Final

Bingyu Xin

December 2020

1 Sheep Dog Bot

1) How many possible sheep/dog configurations might the field be in?

   We first place the two dogs in the board, and then place the sheep in the rest space. The possible configurations $= C_{49}^2 * C_{47}^1 = 55272$

2) For a given field configuration C (position of dogs, position of sheep), formulate a mathematical description for T(C), the minimal number expected rounds needed to corner the sheep. Hint: For what configurations C is T(C) easy to compute?

   For sheep (i,j), When two dogs are on the top and left neighbor of the sheep, the sheep can only move down or move right; After the sheep moving, if we still move the dogs to the top and left neighbor of the moved sheep, so the sheep can directly arrive to the right corner of the field. The expected rounds will be the Manhattan distance between (i,j) and destination (6,6): $12 - i - j$. In this configuration, we can take the minimal rounds to the corner.

   For a given field configuration C, we assume the dogs positions are (a,b),(c,d), sheep position is (i,j). We can divide the corner process into two parts, the first part is two dogs move to the left and top of the sheep, the second part is the process we talked above. In the first part, since the sheep take random moves, we can assume that the average position of the sheep is (i,j) in the first part, so the rounds in this part is the sum of Manhattan distance between (i,j) and two dogs (a,b), (c,d): $abs(a-i) + abs(b-j) + abs(c-i) + abs(d-j)$. The rounds needed for the second part is 12-i-j as we discussed above.

   So, the minimal number expected rounds needed to corner the sheep,

   $$T(C) = abs(a-i) + abs(b-j) + abs(c-i) + abs(d-j) + 12 - i - j$$

   where (a,b),(c,d) are the position of dogs, (i,j) is the position of sheep.

3) If you could calculate T(C) for any configuration C, show that you could construct an optimal dog-controlling algorithm for cornering the sheep.

As described in Q2, the algorithm will be two step, the first step is moving the dogs to the left and top of the sheep, since the sheep is moving randomly, the destination of dogs at each round is changing; the second step is straight forward, since dogs have catching the sheep, they will directly go to the corner.

4) Write an algorithm and code to solve for the function T. Is T always finite? What does it mean when T is infinite?

   When sheep and dogs are in the other three corners, the sheep has no available square to move, so it has to stay in place and the dogs can never move to the left and top of the sheep, in these situation, T will be infinite.

5) Given the initial configuration in the above example, how many rounds do you need (on average) to corner the sheep?

   We run the code for 1000 times and get the average rounds to corner the sheep is 58.

6) What is the worst possible initial state (position of dog bots, sheep), and why? Justify mathematically.

   The worst possible initial state is when sheep is at the left top corner (0,0) and the dogs are at (0,1) and (1,0). The sheep will be stuck at (0,0) and can not move anywhere in our algorithm. The round for this situation is infinite as described in Q4.

   If we exclude the infinite situation, then the worst situation will be when sheep is at (0,0) and dogs are at (5,6) and (6,6). In this situation, the minimal round for the first step and the second step will be max, because the corresponding Manhattan distance is at max.

7) You are allowed to place your dog bots anywhere in the field at the start, then the sheep will be placed uniformly at random in one of the remaining unoccupied cells. Where should you place your dog bots initially, and why? Justify mathematically.

   I would place the dog bots at (3,2) and (3,4), because the average Manhattan distance between these points to possible sheep position will be minimal.

8) Do you think better strategies exist than the one you came up with? Justify.

   Better strategies exist. Firstly, we can avoid the infinite situations, that is , we can easily change the situation to the catching state (dogs at the top and left neighbor of the sheep).Take the sheep at (0,0) and dogs at (0,1) and (1,0) as an example, we can move one dog from (0,1) to (0,1), then the sheep can only move to (1,0); at next round, we move another dog from (1,0) to (0,0), then the sheep can only move to (1,1); at next round, we move the dogs to (0,1) and (1,0), and thus reaching to the catching state.

Another potential improvement to our strategy is that, two dogs can work cooperatively to catch the sheep rather than independently reaching to the top and left neighbor of the sheep, this improvement can reduce the rounds for the algorithm.

2 Bot Negotiations

a) For each of the 10 states, what is the optimal utility (long term expected discounted value) available in that state (i.e., $U^\star(state)$)?

According to the description, we formulate the equation of utility below:

$$U(New) = 101 + \beta * U(Used_1) \tag{1}$$

$$\begin{aligned} U(Used_i) &= max\{100 - 10 * i + \beta * [0.1 * i * U(Used_{i+1}) \\ &+ (1 - 0.1 * i) * U(Used_i)], \\ &- 255 + \beta * U(New)\}, i = 1 \sim 7 \end{aligned} \tag{2}$$

$$\begin{aligned} U(Used_8) &= max\{20 + \beta * [0.8 * U(Dead) + 0.2 * U(Used_8)], \\ &- 255 + \beta * U(New)\} \end{aligned} \tag{3}$$

$$U(Dead) = -255 + \beta * U(New) \tag{4}$$

We use Value Iteration to solve the equations, so we get : $U^\star(New) = 801$, $U^\star(Used_1) = 778$, $U^\star(Used_2) = 642$, $U^\star(Used_3) = 554$, $U^\star(Used_4) = 500$, $U^\star(Used_5) = 472$, $U^\star(Used_6) = 466$, $U^\star(Used_7) = 466$, $U^\star(Used_8) = 466$, $U^\star(Dead) = 466$.

b) What is the optimal policy that gives you this optimal utility - i.e., in each state, what is the best action to take in that state?

Since we have got $U^\star$ in (a), so we can compare the utility of each action. For New and Dead, they only have one action. For other state, we can get the table as Table. 1:

Table 1: Utility of each action at different state when $\beta = 0.9$

method	$Used_1$	$Used_2$	$Used_3$	$Used_4$	$Used_5$	$Used_6$	$Used_7$	$Used_8$
USE	778	642	554	500	472	459	449	439
REPLACE	466	466	466	466	466	466	466	466

So the optimal policy is that we will take action USE when in state $\{Used_1, Used_2, Used_3, Used_4, Used_5\}$, and take action REPLACE when in state $\{Used_6, Used_7, Used_8\}$

c) What is the highest price for which this used machine option would be the rational choice? i.e., what price should MachineSellingBot be selling this option at?

We need to add a new action SELL according to the question, we assume the selling price is x, so the equation of utility (2)(3)(4) will be changed as below:

$$\begin{aligned} U(Used_i) &= max\{100 - 10 * i + \beta * [0.1 * i * U(Used_{i+1}) \\ &+ (1 - 0.1 * i) * U(Used_i)], -255 + \beta * U(New), \\ &- x + \beta * (0.5 * U(Used_0) + 0.5 * U(Used_1))\} \end{aligned} \tag{5}$$

$$\begin{aligned} U(Used_8) &= max\{20 + \beta * [0.8 * U(Dead) + 0.2 * U(Used_8)], \\ &- 255 + \beta * U(New), -x + \beta * (0.5 * U(Used_0) \\ &+ 0.5 * U(Used_1))\} \end{aligned} \tag{6}$$

$$\begin{aligned} U(Dead) &= max\{-255 + \beta * U(New), \\ &- x + \beta * (0.5 * U(Used_0) + 0.5 * U(Used_1))\} \end{aligned} \tag{7}$$

We set x=0 at start, by solving the MDP, we can estimate whether the utility of action SELL is larger than the utility of action REPLACE when in state DEAD. Obviously, the answer is larger when x=0, because when SELL option is free, we will never buy a new machine; Then we iteratively check this by add 1 to x each time, until the utility of action SELL is smaller than the utility of action REPLACE when in state DEAD, which means we arrive at the price x where we prefer to choose REPLACE than SELL in state DEAD. so we get the highest price for this option, x-1.

By implementing the algorithm, we find this price is 172.

d) For different values of (such that 0 ¡ ¡ 1), the utility or value of being in certain states will change. However, the optimal policy may not. Compare the optimal policy for = 0.1, 0.3, 0.5, 0.7, 0.9, 0.99, etc. Is there a policy that is optimal for all sufficiently large ? Does this policy make sense? Explain.

We use the code in (a)(b) to generate optimal policy, When β =0.1,0.3,0.5,0.7, we will take action USE in all Used states; When $\beta = 0.9$,, we will take action USE when in state $\{Used_1, Used_2, Used_3, Used_4, Used_5\}$, and take action REPLACE when in state $\{Used_6, Used_7, Used_8\}$; When $\beta = 0.99$, we will take action REPLACE in all Used states.

We find that when β is sufficiently large, the optimal policy will always be take action REPLACE in all Used states. This doesn't make sense because it is unreasonable to replace the machine only using once. Mathematically, using value iteration method, in kth iteration, the error will be:

$$\max_{s \in S} |U_k^\star(s) - U^\star(s)| \leq \beta^k \max_{s \in S} |U_0^\star(s) - U^\star(s)|$$

When β is close to 1, the error can not be contracted, so we get wrong answer.