Final Project for Classical AI – COMP 5700 – Markov Decision Process (MDP)

Lev Sukherman

Introduction

The main goal in MDPs modeling is also to be able to purposefully apply these techniques to decision-making scenarios that emphasize the indeterminism of the agents' activity as well as the environment it interacts with. Standard determinism studies are inadequate owing to the fact they fail to take into account randomness, performance and uncertainty of the real world. Markov Decision Processes supply a comprehensive mathematical toolkit which is useful in situations that require the action to be stochastic and partially randomized. In addition, the connection between probability theory and, logic within the MDPs motivates the theoretical advancements in the field of academia and the practical applications, resulting in the reinforcement of our knowledge of various stochastic processes and the creation of more sophisticated algorithms for decision-making in complex environments.

Markov property and Markov process

The Markov Property is the fundamental property that is observed in stochastic processes and claims that the conditional probability distribution of future states depends only upon the present state regardless of the previous events which occurred. This lack of memory reduces the complexity of models and analysis by diminishing the need for consideration of previous states resulting in simpler prediction compared to systems with such feature. In this context, Markov Process, or the Markov Chain, has a stochastic character which follows this property, resulting in the probability of transition to any future state being only determined by the current state. These formalisms imply a distinct concept space and mapping functions, including transfer probabilities, which can be defined in discrete or continuous time formats. In a nutshell, this notion greatly simplifies computation and brings any study closer to practice. The most important application of this paradigm is in creating predictive algorithms and analysis models for many sciences and engineering applications.

Problem

Markov Decision Processes (MDP) also widely used for modeling and solving of those decision situations which are inherently uncertain so AI approaches are applicable for complex environment where the outcomes will be determined by both environment's uncertainty and agent decisions. MDPs give a robust mathematical base that covers deterministic control with probabilistic states, which is therefore the most suitable for imitation and analysis of uncontrolled environments, this competence will be indispensable in AI, design the optimal control policy, which is used in reinforcement learning. This means of learning enables agents to adjust and strengthen performance by acting on an environment in which only the stable parts can be regulated. This is a key necessity for applications of autonomous systems and robots that can hardly control some external elements. In a similar vein, stochastic models have probabilistic theory and logical reasoning in the same computational implementation, thus endowing the AI with capability to deal with complex erratic events.

Possible solutions

Dynamic Programming Techniques

Benefits

1) *Theoretical Guarantee of Optimality*: Both Value Iteration and Policy Iteration come with theoretical guarantees that they will converge to the optimal policy, given sufficient iterations and under certain conditions. This is critical for applications where optimal decision-making is paramount.
2) *Comprehensive State Evaluation*: These methods systematically evaluate the value or policy for all states, ensuring that the decision-making strategy is informed by a holistic view of the state space.

Drawbacks

1) *Curse of Dimensionality*: As the number of states and actions increases, the computational and memory requirements grow exponentially, making these methods impractical for very large or continuous state spaces.
2) *Requirement for a Complete Model*: Both methods require a complete model of the environment, including transition probabilities and reward functions, which may not always be available or easy to estimate in real-world scenarios.

Reinforcement Learning Algorithms

Benefits

1) *Flexibility with Incomplete Models*: Reinforcement learning, such as Real-Time Dynamic Programming (RTDP), does not require a complete model of the environment. It can learn optimal policies directly from interaction with the environment, which is particularly useful in complex or poorly understood domains.
2) *Efficiency in Large State Spaces*: RTDP and other reinforcement learning algorithms focus on states that are more relevant to the task, which can greatly reduce the computational burden compared to methods that evaluate all states.

Drawbacks

1) *Convergence Time:* While reinforcement learning methods are powerful, they can sometimes take a long time to converge to an optimal or near-optimal policy, especially in environments with sparse rewards or large state spaces.
2) *Stability and Robustness*: Learning from interaction with the environment can lead to instability in the policy during the learning process. The performance of the algorithm can significantly vary based on the initial conditions, exploration strategy, and the stochastic nature of the environment.

In that regard, dynamic programming provides the most rational tools that the scientific base underpins, however predicting environment and building detailed models may not always be possible in reality. In contrast to how reinforcement learners work, where they learn on the basis of experience and examples; challenges such as convergence and stability may be encountered.

In a nutshell, the preferred method between dynamic programming and reinforcement learning relies on the nature of the issue, which includes the size of the state and action spaces, models availability for the environment, and the scalability of computation while learning is kept stable.

Algorithms

Value Iteration

Initialization: Start with an initial value function V_0, set to zero, for all states.
Iteration: For each state *s* in the state spaces, update the value function based on the Bellman Optimality Equation:

$$V_{i+1}(S) = R(S) + \gamma \underset{a \in A}{max} \sum_{S'}^{S} \{T(S, a, S') * V_i(S')\}$$

where S is the start state, S' is the successor state, R(S) is the reward received in a start state, T(S,a,S') is the transition probability S to S' with action a, $V_i(S')$ is the value of successor state from previous iteration, γ is the discounted factor, and $V_{i+1}(S)$ is the value of a state in the next iteration.
Convergence: Repeat the iteration until the change in value function is below a threshold value.
Benefits:

- Systematic and guaranteed to converge to the optimal solution under proper conditions.
- Suitable for smaller state spaces where exhaustive state evaluation is feasible.

Policy Iteration

Initialization: Start with a random policy π_0 and initialize value function V for this policy.
Policy evaluation: For the current policy π_k, calculate the value function $V^{\pi k}$ by solving:

$$V^{\pi k}(S) = R(S) + \gamma \sum_{S'}^{S} \{T(S, \pi_k(S), S') * V^{\pi k}(S')\}$$

where S is the start state, S' is the successor state, R(S) is the reward received in a start state, $T(S, \pi_k(S), S')$ is the transition probability S to S' with all action according to a policy, $V^{\pi k}(S')$ is the value of successor state according to the value function under policy π_k, and γ is the discounted factor.
Repeat the iteration until the change in value function under policy π_k is below a threshold value.

Policy improvement: Update the policy by choosing actions that maximize the current value function:

$$\pi_{k+1}(S) = \underset{a \in A}{argmax} \sum_{S'}^{S} \{T(S, a, S') * V^{\pi k}(S')\}$$

where S is the start state, S' is the successor state, T(S,a,S') is the transition probability S to S' with action a, $V^{\pi k}(S')$ is the value of successor state according to the value function under policy π_k from precious iteration, and $\pi_{k+1}(S)$ is the policy in the next iteration.

Policy check: Check is a new policy different from a policy from the previous iteration. Repeat both steps until there is no change in a policy.
Benefits:

- More direct and often faster to converge than value iteration since it can find the optimal policy on fewer iterations.

Real-Time Dynamic Programming (RTDP)
Initialization: Begin with an initial value function, often set to zero, for all states.
Learning and Updating: At each step, observe the current state *s,* select an action *a* based on a policy derived from the current value function, execute an action, observe the next state *s'*, and update the value function for the state *s* using the Bellman equation:

$$V = R(S) + \gamma \max_{a \in A} \sum_{S'} \{T(S, a, S') * V(S')\}$$

where S is the start state, S' is the successor state, R(S) is the reward received in a start state, T(S,a,S') is the transition probability S to S' with action a, $V(S')$ is the value of successor state from previous iteration, γ is the discounted factor, and $V(S)$ is the value of a state.

Focus on Relevant States: Unlike traditional methods, RTDP focuses on states visited by the policy.
Benefits:

- Does not require a complete model of the environment and can adapt to changes in real-time.
- Efficient for problems with a large state space where not all states are equally relevant.

Results

Initial setup

- 10 states
- 2 terminal states with rewards of -1 and +1
- Various discount factors (γ) of 0.9, 0.5, and 0.1 to evaluate different sensitivities to future rewards.

Algorithms Tested

- *Value Iteration*
- *Policy Iteration*
- *Real-Time Dynamic Programming (RTDP)*

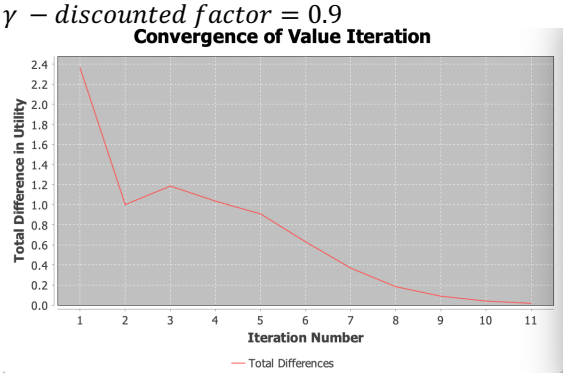
Tests

- *Utility convergence*
- *Policy changes*
- *Real-Time Dynamic Programming changing of actions*

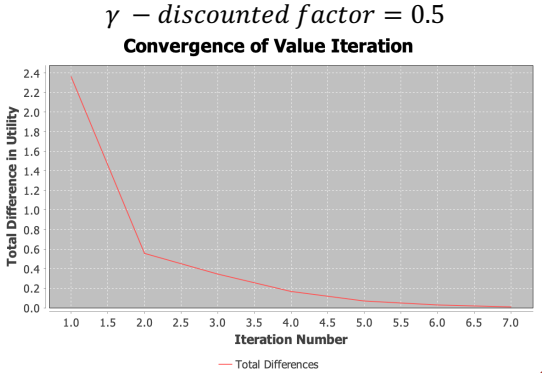
Results of test

Utility iteration

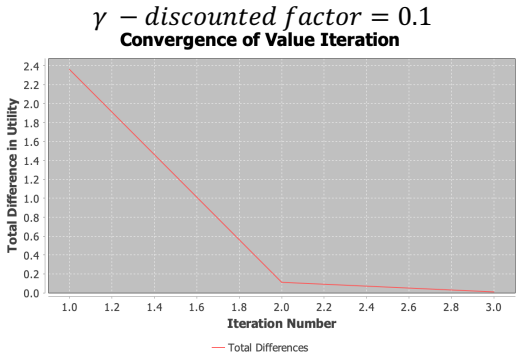
$\gamma - discounted\ factor = 0.9$



$\gamma - discounted\ factor = 0.5$

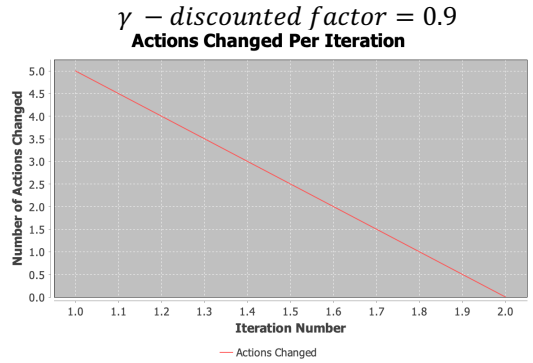


$\gamma - discounted\ factor = 0.1$



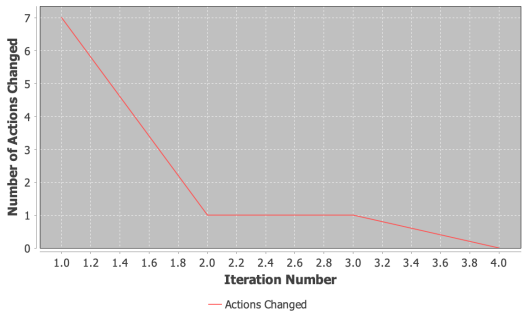
Policy iteration

$\gamma - discounted\ factor = 0.9$



$\gamma - discounted\ factor = 0.5$

$\gamma - discounted\ factor = 0.1$



Real-Time Dynamic Programing

$\gamma - discounted\ factor = 0.9$

```
Iteration 0: Policy changes = 6
Iteration 1: Policy changes = 6
Iteration 2: Policy changes = 3
Iteration 3: Policy changes = 3
Iteration 4: Policy changes = 0

RTDP completed after 4 iterations.
Utility of state 0: -0.010108789397262796
Utility of state 1: 0.11910314335293865
Utility of state 2: 0.2894668332275463
Utility of state 3: -0.17451504216040004
Utility of state 4: -0.11953732899280001
Utility of state 5: 0.48217271780087406
Utility of state 6: -1.0
Utility of state 7: -0.11290155960935203
Utility of state 8: 0.61393019426816
Utility of state 9: 0.7921749719247873
Utility of state 10: 1.0
```

$\gamma - discounted\ factor = 0.5$

```
Iteration 0: Policy changes = 6
Iteration 1: Policy changes = 6
Iteration 2: Policy changes = 2
Iteration 3: Policy changes = 3
Iteration 4: Policy changes = 2
Iteration 5: Policy changes = 0

RTDP completed after 5 iterations.
Utility of state 0: -0.0671828338761605
Utility of state 1: -0.07219697653125001
Utility of state 2: -0.027280655076562493
Utility of state 3: -0.07661612500000001
Utility of state 4: -0.04491656209826089
Utility of state 5: 0.06464800000000004
Utility of state 6: -1.0
Utility of state 7: 0.007208364699528548
Utility of state 8: 0.12547012068000002
Utility of state 9: 0.3823498947051251
Utility of state 10: 1.0
```

$\gamma - discounted\ factor = 0.1$

```
Iteration 0: Policy changes = 6
Iteration 1: Policy changes = 15
Iteration 2: Policy changes = 0

RTDP completed after 2 iterations.
Utility of state 0: -0.044442030513721954
Utility of state 1: -0.04441538209532149
Utility of state 2: -0.04437088416601331
Utility of state 3: -0.044436324
Utility of state 4: -0.04444068548607544
Utility of state 5: -0.043526508905601544
Utility of state 6: -1.0
Utility of state 7: -0.04440494496988866
Utility of state 8: -0.044000000000000004
Utility of state 9: 0.039964377930464624
Utility of state 10: 1.0
```

Conclusion

- *Complexity:*

Value Iteration and Policy Iteration showed predictable linear to quadratic convergence based on the complexity per iteration, which was dependent on the number of states (|S|) and actions (|A|). *Real-Time Dynamic Programming*'s convergence was guided by stochastic approximation, useful in adaptive environments.

- *Convergence Time*:

1) Value Iteration: decrease in convergence time as the number of iterations increases. This might indicate that Value Iteration becomes more efficient over time, but this can also be due to the nature of convergence - as it approaches the optimal solution, less time is required for further refinements.
2) Policy Iteration: has a different convergence profile compared to Value Iteration. Policy Iteration may have fewer iterations overall since it tends to make more significant changes to the policy with each iteration.
3) RTDP: the number of policy changes drops to zero after a few iterations. This suggests that RTDP can quickly stabilize on a policy, which is advantageous in environments where decisions must be made in real-time.

- *Policy stability*

1) Value Iteration: Does not directly produce a policy but rather a value function that can be used to derive a policy. Changes in the value function may imply indirect changes in policy.
2) Policy Iteration: Explicitly updates the policy at each iteration. Policy changes might initially be significant but should decrease as the policy converges.
3) RTDP: The number of policy changes rapidly decreases, often reaching zero policy changes, indicating convergence.

- *Utility stability*

1) Value Iteration and Policy Iteration: Utility values typically converge towards the optimal values, but this process may take many iterations, especially if the state space is large.

2) RTDP: Appears to quickly converge to stable utility values after a few iterations, as indicated by the results. This suggests that RTDP is finding an acceptable solution with potentially less computational effort.

- *Impact of Discount factor (γ)*

Across all methods, different discount factors affect the emphasis on short-term versus long-term rewards. A higher γ (e.g., 0.9) puts more weight on future rewards, possibly leading to longer convergence times but potentially better long-term strategies. A lower γ (e.g., 0.1) focuses the algorithm on immediate rewards, often resulting in faster convergence at the potential expense of long-term optimality.

Future improvement

Exploring More complex Environment: Increase the complexity of the state and action spaces to better simulate real-world environment.
Advanced Algorithms: Implement and test more advanced reinforcement learning algorithms like Deep Q-Network (DQN), where the algorithm uses a convolutional neural network (CNN) to approximate the Q-value function, which represents the value of taking an action in a given state. Partially Observable Markov Decision Process (POMDP), which Extends MDPs to scenarios where the agent does not fully observe the state of the environment, which is more realistic for many real-world problems.
Method Hybridization: approaches that combine the robustness of Policy Iteration with the adaptability of Real-Time Dynamic Programming to enhance performance in dynamic environments.

Reference

- Barto, A. G., Bradtke, S. J., & Singh, S. P. (1995). Learning to act using real-time Dynamic Programming. Artificial Intelligence, 72(1–2), 81–138.
- Bonet, B., & Geffner, H. (n.d.). Solving Stochastic Shortest-Path Problems with RTDP. Research Gate.
- Mausam, & Kolobov, A. (2012). Planning with Markov Decision Processes. Synthesis Lectures on Artificial Intelligence and Machine Learning.
- Russell, S. J., Norvig, P., & Chang, M.-W. (2022). Artificial Intelligence: A modern approach. Pearson Education.