Sieci samouczące się

Laboratorium 2

Przemysław Rośleń 180150

Problem żeglarza rozwiązałem stosując jeden z wariantów algorytmu programowania dynamicznego dla uczenia ze wzmocnieniem Q-learning:

● Iteracji wartości

Poniżej przedstawiłe:

● kod źródłowy funkcji rozwiązującej problem żeglarza stosując dany algorytm

● wyniki w postaci średnich sum nagród uzyskanych w środowiskach:
  ○ map_small
  ○ map_middle
  ○ map_spiral

● wyniki w postaci wizualizacji uzyskanej strategii (zwroty strzałek na mapie)

Iteracja wartości:

Kod:

1) Funkcja obliczająca prawdopodobieństwo ruchu w konkretnym kierunku na podstawie wykonanej akcji:

```
def get_move_probability(action, direction):
    if action == 1:
        if direction == 1:
            return 0.4
        elif direction == 2:
            return 0.2
        elif direction == 3:
            return 0.04
        else:
            return 0.36
    elif action == 2:
        if direction == 1:
            return 0.2
        elif direction == 2:
            return 0.4
```

```
        elif direction == 3:
            return 0.36
        else:
            return 0.04
    elif action == 3:
        if direction == 1:
            return 0.04
        elif direction == 2:
            return 0.2
        elif direction == 3:
            return 0.4
        else:
            return 0.36
    elif action == 4:
        if direction == 1:
            return 0.2
        elif direction == 2:
            return 0.04
        elif direction == 3:
            return 0.36
        else:
            return 0.4
```

2) Funkcja obliczająca nagrodę na podstawie stanu wyjściowego oraz wykonanej akcji:

```
def get_reward(init_state, action, reward_map):
    num_of_rows, num_of_columns = reward_map.shape
    if action == 1:
        return reward_map[init_state[0]][init_state[1] + 1] if
(init_state[1] + 1) < num_of_columns else WALL_COLLID_REWARD
    elif action == 2:
        return reward_map[init_state[0] - 1][init_state[1]] if
(init_state[0] - 1) >= 0 else WALL_COLLID_REWARD
    elif action == 3:
        return reward_map[init_state[0]][init_state[1] - 1] if
(init_state[1] - 1) >= 0 else WALL_COLLID_REWARD
    elif action == 4:
        return reward_map[init_state[0] + 1][init_state[1]] if
(init_state[0] + 1) < num_of_rows else WALL_COLLID_REWARD
```

3) Główna funkcja dynamicznej iteracji wartości realizująca dla danego stanu iterację po wszystkich możliwych akcjach, obliczanie wszystkich składowych potrzebnych do aktualizacji tablicy wartości stanów, a następnie aktualizacje wartości tej tablicy dla danego stanu oraz aktualizacja aktualnego błędu iteracji (parametru delta).

$$V(s) \leftarrow \max_a(r(s,a) + \gamma \sum_{s'} p(s'|s,a)V_{pom}(s'))$$
$$\delta \leftarrow \max(\delta, |V(s) - V_{pom}(s)|)$$

Dodatkowo funkcja realizuje obliczenie tablicy użyteczności stanów Q na podstawie poniższego wzoru:

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

```
def dynamic_value_iteration(reward_map, V, Vpom, init_state,
gamma, delta, Q):
   num_of_rows, num_of_columns = reward_map.shape
   max_reward = -1000
   for action in range(1, 5):
       # r(s,a)
       reward = get_reward(init_state, action, reward_map)
       # Vpom(s')
       right_value = V[init_state[0]][init_state[1] + 1] if
(init_state[1] + 1) < num_of_columns else WALL_COLLID_REWARD
       up_value = V[init_state[0] - 1][init_state[1]] if
(init_state[0] - 1) >= 0 else WALL_COLLID_REWARD
       left_value = V[init_state[0]][init_state[1] - 1] if
(init_state[1] - 1) >= 0 else WALL_COLLID_REWARD
       down_value = V[init_state[0] + 1][init_state[1]] if
(init_state[0] + 1) < num_of_rows else WALL_COLLID_REWARD
       # sum(p(s'|s, a) * Vpom(s')
       right_probability = get_move_probability(action, 1)
       up_probability = get_move_probability(action, 2)
       left_probability = get_move_probability(action, 3)
       down_probability = get_move_probability(action, 4)
       # V(s)
       reward += (gamma * ((right_value * right_probability) +
(up_value * up_probability) + (left_value * left_probability) +
(down_value * down_probability)))
       Q[init_state[0]][init_state[1]][action - 1] = reward

       if reward > max_reward:
           max_reward = reward

   V[init_state[0]][init_state[1]] = max_reward
   delta = max(delta, abs(V[init_state[0]][init_state[1]] -
Vpom[init_state[0]][init_state[1]]))
   return V, delta, Q
```

4) Wywołanie powyższej funkcji w mainie zgodnie ze schematem przedstawionym poniżej oraz ocena jakości modelu:

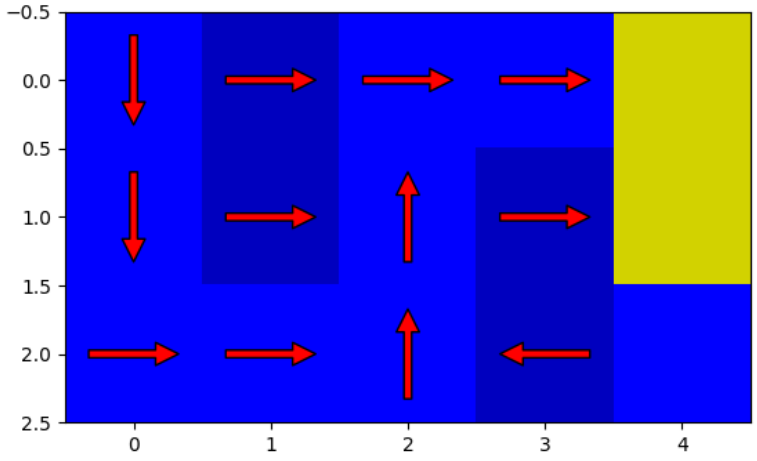
while $\delta \geq \delta_{max}$ **do**
    $V_{pom} \leftarrow V$
    $\delta \leftarrow 0$
    for all $s \in \mathbb{S}$ **do**
        $V(s) \leftarrow \max_a(r(s,a) + \gamma \sum_{s'} p(s'|s,a)V_{pom}(s'))$
        $\delta \leftarrow \max(\delta, |V(s) - V_{pom}(s)|)$
    end for
end while

```
# # DYNAMICZNA ITERACJA WARTOŚCI
V = np.zeros([num_of_rows, num_of_columns], dtype=float)
delta = 1000
while delta >= 4:
   Vpom = np.copy(V)
   delta = 0
   for row in range(num_of_rows):
       for col in range(num_of_columns - 1):
           V, delta, Q =
sf.dynamic_value_iteration(reward_map, V, Vpom, [row, col],
gamma, delta, Q)

   sf.sailor_test(reward_map, Q, number_of_episodes)
   sf.draw(reward_map, Q)
   for raw in V:
       print(raw)

sf.sailor_test(reward_map, Q, number_of_episodes)
sf.draw(reward_map, Q)
```

1) map_small

Tablice V w końcowych iteracjach algorytmu wraz z obliczonymi średnimi sumami nagród w danych iteracjach:

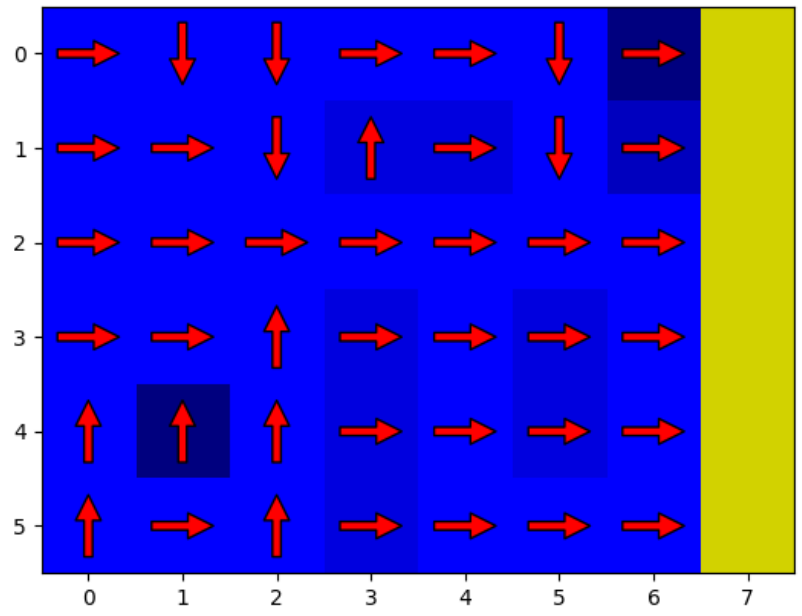
```
test-1000 mean sum of rewards = -8.245299999999999
[-0.038      -0.020444   -0.01079185  9.98058991  0.        ]
[-3.56440000e-02 -5.23883200e-03 -2.20177077e-03  1.18962284e+01
  0.00000000e+00]
[-0.04477236 -0.03689673 -0.01725535  2.2254277   0.        ]
test-1000 mean sum of rewards = 1.8411
[-0.05542908 -0.02699889  3.7718452  14.19284024  0.        ]
[-0.05431518 -0.02064912  3.68586687 13.59779886  0.        ]
[-0.06234064 -0.04704931  1.80356981  3.23473831  0.        ]
test-1000 mean sum of rewards = 1.8085
[-0.06376956  1.40481593  6.68822877 14.8855999   0.        ]
[-0.06423602  1.64901261  5.75760668 14.15333354  0.        ]
[-0.06808358  0.96188175  3.12765437  3.84344203  0.        ]
test-1000 mean sum of rewards = 2.0371000000000006

Process finished with exit code 0
```



2) map_middle

Tablice V w końcowych iteracjach algorytmu wraz z obliczonymi średnimi sumami nagród w danych iteracjach:

```
test-1000 mean sum of rewards = 4.586200000000001
[-0.03326477 -0.01937252 -0.01609639 -0.02674063  0.69523739  4.04768916
 15.94508482  0.        ]
[-1.44923636e-02 -5.54925258e-03 -3.12872000e-03 -2.61145603e-03
  1.26261003e+00  5.79876597e+00  1.92838675e+01  0.00000000e+00]
[-1.29380690e-02 -2.26122704e-03 -1.09775400e-03 -6.85275541e-04
  2.22505946e+00  9.83251039e+00  2.01214484e+01  0.00000000e+00]
[-2.13958969e-02 -6.44336186e-03 -2.86750692e-03 -1.71561024e-03
  1.92353506e+00  1.06699905e+01  2.03190648e+01  0.00000000e+00]
[-4.73471890e-02 -2.07736995e-02 -1.15752128e-02 -1.74778604e-02
  1.81774330e+00  1.08254133e+01  1.89120402e+01  0.00000000e+00]
[-0.06425827 -0.05129237 -0.03401553 -0.04912037  2.48906611  7.27282652
 13.83545504  0.        ]
test-1000 mean sum of rewards = 4.6975
[-0.03511795 -0.02097737 -0.01724389  0.24364182  1.9601929   5.89968316
 16.80027064  0.        ]
[-1.70059452e-02 -6.59417894e-03 -3.82380029e-03  3.31146018e-01
  3.34952160e+00  8.77001312e+00  2.04068473e+01  0.00000000e+00]
[-1.52077926e-02 -4.45156640e-03 -2.13677365e-03  9.07772403e-01
  5.06510739e+00  1.31540637e+01  2.13262756e+01  0.00000000e+00]
[-2.53306967e-02 -1.00026219e-02 -4.99869472e-03  8.97252700e-01
  4.67273062e+00  1.41003718e+01  2.10557242e+01  0.00000000e+00]
[-5.02144820e-02 -2.51227495e-02 -1.51048678e-02  8.43847315e-01
  4.88480268e+00  1.25385751e+01  1.92087791e+01  0.00000000e+00]
[-0.06682705 -0.05443865 -0.03749074  1.07055146  3.69826754  7.74613635
 13.90982121  0.        ]
test-1000 mean sum of rewards = 4.0466999999999995

Process finished with exit code 0
```



3) map_spiral

Obliczona średnia suma nagród w ostatniej iteracji:

```
test-1000 mean sum of rewards = 157.2202

Process finished with exit code 0
```