraints and n variables:

$$\begin{array}{lccccc} \min_{\mathbf{x}} & \sum_{j\in\mathcal{A}_1} c_j x_j + & ... & + \sum_{j\in\mathcal{A}_m} c_j x_j & & \\ \text{s.t.} & \sum_{j\in\mathcal{A}_1} (\mathbf{e}_1 - \gamma \mathbf{p}_j) x_j + & ... & + \sum_{j\in\mathcal{A}_m} (\mathbf{e}_m - \gamma \mathbf{p}_j) x_j & = & \mathbf{e}, \\ & ... & x_j & ... & \geq & 0,\ \forall j, \end{array} \tag{1}$$

where $\mathcal{A}_i$ represents the set of all actions available in state i, $\mathbf{p}_j$ is the state transition probabilities from state i to all states and c_j is the immediate cost when action j is taken, and $0 < \gamma < 1$ is the discount factor. Also, $\mathbf{e} \in R^m$ is the vector of ones, and $\mathbf{e}_i$ is the unit vector with 1 at the i-th position and zeros everywhere else. The variable $x_j,\ j \in \mathcal{A}_i$, is the state-action frequency or flux, or the expected present value of the number of times in which the process visits state i and takes state-action $j \in \mathcal{A}_i$. Thus, solving the problem entails choosing state-action frequencies/fluxes that minimize the expected present value sum of total costs.

The dual of the LP is

$$\begin{array}{rll} \text{maximize}_{\mathbf{y}} & \mathbf{e}^T\mathbf{y} = \sum_{i=1}^m y_i & \\ \text{subject to} & y_1 - \gamma \mathbf{p}_j^T \mathbf{y} & \leq \quad c_j,\ j \in \mathcal{A}_1 \\ & \vdots & \\ & y_i - \gamma \mathbf{p}_j^T \mathbf{y} & \leq \quad c_j,\ j \in \mathcal{A}_i \\ & \vdots & \\ & y_m - \gamma \mathbf{p}_j^T \mathbf{y} & \leq \quad c_j,\ j \in \mathcal{A}_m. \end{array} \tag{2}$$

where y_i represents the cost-to-go value in state i.

Although any MDP can be represented by a linear program and, thus, solved in polynomial time, the time complexity of solving large-scale LPs still makes it inefficient to address MDP in practice. Hence, this project explores approaches to solving MDP, especially focusing on different value iteration methods.

2 Possible Approaches for MDP

Below are suggested value iteration approaches for solving the problem. You are also encouraged to explore policy iteration or Q-learning approaches.

Approach 1: We first introduce the Value Iteration Method. This is a first-order optimization method – starting with any vector $\mathbf{y}^0$, then iteratively updating it

$$y_i^{k+1} = \min_{j \in \mathcal{A}_i} \{c_j + \gamma \mathbf{p}_j^T \mathbf{y}^k\},\ \forall i. \tag{3}$$

Prove the contraction result:

$$\|\mathbf{y}^{k+1} - \mathbf{y}^*\|_\infty \leq \gamma \|\mathbf{y}^k - \mathbf{y}^*\|_\infty,\ \forall k.$$

where $\mathbf{y}^*$ is the fixed-point or optimal value vector, that is,

$$y_i^* = \min_{j \in \mathcal{A}_i} \{c_j + \gamma \mathbf{p}_j^T \mathbf{y}^*\},\ \forall i.$$

Here, we remark that, in the VI method, if starting with any vector $\mathbf{y}^0 \geq (\leq) \mathbf{y}^*$ and assuming $\mathbf{y}^1 \leq (\geq) \mathbf{y}^0$, the following entry-wise monotone property holds:

$$\mathbf{y}^* \leq (\geq) \mathbf{y}^{k+1} \leq (\geq) \mathbf{y}^k,\ \forall k.$$

This monotone property has been used in a recent paper (see [SWWY17]) on the VI method using samples.

Approach 2: Rather than go through all state values in each iteration, we modify the VI method, call it RamdomVI: In the kth iteration, randomly select a subset of states B^k and do

$$y_i^{k+1} = \min_{j \in \mathcal{A}_i} \{c_j + \gamma \mathbf{p}_j^T \mathbf{y}^k\},\ \forall i \in B^k. \tag{4}$$

In RandomVI, we only update a subset of state values randomly in each iteration.

Approach 3: Rather than randomly select a subset of all states in each iteration, suppose we build an "influence tree" from a given subset of states, say B, for all states, denoted by $I(B)$, that are connected by any state in B. When states in B are updated in the current iteration, then we select a subset of states in $I(B)$ for updating in the next iteration. When the transaction matrices are sparse ($\mathbf{p}_j$ is a very sparse distribution vector for each action j) in the MDP network, this approach can avoid many unimportant or irrelevant states in each update, which results in state-reduction.

Approach 4: Here is another modification, called CyclicVI: In the kth iteration, do

- Initialize $\tilde{\mathbf{y}}^k = \mathbf{y}^k$.
- For $i = 1$ to m
$$\tilde{y}_i^k = \min_{j \in \mathcal{A}_i} \{c_j + \gamma \mathbf{p}_j^T \tilde{\mathbf{y}}^k\} \tag{5}$$
- $\mathbf{y}^{k+1} = \tilde{\mathbf{y}}^k$.

In the CyclicVI method, as soon as a state value is updated, we use it to update the rest of the state values.

Approach 5: In the CyclicVI method, rather than with the fixed cycle order from 1 to m, we follow a random permutation order, or sample without replacement to update the state values. More precisely, in the kth iteration, do

0. Initialize $\tilde{\mathbf{y}}^k = \mathbf{y}^k$ and $B^k = \{1, 2, ..., m\}$
1. – Randomly select $i \in B^k$
   –
$$\tilde{y}_i^k = \min_{j \in \mathcal{A}_i} \{c_j + \gamma \mathbf{p}_j^T \tilde{\mathbf{y}}^k\} \tag{6}$$
   – remove i from B^k and return to Step 1.
3. $\mathbf{y}^{k+1} = \tilde{\mathbf{y}}^k$.

We call it the randomly permuted CyclicVI or RPCyclicVI in short.

2.1 Project Goal for MDP

To explore approaches for solving MDP, you may generate different MDPs to test the performance of different algorithms. Then, answer the following key questions: What are the best approaches to solve the MDP? How

do the different structures of the transition matrices, such as density and sparsity, affect the convergence? What are the differences among different methods? The comparison can include algorithm design, theoretical analysis, computation time, and the approximation error of different algorithms.

In addition, to further explore the application of MDPs, you may also generate a $k \times k$ tic-tac-toe game against a random player for $k = 3$ or 4 and find the optimal strategy for this game by solving corresponding MDPs. Here, we consider the random player will take any of the empty cells on the board with equal probability. In this case, what would be the optimal first move? Can the first player always win?

3 Zero-Sum Game and Tic-Tac-Toe

In class, we have discussed the zero-sum game, where no wealth is created or destroyed. Many competitive board games with two players, including the tic-tac-toe game, can be viewed as zero-sum games since if one player wins, another must lose. In Section 3.1, we have investigated the optimal strategy for the tic-tac-toe game against a fixed random player. The random player's strategy will not evolve even if the opposite changes their policy. Can you apply MDPs to find the optimal strategy for both players and the Nash equilibrium?

Apply the Value-Iteration Method to solve the (deterministic) Tic-Tac-Toe game problem.

References

[Ber13] Dimitri P Bertsekas. *Abstract dynamic programming*. Athena Scientific, Belmont, MA, 2013.

[HMZ13] Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *J. ACM*, 60(1):1:1–1:16, February 2013.

[How60] Ronald A. Howard. *Dynamic programming and Markov processes*. The MIT press, Cambridge, MA, 1960.

[LDK95] Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 394–402. Morgan Kaufmann Publishers Inc., 1995.

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Sch13] Bruno Scherrer. Improved and generalized upper bounds on the complexity of policy iteration. In *Advances in Neural Information Processing Systems*, pages 386–394, 2013.

[SWWY17] Aaron Sidford, Mengdi Wang, Xian Wu, Yinyu Ye. Variance Reduced Value Iteration and Faster Algorithms for Solving Markov Decision Processes. SODA2018 and https://arxiv.org/abs/1710.09988

[Ye11] Yinyu Ye. The simplex and policy-iteration methods are strongly polynomial for the markov decision problem with a fixed discount rate. *Mathematics of Operations Research*, 36(4):593–603, 2011.