International Core Journal of Engineering Volume 7 Issue 1, 2021
ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

Planning for Autonomous Robots by Building a Markov Decision Process of a Problem Featuring Uncertain Duration of Actions

Haoyuan Ding

Dalian University of Technology, Dalian, Liaoning 116000, China.

Abstract

Robot mission planning is a major part of AI and robotics which is the process of creating a sequence of actions to achieve an agent's goals. Autonomous robots have to get the capability to achieve a goal where simple reactive behavior leads to poor behavior and to know what it is going to do before doing it. Simple search or logical inference are not suitable here because the former requires domain-specific heuristics to be effective while the latter does not scale well to larger problems. Planning can overcome these problems by using factored representations and action representations. Considering the uncertain duration of actions, this is a kind of planning with non-deterministic models which means actions define multiple outcomes. In this paper, MDPs are built in MATLAB and Python to quantify the uncertainty and create better plans. For some multi-factor MDPs, the priority is set in order to solve nest problems. The existing toolbox is not fully functional so particular programs need to be coded.

Keywords

MDP; Value iteration; Finite horizon; Uncertainty; Nest problem.

1. Introduction

MDP means Markov Decision Process. It is widely used in solving optimization problems. As the outcomes are random and the rewards are different, MDP need to make decisions which leads to optimization[1]. Generally, MDP contains four parts:

$$M = \langle S, \bar{s}, A, T \rangle$$

S is a set of states (typically attributions of values to state variables/features). $\bar{s} \in S$ is the initial state. A is a set of actions. T: $S \times A \to Dist(S)$ is the probabilistic transition function. Considering the probabilities, the outcome of an action is not always what the model describes, which can be unintended outcomes, exogenous events or inherent uncertainty. The solution is to extend the basic action definition to specify multiple outcomes. In other words, a single action can result in one of many possible states. But a plan is a sequence of actions, and doesn't take into account this non-determinism. A structure which allows the actor to look up the next action given the resulting state is needed. Generally, this structure is called "policy", which is a mapping of finite paths of the MDP to (distributions over) actions:

$$\pi : FPath_{M,\bar{s}} \to Dist(A)$$

Provided with a goal state, a policy will maximize probability of reaching it:

$$V^*(\bar{s}) = \Pr^{\max}_{M,\bar{s}}(reach_G) = \sup_{\pi} \Pr^{\pi}_{M,\bar{s}}(reach_G).$$

2. Stochastic shortest path (SSP) MDP

For Stochastic Shortest Path (SSP) MDP[2], two factors are added:

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

$$M = \langle S, A, T, C, G \rangle$$

C is a set of action costs and G is the goal state. Now states are usually sets of variables or values and actions may only be applicable/enabled given certain values of state variables.

2.1 Value iteration

The optimal policy provides the action a to take in state s which minimizes the cost to reach a goal state. Optimal Q value (cost-to-go) is the expected cost to first execute action a in state s, and then follow an optimal policy:

$$Q^*(s,a) = \sum T(s,a,s')\left[C(s,a,s') + V^*(s')\right]$$

And the optimal value function is defined as:

$$V^*(s) = 0 (s \in G)$$
$$= \min Q^*(s,a) (s \notin G).$$

Use dynamic programming to successively approximate V^* with V_n:

$$V_n(s) \leftarrow \min_{a \in A} \sum_{s' \in S} T(s,a,s')\left[C(s,a,s') + V_{n-1}(s')\right]$$

The whole process is shown in figure 1.

1 initialize V_0 arbitrarily for each state
2 $n \leftarrow 0$
3 repeat
4 $\quad n \leftarrow n+1$
5 $\quad$ **for each** $s \in S$ do
6 $\quad\quad$ compute $V_n(s)$ using Bellman backup at s
7 $\quad\quad$ compute $residual_n(s) = |V_n(s) - V_{n-1}(s)|$
8 $\quad$ **end**
9 **until** $\max_{s \in S} residual_n(s) < \varepsilon$;
10 return greedy policy: $\pi^{V_n}(s) = \arg\min_{a \in A} \sum_{s' \in S} T(s,a,s')[C(s,a,s') + V_n(s')]$

Figure.1 Value iteration

2.2 Finite horizon

An MDP has a finite horizon if the number of stages is specified. Given the transition probabilities and rewards, the value function and optimal policy can be calculated by using backwards induction algorithm[3,4]. This allows to recursively evaluate function values starting from the terminal stage. In general, the horizon can be seen as either step or time. Considering the uncertain duration of actions, the policy might change because of different outcomes.

2.3 Nest problem

For real world planning, the non-deterministic factors can be multiple. There are two methods for the agent to take all the factors into consideration. The first method is weighted. By assigning weight to all the factors, it becomes possible for the agent to get “Synthetic optimal policy”. Obviously, when the number of factors is large, this method performs poorly. It is also tough to assign weight as it directly decides the result. So it is easier to use the second method — setting priority for factors. Priority draws into “nest problem” which means solve problems in stages. For the “nest”, the “external” factors are what to be guaranteed while the “internal” factors are what are needed to reach optimization.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

3. Building an MDP in MATLAB

In MATLAB, it is convenient to use the MDP toolbox[5] which has functions related to the resolution of discrete-time Markov Decision Processes: value iteration, policy iteration and finite horizon. It is also available on GNU Octava, Scilab and R. After adding the toolbox to path, it is possible to use all its functions.

3.1 Using policy iteration

Policy iteration is a typical algorithm in solving MDP. Necessarily, it needs transition probability array, reward array and discount factor. The maximum number of iterations and starting policy can also be set. The six-box problem in figure 2 is a great starting point because it has states, probabilities, costs, start and goal state.

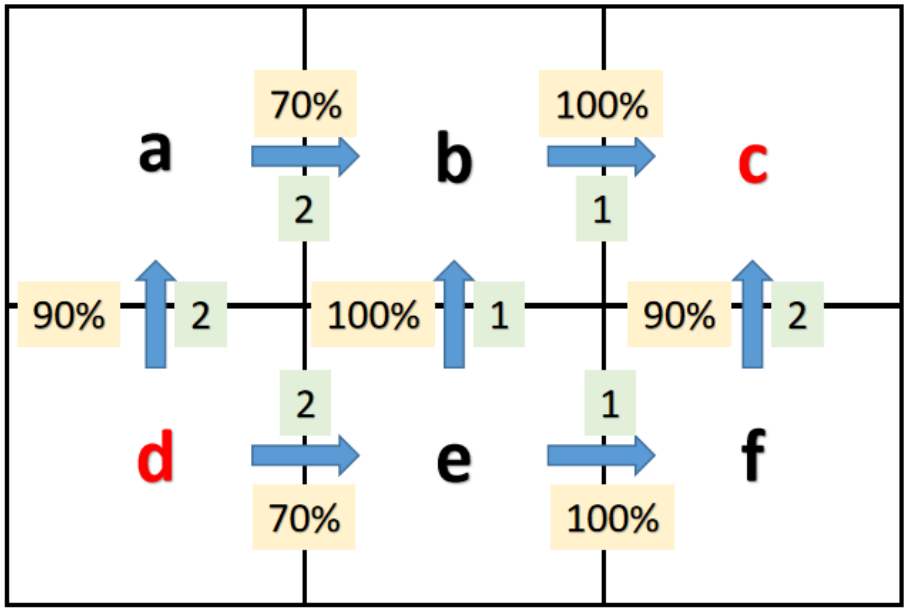

Figure.2 Six-box problem

First, states: a, b, c, d, e, f, and actions: 1. a → b, 2. b → c, 3. b → e, 4. c → b, 5. c → c, 6. d → a, 7. d → e, 8. e → b, 9. e → f, 10. f → c, 11. f → e must be defined.

There is a flaw hidden here: action c → c. This action means the process is infinite. The agent can find the best action in every state, but when it is in state c, the goal state, it still tries to find the best action and doesn't know how to stop. In addition, for every action, the toolbox needs us to input all the probabilities of every state. This leads into plenty of unnecessary numbers. The probabilities and rewards are defined as figure 3.

```
>> P(:,:,1)=P(:,:,1)+eye(6,6);
>> P(:,:,2)=P(:,:,2)+eye(6,6);
>> P(:,:,3)=P(:,:,3)+eye(6,6);
>> P(:,:,4)=P(:,:,4)+eye(6,6);
>> P(:,:,5)=P(:,:,5)+eye(6,6);
>> P(:,:,6)=P(:,:,6)+eye(6,6);
>> P(:,:,7)=P(:,:,7)+eye(6,6);
>> P(:,:,8)=P(:,:,8)+eye(6,6);
>> P(:,:,9)=P(:,:,9)+eye(6,6);
>> P(:,:,10)=P(:,:,10)+eye(6,6);
>> P(:,:,11)=P(:,:,11)+eye(6,6);
```

```
>> P(1,1,1)=0.3;P(1,2,1)=0.7;
>> P(2,2,2)=0;P(2,3,2)=1;
>> P(2,5,3)=1;P(2,2,3)=0;
>> P(3,2,4)=1;P(3,3,4)=0;
>> P(4,1,6)=0.9;P(4,4,6)=0.1;
>> P(4,4,7)=0.3;P(4,5,7)=0.7;
>> P(5,2,8)=1;P(5,5,8)=0;
>> P(5,6,9)=1;P(5,5,9)=0;
>> P(6,3,10)=0.9;P(6,6,10)=0.1;
>> P(6,5,11)=1;P(6,6,11)=0;
```

```
>> R=zeros(6,6,11);
>> R(:,:,1)=R(:,:,1)-eye(6,6);
>> R(:,:,2)=R(:,:,2)-eye(6,6);
>> R(:,:,3)=R(:,:,3)-eye(6,6);
>> R(:,:,4)=R(:,:,4)-eye(6,6);
>> R(:,:,5)=R(:,:,5)-eye(6,6);
>> R(:,:,6)=R(:,:,6)-eye(6,6);
>> R(:,:,7)=R(:,:,7)-eye(6,6);
>> R(:,:,8)=R(:,:,8)-eye(6,6);
>> R(:,:,9)=R(:,:,9)-eye(6,6);
>> R(:,:,10)=R(:,:,10)-eye(6,6);
>> R(:,:,11)=R(:,:,11)-eye(6,6);
```

```
>> R(1,1,1)=-2;R(1,2,1)=-2;
>> R(2,3,2)=-1;
>> R(2,5,3)=-1;
>> R(3,2,4)=-1;
>> R(3,3,5)=0;
>> R(4,1,6)=-2;R(4,4,6)=-2;
>> R(4,4,7)=-2;R(4,5,7)=-2;
>> R(5,2,8)=-1;
>> R(5,6,9)=-1;
>> R(6,3,10)=-2;R(6,6,10)=-2;
>> R(6,5,11)=-1;
```

Figure.3 Probabilities and rewards for policy iteration

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

Set discount = 0.99999; Then use policy iteration:

[V, policy, iter, cpu_time] = mdp_policy_iteration(P, R, discount)

The results are shown in figure 4.

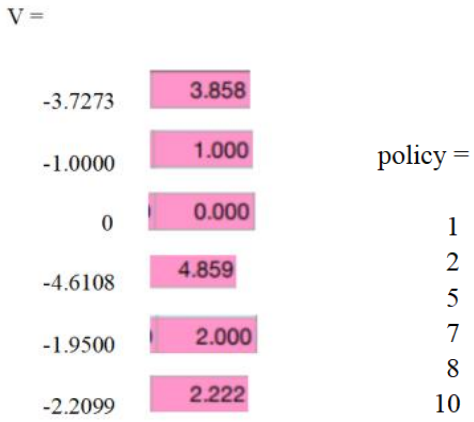

Figure.4 Value and policy for policy iteration

There is another way of using the toolbox. First, all the paths must be listed, and calculate the value of them. The results are shown in Table 1.

①d → a → b → c

②d → a → b → e → f → c

③d → e → b → c

④d → e → f → c

Table 1. Values of routes

Route	①	②	③	④
Value	-6.0793	-8.1903	-4.8571	-6.0793

3.2 Using finite horizon

In order to solve the problem of infinite horizon, "mdl_finite_horizon"in the toolbox can be used. Assume a robot in the shopping mall has 3 states: goods shelves (1), warehouse (2), check-out (3) and 2 actions: move back and forth between the warehouse and the shelves to move the goods or enter the shopping mall again (1), check out at the cashier and go out or hang around outside the shopping mall (2). Every time the goods are delivered, the robot will give a reward of 2. Checking out will receive a reward of 5, but it will leave the shopping mall. It's a long way to re-enter the shopping mall, so the reward is set as -8. The reward for other state changes is -1, because the robot needs time to move. Probabilities and rewards are defined as figure 5.

```
>> P(:,:,1)=[0 1 0;1 0 0;1 0 0];   >> R(:,:,1)=[0 -1 0;2 0 0;-8 0 0];
>> P(:,:,2)=[0 0 1;0 1 0;0 0 1]    >> R(:,:,2)=[0 0 5;0 -1 0;0 0 -1]
```

Figure.5 Probabilities and rewards for finite horizon

The results are shown in figure 6. At the last 2 stages, if the robot is in state 1, it will stop delivering goods and check out. And at the last 3 stages, if the robot is outside the shopping mall, it will not enter again.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

```
>> [V, policy] = mdp_finite_horizon (P, R, 0.99 , 10)
V =
    7.5047    8.4183    6.6572    7.5893    5.7925    6.7435    4.9102
   10.3341    8.5906    9.5134    7.7346    8.6760    6.8611    7.8217
    0.3341   -1.4094   -0.4866   -2.2654   -1.3240   -3.1389   -2.1783

    5.8805    4.0100    5.0000         0
    5.9699    6.9500    2.0000         0
   -2.9701   -1.9900   -1.0000         0
policy =
     1     1     1     1     1     1     1     1     2     2
     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     2     2     2
```

Figure.6 Value and policy for finite horizon

4. Building an MDP in Python

Obviously, the toolbox is more suitable to deal with problems of which actions always affect all the states. In MATLAB, there are several problems:

① Have unnecessary data.

② Need to choose the route artificially.

③ The horizon is infinite.

In order to solve the problems, MDP must be built in Python

4.1 Solving basic MDP

In Python, actions are a dictionary. For every action, it has name, enabled state, goal state, probability and cost. For example, actions['f_c'] = [{'f'}, [['c', 0.9, 2], ['f',0.1, 2]]]. After inputting states, actions, initial state, goal state, initial value and discount factor, value iteration is achieved as shown in figure 7. The problems in MATLAB are solved.

```
['a_b']
['b_c']
['d_e']
['e_b']
['f_c']
{'a': 3.8571432796156753, 'b': 1.0, 'c': 0, 'd': 4.857144269690259, 'e': 2.0, 'f': 2.222222222222223}
3.295943937331458e-06
number of iterations: 17
initial value and epsilon are appropriate
running time: 0.015625s
```

Figure.7 Value iteration in Python

4.2 Solving MDP with uncertain duration

When time is added into consideration, the goal is not simply to take the optimal path to the end [6,7]. Now the agent must reach the end within the given time and on the premise of goal 1, choose the best action.

Now the six-box problem has time, probability(yellow) and value (green), shown in figure 8.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

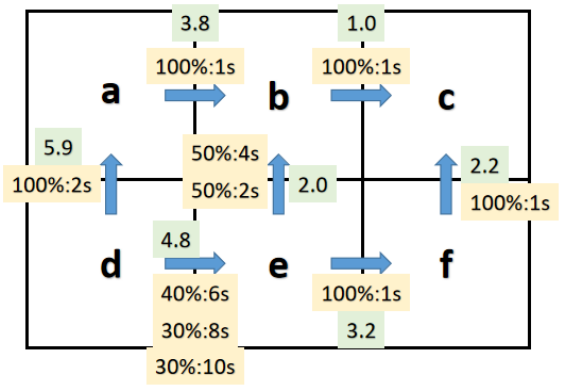

Figure.8 Six-box problem with uncertain duration

First, the allowing time of every state which means the minimum time to ensure that every state can reach the goal needs to be calculated. The result is shown in figure 9.

```
C:\Users\qazml\AppData\Local\Programs\Python\Python
{'a': 2, 'b': 1, 'c': 0, 'd': 11, 'e': 2, 'f': 1}
```

Figure.9 The allowing time of every state

Then the new program can achieve the goal. In short, it will first judge whether it can reach the goal within the given time. In the case of insufficient time, priority will be given to the shortest time-consuming path. While in the case of sufficient time, it will choose the path with the least cost. The results are shown in figure 10. Even though inputting the same time, different results can be obtained because of uncertain duration of actions.

```
please input given time: 12
{'a': 2, 'b': 1, 'c': 0, 'd': 11, 'e': 2, 'f': 1}
local state: d
action running time: 6s
local state: e
action running time: 2s
local state: b
action running time: 1s
reach goal state: c
remaining time: 3s
```

```
please input given time: 12
{'a': 2, 'b': 1, 'c': 0, 'd': 11, 'e': 2, 'f': 1}
local state: d
action running time: 6s
local state: e
action running time: 4s
local state: b
action running time: 1s
reach goal state: c
remaining time: 1s
```

```
please input given time: 12
{'a': 2, 'b': 1, 'c': 0, 'd': 11, 'e': 2, 'f': 1}
local state: d
action running time: 10s
local state: e
action running time: 1s
local state: f
action running time: 1s
reach goal state: c
remaining time: 0s
```

```
please input given time: 12
{'a': 2, 'b': 1, 'c': 0, 'd': 11, 'e': 2, 'f': 1}
local state: d
action running time: 8s
local state: e
action running time: 1s
local state: f
action running time: 1s
reach goal state: c
remaining time: 2s

Process finished with exit code 0
```

Figure.10 Different results

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

4.3 Solving nest MDP

Now if the probability, time and cost are combined, it becomes a nest problem[8]. In order to solve it, priority needs to be set. The goals are:

① calculate the probability of every action reaching the goal state.

② select the actions with the highest probability of current state (If there are more than one action, then select all of them).

③ choose the least cost action and execute it.

④ judge whether the action is successful. Add time and change current state.

Six-box problem has few states and actions, so a nine-box problem is used to show more clearly as shown in figure 11.

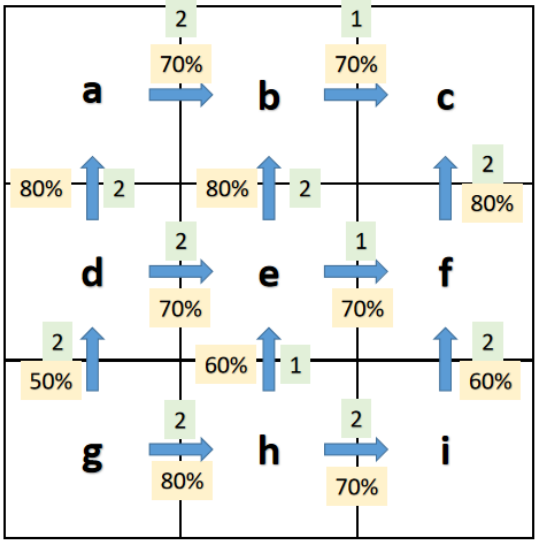

Figure.11 Nine-box problem

After inputting state, probability and cost, the expected probabilities of every action can be calculated. A higher probability indicates an action that is easier to reach the goal which is shown in figure 12.

{'a_b': [{'a'}, {'b'}, 0.7, 2, 0.48999999999999994], 'd_a': [{'d'}, {'a'}, 0.8, 2, 0.39199999999999996],
'd_e': [{'d'}, {'e'}, 0.7, 2, 0.39199999999999996], 'e_b': [{'e'}, {'b'}, 0.8, 2, 0.5599999999999999],
'b_c': [{'b'}, {'c'}, 0.7, 1, 0.7], 'e_f': [{'e'}, {'f'}, 0.7, 1, 0.5599999999999999],
'f_c': [{'f'}, {'c'}, 0.8, 2, 0.8], 'g_d': [{'g'}, {'d'}, 0.5, 2, 0.19599999999999998],
'g_h': [{'g'}, {'h'}, 0.8, 2, 0.2688], 'h_e': [{'h'}, {'e'}, 0.6, 1, 0.33599999999999997],
'h_i': [{'h'}, {'i'}, 0.7, 2, 0.33599999999999997], 'i_f': [{'i'}, {'f'}, 0.6, 2, 0.48]}

Figure.12 The expected probabilities of actions

Thus, in every state, the agent will first consider actions with the highest probabilities. If there is more than one option, it starts to consider about the cost. In this way, the priority of factors is implemented. The results are shown in figure 13.

The previous probability does not consider the number of remaining steps, so it is also important to calculate the probability of every action reaching the goal state in finite steps. However, there is a problem thatthe expected probability of actions is relevant to the path which makes this function not very useful.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

```
current state: g
optional actions: ['g_h']
selected action: g_h
action success
action running time: 2s
current state: h
optional actions: ['h_e', 'h_i']
selected action: h_e
action fail
action running time: 1s
current state: h
optional actions: ['h_e', 'h_i']
selected action: h_e
action success
action running time: 1s
current state: e
optional actions: ['e_b', 'e_f']
selected action: e_f
action fail
action running time: 1s
current state: e
optional actions: ['e_b', 'e_f']
selected action: e_f
action success
action running time: 1s
current state: f
optional actions: ['f_c']
selected action: f_c
action success
action running time: 2s
reach the goal!
total running time: 8s

Process finished with exit code 0
```

Figure.13 The result of nest MDP

For example, in order to get the expected probability of state g reaching the goal in several steps, the calculation need to be done in the best path (g →h →e →f →c). The result is shown in figure 14.

```
C:\Users\qazml\AppData\Local\Programs\Python\P
please input the probability of action 1: 0.8
please input the probability of action 2: 0.6
please input the probability of action 3: 0.7
please input the probability of action 4: 0.8
please input the number of steps: 4
the expected probability: 0.2688

Process finished with exit code 0
```

```
please input the number of steps: 5
the expected probability: 0.5644799999999999
please input the number of steps: 6
the expected probability: 0.7714560000000001
please input the number of steps: 7
the expected probability: 0.8894592000000001

please input the number of steps: 100
the expected probability: 0.9999999999998417

Process finished with exit code 0
```

Figure.14 The expected probability of reaching the goal with steps

There is another way of solving the nest MDP: calculating the "q value" of every action in every remaining step. In this way, the function "When steps are sufficient, select the highest probability action. When steps are insufficient, select the least cost action" can be achieved. As shown in figure 15, when there is only 1 step remained, the agent can't reach the goal, so the probabilities are all 0. It will choose actions according to the cost.

```
q_value_prob for ('g', 1) g_d_1 0.0
q_value_cost for ('g', 1) g_d_1 202.00000000000003
q_value_prob for ('g', 1) g_h_1 0.0
q_value_cost for ('g', 1) g_h_1 201.0
q_value_prob for ('g', 2) g_d_2 0.0
q_value_cost for ('g', 2) g_d_2 203.79999999999998
q_value_prob for ('g', 2) g_h_2 0.0
q_value_cost for ('g', 2) g_h_2 202.0
```

Figure.15 The q value of actions

ISSN: 2414-1895 DOI: 10.6919/ICJE.202101_7(1).0036

5. Conclusion

The MDP has many types considering the complexity of problems, with corresponding algorithms and programs. The MATLAB MDP toolbox mainly aims at resolution of discrete-time MDP. It includes finite horizon, discounted criterion, average criterion and utilities. In the case of actions that affects few states, it is more suitable to program separately in Python. The three MDPs in Python all have their own programs and can be generalized to solve that sort of problems. The more states and actions are input, the more complicated the MDP will be. For the nest MDP, the first thing is to set priority of factors. Then calculate new values floor by floor and add them to corresponding actions. The policy can also be adjusted to choose higher probability or lower cost when the remaining steps are insufficient.

References

[1] Ghallab, M., Nau, D., & Traverso, P. (2016). Automated planning and acting. Cambridge University Press.

[2] Weld, D. S. (2008). Planning with durative actions in stochastic domains. Journal of Artificial Intelligence Research, 31, 33-82.

[3] Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.

[4] LaValle, S. M. (2006). Planning algorithms. Cambridge university press.

[5] Chadès, I., Chapron, G., Cros, M. J., Garcia, F., & Sabbadin, R. (2014). MDPtoolbox: a multi-platform toolbox to solve stochastic dynamic programming problems. Ecography, 37(9), 916-920.

[6] Gillani, R., & Nasir, A. (2016, January). Incorporating artificial intelligence in shopping assistance robot using Markov Decision Process. In 2016 International Conference on Intelligent Systems Engineering (ICISE) (pp. 94-99). IEEE.

[7] Mansouri, M., Lacerda, B., Hawes, N., & Pecora, F. (2019). Multi-robot planning under uncertain travel times and safety constraints. International Joint Conferences on Artificial Intelligence Organization.

[8] Lacerda, B., Parker, D., & Hawes, N. A. (2017). Multi-objective policy.