Implementation of Reinforcement Learning and Game Theory for Optimal Path Planning with Dynamic Obstacles

Jin Woo Park
Aeronautics & Astronautics
Stanford University
jinwoop@stanford.edu

Hyoungju Seo
Aeronautics & Astronautics
Stanford University
hjseo@stanford.edu

***Abstract*— Path planning is a fundamental problem which seeks a dynamically-feasible trajectory and is applied in multidisciplinary fields. This project is inspired from a real road situation where a driverless vehicle is able to find the optimal path from origin to goal that yields no collision with other moving vehicles. The objective of this project is to find the optimal trajectory without having a collision with unknown obstacle dynamics. We propose modified A* algorithm, game theory algorithms, and reinforcement algorithm then compare the performance of algorithms with respect to success rate, evaluation cost, and time complexity. Q-learning algorithm generates the most optimal path with highest success rate.**

I. Introduction

Path planning is a fundamental problem which seeks a dynamically-feasible trajectory connecting an initial state to a goal state avoiding obstacles. This problem has been developed and applied in many fields. In robotics, path planning concerns the movement of a robot with uncertainty, interaction with different robots, and it's dynamics. In artificial intelligence, path planning is interpreted as sequential decision-making of logical actions for optimal trajectories. In controls field, path planning considers the optimality and stability of the plant.

A. Literature Reviews

There are several traditional algorithms for path planning and a few examples are: Breadth First Search (BFS), Depth First Search (DFS), and Dijkstra's algorithm. However, these algorithms are ineffective search algorithms in terms of time and space complexity. The complexity increases exponentially, as the size and/or the dimension of map increases. Furthermore, DFS does not guarantee the optimality of result [1].

A* algorithm is one of the best known path planning algorithm that reduces complexity, which can be applied to metric or topology configuration space. A* additionally implements the heuristic search cost with the traditional path planing algorithm. Typically, $h(v)$ is the heuristic distance from the current state to the goal state. Also, $g(v)$ is the length of the path from the initial state to the current state through the selected sequence of states. Hence, the state with the lowest value of $f(v) = h(v) + g(v)$ is chosen as the next state in the sequence. The computational time is significantly large for high dimensional and dynamically constrained path planning problems. Also, the time complexity of A* largely depends on the chosen heuristic value, $h(v)$ [2].

Sampling-based path planning has emerged as a successful algorithm to solve complex problems. It obtains the solution through uniform or probabilistic sampling from the feasible space. Furtheremore, it probes local connections using collision checking module. The accuracy of the sampling method is proportional to the number of the samples. In short, this method does not guarantee the optimality of the solution. However, the accuracy, time and space complexity increases with the sample size [3].

The Reinforcement Learning (RL) can be implemented to find optimal policies in path planning problems by maximizing a value function. RL takes an action and receives corresponding reward (which can be positive or negative). RL updates its parameters and learns the optimal policy for each state. The reinforcement learning is also capable of finding optimal policy using learned parameters even when new environment is encountered. Also, once the algorithm is learned, generating optimal actions does not depend on the complexity of problems. Hence, the reinforcement learning can be applied to real time

path planning problem [4].

Reinforcement learning techniques have been developed to optimize agents' behavior in a wide range of circumstances. However, multi-agent problems are convoluted in a sense that there is no guarantee of the existence of optimal solution. Furthermore, a dynamic environment makes problem more complicated due to the limited information of rewards and actions. The real world problems such as autonomous vehicles have to account for multi-agents (i.e., other vehicles and pedestrians) in dynamic environment. This led to the development of the interdisciplinary work on reinforcement learning and game theory [5].

B. Proposed work

In this project, we will implement various path planning algorithms to find optimal path in a dynamic environment which is similar to real road situation. We want our agent to move optimally (in terms of distance and time) from the origin state to the goal state without collision with other agents (i.e., obstacles). We use a random sampling-based algorithm for the baseline and a perfect observation algorithm for the oracle. Furthermore, we propose a modified A* algorithm, some game theory algorithms, and a reinforcement learning algorithm for optimal path in our dynamic environment.

II. Problem Formulation and Solution

A. Environment Model

Task environment is a 10 by 10 grid map. For the simple case, there are our agent and five dynamic obstacles which move along the grid cells on the map. For the complex situation, we generate fifteen dynamic obstacles on the map.

1) Initialization: Our agent starts at the origin position. Obstacles are initialized with different random positions and different direction. The direction is one of 'North (N)', 'East (E)', 'South (S)', 'West (W)', 'North West (NW)', 'North East (NE)', 'South West (SW)', or 'South East (SE)'.

2) Movements: Our agent can move in eight directions every time step and chooses the action that returns the optimal policy using different algorithms. Obstacles change their position according to their direction. However, if the next positions are out of the map, they will change their direction opposite way and move to that direction.

3) Terminal: If our agent arrives at goal position, the task is ended without any penalty. If our agent collides with an obstacles, the task is ended with huge penalty.

4) Evaluation cost: The evaluation cost function is the sum of our agent's time cost, travel distance, remain distance to the goal, and penalty at the terminal. Hence, we can conclude that minimum of final evaluation cost indicates the optimal path of our agent.

Evaluation cost = time cost + travel distance + 100 * remain distance + 1000 * [penalty]

The figure below illustrates the setup of our environment.

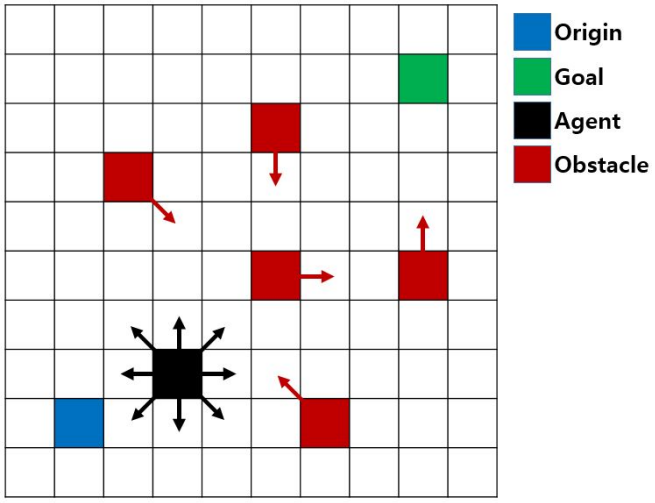

Fig. 1. An example of our project setup.

B. Algorithm

1) Baseline: The baseline of this project is to find a policy that allows the agent to travel from origin to goal without having any collision. For baseline we propose a random sampling-based algorithm. We randomly sample next position every time step with robust collision checking. The robust collision checking makes our agent not sample the grids which can be occupied by obstacles in the next time step and outside of the map. Since this algorithm does not care the previous trajectory, time cost, and distance cost, the result might be deviated from the optimal path. We

also set the time limit to avoid infinite loop between some positions.

Algorithm 1 Baseline Algorithm

Input Agent Position, Obstacles Positions at time t
Output Agent Action at time t
Steps :
    for next possible agent's action
        if pass robust collision check
            add the action to set
    Randomly sample the next action from the set

2) Oracle: The oracle is to find the optimal policy that minimizes the time and distance needed to achieve the baseline. The performance of the oracle is the same to the situation that the agent knows dynamics of all obstacles, which is the upper bound of the performance we can get. To compare the performance of other algorithms, we propose an unrealistic oracle algorithm that the agent knows future movements of all obstacles.

Algorithm 2 Oracle Algorithm

Input Agent Position, Obstacles Positions at time t
    Obstacles Positions at time $t+1$
Output Agent Action at time t
Steps :
    for next possible agent's action
        if not collision with obstacles at time $t+1$
            add the action to set with cost value
    Find the minimum cost value action from the set

3) Modified A:* The modified A* algorithm is a greedy algorithm that searches for the policy that generally move towards to the goal and perform evasive maneuver (action) that avoids collision with obstacles. A heuristic function is implemented in A* algorithm. The function estimates the probability of the next obstacle's position based on its current position and its previous movement direction. It is likely that the object will obey the law of inertia, the relative moving direction is likely to be the same for the next time step. We assigned probability of 0.25 for moving forward, 0.15 for forward cornering (forward + turn) and turning, and 0.025 for reverse cornering, and 0.1 for moving backwards. Also heuristic function accounts for the distance remaining to the goal position. The utility costs are based on only 1 step lookahead with consideration of remaining distance, time elapsed, and penalty for the high probability of having collision. The pros and cons of this algorithm are low time complexity and relatively high success rate and the policy is likely to be suboptimal respectively.

Algorithm 3 Modified A* Algorithm

Input Agent Position, Obstacles Positions at time t,
    Estimate of Obstacles Positions at time $t+1$ using heuristic.
Output Agent Action at time t
Steps :
    for obstacles within 5 x 5 square centered at agent,
        for next possible obstacle's action
            calculate the penalty proportional to the probability obtained using heuristic function.
    for next possible agent's action
        find the minimum cost value (penalty) action from the set.

4) Game Theory: We apply two game theory algorithms, minimax and expectimax, to generate possible routes without collision. In our task, players are our agent and obstacles, the state is the current position, action is the next movement direction, and utility is evaluation cost of current game state. Since our agent has to minimize time and distance value, we implement minimax and expectimax algorithms opposite way, which minimizes evaluation cost value. The game theory algorithm is following.

Algorithm 4 Game theory Algorithm

Input Agent Position, Obstacles Positions at time t
Output Agent Action at time t
Steps :
    for next possible agent's action
        Calculate the current game state value recursively
        Recursion with alpha beta pruning
        Minimax :
            Obstacles : maximum game state value action
            Agent : minimum game state value action
        Expectimax :
            Obstacles : Random action
                Average game state value
            Agent : minimum game state value action
        Until Terminal or Depth limit
    Find the minimum game state value action

The computation complexity of the game theory increases numerously with the branch and depth size. In our task, the branch is b = 8 (possible direction set) and the depth is d = 6K (the number of agent

and obstacles, depth limit). Although we use alpha-beta pruning and limited depth tree search, complexity is $O(8^{6k})$ still high.

Hence, we propose a modified game theory algorithm which only cares a small amount of obstacles. During calculating game state value, we consider only n nearest obstacles around our agent. This algorithm reduces complexity to $O(8^{(n+1)k})$.

5) Reinforcement Learning: Q-learning is one of the most popular model-free reinforcement learning algorithm that searches for the optimal policy of unknown models. The Q-values are obtained by following the incremental update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right).$$

In order to predict the Q-value at the next state, the algorithm estimates the Q-value using linear function of learned weights and state information (feature) as:

$$Q(s',a') = W^{\top} x.$$

The two types of features used in this paper are:

- the relative goal position vector and agent's action: ($x_{goal} - x_a$, action$_a$),
- state of agent, the relative obstacle position vector, and obstacle's action: (x_a, $x_{obs} - x_a$, action$_{obs}$).

The agent has limited obstacles information such that it only knows the presence of obstacle within the visibility of two steps. The performance of Q-learning policy significantly depends on the balance between exploration and exploitation during the training stage. In this paper, the epsilon-greedy method with exploration probability of 0.2 was used to train the feature weights. Also, decaying learning rate ($\alpha = \frac{1}{\sqrt{timestep}}$) and discount factor ($\gamma$) of 0.9 were used in the Q-learning algorithm.

The feature weights were trained by iterating each route 1000 times for 400 routes (episode). The algorithm is delineated in Alg. 5.

Algorithm 5 Q-learning Simulation

```
for goal in {(1,1), (1,10), (10,1), (10, 10)} do
    for init in {(1,1), (1,2), ..., (10,10) } do
        for trial = 1:1000 do
            Q-learning algorithm.
```

After training the feature weights, exploration probability was set to 0.0 when searching for actual policy. The details of Q-learning algorithm is shown in Alg. 7

Algorithm 6 Q-value function

```
for obstacles within 5 x 5 square centered at agent do
    for next possible obstacles action do
        estimate the obstacle's next action based on the
            probability obtained using heuristic function.
    subtract corresponding action from legal actions.
Q_val = 0
for feature, feature value in feature extraction do
    Q_val += weight[feature] × feature value.
return Q_val.
```

Algorithm 7 Q-learning Algorithm

Inputs:
agent position at time t,
obstacle positions and directions at time t.
Outputs:
agent's action at time t.

```
if random.random() < exploration prob. then
    choose random action from legal actions.
else
    choose action maximizes the Q-value.
if x_agent = x_goal then
    Q̂(s,a) = r
else
    Q̂(s',a') = r + γ max_a' Q(s',a').
for feature, feature_value in feature extraction do
    weight[feature] −= α (Q̂(s,a) − Q(s,a)) × feature value.
return action maximizes Q(s,a).
```

III. Simulation and Result

We test baseline, oracle, modified A*, game theory algorithms, and reinforcement learning algorithm with 100 iterations and compare the success rate, average evaluation cost value, and average time complexity. The simple case and complex case results are in the table 1 and 2 respectively. Figure 3, 4 show trajectory examples of algorithms in simple case and complex case.

TABLE I

Results of Success rate, Average Cost, and Time complexity for 5 Obstacles

	Success Rate	Average Cost	Time Complexity (s)
Baseline	0.85	375.0	0.845
Oracle	1.00	17.31	0.0081
Modified A*	0.96	22.01	0.0273
Game Theory(Minimax)	0.86	19.73	0.789
Game Theory(Expectimax)	0.83	19.63	2.543
Modified Game Theory(Minimax)	0.81	19.08	0.143
Modified Game Theory(Expectimax)	0.77	20.33	0.356
Q-learning	**0.94**	**17.44**	**0.016**

As we expected, the results demonstrate that baseline shows the worst performance and oracle shows the best performance in terms of all criteria.

TABLE II

Results of Success rate, Average Cost, and Time complexity for 15 Obstacles

	Success Rate	Average Cost	Time Complexity (s)
Baseline	0.52	408.6	1.10
Oracle	1.00	18.35	0.019
Modified A*	0.85	68.28	0.023
Modified Game Theory(Minimax)	0.38	24.77	0.292
Modified Game Theory(Expectimax)	0.43	24.23	0.582
Q-learning	**0.89**	**18.53**	**0.019**

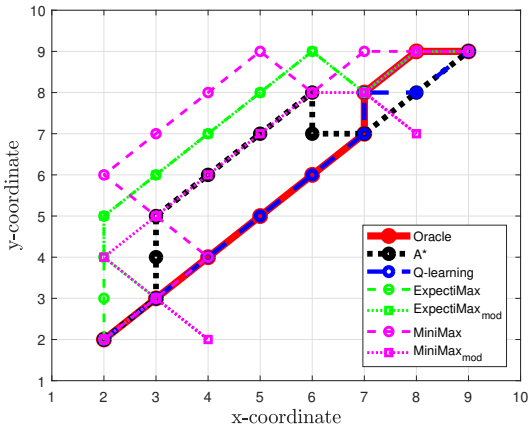

Fig. 2. The *simple* case trajectories.

The modified A* provides high success rate and fast compute time regardless of the complexity of the task. However, the optimality is sacrificed in the complex environment. Since the modified A* considers the collision probability with heuristic future obstacles movements in the 5 x 5 window, it generates conservative policies within a few computations.

The game theory algorithms show relatively optimal policies with high computation time. As we mentioned in the previous section, the time complexity is $O(8^{(n+1)k})$ in every path searching. Furthermore, it provide low success rate in the complex case. The assumption of obstacles movements by game theory (Minimax : moving adversely, Expectimax : moving randomly) can be mostly different from the actual dynamic of obstacles, which affects success rate significantly. Notice that two game theory algorithms, Minimax and Expectimax without any pruning, take more than 12 hours to compute in complex environment.

The Q-learning algorithm presents the highest success rate and most optimal result in the shortest time over all realistic algorithms. Although training feature weights of Q function takes a while (the training of feature weights took about 30 minutes on i7-7800X with 32GB of memory), evaluation of Q function and prediction of next action can be done instantaneously

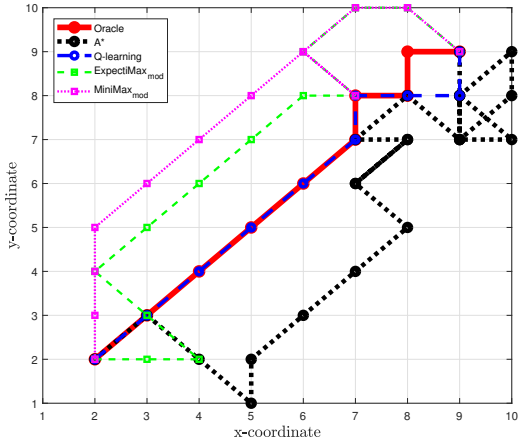

Fig. 3. The *complex* case trajectories.

without compromising success rate and cost. According to Fig. 2 and 3, it is apparent that Q-learning generates the path almost identical to that of oracle.

A. Error Analysis

Trade-off between optimality and success rate is a crucial factor in the application to real-world problem. In some situation that there are pedestrians, we have to more care success rate compared to optimality. However, emergency vehicles should only consider the optimality as other vehicles move with positive effect. Hence, we will analyze how we can control the trade-off.

The results demonstrate that there is a trade-off between optimality and success rate in the modified A* algorithm game theory algorithms. The modified A* algorithm shows high success rate and high evaluation cost (low optimality), as it sets high penalty for collision. If we set penalty for collision to be smaller, the results are different as in the following table.

TABLE III

Comparison of collision penalty in complex game

	Success Rate	Average Cost	Time complexity (s)
A* (Collision Penalty: α)	0.85	68.28	0.023
A* (Collision Penalty: $\frac{\alpha}{100}$)	0.36	18.89	0.0028

We can see that game theory algorithms provide low success rate despite the high penalty for collision. This is because the algorithms do not consider heuristic future obstacles movements(the law of inertia) and assume obstacles movements adversely or randomly. Hence, if we constraint possible action of obstacles

with the law of inertia, we can see the different performance of game theory algorithm in the following table.

TABLE IV

Comparison of inertia consideration in complex game

	Success Rate	Average Cost	Time complexity (s)
Modified Game Theory(Inertia Consideration)	0.66	24.67	0.015
Modified Game Theory(Without Consideration)	0.5	32.98	0.2448

As we can see the results, obstacles inertia consideration generates high success rate, optimal path, and fast computation.

The effect of change in magnitude of penalty and consideration of inertia applies to Q-learning as well. However, Q-learning algorithm has another factor that affects its performance; that is the consideration of features. The Q-value is calculated using the dot product of feature weights and values. However, if we do not consider the features that are related to the obstacles, then it is apparent that the agent will not perform well in dynamic obstacle environment. The performance with and without obstacle feature are tabulated below.

TABLE V

Comparison of collision penalty in complex game

	Success Rate	Average Cost	Time complexity (s)
Q-learning (With obstacle information feature)	0.89	18.53	0.019
Q-learning (Without obstacle information feature)	0.64	18.42	0.014

The results confirm our expectation of the performance of Q-learning algorithm such that the success rate of Q-leaning without obstacle information feature is much lower that that of with features. Therefore, it is crucial to have sufficient number of features to boost the performance.

IV. Future Work

Our agent and obstacles move simultaneously in real road situation. Instead of turn-based game theory, game theory algorithms for simultaneous situation can formulate tasks more accurately. Hence, we can apply Nash equilibrium theorem to solve this task simultaneously.

Future research opportunities include improving the success rate for greater number of obstacles and relaxed assumptions in obstacle dynamics by implementing Deep Reinforcement Learning. The success rate can be improved by adjusting reward or by implementing Deep Reinforcement Learning such as Deep Q Network (DQN) and Duel-DQN (DDQN). The implementation of the DQN and DDQN could potentially lead to a better optimal result and performance.

References

[1] Iadaloharivola Randria, Mohamed Moncef Ben Khelifa, Moez Bouchouicha, and Patrick Abellard. A comparative study of six basic approaches for path planning towards an autonomous navigation. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 2730–2735. IEEE, 2007.

[2] František Duchoň, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, 96:59–69, 2014.

[3] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[4] Amit Konar, Indrani Goswami, Sapam Jitu Singh, Lakhmi C Jain, and Atulya K Nagar. A deterministic improved q-learning for path planning of a mobile robot. *IEEE Trans. Systems, Man, and Cybernetics: Systems*, 43(5):1141–1153, 2013.

[5] Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning*, pages 441–470. Springer, 2012.