S21 - MDL Assignment 2 part 3

Team 7
Rutvij Menavlikar (2019111032)
Tejas Chaudhari (2019111013)

April 10, 2021

Linear Programming

Linear programming involves maximizing or minimizing a linear function subject to certain constraints. We can use linear programming to solve MDPs and create an optimized policy based on the solutions of the linear programming model.

Let
n_a be the total number of actions across all states
n_s be the total number of states

Define the tuple $<A, R, x, a>$, where
$A_{n_s \times n_a}$ is the matrix that stores the probabilities of moving in or out of a given state
$R_{1 \times n_a}$ is the reward for taking an action from a state
$x_{n_a \times 1}$ is the number of times a particular action has to be taken from a state s
$a_{n_s \times 1}$ is the probability distribution of each state in the starting of the MDP.

Solving MDP with LP

To solve a MDP and obtain the optimum policy π^*, the LP should pick the best action possible from the current state. To do this we (In this case) maximize Rx, the final reward value, calculated by multiplying the rewards with the number of actions taken with the constraints:

- $Ax = a$, because the product of A and x should match the initial probability distribution of each state in the start of the MDP (a)
- $x_{i1} \geq 0 \ \forall i \in \{1, 2, 3..., n_a\}$, since x denotes the number of times an action was taken, every value in x is a non-negative integer

---

The Matrix A

The matrix A contains the probabilities of moving in or out of a given state. In an entry A_{ij}, i is an index corresponding to a certain state, and j is an index corresponding to a certain **(state,action)** pair. So, we can calculate A_{ij}:

- For every state s, let p be the probability of obtaining s when performing given action in the state and let a be the index of s
- Then we simply do the following

$$A_{ij} += p$$
$$A_{aj} -= p$$

  Effectively, we are storing the difference of outgoing and incoming probabilities for every **(state,action)** pair

---

Finding the optimal Policy

- x stores the number of times a particular actions has to be taken from a state s for every **(state,action)** pair. So in a solution of x where Rx is maximized, a higher value of any element of x would mean that it corresponds to a better final reward/value of Rx

- To find the optimal policy, we need to find the best action for each possible state
- To find the best action for a state s, consider the set of positions S in x which correspond to actions of s. We find an action a such that the value in x corresponding to **(s,a)** is the maximum among all the values for the positions in S. Then the action a is the best action for state s

The Policy

- In the **West Square**:
  - When *MM* is in dormant state, if *IJ* has sufficient arrows, then he chooses to *SHOOT* in some cases, otherwise *IJ* chooses to *STAY* or move *RIGHT* based on *MM*'s health
  - When *MM* is in ready state, *IJ* chooses to *STAY* in the same location when he has less arrows, and *SHOOT* when he has sufficient arrows. He chooses to go *RIGHT* very rarely
- In the **North Square**:
  - When *MM* is in dormant state, *IJ* moves *DOWN* if he has no materials and has arrows or if he has sufficient arrows. Otherwise, he mostly uses *CRAFT* when he has materials and uses *STAY* if he does not have materials
  - When *MM* is in ready state, *J* mainly uses *CRAFT* if he has materials and less than three arrows, otherwise he uses *STAY*
- In the **East Square**:
  - When *MM* is in dormant state, *IJ* always attacks. Depending on how many arrows *IJ* has, he chooses to attack with *SHOOT* or *HIT*
  - When *MM* is in ready state, *IJ* always attacks with the same conditions as in the dormant state, except *IJ* attacks with *HIT* irrespective of the number of arrow he has when *MM* has full health
- In the **South Square**:
  - When *MM* is in dormant state, *IJ* chooses to go *UP* mainly when he has sufficient arrows or *MM* has high health. Otherwise, if he has less than 2 materials, he uses *GATHER* depending on the number of arrows he has, and in the other cases uses *STAY*
  - When *MM* is in ready state, *IJ* usally chooses to *GATHER* when *MM* does not have full health and chooses *STAY* in all other cases.
- In the **Center Square**:
  - When *MM* is in dormant state, *IJ* attacks depending on how many materials he has. If he has no materials, depending on the number of arrows and *MM*'s health, he chooses to *SHOOT*, *HIT* or go *RIGHT* to attack with a better chance. If he has materials, he might choose to go *UP* to *CRAFT* some arrows
  - When *MM* is in ready state, if *IJ* is low on materials, he goes *DOWN*, if he is low on arrows and has materials, he goes *UP*. Otherwise depending on *MM*'s health, he attacks with *SHOOT* or goes *LEFT* to get away from *MM*'s attack zone

---

Multiple Policies

There are multiple policies *possible* for the problem.

- In the method described to find the policy, if for a state s, S denotes the set of all positions in x which correspond to actions on s, then it is possible that the maximum of values of x for positions in s is shared by more than one positions. In our code, we choose the first position in S whose value in x is the maximum among all values of the positions in S. If we randomize or pick any other position except the first maximum one, we obtain a different policy
- Changing the matrix A can change the policy as well. Matrix A stores the probabilities of moving in or out of a given state, and changing them would change state utilities, reward matrix R, and variable matrix x
- The policy starting space a also affects the choice of the policy at it changes the constraints on x
- Finally, changing the reward matrix R would change the condition of maximizing Rx and so it too changes the policy