SINCE 1947
SEO KYEONG
UNIVERSITY

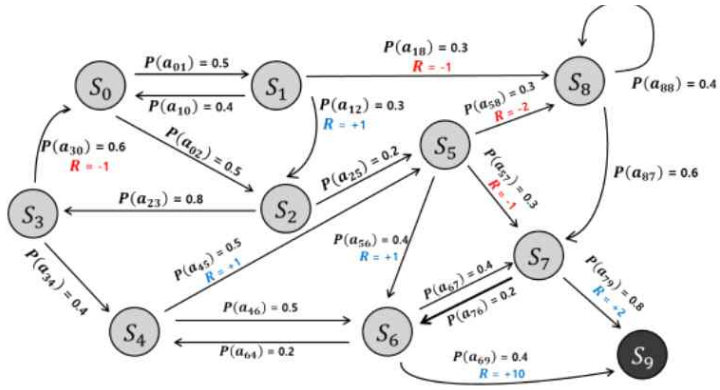
[지능시스템
2차 과제]

과목명	지능 시스템
담당 교수님	조영완 교수님

전공	컴퓨터공학과
학번	2018305068
이름	정동기

제출일	2021.04.09

2차 과제: 정책 이터레이션을 이용하여 가치함수 및 최적 정책을 구하는 프로그램을 작성하시오.



위에 있는 다이아그램을 이용하여 구하는 문제이다.
문제를 풀기 전 알고리즘은 3주차 ppt 17쪽, 22쪽에 있는 알고리즘을 참고하였다.

Iterative policy evaluation

Input π, the policy to be evaluated
Initialize an array $V(s) = 0$, for all $s \in \mathcal{S}^+$
Repeat
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ (a small positive number)
Output $V \approx v_\pi$

17쪽 반복 정책 평가 알고리즘

Policy iteration (using iterative policy evaluation)

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Repeat
     $\Delta \leftarrow 0$
     For each $s \in \mathcal{S}$:
       $v \leftarrow V(s)$
       $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))\left[r + \gamma V(s')\right]$
       $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number)

   과제 알고리즘과 같음

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
     old-action $\leftarrow \pi(s)$
     $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)\left[r + \gamma V(s')\right]$
     If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

22쪽 정책 이터레이션 알고리즘

위의 알고리즘을 보고 내가 이해한 대로 해석을 해보았다.

one array 방식을 이용.

1. 임의로 가치함수와 정책을 지정한다.

2. 정책 평가
모든 상태에 대해서 가치함수 array 값: 0
세타값: 임의의 값
델타값: 0
모든 상태에 대해서 현재 기댓값을 스칼라 변수에 저장을 한다.
벨만 기대 방정식을 이용하여 가치함수를 업데이트한다.
이후 이전값에서 현재값의 차이가 델타값보다 더 크면 델타에 갱신한다.
어떤 정해진 세타 보다 델타가 작으면 끝난다.

3. 정책 발전
정책이 바뀌지 않았을 때: True
모든 상태에 대해서
old action에 이전 정책값을 넣어 놓는다.
시작지점이 같은 큐함수들을 구하고 그 중 가장 큰 값을 구한다.
가장 큰 큐함수를 가지고 있는 정책을 1로 바꾼다. 만약 가장 큰 함수가 여러개라면 1/n값을 여러개에 다 넣어준다.
새로운 정책과 이전 정책을 비교하여 값이 같지 않으면 False로 바꾼다.
만약 새로운 정책과 이전 정책이 같아 True값이라면 거기서 프로그램을 종료하고, False값이라면 2. 정책 평가 단계로 다시 돌아간다.
true가 나오면 최적의 정책이다.

위에 문제의 다이아그램을 보면 움직이는 방향이 한 방향이기 때문에 상태변환 확률이 1이다.

위에 짠 알고리즘을 이용하여 파이썬 언어로 프로그램을 짜주었다.

*코드를 짤 때 list의 이름을 array라고 두었는데 이름만 array고 형식은 list이다.

```python
# 1번
# array = [출발, 목적, 정책, 보상]
array = [[0, 1, 0.5, 0], [0, 2, 0.5, 0],
         [1, 0, 0.4, 0], [1, 2, 0.3, 1], [1, 8, 0.3, -1],
         [2, 3, 0.8, 0], [2, 5, 0.2, 0],
         [3, 0, 0.6, -1], [3, 4, 0.4, 0],
         [4, 5, 0.5, 1], [4, 6, 0.5, 0],
         [5, 6, 0.4, 1], [5, 7, 0.3, -1], [5, 8, 0.3, -2],
         [6, 4, 0.2, 0], [6, 7, 0.4, 0], [6, 9, 0.4, 10],
         [7, 6, 0.2, 0], [7, 9, 0.8, 2],
         [8, 7, 0.6, 0], [8, 8, 0.4, 0]]
# 2번
Value_array = [0, 0, 0, 0, 0, 0, 0, 0, 0]  # 가치함수 0으로 초기화

seta = 0.0001  # 임의의 값 지정
number = 1  # 몇번 돌았는지
while True:
    while True:  # Repeat
        delta = 0
        for s in range(9):  # For each s
            v = Value_array[s]  # c <- V(s)
            for i in range(9):  # 벨만 기대 방정식
                Value = 0
                for j in range(len(array)):
                    if array[j][0] == i:
                        Value += array[j][2] * (array[j][3] + 0.9 * Value_array[array[j][1]])
                Value_array[i] = Value
            delta = max(delta, (v - Value_array[s]))  # delta max
        if delta < seta:  # until
            break
    # 3번
    policy_stable = True  # 정책이 바뀌지 않으면
    for s in range(9):
        list1 = []  # 큐함수 값
        list2 = []  # array의 위치 저장
        old_action = []  # 원래있던 정책 값
        pie = 1  # 정책값
        count = 1  # 나중에 정책값 나눌 때 사용
        for j in range(len(array)):
            if array[j][0] == s:  # 시작지점에서의 큐함수 구하기
                old_action.append(array[j][2])
                list1.append((array[j][3] + 0.9 * Value_array[array[j][1]]))  # 큐함수
                list2.append(j)  # array의 위치 저장
        MAX = max(list1)  # 리스트의 최댓값을 얻음
        for i in range(len(list1)):  # 리스트 크기만큼 비교
            if max == list1[i]:  # 만약 최대값과 큐함수 값이 같다면
                count += 1  # 카운트를 1 올림, 마지막에 정책을 나누어줄 때 사용
        pie = pie / count  # 정책값 1을 최댓값이 몇개 더 있다면 그 수만큼 나누어줌
        for i in range(len(list1)):
            if list1[i] == MAX:  # 큐함수가 최대값이라면
                array[list2[i]][2] = pie  # 정책값을 갱신해줌
            else:
                array[list2[i]][2] = 0
        for i in range(len(list1)):  # 정책이 바뀌었다면 False
            if old_action[i] != array[list2[i]][2]:
                policy_stable = False
    # 결과값 출력
    print(number, "번째 기댓값과 정책값")
    for i in range(10):
        print("V(", i, "):", Value_array[i])
    for i in range(len(array)):
        print(array[i][0], "에서", array[i][1], ":", array[i][2])

    if policy_stable == True:  # 정책이 바뀌지 않는다면 나간다.
        print(number, "번 돌았습니다.")
        break

    number += 1  # 횟수 더해주기

print("최적의 경로")
root = 0  # 경로의 목적지를 저장
for i in range(len(array)):
    if array[i][2] != 0:
        if i == 0:
            print("[", array[i][0], "->", array[i][1], "]", end=' ')
            root = array[i][1]
        if root == array[i][0]:  #목적지와 출발지가 같을 때
            root = array[i][1]
            print("[", array[i][0], "->", array[i][1], "]", end=' ')
```

위의 프로그램을 돌린 후 결과

```
1 번째 기댓값과 정책값
V( 0 ): 1.4209378711297773
V( 1 ): 1.560162191185685
V( 2 ): 1.730710217239784
V( 3 ): 1.7411164961225871
V( 4 ): 4.373506694881403
V( 5 ): 2.874410683605988
V( 6 ): 5.7348216979671065
V( 7 ): 2.6322679056340794
V( 8 ): 2.2205262514178283
V( 9 ): 0
0 에서 1 : 0
0 에서 2 : 1.0
1 에서 0 : 0
1 에서 2 : 1.0
1 에서 8 : 0
2 에서 3 : 0
2 에서 5 : 1.0
3 에서 0 : 0
3 에서 4 : 1.0
4 에서 5 : 0
4 에서 6 : 1.0
5 에서 6 : 1.0
5 에서 7 : 0
5 에서 8 : 0
6 에서 4 : 0
6 에서 7 : 0
6 에서 9 : 1.0
7 에서 6 : 1.0
7 에서 9 : 0
8 에서 7 : 1.0
8 에서 8 : 0
```

```
2 번째 기댓값과 정책값
V( 0 ): 8.1
V( 1 ): 9.1
V( 2 ): 9.0
V( 3 ): 8.1
V( 4 ): 9.0
V( 5 ): 10.0
V( 6 ): 10.0
V( 7 ): 9.0
V( 8 ): 8.1
V( 9 ): 0
0 에서 1 : 1.0
0 에서 2 : 0
1 에서 0 : 0
1 에서 2 : 1.0
1 에서 8 : 0
2 에서 3 : 0
2 에서 5 : 1.0
3 에서 0 : 0
3 에서 4 : 1.0
4 에서 5 : 1.0
4 에서 6 : 0
5 에서 6 : 1.0
5 에서 7 : 0
5 에서 8 : 0
6 에서 4 : 0
6 에서 7 : 0
6 에서 9 : 1.0
7 에서 6 : 1.0
7 에서 9 : 0
8 에서 7 : 1.0
8 에서 8 : 0
```

```
3 번째 기댓값과 정책값
V( 0 ): 8.19
V( 1 ): 9.1
V( 2 ): 9.0
V( 3 ): 9.0
V( 4 ): 10.0
V( 5 ): 10.0
V( 6 ): 10.0
V( 7 ): 9.0
V( 8 ): 8.1
V( 9 ): 0
0 에서 1 : 1.0
0 에서 2 : 0
1 에서 0 : 0
1 에서 2 : 1.0
1 에서 8 : 0
2 에서 3 : 0
2 에서 5 : 1.0
3 에서 0 : 0
3 에서 4 : 1.0
4 에서 5 : 1.0
4 에서 6 : 0
5 에서 6 : 1.0
5 에서 7 : 0
5 에서 8 : 0
6 에서 4 : 0
6 에서 7 : 0
6 에서 9 : 1.0
7 에서 6 : 1.0
7 에서 9 : 0
8 에서 7 : 1.0
8 에서 8 : 0
3 번 돌았습니다.
최적의 경로
[ 0 -> 1 ] [ 1 -> 2 ] [ 2 -> 5 ] [ 5 -> 6 ] [ 6 -> 9 ]
```