Path Planning of Multi-Terrain Surfaces using Temporal Difference Learning Techniques

Amar Bhatt, *Computer Engineer, BS/MS*

***Abstract*—When born, animals and humans are thrown into an unknown world forced to use their sensory inputs for survival. As they begin to understand and develop their senses they are able to navigate and interact with their environment. The process in which we learn to do this is called reinforcement learning. This is the idea that learning comes from a series of trial and error where there exists rewards and punishments for every action. The brain naturally logs these events as experiences, and decides new actions based on past experience. An action resulting in a reward will then be higher favored than an action resulting in a punishment. Using this concept, autonomous systems, such as robots, can learn about their environment in the same way. Using simulated sensory data from ultrasonic sensors, moisture sensors, encoders, shock sensors, pressure sensors, and steepness sensors, a robotic system will be able to make decisions on how to navigate through its environment to reach a goal. The robotic system will not know the source of the data or the terrain it is navigating. Given a map of an open environment simulating an area after a natural disaster, the robot will use model-free temporal difference learning with exploration to find the best path to a goal in terms of distance, safety, and terrain navigation. Two forms of temporal difference learning will be tested; off-policy (Q-Learning) and on-policy (Sarsa). Through experimentation with several world map sizes, it is found that the off-policy algorithm, Q-Learning, is the most reliable and efficient in terms of navigating a known map with unequal states.**

Keywords*—Temporal Difference Learning, Q-Learning, Sarsa, Navigation, Robot.

I. Introduction

Temporal difference learning techniques are used in several reinforcement based applications. Its popularity stems from its versatility in being a general-purpose learning algorithm used in several applications [1]. In this paper, two types of temporal difference learning techniques are explored, Q-Learning and Sarsa. Q-learning is an off-policy algorithm that allows a system to learn by exploring actions and their consequences [2] [3]. Sarsa is an on-policy algorithm that updates based on actions taken rather than the best possible action [2] [4]. Each of these algorithms will follow an ϵ-greedy action selection policy allowing for random action selections in early trials, and best action selections in later trials [2]. Each algorithm will be applied to a simulated robotic system navigating a known map with unknown terrain. Each algorithm will also utilize a punishment function with its value function based on normalized environment sensor inputs coming from the robotic system. The algorithms will be tested and analyzed on varying map sizes, complexities, and algorithm variations.

The rest of this paper is organized as follows. After the introduction, Section 2 overviews temporal difference learning, off-policy valuation, on-policy valuation, and selection policies. Section 3 introduces the Q-Learning algorithm, Sarsa algorithm, and testing methodology. Section 4 presents the experimental results and analysis, and Section 5 contains concluding remarks.

II. Background

Reinforcement learning is used in many applications aimed to reflect the way a human/animal's brain learns [5]. Some of these applications include the inverse pendulum, mountain car problem [6], robotic navigation, and decision based systems. In this paper, robotic navigation in an open, multi-terrain world will be explored. For a world to be open it must have no barriers between states, meaning there will always be multiple paths from a starting point to a goal. A multi-terrain surface presents its own challenges. Here, we consider a general purpose system that can navigate on any surface by land. However, the system will navigate with different efficiency levels on different surface terrains. Therefore, sensors on the system detecting obstacles, terrain smoothness, speed, moisture, pressure, and steepness are used to modify the reward of the state the system will be in at any given point.

A. Temporal Difference Learning

Temporal difference learning is branch of reinforcement learning that attempts to model the way an animal learns by predicting a reward in terms of a given stimulus. The heart of this algorithm comes from its prediction error signal, which constantly adjusts expected rewards based on the current given state [7]. This algorithm estimates the value function of a state, allowing it to estimate the final reward at each state and action taken. If it were not estimated, and instead calculated, the agent would need to wait until the final reward before updating state and action reward values. The non-estimated value function is shown in (1) and the temporal difference estimation of the value function is shown in (2) [2].

$$V(s_t) = V(s_t) + \alpha[R_t - V(s_t)] \tag{1}$$

$$V(s_t) = V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \tag{2}$$

As shown the major difference between the original value function and its estimate is the consideration of the reward in the next step as well as the difference in expected reward between the current state and next state. Estimating in this way allows for the continuous update of state values as more information is provided. The r value in (2) is the estimated

reward for the action selected at that state in current time, whereas R in (1) is the final reward. The γ value (0 to 1) is the discount factor used for convergence, where a low γ causes the system to over emphasize the current state, and a higher γ will give a higher weight to future states [2]. Temporal difference learning requires several iterations, called episodes, to converge. There are two types of Temporal Difference Learning methods; Off-Policy and On-Policy.

B. Off-Policy: Q-Learning

Q-learning is an off-policy method of temporal difference learning. Its value function can be seen in (3).

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_{\alpha'} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (3)$$

It learns soft policies allowing it to have a high factor of exploration. Off-Policy algorithms update value functions using assumed actions which have not been tried, as shown by taking the maximum valued action at any given state. The allows the agent to learn actions it did not actually perform [2]. This shows that this algorithm favors exploration more so than exploitation. Q-Learning builds a table (Q) of expected values. These values are updated as states and actions are chosen, and requires several iterations at different states to reach convergence [3]. This is because unlike other neural-network based applications, Q-Learning is unsupervised and must learn an environment dynamically. Typically, this causes massive overhead when scaling this algorithm to larger environments, causing the traditional Q-Learning methods to be unpopular [8]. However, recent advances in this algorithm have generated new ways to define its policy making it capable to scale to larger environments [9].

C. On-Policy: Sarsa

Sarsa is an on-policy method of temporal difference learning. In this case the value function in (4) is updated according to a policy designed more around experience. This ties this algorithm into considering control and exploration as one entity, whereas its Off-Policy counterpart can separate the two.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (4)$$

Unlike the Q-Learning value function equation, Sarsa does not consider the maximum value of an action at a given state, but instead updates with an action that is taken [2]. This shows that this algorithm favors exploitation more so than exploration. This also allows this algorithm to populate Q-values with a closer true approximate than the Q-Learning algorithm [4]. Therefore, it can be said that while Q-Learning updates based on states, Sarsa updates in accordance to state-action pairs. This allows for a better convergence policy, and leads to less unnecessary exploration by the system at hand [6].

D. Epsilon Selection Policies

Both off-policy and on-policy methods of temporal difference learning rely on the selection of states and actions. Policies are meant to find a balance between exploration and exploitation. Popular policies used in both Q-Learning and Sarsa are ϵ-greedy, ϵ-soft, and softmax. In both ϵ-greedy and ϵ-soft selection policies an action is chosen with the highest estimated reward for most of the iterations. They choose a random action with a probability of ϵ for ϵ-greedy and probability 1-ϵ for ϵ-soft where ϵ is a small value between 0 to 1. ϵ-greedy is chosen to limit exploration using ϵ between 0 to 0.5. This results in 0% to 50% of the trials having actions chosen at random. ϵ-soft is chosen to favor exploration using ϵ between 0 to 0.5. This results in 50% to 100% of the trials having actions chosen at random. Unlike the ϵ selections, softmax chooses random actions with respect to weights assigned to each action. This makes it so worst actions are less likely to be chosen, whereas in the ϵ selections the worst action has the same probability as the best action of being randomly chosen. Each selection method has its own benefits, but none have proven to be better than the other [2]. For the application explored in this paper, the ϵ-greedy selection policy will be used due to its limit of exploration which is crucial in time-sensitive natural disaster missions. In this policy a low ϵ value is chosen to set the percentage of episodes set aside for random selection (exploration). The rest of the episodes will choose the action that returns the best reward (exploitation) [9].

E. Application

In many cases robotic path planning is used in controlled environments, whether in a hospital, nursing home, hotel, etc. Reinforcement learning is also used to navigate known city maps and other open environments [8]. However, many of these cases fall into the category of finding the best path of a known map which can also be achieved with less invasive path planning algorithms. Consider the case of a city that underwent a natural disaster. While the city map has stayed the same, the terrain of the city may have vastly altered. Clear roads may be flooded with water, and buildings may have been knocked down creating alternative paths. In this case, it will be pertinent for a land-based robotic system to efficiently navigate the city to reach potential resources and survivors in the safest way. If the system took the known path as shown on a city map before the disaster, the robot may get stuck, hurt, or take too much time. Understanding the landscape and making decisions based on the type of terrain can ensure an efficient and safe path is chosen.

III. Methods

To test the temporal difference learning algorithms, Q-Learning and Sarsa, MATLAB was used. An MxN matrix was created consisting of M*N states. Each state had between 2-4 neighbors, depending on where on the grid it fell. Each state also was associated to one of five terrain types; sand, forest, pavement, water, mountainous rock/debris. These states represented an area on a map after a natural disaster. This meaning

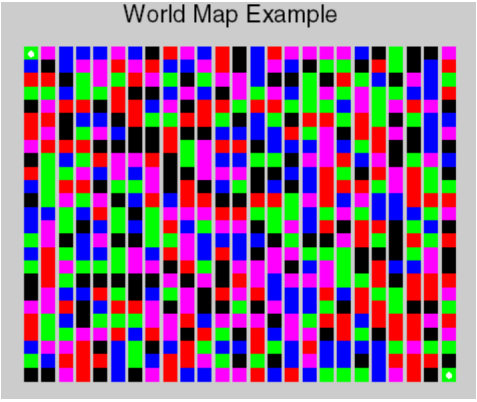

Fig. 1: Map Example with Terrain (Sand = Magenta, Forest = Green, Pavement = Black, Water = Blue, Misc. Debris = Red, Start/Goal = White dots)

```
Choose a Q(s,a)
For each episode
    Set state s
    while s is not goal
        Choose action a from s using
            epsilon policy from Q
        Take action a, observe reward r
            and next state s'
        Update Q-value as follows
        Q(s,a) <- Q(s,a) + α [r+ γ max_α(
            Q(s',a')) - Q(s,a)]
        set s to s'
    Repeat until s is the goal state
```

Fig. 2: Q-Learning Algorithm [2]

the defined paths from a start to a goal have been severely altered. An example map is shown in Fig. 1. Each terrain was associated to certain punishments, as decided by the simulated sensor readings on the robot. The sensors used were, encoders (speed detection), ultrasonic sensors (visibility), shock sensor (smoothness), moisture sensor, pressure sensor (stability), and steepness sensor. Each sensor reading was statically defined and normalized between 0 and 1, where 0 is the safest/most efficient and 1 is the most dangerous/least efficient for that particular sensor. Further testing of the system would call for the sensor readings to be introduced to natural noise.

The world map matrix were at first predefined to validate the two algorithms (defined path from start to goal) and to find appropriate parameter values for the algorithms (punishment weight, discount factor, goal reward, etc). The maps were then randomized for further testing. The simulated robot navigating the map from the start (top left corner) to the goal (bottom right corner) had no prior knowledge of the terrain of the state it was in. It did know the map, however. As each algorithm was run, the value at each state was updated using a reward for the action chosen as well as a punishment which was determined by the sensor readings for each state. On a normalized average the punishments for each terrain type was as follows: sand 1, forest 2, pavement 0, water 3, mountainous rock/debris 4. This is much like the penalties incurred in James Sutton's puddle world problem, however, in this case the robot does not know about the terrain in its world, only what it reads through its sensors [6]. Varying punishments made one state more favorable than another in terms of proximity to the goal, safeness, and efficiency. A rewards matrix of M*NxM*N was created, where each row symbolized a state and each column symbolized an action (next state). Each neighbor state in the actions column were given a value of 0, non-neighbors were given a value of -∞. The goal state and any state that was a neighbor to the goal received the highest reward value. A Q-matrix was also created the same way as the reward matrix, however, the goal was set to 0 to be updated by the temporal difference learning algorithms. Both the reward and Q-matrix represented the true map of the world without terrain information [10]. Other factors needing tuning were the goal reward, the discount factor, and the punishment weight. These were found for both algorithms through the control testing on a predefined map. Also, each algorithm was run for 10,000 episodes with an epsilon of 0.2 meaning that 20% of the episodes would have next state actions chosen at random under the ϵ-greedy selection policy. The updated Q-matrix is used to determine the best path based on the highest rewards for each action at a given state.

A. Q-Learning Algorithm

To implement the Q-learning algorithm, the high-level steps outlined in Fig. 2 were used. As shown by the algorithm, a random state is chosen from the generated Q-matrix used to hold the values to determine the best path. Then an action is taken in the current state based on the ϵ-greedy selection policy. Using this action, the next state transition is determined as well as the reward for taking the action. These values then update the Q-matrix at the current state as described by (3). This is then repeated with the next state, until the current state is the goal. As shown in the algorithm, the Q-matrix at each state is updated based on a learning parameter, α which is set to 1 in this application to assure quick learning. This matrix also depends on a discount factor denoted by γ which is set to 0.8 to give importance to future rewards [2]. This was repeated for 10,000 episodes where at the start of each episode a random starting state was chosen.

B. Sarsa Algorithm

To implement the Sarsa algorithm, the high-level steps outlined in Fig. 3 were used. As shown by the algorithm a random state is chosen from the generated Q-matrix used to hold the values to determine the best path. Then an action is taken in the current state based on the ϵ-greedy selection policy. Then until the state present is not the goal, the action will be taken and the next state transition is determined as well as the reward for taking the action. Another action is chosen using

```
Choose a Q(s,a)
For each episode
    Set state s
    Choose action a from s using epsilon
        policy from Q
    while s is not goal
        Take action a, observe reward r
            and next state s'
        Choose action a' from s' using
            epsilon policy from Q
        Update Q-value as follows
        Q(s,a) <- Q(s,a) + α [r+γ Q(s',a')
            - Q(s,a)]
        set s to s'
        set a to a'
    Repeat until s is the goal state
```

Fig. 3: Sarsa Algorithm [2]

the ϵ-greedy selection policy for the next state. The Q-matrix at the current state is then updated as described in (4). As shown in the algorithm, the Q-matrix at each state is updated based on a learning parameter, α which is set to 1 to assure quick learning. This matrix also depends on a discount factor denoted by γ which is set to 0.3 to give importance to future rewards [2]. This was repeated for 10,000 episodes where at the start of each episode a random starting state was chosen.

IV. Results

To validate the effectiveness of the two algorithms, each was run with a 5x5, 10x10, 15x15, and 25x25 size world with randomly generated terrains. Each was run in 10 different terrain maps at each world size. The results from this can be found in Table I. As shown, it is apparent that the Q-Learning algorithm performs best in terms of cost. It chooses the shortest and safest path. Also, as the world size gets bigger, the Q-Learning algorithm takes advantage of the increase in space and terrain to lower its cost per move. Sarsa also does this, but only slightly. The Sarsa algorithm is the fastest, while it may not be the most efficient. It is shown to finish well before the Q-Learning algorithm on larger maps. However, the Q-Learning algorithm has a higher rate of convergence, where the Sarsa algorithm is prone to diverge at larger map sizes. In the 25x25 world size, the Sarsa algorithm failed to converge, increasing the discount factor for the algorithm at this world size to 0.8 allowed for convergence. Examples of paths found by the Q-Learning algorithm and Sarsa algorithm at a world size of 25x25 are shown in Fig. 4 and Fig. 5, respectively.

In consideration of natural disaster relief and rebuilding, the Q-Learning algorithm is the best. It takes the longest to converge, but will find the most efficient path, that is time conscience and safe for itself. Sarsa, tends to favor the goal over safety in these situations because of its exploitation over exploration strategy.

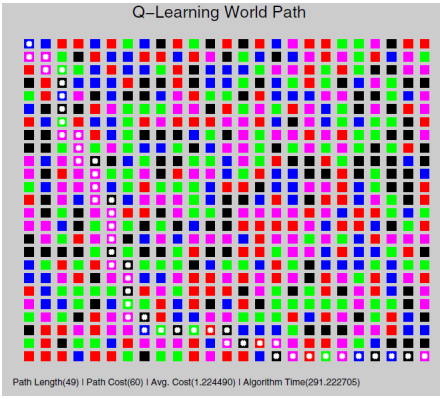

Fig. 4: Q-Learning Algorithm Path Example with Punishment

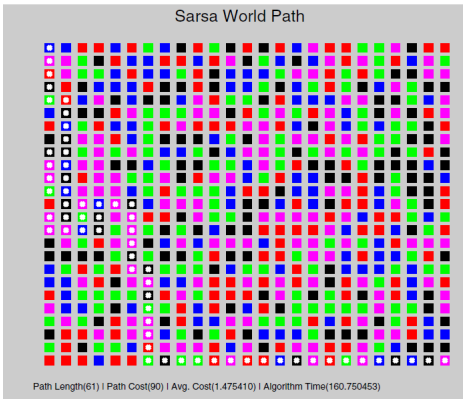

Fig. 5: Sarsa Algorithm Path Example with Punishment

TABLE I: Q-Learning and Sarsa results across several world map sizes

		Q-Learning	Sarsa
5x5	Avg. Path	9	9
	Avg. Cost	12.3	18
	Avg. Unit Cost	1.3667	2.000
	Avg. Time (s)	2.396	2.2731
	% Converge	100	100
10x10	Avg. Path	19	23.667
	Avg. Cost	23.4	38.8
	Avg. Unit Cost	1.2316	1.6260
	Avg. Time (s)	12.7255	12.2337
	% Converge	100	60
15x15	Avg. Path	29	60.75
	Avg. Cost	32.2	80.0
	Avg. Unit Cost	1.1103	1.3110
	Avg. Time (s)	37.8544	33.4482
	% Converge	100	40
25x25	Avg. Path	49	63.2*
	Avg. Cost	51.3	101*
	Avg. Unit Cost	1.0469	1.6005*
	Avg. Time (s)	337.8118	186.467*
	% Converge	100	50*
	*with a discount factor of 0.8		

V. Conclusion

Reinforcement learning is a series of methods and algorithms used to pseudo map out the way a living being makes decisions. Just like a living being, a decision cannot be proven right or wrong until it has been made [10]. This methodology gives way for a system to learn its environment and discover patterns not easily recognizable. Used in the case of natural disaster response or ruins exploration, reinforcement learning, specifically temporal difference learning, can be used to explore the area, as well as build an efficient navigation map. In terms of the experiments done in this paper Q-Learning does much better than Sarsa in creating an efficient and safe path from a start to a goal. Using multiple synchronized systems across a map, where each system would represent an episode, could reduce the time limitation found in Q-learning. In this type of problem, Q-Learning does the best because of the importance it places on exploration. On known, non-variable maps, Sarsa will do better because it can exploit rewards in future states early.

References

[1] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, pp. 58–68, Mar. 1995. [Online]. Available: http://doi.acm.org/10.1145/203330.203343

[2] T. Eden, A. Knittel, and R. van Uffelen, "Reinforcement learning," http://www.cse.unsw.edu.au/ cs9417ml/RL1/index.html, accessed: 2015-11-22.

[3] C. J. C. H. Watkins and P. Dayan, "Technical note: q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, May 1992. [Online]. Available: http://dx.doi.org/10.1007/BF00992698

[4] N. Sprague and D. Ballard, "Multiple-goal reinforcement learning with modular sarsa (0)," in *IJCAI*. Citeseer, 2003, pp. 1445–1447.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[6] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, pp. 1038–1044, 1996.

[7] J. P. O'Doherty, P. Dayan, K. Friston, H. Critchley, and R. J. Dolan, "Temporal difference models and reward-related learning in the human brain," *Neuron*, vol. 38, no. 2, pp. 329–337, 2003.

[8] J. Peng, "Mobile robot path planning based on improved q learning algorithm," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 10, no. 7, pp. 285–294, 2015.

[9] "How to implement epsilon greedy strategy / policy," https://junedmunshi.wordpress.com/2012/03/30/how-to-implement-epsilon-greedy-strategy-policy/, accessed: 2015-11-21.

[10] "Q-learning by examples," http://people.revoledu.com/kardi/tutorial/ReinforcementLearning/, accessed: 2015-11-15.