Eric Komperud

A8 – CS4300 Lab Report –11/30/17
Modified Policy Iteration and Policy Evaluation

1. Introduction

For this assignment, I implemented the Modified Policy Iteration algorithm which is used to find an optimal policy to traverse a world by iterating over and improving bad policies until a good one is found. This algorithm, like the Value Iteration algorithm, takes in an MDP as its parameter and returns a policy once it is completed (unlike the Value Iteration algorithm which only returns a vector of utilities). My implementation of the algorithm was tested on Wumpus World as were previous algorithms/agents. That is, it had to find an optimal policy which would take into account the -1000 reward for dying, the +1000 reward for finding gold, and the -1 reward for taking an action. The board, shown below in Fig 1, was fixed. It also ran on the board shown in Russell & Norvig's *Artificial Intelligence: A Modern Approach* 2nd edition on page 646 (Fig 2). This board has two fixed reward states of -1 and +1 as well as an unreachable state. It is also set up to make it easy to adjust the living reward and consequently change the policy.

0	0	0	G
0	0	P	0
0	0	W	0
0	0	P	0

Fig 1.

Wumpus Board

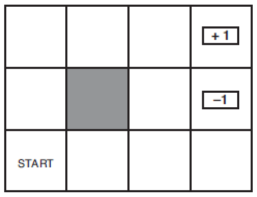


Fig 2.

R&N board

The Modified Policy Iteration algorithm is also similar to the Value Iteration algorithm in that policies it generates can be radically different based on the living rewards. I aimed to prove that it could produce similar results to that of Value Iteration in this regard. However, if it essentially serves the same purpose as Value Iteration, the question of which algorithm is quicker still remains. I aimed to test the running time of each algorithm to see which consistently ran quicker.

2. Method

The parameters for the Modified Policy Iteration algorithm are as follows:

- S: A vector of all the states in the MDP
- A: A vector of all the actions in the MDP
- P: A transition model
- R: A vector of the rewards for each state in the MDP
- k: The number of iterations to use for Policy Evaluation (explained later)
- gamma: the discount factor applied to the rewards for potential future states

The pseudocode for the Modified Policy Iteration algorithm is as follows:

```
function CS4300_Policy_Iteration(mdp) returns a policy
    inputs: mdp, an MDP with states S, actions A, a transition model P, and rewards R
    locals: U, a vector of utilities for states in S, initially 0
            pi, a vector of policies for states in S, initially random
    **repeat**
        U = CS4300_Policy_Evaluation(pi, U, mdp)
        unchanged = true
        **for each** state s **in** S **do**
            **if** $\max_{a \in A(s)} \Sigma_{s'} P(s'|s,a)\, U[s'] > \Sigma_{s'} P(s'|s,pi[s])\, U[s']$ **then do**
                pi[s] = $\text{argmax}_{a \in A(s)}\, \Sigma_{s'} P(s'|s,a)\, U[s']$
                unchanged = false
    **until** unchanged == true
    **return** pi
```

That is, the algorithm performs a loop, potentially changing its policy each iteration until it doesn't change its policy and then returns. Within the loop, it performs a Policy Evaluation to get utilities based on its current policy. It will then evaluate each state to see if there's an action that will yield a higher potential reward than the current policy's given action, based on the utilities given from Policy Evaluation. Once the policy isn't changed in a loop, this means that an optimal policy has been found! What is Policy Evaluation though? Policy Evaluation takes in a policy and calculates the utility of each state if the policy were to be executed for each of those states. Overall, this method was much simpler in terms of the amount of code. The pseudocode is shown below:

```
**function** CS4300_Policy_Evaluation(mdp) **returns** a vector of utilities, U
    **inputs:** mdp, an MDP with states S, actions A, a transition model P, and rewards R
        policy, a vector of actions to take for each state S
        K, number of iterations to run
    **locals:** U, a vector of utilities for states in S, initially 0
    **repeat**
        **for each** state s **in** S **do**
            $U(s) = R(s) + \text{gamma} * (\Sigma P(s'|s,pi[s])\, U[s'])$
    **until** K is reached
    **return**
```

To gather data on whether or not the Modified Policy Iteration algorithm would produce policies that matched those produced by the Value Iteration algorithm, I simply ran my Modified Policy Iteration implementation on the same board that I did for the Value Iteration implementation to confirm that they were the same. To collect data on the times it took to run each algorithm, I ran each algorithm 30 times, recording the time it took for each to run from start toP finish and produce a policy.

3. Verification of Program

I verified my program by running it with the same parameters of the boards shown in R&N on page 648 to see if the policies generated by my algorithm matched the results shown in the book.

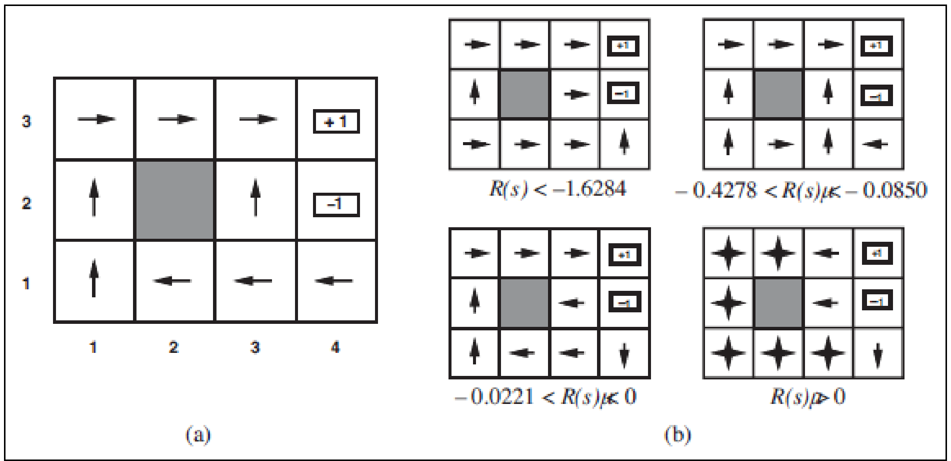


Figure 17.2 (a) An optimal policy for the stochastic environment with $R(s) = -0.04$ in the nonterminal states. (b) Optimal policies for four different ranges of $R(s)$.

Fig 3. R&N policies

The policies of my algorithm for the various R(s) values are shown below. Note that terminal states use '1 X' as a placeholder value:

R(s) = -0.04

4 ->	4 ->	4 ->	1 X
1 ^	1 X	1 ^	1 X
1 ^	2 <-	2 <-	2 <-

Fig 4.

R(s) = -2.00 / R(s) < -1.6284

4 ->	4 ->	4 ->	1 X
1 ^	1 X	4 ->	1 X
4 ->	4 ->	4 ->	1 ^

Fig 5.

R(s) = -0.30 / -0.4278 < R(s) < -0.0850

4 ->	4 ->	4 ->	1 X
1 ^	1 X	1 ^	1 X
1 ^	4 ->	1 ^	2 <-

Fig 6.

R(s) = -0.01 / -0.0221 < R(s) < 0

4 ->	4 ->	4 ->	1 X
1 ^	1 X	2 <-	1 X
1 ^	2 <-	2 <-	3 v

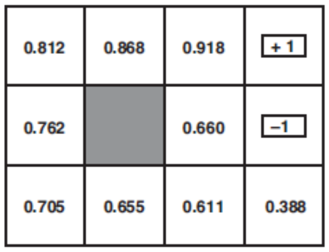
Fig 7.

1 ^	1 ^	2 <-	1
1 ^	1	2 <-	1
1 ^	1 ^	2 <-	3 v

R(s) = 2.00 / R(s) > 0 Fig 8.

In addition, the utilities returned by my algorithm for R(s) = -0.04 also closely match those shown in the book for the same R(s) on page 651.

0.811558	0.867808	0.917808	1
0.761558	0	0.660274	-1
0.705308	0.655308	0.611416	0.387925



0.812	0.868	0.918	$\boxed{+1}$
0.762	▨	0.660	$\boxed{-1}$
0.705	0.655	0.611	0.388

Fig 9(a) and 9(b).

Since my algorithm indeed produced policies and utilities that matched those shown in the book, I conclude that it is correct for the problem at hand.

4. Data

The policy generated by my algorithm for the Wumpus board shown in Fig 1 is as follows:

4 ->	4 ->	1 ^	1 X
1 ^	2 <-	1 X	4 ->
1 ^	2 <-	1 X	4 ->
1 ^	2 <-	1 X	4 ->

Fig 10.

The times recorded in seconds for 30 runs of the Value Iteration and Modified Policy Iteration algorithms to generate policies for the Wumpus board in Fig 1 are shown below. The mean, standard deviation, and 95% confidence interval are also shown:

Algorithm	Modified Policy Iteration	Value Iteration
Time (s)	0.00467	0.046531
	0.004469	0.025985
	0.004222	0.02373
	0.003686	0.02733
	0.003078	0.0251
	0.00272	0.023194
	0.004622	0.022799
	0.003068	0.022985
	0.003072	0.023762
	0.003251	0.024531
	0.003345	0.022623

	0.004099	0.024971
	0.002609	0.022816
	0.002572	0.021946
	0.003644	0.022197
	0.002866	0.022193
	0.005192	0.023604
	0.002978	0.020817
	0.00258	0.022207
	0.002387	0.022941
	0.002893	0.023102
	0.004495	0.024015
	0.002393	0.0226
	0.002321	0.0219
	0.002332	0.022075
	0.003156	0.022837
	0.00481	0.021824
	0.002489	0.023109
	0.003022	0.022404
	0.003101	0.023306
Mean	0.003338	0.023981
Std Deviation	0.000833	0.004389
95% Confidence Interval	0.003338 ±2.98e-4	0.023981 ±0.001571

Fig 11.

5. Analysis

The fact that my implementation of the Modified Policy Iteration algorithm produces the same policies as the Value Iteration algorithm reveals two things. The first is that I now know Modified Policy Iteration can produce reliable policies. That's pretty cool. The second is that these two algorithms now contend with each other for the privilege of being called the most efficient policy generator. That's where my timing data comes in. The results of my timing tests clearly show that Modified Policy Iteration is the faster algorithm with 95% confidence.

6. Interpretation

The results of assignments A7 + A8 reveal that both the Value Iteration and Modified Policy Iteration algorithms are effective ways to generate policies for Wumpus World. However, A8 reveals the Policy Iteration is the more efficient of the two (at least for Wumpus World). Though the differences are small in this problem, larger problems may reveal more dramatic differences.

7. Critique

During this assignment, I spent a large amount of time trying to get Policy Evaluation working by way of solving a system of linear equations. That original method actually did work for many problems, but failed to accurately solver problems in which the living reward was positive. I then attempted an iterative method and was able to get that working within an hour. My advice to anyone reading this would be to NOT use the system of linear equations method. Definitely use the iterative method instead. Further research might be done on larger problems to see the difference in computational complexity of using Value Iteration over Policy Iteration.

8. Log

I spent approximately 6 hours writing the methods for my code and debugging them. I spent approximately an hour gathering data for analysis and verification. I spent approximately 2 hours writing this lab report.