RS-HL-5: MDP-DP Simulation

UT Austin Oden Institute

Hongseok Kim

06/22/2024

Scope

- Designed Markov Decision Process (MDP) framework using Dynamic Programming (DP) in satellite network control domain.
- For the simplicity of the problem formulation, we have supposed single ground station to single ground station only.
- Both utilized value iteration method and policy iteration method.

I. MDP Formulation

For creating MDP framework environment, we need 5 necessary variable spaces $M = (S, A, P, R, \gamma)$ and policy π. We are simulating the network structure, transmitting one data packet from Ground Station m to Ground station n by using satellite network (SAT1 - SAT48)

1. S : Finite **Set** of States - Location of the data packet. Total 78 States (30 GS + 48 SATS)
2. A : Finite **Set** of Actions - Ordering the agents (GS + SAT) to mover the data packet from one place to another (GS -> SAT, SAT -> SAT, SAT -> GS)
3. P : State Trasmission Probability **Matrix**- Probability when the data is successfully transmitted by given action. (e.g. 0.8 indicates that failure of data transmittion in 20% occation)
4. R : Reward **Function** - Reward when the data is moved to certain agent. GS -> SAT / SAT -> SAT: minus reward. Only positive reward is when the data is transmitted to final destination SAT which can directly transmit to GS.
5. γ : Discount **Factor** - Discount factor of the reward in order to sum of series of rewards are converged.
6. $\pi(a \mid s)$: Policy probability - Wheter deterministic $\pi = 1$ for optimal and 0 for all other case or stocastic $\pi \sim (0, 1)$

Caution: The Satellites and Ground Stations themselves are not the states. Instead, they are agents and formulate the grid area. The phenomenons that the data is located at each agent are states.

II. Review of basic MDP Theory

We have to define $M = (S, A, P, R, \gamma)$ from given dataset. In order to do that, we have to understand the basic framework of the process of MDP simulation, include knowing know to define each variable.

II.1 State Transition Probability Matrix P using S, A

$P^{a}_{ss'} = p(S_{t+1} = s' | S_t = s, A_t = a)$

- Currently, each states may be connected to other state by network graph. The created network graph is composed by 0,1 - 0 indicates no connection and 1 indicates connection.
- If the probability is 0.8, the 0.2 probability should be returning to current state.
- The actions should be deciding the probabilty to transmite to other states (then where?) or staying in current state.
- There are different scale of action for different states because the number of satellites or ground stations connected to each agent should be different.

II.2 Policies π : Mapping from State to Actions

$\pi(a|s) = p(A_t = a \mid S_t = s)$

- Initially, we assume there is same possibility for all available actions, then converge over the **policy iteration** process.
- In value iteration process, we don't have to consider policy before all the state value functions are converged.
- Also, we have to choose between deterministic of stochastic policy

II.3 Value Function

State-value function $v_\pi(s)$: Expected return starting from state s, in the case following policy π after that.

- $v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$

- Another Expression [Sutton]

- $$v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right] = E_\pi\left[R_{t+1} + \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s\right]$$
  $$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r \mid s, a) \left[r + \gamma E_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_{t+1} = s'\right]\right]$$
  $$= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')] \rightarrow \text{Bellman Equation for } v_\pi$$
- It is really a sum over all values of the three variables, $a. s'$, and r,.
- For each triple, we compute its probability, $\pi(a|s)\, p(s', r \mid s, a)$, weight the quantity in brackets by that probability in brackets by that tprobability, then sum over all possibilities to get an expected value.

Action-value function $q_\pi(s, a)$: Expected return starting from state s and action a, in the case following policy π after that.

- $q_\pi(s,a) = E_\pi[G_t \mid S_t = s,\ A_t = a] = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}) \mid S_t = s,\ A_t = a\ ]$

II.4 Bellman Expectiation Equation

Relationship between $v_\pi(s)$ and $q_\pi(s)$ and **Reward function** $R(s,a)$

- $v_\pi(s) = \sum_{a \in A} \pi(a|s)\, q_\pi(s,a)$
- $q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_\pi(s')$
- $R_s^a = E[R_{t+1} \mid S_t = s, A_t = a]$

Bellman expection equation drtivation for $v_\pi(s)$ **and** $q_\pi(s,a)$

- For determininstic optimal policy,
- $v_\pi(s) = \sum_{a \in A} \pi(a|s)\, q_\pi(s,a) = \sum_{a \in A} \pi(a|s) \left[R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_\pi(s') \right]$
- $q_\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_\pi(s') = R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \left[\sum_{a' \in A} \pi(a'|s')\, q_\pi(s',a') \right]$

II.5 Bellman Optimality Equation

Optimal state-value function $v_*(s)$: Maximum value function for all policies

- $$v_*(s) = \max_\pi v_\pi(s) = \max_a q_*(s,a)$$
$$= \max_a \left(R(s,a) + \gamma \sum_{s' \in S} p(s' \mid s,a) v_*(s') \right)$$
$$= R(s,a) + \gamma \max_a \sum_{s' \in S} p(s' \mid s,a) v_*(s')$$

- Another Expression[Sutton]: Last two equations are two forms of the Bellban optimality equation for v_*

- $$v_*(s) = \max_{a \in A(s)} q_{\pi_*}(s,a) = \max_a E_{\pi*}[G_t \mid S_t = s,\ A_t = a]$$

$$= \max_a E_{\pi*}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right]$$

$$= \max_a E_{\pi*}\left[R_{t+1} + \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \mid S_t = s,\ A_t = a\right]$$

$$= \max_a E[R_{t+1} + \gamma\, v_*(S_{t+1}) \mid S_t = s, A_t = a]$$

$$= \max_{a \in A(s)} \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_*(s')]$$

Optimal action-value euqation $q_*(s,a)$: Maximum action value function for all policies

- $$q_*(s,a) = \max_\pi q_\pi(s,a)$$

$$= \max_a \left(R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_\pi(s')\right)$$

$$= R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_\pi v_\pi(s')$$

$$= R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_*(s')$$

$$= R(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) \max_{a'} q_*(s', a')$$

- Another Expression[Sutton]

- $$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \mid S_t = s,\ A_t = a\right]$$

$$= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right]$$

II.6 Solving the Bellman Optimality Equation

We can directly define the optimal value function using Bellman optimality equation.

- $$v_*(s) = \max_a \left(R(s) + \gamma \sum_{s' \in s} p(s'|s,a)\, v_\pi(s') \right)$$

$$= R(s) + \gamma \max_a \sum_{s' \in s} p(s'|s,a)\, v_*(s')$$

$$q_*(s,a) = R(s) + \gamma \sum_{s' \in S} p(s'|s,a)\, v_*(s')$$

And optimal policy is simply the action that attains this max

- $$\pi_*(s) = \arg\max_a \sum_{s' \in s} p(s'|s,a)\, v_*(s')$$
- For determininstic optimal policy, $\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$

Bellman optimality equation is non-linear, ane there is no closed form in general. So there are many iterative solution methods including Value Iteration, Policy Iteration, Q-learning and SARSA.

II.7 Value Iteration

Algorithm

1. Initialize an estimate for the value function arbitarily (or zeros): $v(s) \leftarrow 0 \;\; \forall s \in S$
2. Repeat, update $v(s)$: $v(s) \leftarrow R(s) + \gamma \max_a \sum_{s' \in s} p(s'|s,a)\, v_*(s'), \forall s \in S$
3. Output a deterministic policy, π such that $\pi(s) = \arg\max_a \sum_{s',r} p(s', r\,|s,a)[r + \gamma\, v(s')]$

II.8 Policy Iteration

- Given a policy π, then evaluate the policy π
- Improve the polocy by acting greedy with respect to v_π
- Policy iteration requires fewer iterations than value iteration, but each iteration requires solving a linear system instead of just applying bellman operator.
- In practice, policy iteration is often faster, especially if the transision probabilites are structured (e.g. sparse) tomake solution of linear system different.

Algorithm

1. Initialize policy $\widehat{\pi}$ (e.g. randomly)
2. Compute a value function of policy, v_π (e.g. via solving linear system or Bellman expectiation equation iteratively).
3. Update π to be greedy policy with respect to v_π: $\pi(s) \leftarrow \arg\max_a \sum_{s' \in s} p(s'|s,a)\, v_\pi(s')$
4. If policy π changed in last iteration, return to step 2.

[Another Approach from Sutton]

Policy Evaluation

First we consider how to compute the state-value function v_π for an arbitary policy π. This is called policy evaluation in the DP literature.

- $$v_\pi(s) = E_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s\right]$$
$$= E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_\pi(s')]$$

If the environment's dynamics are completely knowm then given equation is a system of $|S|$ simultaneous linear equations in $|S|$ unknowns (the $v_\pi(s), s \in S$). Consider a sequence of approximate value functions $v_0, v_1, v_2, \ldots$ each mapping $S \to R$. Initial approximation v_0 is chosen arbitrarily and each successive approximation is obtained by using the Bellman equation for v_π as an update rule.

- $$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma v_k(s')]$$

Policy Improvement

We know how good it is to follow the current policy from s - that is $v(\pi)$- but would it better or worse to change to the new policy? One way to answer this question is to consider selecting a in s and thereafter following the existing policy π.

The value of this way of behaving is:

- $$q_\pi(s,a) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a\ ]$$
$$= \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')]$$

The key criterion is whetheher this is greator than or less than $v_\pi(s)$. If it is greater-that is, if it is better to select a one in s and thereafter follow π than it would be to follow π all the time-then one would expect it to be better still to select a evey time s is encountered, and that the new policy would in fact be a better one overall.

that this is true is a special case of a general result called the policy improvement theorem. Let π and π' be any pair of deterministic policies such that, for all $s \in S$.

- $q_\pi(s, \pi'(s)) \geq v_\pi(s)$

Then the policy π' must be good as good as, or better than, π. That is, it must obtain greater or equal return from all states $s \in S$.

- $v_{\pi'}(s) \geq v_\pi(s)$

So far we have seen how, given a policy and its value function, we can easily evaluate a change in the policy at a single state to a particular action.It is a natural extension to consider changes at all states and to all possible actions, selecting at each state the action that appears best according to $q_\pi(s, a)$.

In other words, to consider the new greedy policy π' given by:

- $$\begin{aligned}\pi'(s) &= \arg\max_a q_\pi(s, a) \\ &= \arg\max_a E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a\ ] \\ &= \arg\max_a \sum_{s', r} p(s', r|s, a)[r + \gamma v_\pi(s')]\end{aligned}$$

where $\arg\max_a$ denots the value of a at which the expression that follows is maximized.

Suppose the new greedy policy, π', is as good as, but not better than the old policy π. Then $v_\pi = v_{\pi'}$, and it follows that for all $s \in S$:

- $$\begin{aligned}v_{\pi'}(s) &= \max_a E[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s,\ A_t = a\ ] \\ &= \max_a \sum_{s', r} p(s', r|s, a)\left[r + \gamma v_{\pi'}(s')\right]\end{aligned}$$

But this is the same as the Bellman optimality equation, and therefore, $v_{\pi'}$ must be v_*, and both π and π' must be optimal policies. Policy improvement thus give us a strictly better policy except when the original policy is already aoptimal.

In case of stochastic policy π, the policy improvement theorem carries through as stated for the shochastic case, user the natural definition

- $q_\pi(s, \pi'(s)) = \sum_a \pi'(a|s) q_\pi(s, a)$
- If $q_\pi(s, \pi'(s)) \geq v_\pi(s)$, then $v_{\pi'}(s) \geq v_\pi(s)$

Policy Iteration

Once a plicy π has been improved using v_π to yield a better policy π', we can then compute $v_{\pi'}$ and improve it again to yield an even better π''. We can thus obtain a sequence of monotonically improving policies and vlue functions.

- $\pi_0 \rightarrow$ evaluation $\rightarrow v_{\pi 0} \rightarrow$ improvement $\rightarrow \pi_1 \rightarrow$ evaluation $\rightarrow v_{\pi 1} \rightarrow$ improvement $\rightarrow \cdots \rightarrow \pi_* \rightarrow$ evaluation $\rightarrow v_*$

III. MDP application on Satellite Network System

III.1 Recap for Value Iteration and Policy Iteration Algorithm

Value Iteration Algorithm

1. Initialize array V arbitrarily $(e.g., V(s) = 0 \text{ for all } s \in S)$

2. Repeat

$\Delta \leftarrow 0$

For each $s \in S$:

$v \leftarrow V(s)$

$V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

3. Output a deterministic policy, π, such that

$$\pi(s) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$$

Policy Iteration (Using iterative policy evaluation)

1. Initialization

$V(s) \in R$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$

2. Policy Evaluation

Repeat

$\Delta \leftarrow 0$

For each $s \in S$

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ (a small positive number)

3. Policy Improvement

$policy - stable \leftarrow$ true

For each $s \in S$

$a \leftarrow \pi(s)$

$\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$

If $a \neq \pi(s)$, then $policy - stable \leftarrow$ false

If policy − stable, then stop and return V and π;

also go to 2

Iterative Policy Evaluation

$$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s]$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$$

III.2 Application to Satellite Network Structure

III.2.1 Key point of two algorithms:

How to define $\sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$ for each action a from given policy $\pi(a \mid s)$?

- In this statement, all of required variable for formulating MDP are included: $M = (S, A, P, R, \gamma)$
- Therfore, it is important to define given 5 variables and modulate the process of generating each probability in each case.
- Also, defining the relationship and interation between each variable should be also important.

III.2.2 Defining Variables

1. S : Finite **Set** of States - Location of the data packet. Total 78 States (30 GS + 48 SATS)
2. A : Finite **Set** of Actions - Ordering the agents (GS + SAT) to mover the data packet from one place to another (GS -> SAT, SAT -> SAT, SAT -> GS)
3. P : State Trasmission Probability **Matrix**- Probability when the data is successfully transmitted by given action. (e.g. 0.8 indicates that failure of data transmittion in 20% occation)
4. R : Reward **Function** - Reward when the data is moved to certain agent. GS -> SAT / SAT -> SAT: minus reward. Only positive reward is when the data is transmitted to final destination SAT which can directly transmit to GS.
5. γ : Discount **Factor** - Discount factor of the reward in order to sum of series of rewards are converged.
6. $\pi(a \mid s)$: Policy probability - Wheter deterministic $\pi = 1$ for optimal and 0 for all other case or stocastic $\pi \sim (0, 1)$

III.2.3 S, A, T, R, γ, π formulation

```
clear;clc;

number_of_states = 48;

load('/workspace/RS_Dataset/RS_HL_3_dataset.mat')

time_indics = 1;

sat_to_sat_contact_matrix = sat_to_sat_contact_3d_matrix(:,:,time_indics);

sample_1 = find(sat_to_sat_contact_matrix(1,:) == 1);

gamma = 0.99;

for i = 1:48

  actions_info = find(sat_to_sat_contact_matrix(i,:) == 1);
  number_of_actions = length(actions_info);
   for j = 1:number_of_actions
     MDP.(['state' num2str(i)]).(['action' num2str(actions_info(j))]).pi = 1/
number_of_actions;

    % State Transmission Possibility: Default 0.8 and should be desgined
    % returning to current state for 0.2 possibility

     MDP.(['state' num2str(i)]).(['action' num2str(actions_info(j))]).T =
0.8;

     % Design Reward Function - Need to allocation different reward value
```

```
     % for different i and j pair i indicates the starting state and j
     % inticates the end state

     MDP.(['state' num2str(i)]).(['action' num2str(actions_info(j))]).R = -1;

   end
end
```

III.2.4 Value Iteration

```
for k = 1:48
    MDP.(['state' num2str(i)]).value = 0;
end


for l = 1:48
  Delta = 0;
  v = MDP.(['state' num2str(l)]).value;

end

%
% % Create the MDP structure
% MDP.state1.action1 = 0;
% MDP.state1.action2 = 0;
% MDP.state1.action3 = 0;
%
% MDP.state2.action1 = 0;
% MDP.state2.action2 = 0;
% MDP.state2.action3 = 0;
%
% % Add more states and actions as needed
% for i = 3:number_of_states
%     MDP.(['state' num2str(i)]).action1 = 0;
%     MDP.(['state' num2str(i)]).action2 = 0;
%     MDP.(['state' num2str(i)]).action3 = 0;
% end
%
% % Search for a specific state and action
% state_num = 25;
% action_num = 2;
%
% % Access the state and action
% state_name = ['state' num2str(state_num)];
% action_name = ['action' num2str(action_num)];
% value = MDP.(state_name).(action_name);
% fprintf('The value of MDP.%s.%s is %f\n', state_name, action_name, value);
```