Markov Decision Process Experiments on Grid World

January 30, 2019

1 Introduction

We implemented value iteration(VI), policy iteration(PI) and modified policy iteration(MPI) based matrix based dynamic programming. Our goal is to find the optimal value functions in the grid world environment. The upper right corner has positive reward +10 and the upper left corner has positive reward +1, all other rewards are 0. The upper right corner and upper left corner are absorbing state so that they stay with probability one, while all other states moves in desired direction with probability p and a random direction with probability 1-p. We compare the empirical performances of three algorithms, including the number of steps and the time required to reach convergence. The analysis of results is conducted int the last section.

2 Algorithms

Algorithm 1-3 specified our implementation of PI, VI and MIP respectively. We first specified values of reward vector R and initialize the transition matrix P^π with under policy. We then compute the action-specific stochastic matrices P^a and reward vectors R^a, $a \in [up, down, left, right]$, considering every move has successful rate p and failure rate 1-p. The rest of the steps in each algorithm are directly translated from the summation based algorithm descriptions in the original papers.

Initialization: Set $V \in R^{n^2}$ to zero and compute
$R \in R^{n^2}, P^\pi \in R^{n^2 * n^2}, P^a \in R^{n^2 * n^2}, R^a = P^a R \in R^{n^2}, a \in [up, down, left, right]$
while $\Delta > \epsilon$*(a small number)* **do**
  Policy evaluation
  $V_{prev} \leftarrow V$
  $V \leftarrow (I - \gamma P^\pi)^{-1} P^\pi R$
  Policy Improvement
  forall i **do**
    $P_i^\pi \leftarrow P_i^a$, where $a = \underset{a'}{\text{argmax}}\, R^{a'} + \gamma P^{a'} V$ and P_i^π is the i-th row of P^π
  end
  $\Delta \leftarrow \|V - V_{prev}\|$
end

Algorithm 1: Policy Iteration

3 Results and Analysis

We run all three algorithms under four different environment settings: p = 0.7 and p = 0.9, n = 5 and n = 50. The discount factor is set to 0.9 and k in modified iteration is set to 10. If the

Initialization: same as above
Value Iteration
while $\Delta > \epsilon$ **do**
  $\mathrm{V}_{prev} \leftarrow V$
  $\mathrm{V} \leftarrow \max R^{a'} + \gamma P^{a'} V, \forall a$
  $\Delta \leftarrow \|V - V_{prev}\|$
end
Policy Improvement
forall i **do**
  $\mathrm{P}_i^{\pi} \leftarrow P_i^a, a = \underset{a'}{\operatorname{argmax}}\, R^{a'} + \gamma P^{a'} V,$
end

Algorithm 2: Value Iteration

Initialization: same as above
while $\Delta > \epsilon$ **do**
  Policy evaluation
  $\mathrm{V}_{prev} \leftarrow V$
  for k *times* **do**
    $\mathrm{V} \leftarrow P^{\pi}(R + \gamma V)$
  end
  Policy Improvement
  forall i **do**
    $\mathrm{P}_i^{\pi} \leftarrow P_i^a, a = \underset{a'}{\operatorname{argmax}}\, R^{a'} + \gamma P^{a'} V,$
    P_i^{π} is the i-th row of P^{π}
  end
  $\Delta \leftarrow \|V - V_{prev}\|$
end

Algorithm 3: Modified Policy Iteration

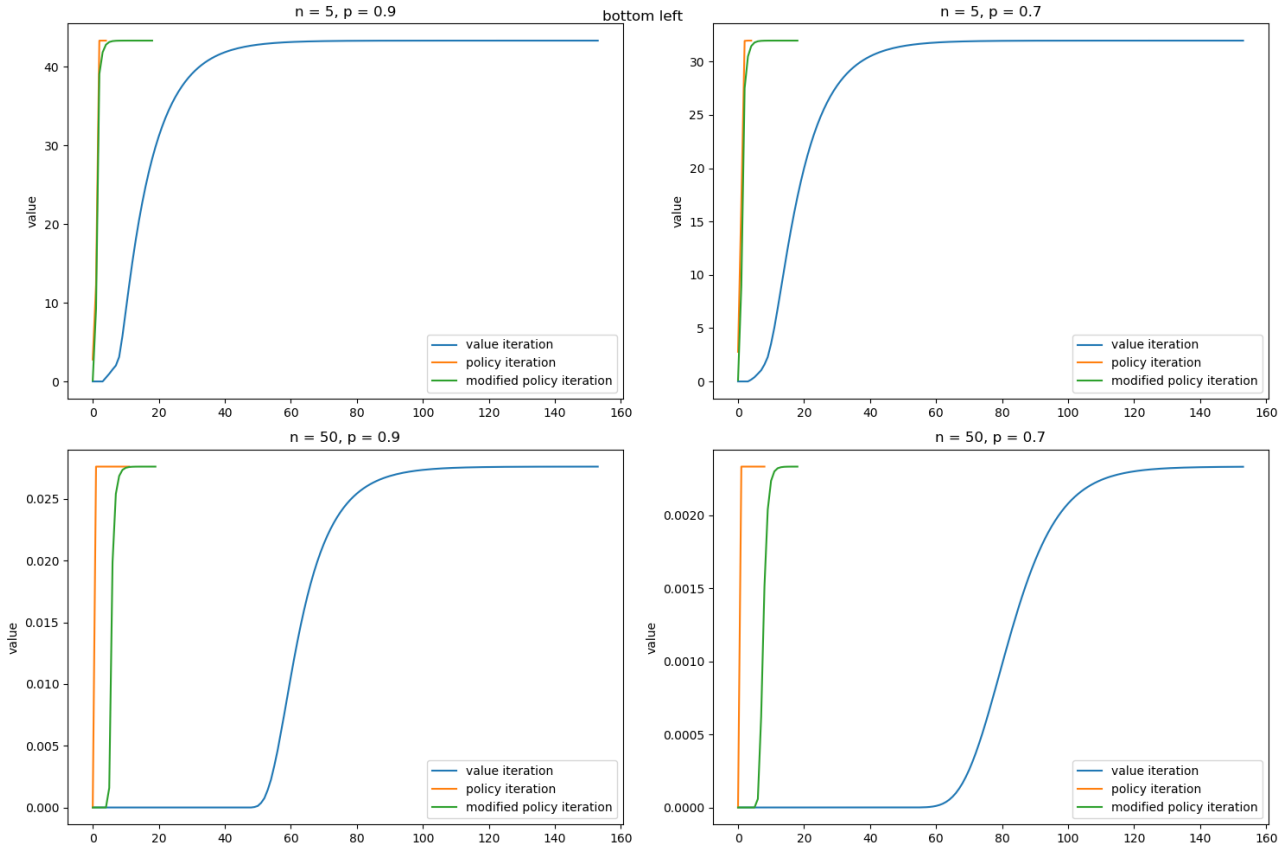

Figure 1: Value of greedy policy at bottom left state

agent bumps into the edge of the grid as a result of a transition, it stays where it was. Figure 1 and figure 2 summarize the values of the bottom left state and bottom right state with respect to number of updates computed by each algorithm. We observe that all three algorithm reached the convergence criterion and they converged to the same values, which are in accordance with the Bellman Optimality theory. The time consumption of each algorithms are summarized in Table 1.

Policy iteration(PI) takes the least number of updates under all four settings. The reason is that we fully evaluate the values under a policy after each policy improvement step by taking matrix inverse, which is computationally expensive. While this is ideal at n = 5, the matrix inversion becomes the bottleneck for time consumption when the state space becomes large. Value iteration(VI), on the other hand, performs a single step of value update following each implicit step of policy improvement, i.e. taking the maximum of returns corresponding to each action. In this manner, it takes more updates for the values to converge, yet each update is much faster.

We can regard PI and VI as two extreme cases of modified policy iteration(MPI), where PI sets $k = \infty$ and VI sets $k = 1$. Instead of evaluating the value function fully after policy improvement, MPI tolerates some error by doing k iterative updates, upon which policy gets improved again. The results illustrates that MPI finds optimal policy using moderately more iterations than PI yet takes much less time when state space gets large. When state space is small, MPI is faster than VI because of more accurate value approximation.

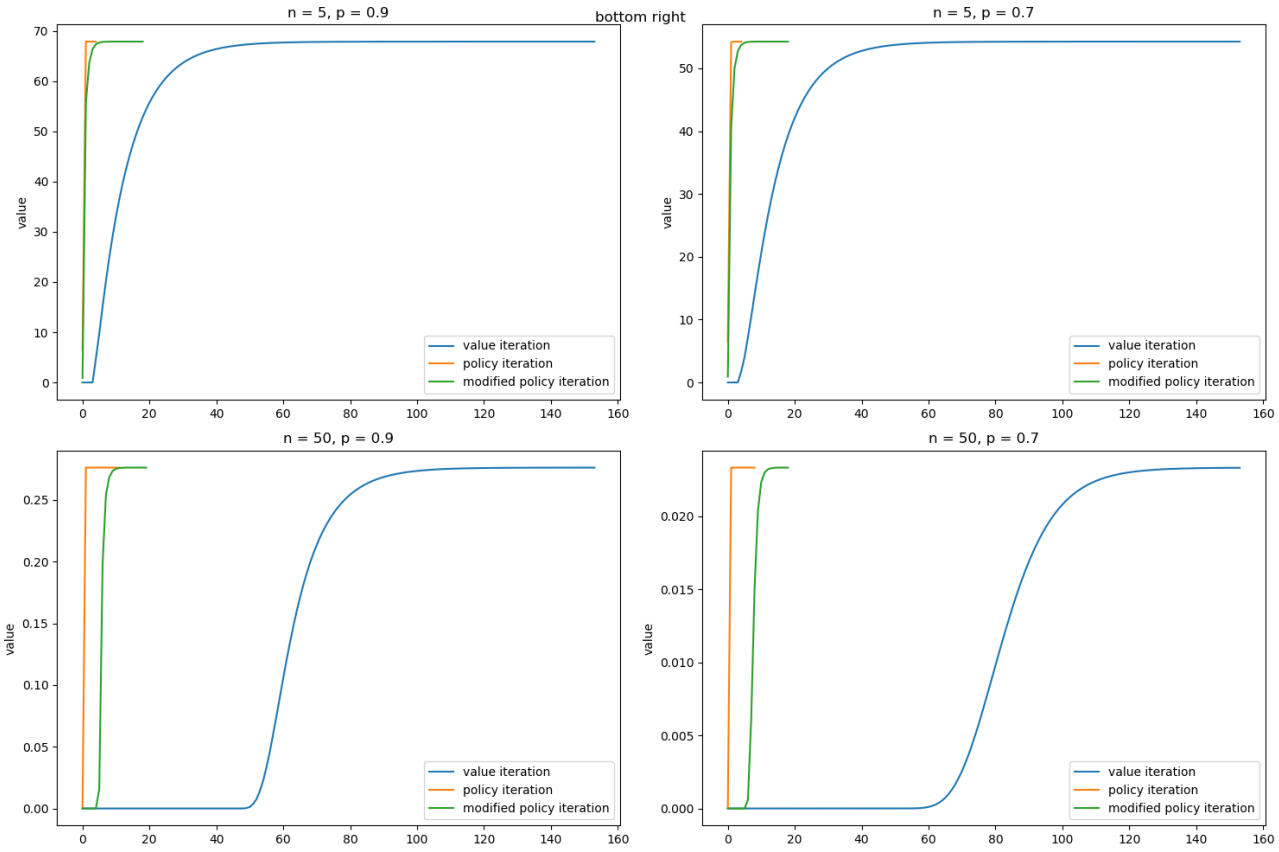

Figure 2: Value of greedy policy at bottom right state

Algorithm	n = 5, p = 0.9	n = 5, p = 0.7	n = 50, p = 0.9	n = 50, p = 0.7
VI	6.26	6.28	**747.74**	**609.73**
PI	2.31	**0.68**	9933.76	7272.85
MPI	**1.71**	1.67	1537.98	1436.72

Table 1: Time consumption for VI(value iteration), PI(policy iteration), MPI(modified policy iteration) in four environment settings(milliseconds)