RS-HL-9: Time Variant MDP - Code

UT Austin Oden Institute

Hongseok Kim

07/09/2024

Scope

- In this report, the code of Time Variant MDP of Satellite Simulation is Presented.

II. Code Demonstration

II.1 Call the SAT-to-SAT Dataset

```
clear;clc;


% Load the Satellite Contat Dataset
load('/workspace/RS_Dataset/RS_HL_3_dataset.mat')

% Select Elapsed time slot for the simulation:
% 15 seconds timestep,so n time_indices indicates start time = 15*n seconds

% time_index = 1000;
```

II.2 Configure MDP Structure (Level 1 - 6)

```
% Level 1: Initialize MDP structure
MDP = struct();

% Level 2: Initialize Time Index Vector

time_index_vector = 1:200;

for time_index = time_index_vector
    MDP.(['time' num2str(time_index)]) = {};
end

% Level 3: Initialize State and Policy Distribution

number_of_states = length(sat_to_sat_contact_3d_matrix(1,:,1));

for time_index = time_index_vector
    for state_index = 1:number_of_states
```

```
    % Level 3.1: Initialize State
    MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]) = {};
    % Level 3.2: Initialize Policy Distribution
    MDP.(['time' num2str(time_index)]).('policy_distribution') =
zeros(number_of_states);
    end
end

% Level 4: Initialize Action, State Value


for time_index = time_index_vector
    sat_to_sat_contact_matrix =
sat_to_sat_contact_3d_matrix(:,:,time_index);
    for state_index = 1:number_of_states

        % Find the availabe next state candidates for each current state
        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);

        for action_index = 1:number_of_actions
        % Level 4.1: Initialize Action
        MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]) = {};

        % Level 4.2: Initialize State Value with 0 initial value
        MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
('state_value') = 0;

        end
    end
end

% Level 5: Initialize Success/Fail, Action Value and Policy Function

for time_index = time_index_vector
    sat_to_sat_contact_matrix =
sat_to_sat_contact_3d_matrix(:,:,time_index);
    for state_index = 1:number_of_states

        % Find the availabe next state candidates for each current state
        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);

        for action_index = 1:number_of_actions
            % Level 5.1: Initialize Success / Fail and Next State (Level 6)
```

```
            MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('next_state') = {};
            MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('next_state') = {};

            % Level 5.2: Initialize Action Value
            MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('action_value') =
{};

            % Level 5.3: Initialize Policy Function
            MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('policy_function') = 1/number_of_actions;

        end
    end
end

% Level 6-1: Define Next State value, State Transition Probability


for time_index = time_index_vector
    sat_to_sat_contact_matrix =
sat_to_sat_contact_3d_matrix(:,:,time_index);
    for state_index = 1:number_of_states

        % Find the availabe next state candidates for each current state
        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);

        for action_index = 1:number_of_actions

         % Level 6.1: Case 1 - Next state value if Action is success
         MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('success').('next_state') =
next_state_candidates(action_index);

         % Level 6.1: Case 2 - Next state value if Action is failure
         MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('fail').('next_state') = state_index;

         % Level 6.2: Define State transition probability T (S - 0.8 / F -
0.2)
         MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('success').('transition_probability') =
0.8;
```

```
            MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('fail').('transition_probability') = 0.2;

        end
    end
end

% Level 6-2: Define Reward

for time_index = time_index_vector
    sat_to_sat_contact_matrix =
sat_to_sat_contact_3d_matrix(:,:,time_index);
    for state_index = 1:number_of_states

        % Find the availabe next state candidates for each current state
        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);

        for action_index = 1:number_of_actions

            % Define Failure Reward = -30
            MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('reward') = -30;
            current_state = state_index;
            next_state = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('success').('next_state');

             % Regular Transition:
             % Transition in Same orbit -> reward = -1, Transition between
Different orbit -> reward = -15
            if current_state < 25
                if next_state < 25
                    MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = -1;
                end
                if next_state > 23
                    MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = -15;
                end
            end

            if current_state > 23
                if next_state > 23
```

```
                    MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = -1;
                end
                if next_state < 25
                    MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = -15;
                end
            end

            % If success but returning to current state: Reward = -50
            if next_state == state_index
                MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = -50;
            end

        end
    end
end
```

II.3 Define and Modify Destination State Information

```
% Define destination satellite number
destination_state = 38;

for time_index = time_index_vector
MDP.(['time' num2str(time_index)]) = rmfield(MDP.(['time'
num2str(time_index)]), ['state' num2str(destination_state)]);
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
('state_value') = 0;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('action_value') = {};
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('policy_function') = 1;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('success').('next_state') = destination_state;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('success').('transition_probability') = 1;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('success').('reward') = 0;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('fail').('next_state') = destination_state;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('fail').('transition_probability') = 0;
MDP.(['time' num2str(time_index)]).(['state' num2str(destination_state)]).
(['action' num2str(1)]).('fail').('reward') = 0;
end
```

```
% update corresponding reward information based on destination state
for time_index = time_index_vector

    sat_to_sat_contact_matrix =
sat_to_sat_contact_3d_matrix(:,:,time_index);
    % We don't count the case of destination state: already defined
    for state_index = 1:number_of_states

        if state_index == destination_state
            continue
        end

        % Find the availabe next state candidates for each current state
        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);
        for action_index = 1:number_of_actions

        % Define State transition Probability T (Level 5-2 = 4-1-2)
        current_state = state_index;
        next_state = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('success').('next_state');

        %Filter: Next state is destination state
            if next_state == destination_state

            % Positive Reward function for Next state is destination case
            % Same orbit reward = 100, Different orbit reward = 50
                if current_state < 25
                    if next_state < 25
                        MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = 100;
                    end
                    if next_state > 23
                        MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = 50;
                    end
                end

                if current_state > 23
                    if next_state > 23
                        MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = 100;
                    end
```

```
                    if next_state < 25
                        MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward') = 50;
                    end
                end
            end
        end
    end
end

fprintf('simulation set up complete!')
```

```
simulation set up complete!
```

II.4 Policy Iteration Process for Last Step

II.4.1 Policy Evaluation Process

```
pi_distribution = zeros(number_of_states);
pi_distribution_updated = zeros(number_of_states);

t_n = max(time_index_vector);

policy_iteration_count = 1;

state_value_struct = struct();
state_value_matrix_cumulative = [];
value_iteration_no = [];


% Start of the Algorithm
while true

% Initialize pi_distribution

gamma = 1;

tolerance = 1e-1;
max_iterations = 1000;
Delta = inf;
value_iteration_count = 0;

state_value_matrix = [];


while(abs(Delta) > tolerance) && (value_iteration_count < max_iterations)
    Delta_vector = zeros(number_of_states,1);
```

```
    state_value_vector = zeros(1,number_of_states);

    for state_index = 1:number_of_states

        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);

        if state_index == destination_state
        number_of_actions = 1;
        end


        v = MDP.(['time' num2str(t_n)]).(['state'
num2str(state_index)]).state_value;
        v_update = 0;


        for action_index = 1:number_of_actions
            T_s = MDP.(['time' num2str(t_n)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('transition_probability');
            r_s = MDP.(['time' num2str(t_n)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('reward');
            T_f = MDP.(['time' num2str(t_n)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('transition_probability');
            r_f = MDP.(['time' num2str(t_n)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('reward');
            s_s = MDP.(['time' num2str(t_n)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('success').
('next_state');
            s_f = MDP.(['time' num2str(t_n)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('next_state');
            v_s_s = MDP.(['time' num2str(t_n)]).(['state' num2str(s_s)]).
('state_value');
            v_s_f = MDP.(['time' num2str(t_n)]).(['state' num2str(s_f)]).
('state_value');


            q_s_a = T_s * (r_s + gamma* v_s_s) + T_f * (r_f + gamma* v_s_f);
            MDP.(['time' num2str(t_n)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('action_value') = q_s_a;
```

```
            pi = MDP.(['time' num2str(t_n)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('policy_function');

            v_update = v_update + pi*q_s_a;
        end

       MDP.(['time' num2str(t_n)]).(['state'
num2str(state_index)]).state_value = v_update;
       Delta_vector(state_index) = abs(v_update-v);
       v = v_update;
       state_value_vector(state_index) = v_update;
    end

    state_value_matrix = [state_value_matrix; state_value_vector];
    state_value_matrix_cumulative =
[state_value_matrix_cumulative;state_value_vector];

    Delta = max(Delta_vector);
    value_iteration_count = value_iteration_count + 1;
    if value_iteration_count >= max_iterations
        fprintf('Maximum number of iterations reached for state %d\n',
state_index);
    end
end
state_value_struct.(['iteration' num2str(policy_iteration_count)]) =
state_value_matrix;
value_iteration_no = [value_iteration_no; length(state_value_matrix(:,1))];
state_value_matrix = [];
```

II.4.2 Policy Improvement Process

```
for state_index = 1:number_of_states

    next_state_candidates = find(sat_to_sat_contact_matrix(state_index,:) ==
1);
    number_of_actions = length(next_state_candidates);

    if state_index == destination_state
      number_of_actions = 1;
    end

    action_value_vector = zeros(number_of_actions,1);

    for action_index = 1:number_of_actions
    action_value_vector(action_index) = MDP.(['time' num2str(t_n)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('action_value');
    end
```

```
    maximum_action_value = max(action_value_vector);
    argmax_a = find(action_value_vector == maximum_action_value);
    for action_index = 1:number_of_actions
      if any(action_index == argmax_a)
        MDP.(['time' num2str(t_n)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('policy_function') = 1/length(argmax_a);
      else
        MDP.(['time' num2str(t_n)]).(['state' num2str(state_index)]).
(['action' num2str(action_index)]).('policy_function') = 0;
      end

    pi_distribution_updated(state_index,action_index) = MDP.
(['time' num2str(t_n)]).(['state' num2str(state_index)]).(['action'
num2str(action_index)]).('policy_function');


    end
end

   if pi_distribution == pi_distribution_updated
       break;
   end

   pi_distribution = pi_distribution_updated;
   policy_iteration_count = policy_iteration_count +1;

end

total_policy_iteration = policy_iteration_count;

MDP.(['time' num2str(t_n)]).policy_distribution = pi_distribution;
```

II.5 Value and Action Function Propagation to $t_{n-1}, t_{n-1}, \cdots t_1$

```
inverse_time_index = flip(time_index_vector(1:end-1));


for time_index = inverse_time_index
    pi_distribution_updated = zeros(number_of_states);
    sat_to_sat_contact_matrix =
sat_to_sat_contact_3d_matrix(:,:,time_index);
    for state_index = 1:number_of_states
        % Find the availabe next state candidates for each current state
        next_state_candidates =
find(sat_to_sat_contact_matrix(state_index,:) == 1);
        number_of_actions = length(next_state_candidates);
```

```
        if state_index == destination_state
            number_of_actions = 1;
        end

        action_value_collector = zeros(number_of_actions,1);

        for action_index = 1:number_of_actions

            T_s = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('success').('transition_probability');
            r_s = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('success').('reward');
            T_f = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('transition_probability');
            r_f = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('reward');
            s_s = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('success').('next_state');
            s_f = MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).('fail').
('next_state');
            v_s_s = MDP.(['time' num2str(time_index+1)]).(['state'
num2str(s_s)]).('state_value');
            v_s_f = MDP.(['time' num2str(time_index+1)]).(['state'
num2str(s_f)]).('state_value');


            q_s_a = T_s * (r_s + gamma* v_s_s) + T_f * (r_f + gamma* v_s_f);
            MDP.(['time' num2str(time_index)]).(['state'
num2str(state_index)]).(['action' num2str(action_index)]).('action_value') =
q_s_a;
            action_value_collector(action_index) = q_s_a;
        end
        state_value = max(action_value_collector);
        MDP.(['time' num2str(time_index)]).(['state' num2str(state_index)]).
('state_value') = state_value;

        argmax_a = find(action_value_collector == state_value);

        for action_index = 1:number_of_actions
            if any(action_index == argmax_a)
                MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('policy_function') = 1/length(argmax_a);
            else
```

```
                MDP.(['time' num2str(time_index)]).
(['state' num2str(state_index)]).(['action' num2str(action_index)]).
('policy_function') = 0;
            end
        pi_distribution_updated(state_index,action_index) = MDP.
(['time' num2str(time_index)]).(['state' num2str(state_index)]).(['action'
num2str(action_index)]).('policy_function');
        end
    end

    MDP.(['time' num2str(time_index)]).policy_distribution =
pi_distribution_updated;
end
```

II.6 Test

```
start_time = 50;
start_state = 27;

time_index = start_time:max(time_index_vector);

simulation_time = length(time_index);

state_list = zeros(simulation_time,1);
reward_list = zeros(simulation_time,1);
cumulative_reward = 0;

state_list(1) = start_state;

% We should consider the possibility of the failure (80% success and 20%
% failure)

for t = 1:simulation_time
    current_state = state_list(t);

    pi_dist = MDP.(['time' num2str(time_index(t))]).('policy_distribution');
    action_number = find(pi_dist(current_state,:));

    if length(action_number) > 1
        action_number = randsample(action_number,1);
    end

    next_state = MDP.(['time' num2str(time_index(t))]).(['state'
num2str(current_state)]).(['action' num2str(action_number)]).('success').
('next_state');
```

```
    reward = MDP.(['time' num2str(time_index(t))]).(['state'
num2str(current_state)]).(['action' num2str(action_number)]).('success').
('reward');
    cumulative_reward = cumulative_reward + reward;

    reward_list(t+1) = cumulative_reward;
    state_list(t+1) = next_state;
end

[state_list, reward_list]
```

```
ans = 152×2
    27     0
    28    -1
    29    -2
    30    -3
    31    -4
    32    -5
    33    -6
    34    -7
    35    -8
    36    -9
     ⋮
```

III.Analysis

III.1 State Value changing over time

```
state_value_over_time = zeros(length(time_index_vector),number_of_states);



for t = 1:length(time_index_vector)
    for state_index = 1:number_of_states
        state_value_over_time(t,state_index) = MDP.
(['time' num2str(time_index_vector(t))]).(['state' num2str(state_index)]).
('state_value');
    end
end

for state_index = 1:number_of_states
    plot(time_index_vector,state_value_over_time(:,state_index))
    hold on
end

hold off
```

```
title('State Value Function Change Over Time')
xlabel('timestep (x15 seconds)')
ylabel('state value')
```