Monish Gupta (Odds) and Eric Waugh (Evens)

CS 4300

November 28 2017

Assignment 8 Lab Report

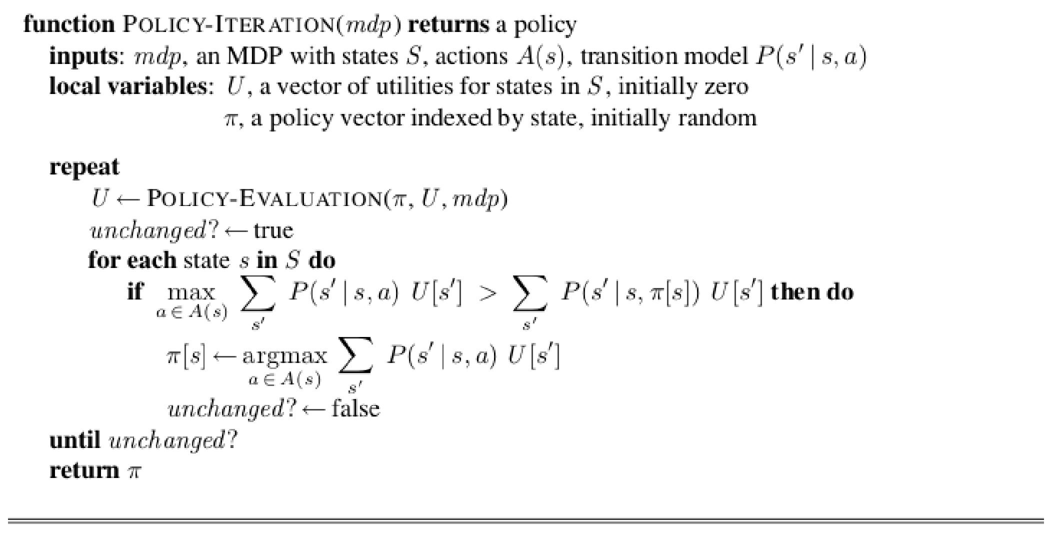
1. *Introduction*

   The policy iteration algorithm for Markov Decision Problems (MDPs) is a variant of the generalized policy iteration algorithm that relaxes policies on the board until they have reached an optimal and unchanging status. Similar to the value iteration algorithm , the policy iteration derives a new policy based off an old policy until the new policy and the old policy are the same. The key advantage to using policy iteration is its lack of a max function. This allows the algorithm to avoid non-linear running times as it does not need to compute precise utilities to designate an optimal action onto a particular state. There are two phases to policy iteration, policy evaluation and policy improvement. Policy evaluation provides new calculations of utilities based on changes from policy changes from previous iterations, and policy improvement changes the current working policy based off of utilities retrieved from policy evaluation. There are two ways to perform policy evaluation, one attempts to solve for utilities in a system of equations, the other opts for a value iteration hybrid. The implementation opts for the latter.

   Question : What impact does the amount of iterations (k) in the policy iteration algorithm have on the resulting utilities and policies? The cost to compute the utilities?

2. *Method*

   For the value iteration function we followed the algorithm from the book.



```
function POLICY-ITERATION($mdp$) returns a policy
  inputs: $mdp$, an MDP with states $S$, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: $U$, a vector of utilities for states in $S$, initially zero
                   $\pi$, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION($\pi, U, mdp$)
    $unchanged? \leftarrow$ true
    for each state $s$ in $S$ do
      if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']$ then do
        $\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return $\pi$
```

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.

The algorithm provided from the book

This takes in generic policies, in our case every state going up, and determines what the best current policy is by determining the current utilities from Policy Evaluation. It does this as long as the current policy is being changed and once it is no longer being changed it is the final policy. We used our previous helped function to find the best policy instead of doing the summations shown in the algorithm from the book.

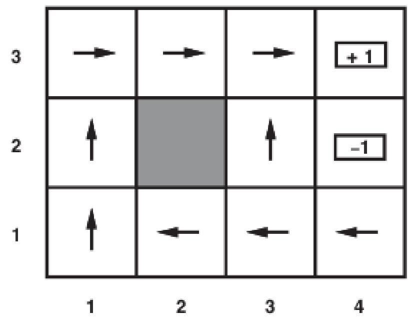
For policy evaluation we used the summation of each of the state’s probability with the current policy to get the new utilities for each state.

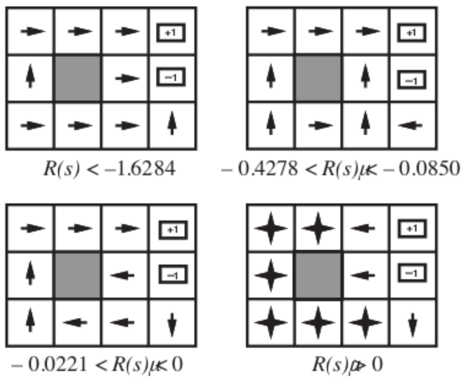
```
n = size(S,2);
U = zeros(1,n);
policy = ones(1,n);
Ut = [];

unchanged = 0;

while unchanged == 0
    unchanged = 1;
    U = CS4300_Policy_Evaluation(U,S,A,P,R,policy,k,gamma);

    policy_new = CS4300_MDP_policy(S,A,P,U);
    for s = 1:n
        if policy_new(s) ~= policy(s)
           policy(s) = policy_new(s);
           unchanged = 0;
        end
    end

    Ut = [Ut;U];

end
```

Our algorithm

3. *Verification*

To verify our policies and value iteration we used the 3 by 4 board provided that in the header that tells us the policies we should have (also in the book).





The policy for the 3 by 4 board

Policies for different rewards

RIGHT	RIGHT	RIGHT	+1
UP	X	UP	-1
UP	LEFT	LEFT	LEFT

The policy for R = -0.04

RIGHT	RIGHT	RIGHT	+1
UP	X	RIGHT	-1
RIGHT	RIGHT	RIGHT	UP

The policy for R < -1.6284

RIGHT	RIGHT	RIGHT	+1
UP	X	UP	-1
UP	RIGHT	UP	LEFT

The policy for 0.4278 < R < -0.0850

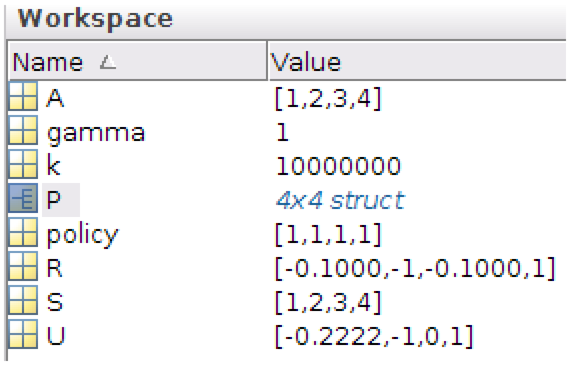
RIGHT	RIGHT	RIGHT	+1
UP	X	LEFT	-1
UP	LEFT	LEFT	DOWN

The policy for -0.0221 < R < 0

RIGHT	RIGHT	LEFT	+1
UP	X	LEFT	-1
UP	LEFT	UP	DOWN

The policy for R > 0

We also used the 2 by 2 board example from class to verify our Policy Evaluation function. This board has the first and third spots equal to -.1, the second spot equal to -1 and the fourth spot equal to 1. The probabilities are as expected when moving in any direction with the start policy also being set to all ones. The expected returned U is [-0.222,-1,0,1]. Our workspace shows that we get the correct U from using this example.



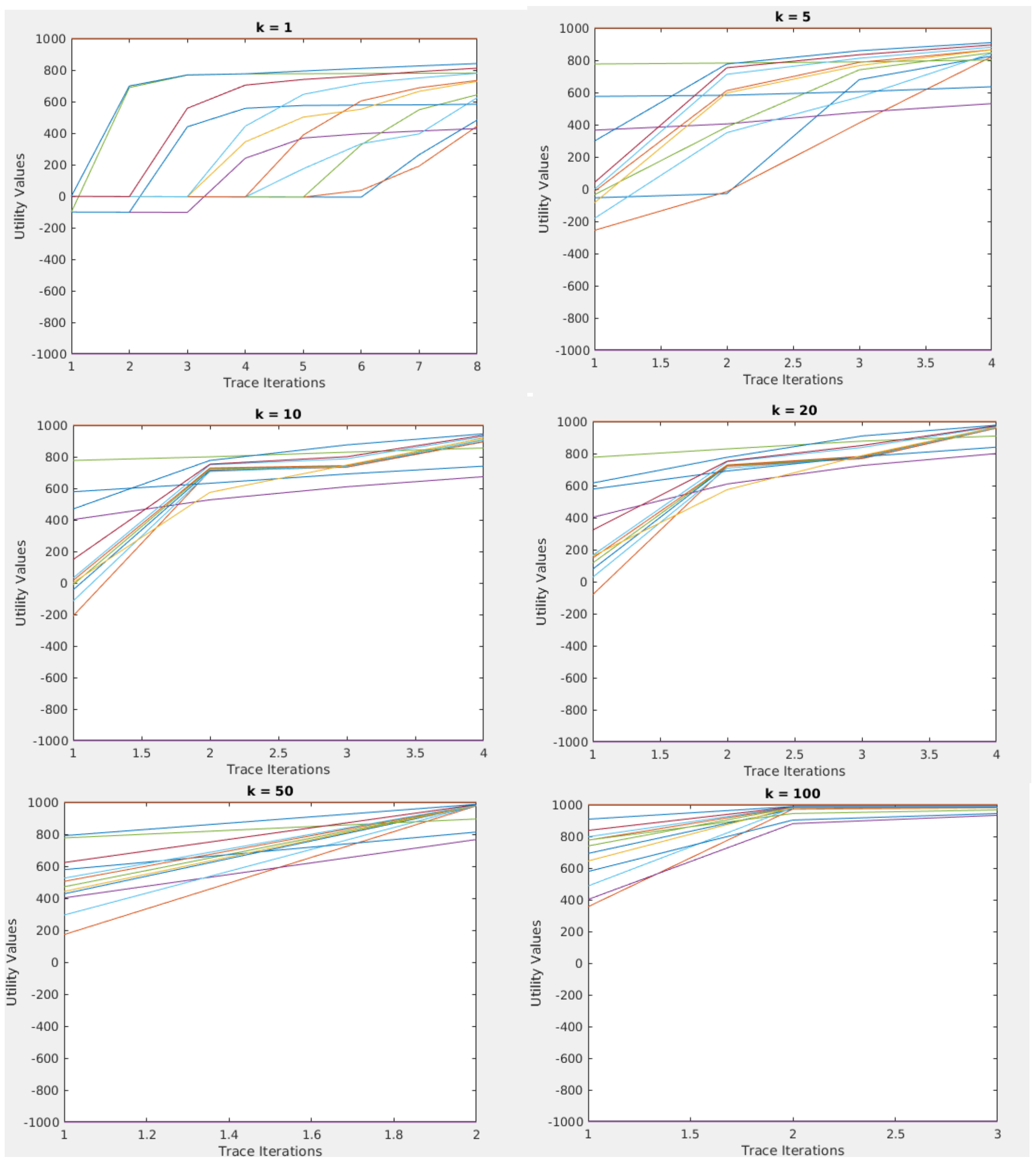
Workspace

Name	Value
A	[1,2,3,4]
gamma	1
k	10000000
P	4x4 struct
policy	[1,1,1,1]
R	[-0.1000,-1,-0.1000,1]
S	[1,2,3,4]
U	[-0.2222,-1,0,1]

2 by 2 example workspace

4. *Data and Analysis*

Graphs of Ut using various k iterations.



5. *Interpretation*

For lower k’s we can see that the unchanged loop goes on longer since there are more utilities that are spread out. Each of these different k’s will return the same policy but for each Policy Evaluation call the utilities returned will not well represent the final utilities for that given policy that was found so it takes more time to find the final policy. To the naive eye, it may seem as though a higher k ensures the algorithm suffers a lesser cost to find optimal, but although the amount of trace iterations is less, we are performing these trace iterations k times. This leads us to do more work overall (1(k) * 8(trace count) < 5 * 4 < 10 * 4). The utilities and policy rely on each other and having a larger k allows us to have a more consistent way to find the optimal policy coming from evaluation giving the most proper utilities instead of many new policies being generated. You can also see in our graphs that there a less steep jumps in the utilities with a higher k due to it not taking multiple evaluations to figure out a more proper utility.

6. *Critique*

   During this assignment I didn't feel like there was any reason why this policy iterator is a better way to get the policy for a wumpus world instead of using what we did last week. For example they give the same outputs for the 3 by 4 board for that boards policies and neither of them seem to run faster than the other. Im mostly curious as to why we would focus on solving the same problem in a different way. If we have a fast way to find policies why spend time on another assignment writing another way to find policies that seems not different in speed or outcomes.

7. *Log:*

   Eric:

   Assignment: 8 hours

   Lab: 4 hours

   Monish:

   Assignment: 8 hours

   Lab: 4 hours