Reinforcement Learning

CC482 : Artificial Intelligence

Assignment 3

Abstract

maze solver using Markov Decision Process' Policy Iteration and Value Iteration algorithms

26/1/2019

TEAM

Dahlia Chehata 27

Fares Mehanna 52

Omar Shawky 43

Table Of Contents

Policy Iteration……………………………………………………………………………………. 3

Approach………………………………………………………………………………….. 3

Algorithms and Data Structures………………………………………………………………… 4

path to goal…………………………………………………………………………………... 5

initial maze with S as starting point and E as terminal point……………………………….. 5

path from start to goal……………………………………………………………………... 5

Cost……………………………………………………………………………………………5

Running time and state value function……………………………………………………… 6

Value Iteration…………………………………………………………………………………... 6

Approach………………………………………………………………………………….. 6

Algorithms and Data Structures………………………………………………………………… 7

path to goal…………………………………………………………………………………....8

initial maze with S as starting point and E as terminal point……………………………….8

path from start to goal……………………………………………………………………... 8

Cost……………………………………………………………………………………………8

Running time and state-value function……………………………………………………….9

Extra-Work……………………………………………………………………………………….....10

Visualization and GUI……………………………………………………………………....10

performance Measure…………………………………………………………………….. 11

changing maze size NXN with respect to the number of iterations………………….. 11

changing gamma values for a constant 20x20 maze size……………………………..11

Time taken to solve the maze with different sizes NxN………………………………. 12

Sample Runs……………………………………………………………………………………....12

User Manual……………………………………………………………………………………. 12

Policy Iteration

- **Approach**

  Starting with a random policy π_0, this approach consists of two steps:

  I. **Policy Evaluation:** Calculate the utilities of a non-optimal policy π_i

  $$Q^{\pi_i}(s,a) = r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^{\pi_i}(s')$$

  II. **Greedy Policy improvement:** Update the policy using the resulting converged utilities from the previous step to obtain π_{i+1}.

  $$\pi_{i+1}(s) = \arg\max_a Q^{\pi_i}(s,a)$$

  These steps are repeated until **π** converges to **π***

- **Algorithms and Data Structures**

DS: dictionary

```
def policy_iteration(grid, gamma):
   policy_changed = True
   policy = [['up' for i in range(len(grid[0]))] for j in range(len(grid))]
   actions = ['up', 'down', 'left', 'right']
   iters = 0

   '''Policy iteration'''
   while policy_changed:
       policy_changed = False

       ''' 1- Policy evaluation '''
       # no transition probabilities = deterministic
       value_changed = True

       while value_changed:
           value_changed = False
           # Run value iteration for each state
           for i in range(len(grid)):
               for j in range(len(grid[i])):
                   if grid[i][j] == '$':
                       policy[i][j] = '$'
                   else:
                       neighbor = getattr(grid[i][j], policy[i][j])
                       # V = R + 𝛾  Σ  PV
                       v = grid[i][j].reward + gamma * grid[neighbor[0]][neighbor[1]].value
                       # Compare to previous iteration
                       if v != grid[i][j].value:
                           value_changed = True
                           grid[i][j].value = v

       '''2- Greedy Policy'''
       # Once values have converged for the policy, update policy with greedy approach
       for i in range(len(grid)):
           for j in range(len(grid[i])):
               if grid[i][j] != '$':
                   action_values = {a: grid[getattr(grid[i][j], a)[0]]
                                   [getattr(grid[i][j], a)[1]].value for a in actions}
                   best_action = max(action_values, key=action_values.get)
                   # Compare to previous policy
                   if best_action != policy[i][j]:
                       policy_changed = True
                       policy[i][j] = best_action
       iters += 1
   return policy
```

- **path to goal**
  - *initial maze with S as starting point and E as terminal point*

```
['$', '$', '$', '$', '$', '$', '$', '$', '$', '$', '$']
['$', 'E', ' ', ' ', ' ', ' ', '$', ' ', ' ', ' ', '$']
['$', ' ', '$', '$', '$', ' ', '$', ' ', '$', ' ', '$']
['$', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', '$']
['$', '$', '$', ' ', '$', '$', '$', '$', '$', ' ', '$']
['$', ' ', ' ', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$']
['$', ' ', '$', '$', '$', '$', '$', '$', '$', ' ', '$']
['$', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$', ' ', '$']
['$', ' ', '$', ' ', '$', '$', '$', '$', '$', ' ', '$']
['$', 'S', ' ', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$']
['$', '$', '$', '$', '$', '$', '$', '$', '$', '$', '$']
```

  - *path from start to goal*

```
$ $ $ $ $ $ $ $ $ $ $
$ ↑ ← ← ← ← $ ↓ ← ← $
$ ↑ $ $ $ ↑ $ ↓ $ ↑ $
$ ↑ $ → → ↑ ← ← $ ↑ $
$ $ $ ↑ $ $ $ $ $ ↑ $
$ → → ↑ $ → → → → ↑ $
$ ↑ $ $ $ $ $ $ $ ↑ $
$ ↑ $ ↓ ← ← ← ← $ ↑ $
$ ↑ $ ↓ $ $ $ $ $ ↑ $
$ ↑ ← ← $ → → → → ↑ $
$ $ $ $ $ $ $ $ $ $ $
```

- **Cost**

```
Path cost is: 16
Path to goal is: [(9, 1), (8, 1), (7, 1), (6, 1), (5, 1), (5, 2), (5, 3), (4, 3), (3, 3), (3, 4), (3, 5), (2, 5), (1, 5), (1, 4), (1, 3), (1, 2), (1, 1)]
```

- **Running time and state value function**

Value Iteration

- **Approach**
  The value iteration approach finds the optimal policy π* by calculating the optimal value function, V*. Starting with V(s) = 0 for all states s, the values of each state are iteratively updated to get the next value function V, which converges towards V*.

$$V_{k+1}(s) \quad = \quad \max_a R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) V_k(s')$$

- **Algorithms and Data Structures**

DS : dictionary

```
def value_iteration(grid, gamma):
   policy = [['up' for i in range(len(grid[0]))] for j in range(len(grid))]
   actions = ['up', 'down', 'left', 'right']
   value_changed = True
   iters = 0
   # iterate values until convergence
   while value_changed:
       value_changed = False
       for i in range(len(grid)):
           for j in range(len(grid[i])):
               if grid[i][j] != '$':
                   q = []
                   for a in actions:
                       neighbor = getattr(grid[i][j], a)
                       q.append(grid[i][j].reward
                         + gamma * grid[neighbor[0]][neighbor[1]].value)
                       v = max(q)
                       if v != grid[i][j].value:
                          value_changed = True
                          grid[i][j].value = v
       iters += 1
   for i in range(len(grid)):
       for j in range(len(grid[i])):
           if grid[i][j] != '$':
              action_values = {a: grid[getattr(grid[i][j], a)[0]]
              [getattr(grid[i][j],  a)[1]].value for a in actions}
               policy[i][j] = max(action_values, key=action_values.get)
               # Compare to previous policy
           else:
               policy[i][j] = '$'
   return policy
```

- **path to goal**
    - *initial maze with S as starting point and E as terminal point*

```
['$', '$', '$', '$', '$', '$', '$', '$', '$', '$', '$']
['$', 'E', ' ', ' ', '$', ' ', ' ', ' ', ' ', ' ', '$']
['$', '$', '$', ' ', '$', '$', '$', '$', '$', ' ', '$']
['$', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '$', ' ', '$']
['$', ' ', '$', '$', '$', '$', '$', ' ', '$', ' ', '$']
['$', ' ', '$', ' ', ' ', ' ', '$', ' ', ' ', ' ', '$']
['$', ' ', '$', ' ', '$', ' ', '$', '$', '$', '$', '$']
['$', ' ', '$', ' ', '$', ' ', '$', ' ', ' ', ' ', '$']
['$', ' ', '$', ' ', '$', ' ', '$', '$', '$', ' ', '$']
['$', ' ', ' ', 'S', '$', ' ', ' ', ' ', ' ', ' ', '$']
['$', '$', '$', '$', '$', '$', '$', '$', '$', '$', '$']
```

- *path from start to goal*

```
$ $ $ $ $ $ $ $ $ $ $
$ ↑ ← ← $ → → → → ↓ $
$ $ $ ↑ $ $ $ $ $ ↓ $
$ → → ↑ ← ← ← ← $ ↓ $
$ ↑ $ $ $ $ $ ↑ $ ↓ $
$ ↑ $ ↓ ← ← $ ↑ ← ← $
$ ↑ $ ↓ $ ↑ $ $ $ $ $
$ ↑ $ ↓ $ ↑ $ → → ↓ $
$ ↑ $ ↓ $ ↑ $ $ $ ↓ $
$ ↑ ← ← $ ↑ ← ← ← ← $
$ $ $ $ $ $ $ $ $ $ $
```

- **Cost**

```
Path cost is: 14
Path to goal is: [(9, 3), (9, 2), (9, 1), (8, 1), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1), (3, 2), (3, 3), (2, 3), (1, 3), (1, 2), (1, 1)]
```

- **Running time and state-value function**

where the golden circles in the grid squares are the ones with the highest value function close to 100 and above and the golden color intensity decreases with the value decrease

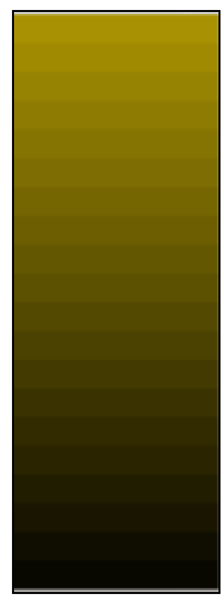
The black squares are the ones with lowest values

Extra-Work

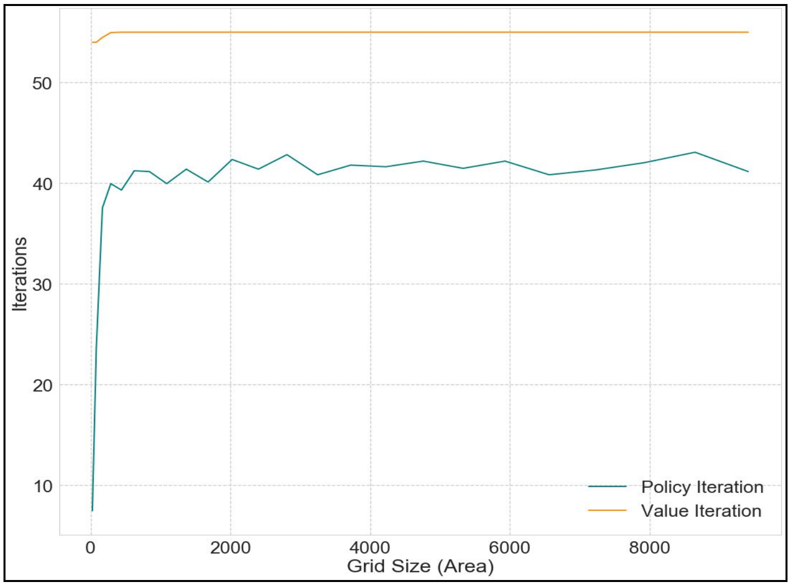
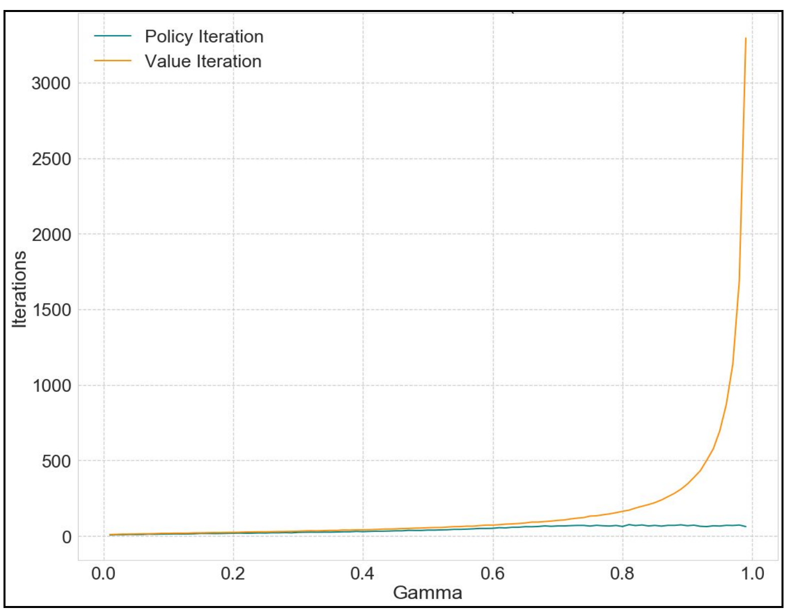
- **Visualization and GUI**
    - We have 2 versions the **GUI version** and the **console printed maze version** for both **policy iteration** and **value iteration**
    - **The policy iteration algorithm:**
        - The GUI version shows the transition between states by optimizing the policy each iteration (represented by the arrows) till reaching the optimal policy. The running time and the value function as well as the number of iterations are printed every iteration as shown in fig 1

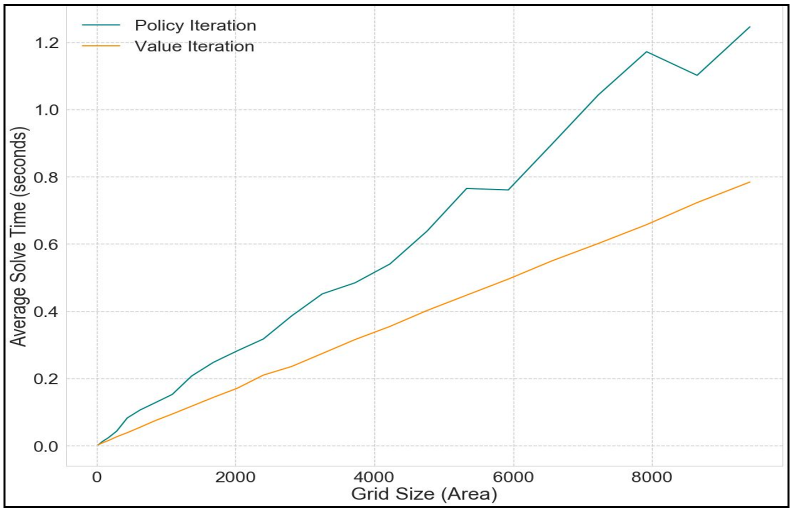
        - The printed version prints each step sequentially with choosing a random start state and compute the path cost as shown in fig 3 , fig 4 and fig 5

    - **The value iteration algorithm**

<table>
<tr><td>example of the value iteration case</td><td>scale of V(s)</td></tr>
<tr><td>
</td><td>105

-10</td></tr>
<tr><td colspan="2">The gradient between the golden and the black colors are used to represent the value range of the state value function of each cell till the optimization</td></tr>
</table>

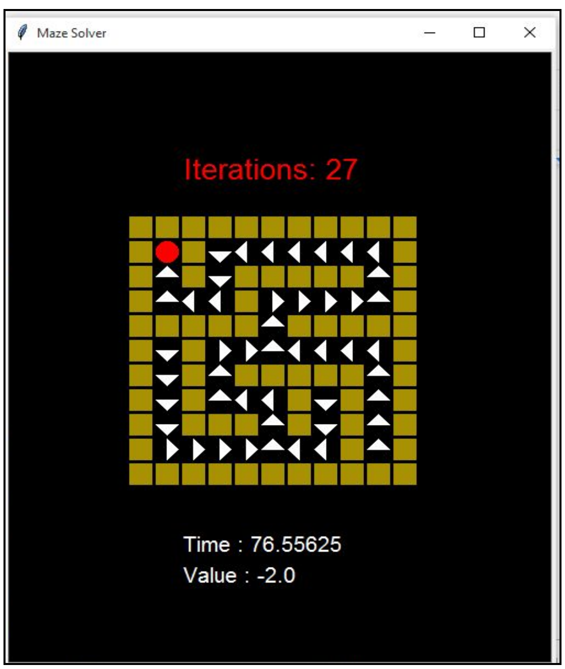
- **performance Measure**

changing maze size	Description
	when changing maze size by different NxN values: - The number of iterations in policy iteration increases gradually with the size till certain threshold (100 X 100) and then it fluctuates around certain point or nearly constant - The number of iterations in value iteration is nearly constant and always higher than its corresponding value in the policy iteration
changing gamma values for a constant 20x20 maze size with respect to the number of iterations	**Description**
	when changing the gamma values with the range (0.01, 1.0) with increase of 0.01 : - The policy iteration increases gradually with a maximum threshold of 100 (linear increase) - The value iteration increases gradually too but suffers from unbounded increase with gamma = 1 (exponential increase)

Time taken to solve the maze with different sizes NxN	Description
	value iteration surpasses policy iteration in terms of runtime, despite requiring more iterations to converge to the optimal policy.

Sample Runs

policy iteration



value iteration

User Manual

- for Policy Iteration, run **PI_visualization** for GUI and after closing the GUI window, the path information are printed in the console
- for Value Iteration, run **VI_visualization** for GUI and after closing the GUI window, the path information are printed in the console