Planning the Pacman problem with Markov decision processes

Li Siyao

Abstract

MDP is a classical planning algorithm which is widely used to make decision for the agent. This report is divided into two parts. The first part introduced the background of Markov decision process, then described my train of thought to write the code, which helps me to solve the Pacman problem under the strategy of MDP. The second part shows resulting performance of my code and made hypothesis about the relationship between value of ghosts and the performance of Pacman. At the end, I designed experiment to prove my hypothesis, and the results show that the value of ghost is inversely proportional to the rate to win the Pacman game.

Part I: Introduction and description

MDP theory

A sequential decision problem for a fully observable stochastic environment with a Markovian transition model and additive rewards is calls a Markov decision process, or MDP (Russell et al.,1995). MDP is consist with the following components: a set of states $s \in S$; a set of actions: $x \in X$; a state transition function: T; and a reward, $R(s,x)$, for executing action x in states. At each time step, the decision-maker observes the state of the whole system and selects an optimal action. The state and action choice produce two results: the decision-maker receives a reward and the system transitions from one stage to the next.

The algorithm proceeds as follows (Schapaugh and Tyre 2013):

- Set $t = T + 1$ and $V^{T+1}(s) = f(s)$ for all $s \in S_{T+1}$ (terminal value is a function of the state)
- Substitute $t-1$ for t and compute $V(t,T,s)$ for each $s \in S$ using (Bellman 1957)
- $V(t,T,s) = \max_{x\in X}\{R(s,x) + \sum_{s'} \Pr(s' \mid s,x)V(t+1,T,s')\}$
- If $t = 1$, stop, otherwise return to step (2).

Description

The coursework of MDPAgent is to use Markov decision process to guide Pacman to win as much as possible. Pacman can grasp the information of objects' positions under two kinds of layout, which includes 'small' and 'medium'.

Primary functions and flow chart

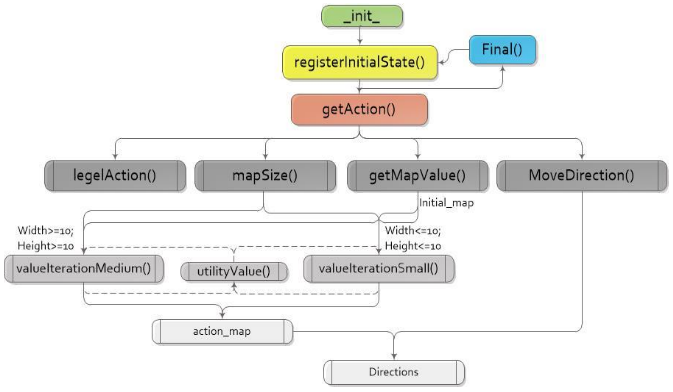


Figure 1. Flow chart of MDP process

Before the description of my code, figure 1 shows the whole train of thought of me to solve this problem.

Bellman function and implementation

Bellman equation is the significant component of MDP, and it consist of three parts:

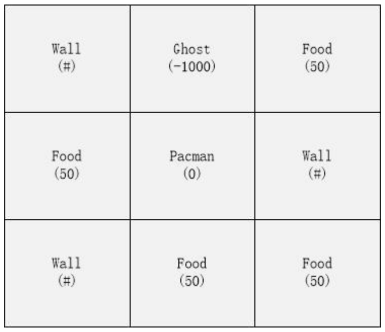
$$U(s) = R(s) + \gamma \max_{a\in A(s)} \sum_{s'} P(s' \mid s,a)U(s')$$

Reward and discount factor

$R(s)$ and γ are two constants, I set these two values under the different circumstances, which includes smallGrid and mediumClassic. When we start the game under smallGrid layout, reward is equal to -1, which means that the agent does not like the environment and try to finish the game by eating food and capsule whose value is larger than 0. The discount factor is equal to 0.95, which make Pacman regard future as important. In another occasion, we play Pacman in the mediumClassic layout. Reward is equal to -1 and the discount factor is set to 0.9, respectively.

Initial value of map

In order to get the utility value of next state in the Bellman equation, I use the getValueMap() function to set initial value of map. In the first map, I set the different objects in the map to different value. Among them, value of foods and capsules is equal to 50, value of ghost is set to -1000. Besides, the value of walls is represented by #. The rest of the normal places' value is set to 0. In this way I formed the initial configuration of the map.



Wall (#)	Ghost (-1000)	Food (50)
Food (50)	Pacman (0)	Wall (#)
Wall (#)	Food (50)	Food (50)

Figure 2. value map example

Utility calculation

The last important part of the Bellman equation is to calculate the maximum value of utility at the certain point. I use utilityValue() function to realize this. In this function, I suppose the action of Pacman is non-deterministic. Therefore, I assume Pacman have one target direction to go which possibility is 0.7.

Besides, Pacman may not follow the instruction to the target places, which leads the outcome that Pacman have 0.1 possibility to go to the other three directions. For example, if next step Pacman is guided to go north, there are 0.7 possibility that Pacman go north, but there still exist 0.1 possibility Pacman go south, 0.1 possibility Pacman go east and 0.1 possibility Pacman go west.

Then multiply the possibility of corresponding direction and the value of aim position. Next, sum all this utility together. Finally, the maximum result is regard as utility of current point.

Utility iteration

The three important components in the Bellman equation is realized in the functions above. Then need to iterate the utility value to get the final value map, which is consist of all the iterated utilities on the map. At this time, iteration makes all the utilities on the map reach to a stable status. I divided iteration process into two type, which is determined by the layout of the map.

When layout is smallGrid, use valueIterationSmall() function to iterate values in map. In this function, as I said earlier, the values of gamma and reward are set to 0.95 and -1 respectively. The number of iterations is set to 100.

I consider the positions which is three steps from ghosts as dangerous places, which values is set proportionally to the distance from this position to ghost. In order to let Pacman get away from these dangerous places as far as possible, finally, value of dangerous places is set to $-700 + 15 * i$ (i is the distance to the ghost, and it range from 1 to 3). In this small map, the range of dangerous positions is a radial range from ghost centered to three steps outward in eight directions.

Besides, I use wall mark to indicate whether there's a wall exist in this direction. Because when there stands a wall between Pacman and ghost, this current position is not a dangerous place anymore (wall defends Pacman from the ghosts). Then I compute the utility of positions which not includes safety foods and capsule positions and walls and ghost positions use Bellman equation. In this process, I get the value of target place by the map which calculate by iteration function. Therefore, iteration function and utility calculate function are mutually restrictive.

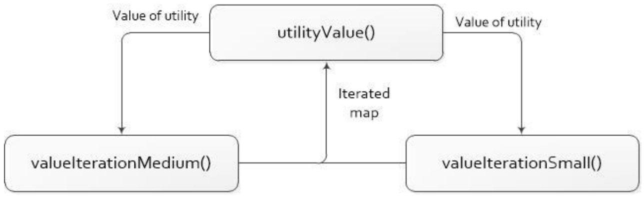


Figure 3. Relationship between utility and Iteration

When layout is mediumClassic, the configuration of valueIterationMedium() is similar to valueIterationSmall().For the reason that the medium map is more intricate and complex, in which the code takes a long time to run , the number of iterations is set to 50. Value of dangerous places is set to $-70 + 10 * i$ (i is the distance to the ghost, and it range from 1 to 3).

However, medium layout has more complex context than small one. Therefore, in the valueIterationMedium(), the range of dangerous positions is a seven times seven square around ghost.

Make decision and get action

After iteration the function return the final map which can guide Pacman to make decisions. In this procedure, I use moveDirection(), this function needs the current coordinate of Pacman, then output the direction which has the max value around to Pacman. Next, execute the getAction() function, which guide Pacman to make move.

Part II Result and data analysis

Result

According to the instructions which are given in smallGrid and mediumClassic:

"python pacman.py -q -n 25 -p MDPAgent -l smallGrid"

"python pacman.py -q -n 25 -p MDPAgent -l med-umClassic"

These two orders mean that the MDP agent that I wrote the would be run 25 times under the two types of layout, which includes small grid and medium grid. I took 10 samples of Pacman running 25 times in small map and medium map respectively.

The result of Pacman running 25 times in small map is shown below with γ=0.95:

NO.	Number of wins	Wining rate	Average score
1	19	0.76	258.88
2	17	0.68	177.24
3	17	0.68	177.60
4	15	0.60	97.68
5	21	0.84	336.16
6	19	0.76	258.16
7	15	0.60	97.40
8	18	0.72	217.88
9	16	0.64	137.68
10	21	0.84	339.08

Table 1. Results of small grid

The average of win is 17.8, the average of winning rate is 0.71, the average score of the Pacman game is 212.78.

The result of Pacman running 25 times in medium map is shown below with γ=0.90:

NO.	Number of wins	Wining rate	Average score
1	13	0.52	633.92
2	13	0.52	696.64
3	15	0.60	741.16
4	11	0.44	538.00
5	18	0.72	959.72
6	18	0.72	914.00
7	12	0.48	617.76
8	13	0.52	686.84
9	11	0.44	505.20
10	13	0.53	665.88

Table 2. Results of medium grid

The average of win is 13.7, the average of winning rate is 0.55, the average score of the Pacman game is 695.91.

Hypothesis

When I test the performance of my code, I find out there may exist some relationship between the value of ghost and the performance of Pacman. I suppose maybe when I set lower value to ghost, for the reason of the low reward, Pacman would stay away from ghost.

Therefore, I made a hypothesis that the possibility to win the game is inversely proportional to the value of ghost.

Experiment and outcome

In order to prove my hypothesis, I design to set the value of ghost to three type (control the other parameters are unchanged). I give Ghost a value of – 10, -100, -1000, to test whether it would affect the performance of Pacman.

I took 10 samples of Pacman running 25 times in small map, When the value of ghost is -10, -100, -1000 respectively. Result shows that when ghost value is equal to -10, the average winning rate is 13.6, which means Pacman wins 13.6 games per 25 games, the standard deviation is 3.06. When the ghost value is equal to -100, the average rate is 17.3, which means Pacman wins 17.3 games per 25 games, the standard deviation is 3.02. When the ghost value is equal to -1000, the average rate is 17.8, which means Pacman wins 17.8 games per 25 games, the standard deviation is 2.20.

Ghost value	Average win times (in 25 games)	deviation
-10	13.6	3.06
-100	17.3	3.02
-1000	17.8	2.20

Table 3. Win rate under different ghost value

The experiment result shows that the lower the ghost value, the higher the winning rate of Pacman and the more stable performance. When ghost value is equal to -10, the winning rate dropped significantly, and the standard deviation is much higher which means the win rate fluctuates greatly. Therefore, I conclude that the value of ghost is inversely proportional to the rate to win the Pacman game.

References

Russell, S.J., Norvig, P., Canny, J. F., Malik, J. M., & Edwards, D. D. (1995). *Artificial intelligence: a modern approach, vol. 2*. Englewood Cliffs: Prentice Hall.

Bellman, R., (1957). Dynamic Programming. Princeton University Press, Princeton, New Jersey

Schapaugh, A. W., & Tyre, A. J. (2013). Accounting for parametric uncertainty in Markov decision processes. *Ecological modelling*, *254*, 15-21.