RS-HL-5: MDP-DP Satellite Network Simulation

Hongseok Kim

UT Austin Oden Institute

6/26/2024

Scope

- Design the concept of Satellite Network Structure based on MDP framework reviewed in RS-HL-5: MDP-DP Theory.
- Briefly Review the Theory and Setup the Structure of required variables.
- Starting from simple structure (1 GS -> 1 GS, time invariant) and expand to complicated structure (N GS -> N GS, time variant)

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 Transition Probability **Matrix** - Probability when the data is successfully transitted by given action. (e.g. 0.8 indicates that failure of data transittion 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 transitted to final destination SAT which can directly transit 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

Equations to Acquire:

1. **Action Value Function** : $q(s,a) = \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$

   $\Rightarrow$ Very Basic Function for Furture Equations
2. **State Value Function** : $v(s) = \sum_a \pi(a|s)\, q(s,a) = \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]$

   $\Rightarrow$ Required for Policy Evaluation
3. **Optimal Policy Function** : $\pi(a|s) = \arg\max_a q(s,a) = \arg\max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')]$: In case of deterministic

   $\Rightarrow$ Collect all cases of action value functions $q(s,a)$ and pick the action which makes highest $q(s,a)$

   $\Rightarrow$ If there is multiple $q_*(s,a)$, then equally distribte the $\pi(a|s)$

Using MATLAB structure array and pupose of the code design format:

- We are using MATLAB structure array for allocating data structure of each variable by different hierarchy.
- The purpose of explaining structure hiearchy explicitely is to open the future possbility to expand this algorithm in various ways.
- Also, we are using this structure as a testbed for various constellation formation and satellite specification, so structure change shall be flexible.
- The Network Structure will be more complicated in the future, and the conops will be changed in time.
- Therefore, we have to design the code open to be changed.

MDP structure hierarchy

Level 1: MDP structure array

- Top hierarchy, which formulates total MDP structure

Level 2: State s string array

- Represents states $s \in S$. One state indicates the location where certain data packet locates.
- Currently, we have total 78 states $S = \{s_1, s_2, \dots s_{78}\}$, 78 situations in which data locates in 30 ground stations and 48 satellites.
- But, in this situation, we only simulating satellite network, so utilizaing only 48 states (48 satellites)
- We can define 3D state array and allocate action for different layer for each iteration in time-variant MDP, which will be simulated in the future.

Level 3-1: Action a string array

- Represents actions $a \in A$. The decisions that indicates what state the agent **'wants'** to transit
- We have Matrix indicates connectivity between each states. So we can extract the states that agent in certain state can reach.
- For example let's say we are in state s_{10}, and from contact chart, the available states that data packet can go are $\{s_1, s_3, s_7, s_{10}, s_{18}, s_{30}\}$. Then, we have available action corresponding to available states: $\{a_1, a_3, a_7, a_{10}, a_{18}, a_{30}\}$.
- **Note: we also consider the action whcih make not to move around, stay in the current state.**
- **This is intended to consider the fastly changing network structure, there is possibility that agenet decide it may be optimal choice to stay -> Future application**

Level 3-2: State value $v(s)$ double array

- Each State has its state value in constructor of state.
- We update the State value $v(s)$ iteratively from action values and policy functions we get from level 4
- $v(s) = \sum_a \pi(a|s)\, q(s,a)$

Level 4-1: Next State s' string array (3-1-1)

- The probability of successfully transiting to other state is not 100%.
- If success rate is 80%, there exist two next states: Success / Return $s_{\text{next}} = \{s_s\ , s_r\}$
- The corresponding reward for Success and Return will be different
- **Note: For end node (termination state), we define the next state is only returning state** $s_{\text{next}} = s_r$.

Level 4-2: Action value $q(s,a)$ double array (3-1-2)

- Collecting all cases in single action, we get action value from action value function

- $q(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma V_k(s')] = T_s(r_s + \gamma V(s_s)) + T_r(r_r + \gamma V(s_r))$
- We can get T_s and r_s from level 5 array, γ is global factor and V from level 3 array

Level 4-3: Policy Function $\pi(a|s)$ double array (3-1-3)

- The possibility distribution which agent pickes certain action is policy function.
- Since we have defined action in level 3, we can define each action's possibility for each action.

Level 5-1: State Transition Probabilty $T = p(s'|s,a)$ double array (4-1-1)

- From corresponding 'Next state' we can define state transition probability. ($T_s = 0.8$ for success and $T_r = 0.2$ for failure and returning)
- Note: For end node (termination state), we define returing State transmission probabilty $T_r = 1$.

Level 5-2: Reward r double array (4-1-2)

- We have to allocate for every situation, and pair between starting state and next state.
- For the most cases, we allocate negative reward.
- Also, we know the starting state and end state, so we can customize different reward system by classification of what orbit the data os tramitted to what orbit.
- When the next state is end node from different start node, we allocate high posibie reward.
- For starting from end node, we define 0 reward.

Reward System by next state by different cases

Case 1: Regular transition

- Currently, we have two sub-orbits of the satellite system: Orbit 1 and Orbit 2.
- Transiting between different orbit may take more cost, which means less reward.
- Also, returning to start state should be the worst.

1-1: From orbit 1 to orbit 1: -1

1-2: From orbit 1 to orbit 2: -15

1-3: From orbit 2 to orbit 1: -1

1-4: From orbit 2 to orbit 2: -15

1-5: Returning to starting state: -50

Case 2: Next State = end state

2-1: From orbit 1 to orbit 1: 100

2-2: From orbit 1 to orbit 2: 50

2-3: From orbit 2 to orbit 1: 100

2-4: From orbit 2 to orbit 2: 50

2-5: Returning to starting state: -50

Case 3: Starting State = end state

3-1: Returing to starting state: 0

```
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);
%       v_updated(j) =
%   end
%
%   MDP.(['state' num2str(i)]).value = sum(v_updated);
% end
gamma = 0.99;


end_destination = 37;


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 Transition 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))]).T1 =
0.8;
```

```
     MDP.(['state' num2str(i)]).(['action' num2str(actions_info(j))]).T2 =
0.2;

     % 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

       if i < 25
            if actions_info(j) < 25
                MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = -1;
            end
            if actions_info(j) > 24
                MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = -15;
            end
        end

       if i > 24
            if actions_info(j) > 24
               MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = -1;
            end
            if actions_info(j) < 25
               MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = -15;
            end
       end


      if i == actions_info(j)
           MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = -50;
      end

   % Definition of End_destination scenario

      if i == end_destination
        MDP = rmfield(MDP, ['state' num2str(i)]);
        MDP.(['state' num2str(i)]).(['action' num2str(i)]).T1 = 0.8;
        MDP.(['state' num2str(i)]).(['action' num2str(i)]).T2 = 0.2;
        MDP.(['state' num2str(i)]).(['action' num2str(i)]).R = 0;
      end



      if actions_info(j) == end_destination
        if i < 25
            if actions_info(j) < 25
```

```
                MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = 50;
            end
            if actions_info(j) > 24
                MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = 100;
            end
        end

       if i > 24
            if actions_info(j) > 24
               MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = 50;
            end
            if actions_info(j) < 25
               MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).R = 100;
            end
       end

      end

   end
end
```

III.2.4 Value Iteration

```
% for k = 1:48
%     MDP.(['state' num2str(k)]).value = 0;
% end
%
%
% for i = 1:48
%   Delta = 0;
%   v = MDP.(['state' num2str(i)]).value;
%   actions_info = find(sat_to_sat_contact_matrix(i,:) == 1);
%   number_of_actions = length(actions_info);
%
%   v_updated = zeros(number_of_actions,1);
%   for j = 1:number_of_actions
%       T1 = MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).T1;
%       T2 = MDP.(['state' num2str(i)]).(['action'
num2str(actions_info(j))]).T2;
%       R =
%
% %       v_updated(j) =
%   end
```

```
%
%   MDP.(['state' num2str(i)]).value = sum(v_updated);
% 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);
```