Reinforcement Learning CS/CE 353/368

Assignment 02

Ali Muhammad - aa07190

Question:	1	2	3	4	Total
Points:	10	10	25	5	50
Score:					

Q1) Value-Iteration Vs. Policy Iteration

- (5 points) How do the convergence speeds (i.e., number of iterations) differ between the two algorithms in practice? Support your analysis with visual plots.
- (3 points) How does the overall computational complexity (in terms of state and action space sizes) differ between the two methods?
- (2 points) How do the intermediate policies generated during the iterative process compare between the two algorithms?

Solution:

(a) In practice, the convergence speeds of Value Iteration (IV) and Policy Iteration (PI) depends on the problem's states and actions spaces, as well as the discount factor γ.

- **Value Iteration:** VI updates the value function for all states in each iteration using the Bellman equation, which usually requires more iterations than PI because it incrementally improves the value function until convergence. The number of iterations is often proportional to the inverse of the discount factor γ. Each iteration requires evaluating all state-action pairs, which can be computationally expensive for large state spaces.
- **Policy Iteration:** PI alternates between policy evaluation and policy improvemen (updating policy greedily). PI usually converges in fewer iterations then VI because each policy improvement step ensures a strictly better policy. However, each policy evaluation step can be time-consuming, especially if the value function requires many iterations to converge.

The plot below on some sample values reflects the convergence speeds of VI and PI.

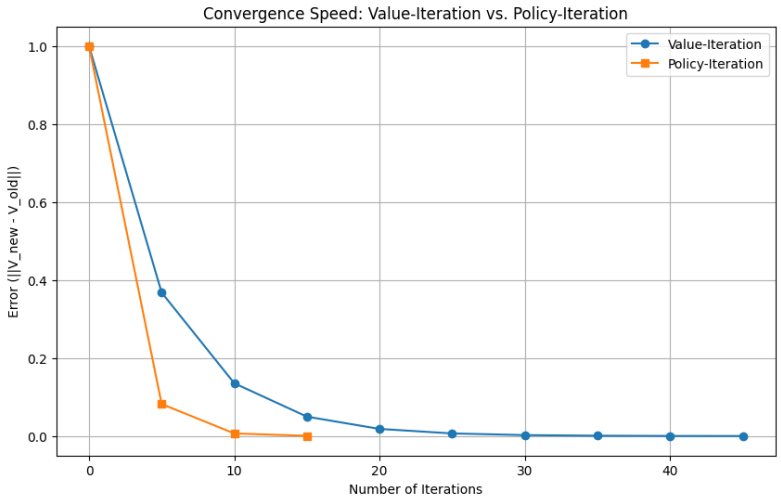

Figure 1: Convergence Speed of Value Iteration and Policy Iteration

(b) The computational complexity of VI and PI depends on the size of the state space |S| and action space |A|.

- **Value Iteration**
  - Each iteration updates the value function for all states and requires $O(|S| \cdot |A|)$ operations.
  - The number of operations until convergence is typically $O(\log(\frac{1}{\varepsilon}))$ where ε is the error tolerance.
  - Total complexity: $O(|S| \cdot |A| \cdot \log(\frac{1}{\varepsilon}))$.
  - Space complexity is $O(|S|)$ for storing the value function.
- **Policy Iteration**
  - Solves a system of linear equations for all states, which is $O(|\mathrm{S}|^3)$ per evaluation if using matrix inversion, or $O(|\mathrm{S}|)$ per sweep if using iterative methods.
  - $O(|S| * |A|)$ to udpate the policy for all states
  - The number of policy improvement cycles is typically smaller than the number of iterations in VI
  - Total complexity is $O(|S|^3 + |S| \cdot |A| \cdot k)$, where k is the number of iterations for policy evaluation.

VI is generally simpler and has lower per-iteration complexity but more iterations. PI has higher per-iteration complexity (due to policy evaluation) but fewer iterations. In FrozenLake, with a small state and action space, VI might be more efficient, but for larger problems, PI could be preferable if k is small.

(c)
- **Value Iteration:** VI does not explicitly maintain or improve a policy until the value function converges. Intermediate policies can be extracted at any point by choosing the action that maximizes the Q-value for each state (greedy policy). These intermediate policies are typically suboptimal and improve gradually as the value function approaches the optimal V*. In FrozenLake, early VI policies might lead the agent into holes due to inaccurate value estimates.
- **Policy Iteration:** PI explicitly maintains and improves a policy in each iteration. After each policy evaluation, the policy improvement step ensures that the new policy is strictly better (or equal) to the previous policy. Thus, intermediate policies in PI are generally better than those in VI at the same stage. In FrozenLake, PI's intermediate policies are more likely to avoid holes earlier in the process.

PI's intermediate policies are more stable and closer to optimal at each step, while VI's are noisier and less reliable until convergence.

Q2) First-Visit vs. Every-Visit

- (4 points) Under what conditions do the two methods produce unbiased estimates of the value function, and are there any scenarios where one method might introduce bias over the other?
- (4 points) What role do correlations between returns (from multiple visits to the same state) play in the stability of the learning process for every-visit MC?
- (2 points) In empirical studies, under what circumstances does one method outperform the other in terms of convergence speed and final accuracy of the value function?

Solution:

(a) Both First-Visit Monte Carlo (FVMC) and Every-Visit Monte Carlo (EVMC) are unbiased estimators of the value function under certain conditions

- **Unbiased Estimates:** Both methods produce unbiased estimates if the policy is stationary (fixed) and the episodes are sampled from the same distribution. The key condition is that returns (discounted sums of rewards) must be averaged over sufficient samples to approximate the expected return $E[G_t|S_t = s]$.
- **Bias Scenario:** FVMC only uses the first occurrence of a state in an episode to update the value estimate. It can introduce bias if the first visit to a state is not representative of all possible visits (e.g., if the policy makes it unlikely to reach certain states later). In FrozenLake, if the agent rarely reaches a state after the first visit (e.g., due to slipping into a hole), FVMC might underestimate or overestimate the value. On the other hand, EVMC uses all visits to a state in an episode, which reduces bias by averaging more samples. However, it can still be biased if the episode length is short or if there are strong correlations between visits (e.g., the agent repeatedly visits the same state due to a poor policy). In FrozenLake, EVMC might be less biased if the agent slips and revisits states, but it could overfit to noisy trajectories.

(b) Correlations between returns in EVMC can impact stability of the learning process. If returns from multiple visits to the same state are positively correlated, the variance of the value estimates increase, which can lead to instability as the updates diverge, especially in the FrozenLake example where slipping may increase chances of revisiting states. Further, negative correlations can stabilize learning by canceling out noise. So high correlations can increase variance which can make convergence slower, and less reliable. Thus, techniques such as discounting factor can decorrelate returns.

(c) Emperically FVMC is faster and more accurate where states are visited rarely or episodes are short. In FrozenLake, if the agent rarely visits states, FVMC converges faster with lower variance, and also requires less memory and computation per episode. EVMC would be better with high revisit rates or noise transitions, like FrozenLake with slippery ice. It leverages more data leading to better accuracy if correlations are managed, but may converge slower due to higher variance.

Q3) Dynamic Programming Vs. Monte-Carlo Methods

- (10 points) Discuss and compare the two methods in terms of:
  - convergence speed
  - computational complexity
  - practical applicability to different environments

  Include visualizations (e.g., plots, heat maps, etc) to support your analysis
- (5 points) Monte Carlo methods are often subject to high variance because of sampling over complete episodes. How does this variance impact the stability and accuracy of the estimated value functions compared to the deterministic updates in DP methods?
- (5 points) Discuss possible ways to improve the performance of each method.
- (5 points) What trade-offs emerge in practice when choosing between DP and MC approaches for a given reinforcement learning problem?

Solution:

(a)
- Convergence speed: DP converges faster since it uses the full model and updates values deterministically, whereas MC methods rely on sampling which is stochastic in nature and leads to a slower convergence.
- Computational Complexity: DP's complexity is $O(|S|^2 * |A|)$ per iteration whereas MC's complexity is $O(E * |S|)$ where E is the number of episodes. So DP would be more suitable for smaller state spaces, while MC is better for larger state spaces.
- Practical applicability: DP requires a full model, which is ideal for smaller environments with known dynamics. MC methods are more flexible and can be applied to larger environments where the model is unknown or too complex to compute.

The below plots further reinforce the above points.

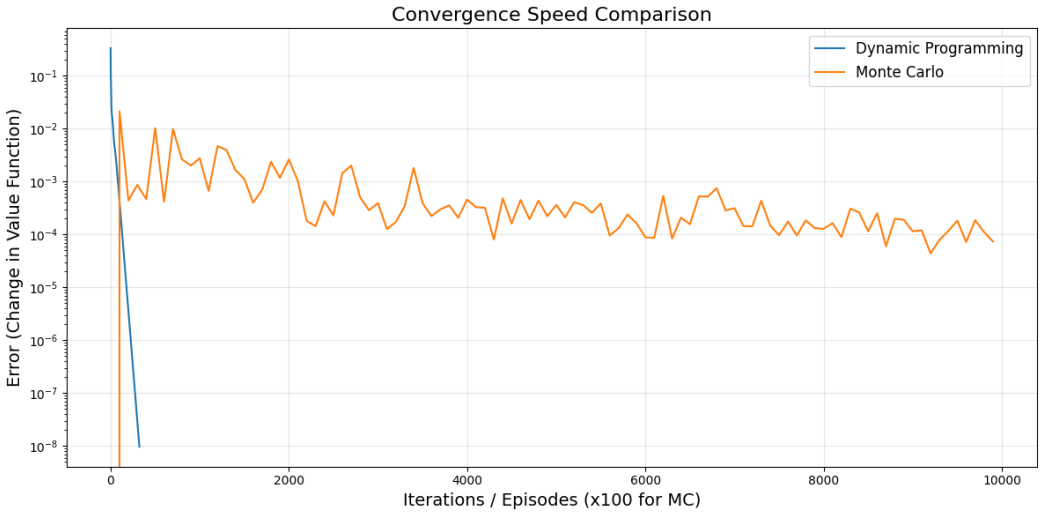

Figure 2: Convergence Speed of DP and MC

0.001111s

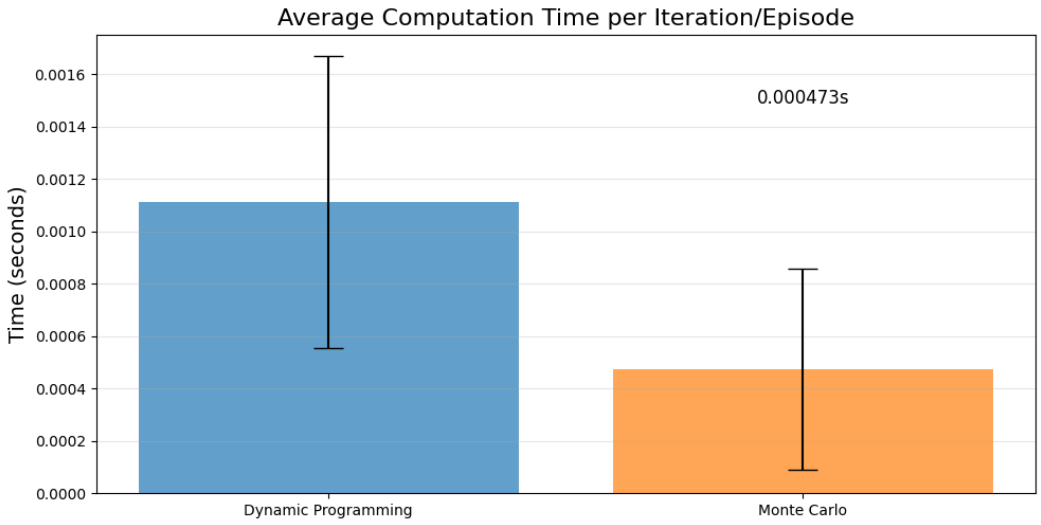

Figure 3: Computational Complexity of DP and MC

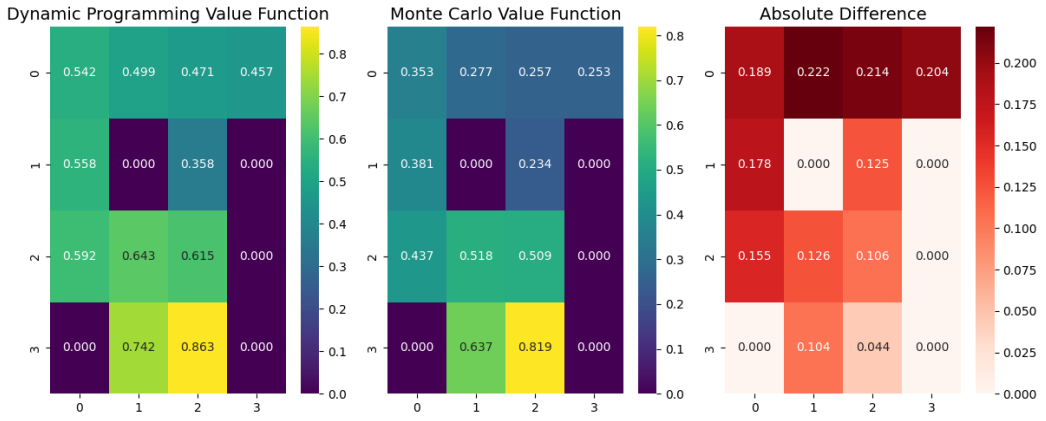

Figure 4: Practical Applicability of DP and MC

Figure 2 shows that DP is faster than MC in terms of convergence speed, figure 3 shows that DP is more computationally expensive than MC, and figure 4 shows that DP is more applicable to smaller environments while MC is more applicable to larger environments. The visuals were plotted by using OpenAI's gym environment values on the FrozenLake environment.

(b) MC's high variance arises from random sampling of episodes, leading to fluctuating value estimates. This instability can cause slow convergence or poor policies, especially in FrozenLake where slipping introduces noise. DP's deterministic updates ensure stability and consistent improvement. However, MC's accuracy improves with more samples but is initially worse than DP's. In FrozenLake, DP might accurately value safe paths, while MC might overestimate risky paths due to rare successful episodes.

(c) For DP, we can use asynchronous updates to reduce computation, apply prioritization such as focusing on high error states, and adjust the discount factor for faster convergence. For MC, we can use importance sampling to reduce the variance, apply function approximations such as Neural Networks for better generalization, and introduce baselines or control variants to stabilize estimates.

(d) DP would require a full model but is faster and more stable, MC is much more flexible and can be used where aspects of the model are unknown but is slower and noisier. DP scales poorly with size whereas MC can handle larger spaces but requires more samples. So DP offers more stability but less flexibility while MC offers flexibility but less scalability. In practice, the choice depends on the problem size, model availability, and computational resources. For example, in a small grid world with known dynamics, DP would be preferred for its speed and stability. In contrast, for a large and complex environment like Atari games, MC methods would be more suitable due to their flexibility and ability to learn from experience.

Q4) Assignment Feedback

- How clear were the objectives and topics covered in the recitation module (e.g., dynamic programming and Monte Carlo methods)?
- Were the concepts explained in a way that helped you understand the material? If not, what could be improved?
- How well did recitation module 2 prepare you for this part of the assignment tasks?
- Do you prefer this style of assignment, i.e., assignment being paired with Lab style Recitation Module 2 in 50% and 50% division?
- How many hours did it take to complete the assignment?
- How difficult did you find the complete assignment?

Solution:

(a) The objectives and topics covered in the recitation were clear. More practice problems would be helpful though.

(b) It was a good preparation, as one could relate back to the recitation.

(c) Yes, I prefer this style of assignment as it introduces both theory and practical aspects of the concepts. Having such assignments in pairs would be more beneficial.

(d) It took me around 10 hours to complete the assignment.

(e) It was somewhat difficult as I had to refer to the book a lot, read, see videos and search up the internet, and also GPT some stuff for clarification.