Fast Reinforcement Learning Techniques Using the Euclidean Distance and Agent State Occurrence Frequency

H. S. Al-Dayaa  
CMINDS Research Center  
Electrical and Computer Engineering Department  
University of Massachusetts Lowell  
Lowell MA, USA

D. B. Megherbi  
CMINDS Research Center  
Electrical and Computer Engineering Department  
University of Massachusetts Lowell  
Lowell MA, USA

Abstract – Reinforcement learning techniques like the Q-Learning one and the Multiple-Lookahead-Levels one that we introduced in our prior work require the agent to complete an initial exploratory path followed by as many hypothetical and physical paths as necessary to find the optimal path to the goal. This paper introduces a reinforcement learning technique that uses the Euclidean distance to the goal as a main measure for an autonomous agent’s action selection. As we show in this paper, no exploratory or hypothetical paths are required, there is no need to cover all possible states, and the agent requires a maximum of two physical paths to find the optimal path to the goal. The agent’s state occurrence frequency is introduced here and used to support the proposed Distance-Only technique. A computation speed performance analysis is carried out, and the Distance-and Frequency technique is shown to require less computation time than the Q-Learning one.

Keywords: Reinforcement Learning, Machine Learning, Artificial Intelligence, Robotics.

1 Introduction

Machine learning refers to a system capable of the autonomous acquirement and incorporation of knowledge. It is a system that can continuously self-improve and becomes more efficient as a result of learning from experience, analytical observation, and other means [1][12]. The Euclidean distance has been effectively used before in the Distance-Weighted Nearest Neighbor machine learning algorithm, an instance-based learning method that weights the contribution of each of the k neighbors according to their distance to the goal [2][4][7][9][12]. Equation (1) shows the real-valued target function of a query point \(x_q\) in this algorithm, where \(f(x)\) is the return value of a training instance \(x\), and \(w_i\) is the weight of the \(i^{th}\) k neighbor and inversely proportional to the Euclidean distance of that neighbor to the goal as shown in equation (2). In the literature including [2][4][7][9][12], the Distance-Weighted Nearest Neighbor algorithm assigns \(f^*(x_q)\) to be \(f(x_i)\) when \(x_q\) exactly matches one of the training instances and the denominator \(d(x_q,x_i)^2\) is therefore equal to zero, creating an exception to equation (1).

\[
\hat{f}(x_q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} 
\]

\[
w_i = \frac{1}{d(x_q,x_i)^2} 
\]

In this paper, we present and demonstrate how the Euclidean distance can also be used in reinforcement learning to result in a faster learning process that requires less iterations and processor run time. We tackle the problem of how an autonomous agent put in an environment that it does not initially know anything about can learn the optimal path (shortest path) to the goal. Although the goal state is not actually known by the agent, it acts as a magnetic field as shown in Figure 1. The effect of the field on the agent is inversely proportional to the Euclidean distance between the agent and the goal.

Figure 1 - Effect of the Goal on Agent

There are several machine learning applications to this technique; they range from data-mining programs, to error-detection systems, medical systems, to autonomous vehicles, to robots, etc. The type of environment the intelligent system acts in is different for each application. In this paper, we consider a generic environment \(E\) that consists of a 2-dimensional maze of possible states, one of which is the starting state marked with ‘S’, and one is the goal state marked with ‘G’. The states marked with ‘B’ are barriers and cannot be visited by the agent, and the others are accessible states. All the states are distinguishable by their x-y coordinates (x for horizontal and y for vertical) in the maze. From each state \(s_k\), the agent can take an action \(a_k\) from A, where A is the set of actions and contains UP, RIGHT, DOWN, and LEFT. Each action changes the
agent’s location accordingly, except where such action would take the agent into a barrier or outside the maze. The agent is rewarded a real value \( R(s_k, a_k) \) after each action \( a_k \), where \( k \) represents the x-y space coordinates defining the state \( s_k \). The main task \( T \) of the agent is to find the optimal path to the goal. The different illustrations and equations in this paper are based on this representation.

This paper is organized as follows: We illustrate the Distance-Only based technique in section 2 and the Distance-and-Frequency based technique in section 3. In section 4, we present a method to detect an unreachable goal. In section 5, we show the computation speed performance analysis of the Distance-and-Frequency based technique in comparison to the Q-Learning one.

2 Proposed Distance-Only Technique

The first contribution of this paper is a technique where the agent relies on the Euclidean distance to the goal only to take its actions and find the optimal path.

2.1 Proposed Algorithm

Let \( d_{k,G} \) be the Euclidean distance between from the agent’s current state \( s_k \) and the goal state ‘G’ as shown in equation (3). The distance reward \( D(s_k,a_k) \), equal to the \( R(s_k,a_k) \) reward in this case, is inversely proportional to the distance to the goal and is never zero as shown in (4).

\[
d_{k,G} = \sqrt{(x_k - x_G)^2 + (y_k - y_G)^2} \tag{3}
\]

\[
R(s_k, a_k) = D(s_k, a_k) = \frac{1}{d_{k+1,G} + 1} \tag{4}
\]

Note that the reward \( D(s_k, a_k) \) reaches its maximum value of 1 not 1/0 (different from equation (2) for the Distance-Weighted Nearest Neighbor method) when the agent reaches the goal and decreases when the distance to the goal increases.

The agent will take the action that returns the highest reward, hence the action that leads closest to the goal as shown in equation (5). Note that in order to avoid choosing an action that leads to a barrier, the distance from a barrier to the goal is set to a value greater than the maximum possible distance to the goal.

\[
a = \arg \max_{a_k \in A} R(s_k, a_k) \tag{5}
\]

Figure 2 depicts the Distance-Only algorithm flowchart.

2.2 Experimental Results

Figure 3 and Figure 4 illustrate two instances of running the Machine-Learning Program MLProg (a software utility we developed) using the Distance-Only technique. As described in the last paragraph of section Introduction, ‘S’ is the starting state, ‘G’ is the goal state, and ‘B’ is a barrier.

Note that the filled barrier states are the ones discovered by the agent as the latter does not need to discover every barrier or state in the environment to find the optimal path (shortest path) to the goal.

Figure 2 - Distance-Only Algorithm Flowchart

Figure 3 - First Instance Using Distance Only
3 Proposed Distance-and-Frequency Technique

Another contribution of this paper is an enhancement to the Distance-Only technique. It uses the agent’s state occurrence frequency (the number of times a state has been visited) in addition to the distance to goal. In this section, we also introduce and demonstrate the Closed-Loop Omitting (CLO) and Loop Pinching (LP) methods to further enhance the proposed Distance-and-Frequency based technique.

3.1 Need for State Occurrence Frequency

In some environments, using the distance to the goal only can lead to a deadlock in which the agent will never reach the goal although the goal is reachable. This situation happens in two cases:

1. The Dead End case: In this case, the agent keeps moving closer to the goal until it reaches a barrier causing a dead end. Figure 5 illustrates this case, where the agent keeps moving from the starting state at \([x=0,y=2]\) to \([1,2]\) to \([2,2]\) and reaches a dead end at \([4,2]\). Relying on the distance to goal only, the agent will always choose the state \([4,2]\) from \([3,2]\) but can never reach it because it is a barrier.

2. The Infinite Repetition case: In this case, the agent keeps repeating the same actions between two or more states seeking the closest possible state to the goal but never leaving this repetitive condition. Figure 6 illustrates this case, where the agent moves from the starting state at \([x=1,y=1]\) to \([2,1]\) to \([2,2]\) then keeps moving between \([2,1]\) and \([2,2]\) an infinite number of times. Relying on the distance to goal only causes a deadlock because \([2,2]\) is the closest possible state to the goal from \([2,1]\), and \([2,1]\) is the closest possible state to the goal from \([2,2]\).

3.2 Proposed Algorithm

The agent’s state occurrence frequency is used in addition to the Euclidean distance to the goal in order to avoid the two deadlock situations described earlier. Let \(v_k\) be the number of times a state has been visited. The reward \(R(s_k, a_k)\) in this case is a function of both the distance reward \(D(s_k, a_k)\) and visited reward \(V(s_k, a_k)\) where \(V(s_k, a_k)\) is inversely proportional to \(v_k\) as shown in equations (6) and (7). The reward equation is shown in equation (8).

\[
V(s_k, a_k) = \frac{1}{v_{k+1} + 1} \quad (6)
\]

\[
R(s_k, a_k) = f(s_k, a_k) \quad (7)
\]

The function \(f\) is represented in equation (8) where \(V\) is the set of four visited weight values from (6), and \(D\) is the set of the four distance weight values from (4). As in the Distance-Only technique, the agent chooses the action that returns in the highest reward value, as shown in (9).

\[
f(s_k, a_k) = \max_{a_{k+1}} \left( \max_f (s_k, a_k) \right) \quad (8)
\]

\[
a = \arg\max_{a_k \in D} (R(s_k, a_k)) \quad (9)
\]

The function \(f\) denotes that the agent takes the action that leads it to the state that is the closest to the goal among the least visited ones as illustrated in the algorithm flowchart shown in Figure 7.
1.a. From current state $s_k$, group the possible states in terms of occurrence frequency.

1.b. $V$ is the set of possible states that are the least visited (i.e., the set of states with the highest visited rewards).

2.a. Choose from the set $V$ the state $s_{k+1}$ that is closest to the goal (i.e., the state with the highest distance reward).

2.b. Take action that leads from current state $s_k$ to $s_{k+1}$ using the algorithm illustrated in the Distance-Only flowchart in Figure 2.

**Figure 7 - Distance-and-Frequency Algorithm Flowchart**

Lemma 1: The proposed Distance-and-Frequency algorithm depicted in Figure 7 avoids deadlocks and lets the agent escape Dead End and Infinite Circle cases under any conditions in an environment $E$ as defined earlier.

Proof: Let $s_{k+1}$ be the state that is closest to the goal and faced by a barrier. Using the Distance-and-Frequency algorithm, the agent has two possible states to consider from this state: the state $s_k$ it moved from to $s_{k+1}$ and the barrier. It will pick $s_k$, because it was less visited (the state occurrence frequency of a discovered barrier is set to a value higher than the maximum possible one) which allows it to back up from the deadlock path faced by the barrier. Q.E.D.

3.3 Closed-Loop Omitting (CLO) Method

The Distance-and-Frequency technique lets the agent avoid being stuck in dead ends and infinite circles, but this leads to a longer path in most cases. A second physical path is required in this case for the agent to find the shortest path to the goal. Since escaping dead ends and infinite circles means the agent has to back up to a state it already visited and goes toward the goal from there, it was necessary to omit the closed loop created by such situation. The optimal path to the goal can be generated from the first one after omitting any closed loops in it using the CLO method as shown in Figure 8.

**Figure 8 - Closed-Loop Omitting Method**

3.4 Loop Pinching (LP) Method

In some environments, Closed-Loop Omitting as discussed in the previous section is not enough to lead to the optimal path, such as illustrated in the case of Figure 9. Hence, Loop Pinching should be applied on the first path (applying LP after CLO is preferable). It results in removing unnecessary states that make the agent’s path longer. The LP method functions as follows: in a current state $s_k$, the already-visited state that can lead to $s_k$ and is closest to the starting state $S$ will be the state that leads directly to $s_k$. Figure 9 further illustrates the LP method.

**Figure 9 - Loop Pinching Method**

3.5 Experimental Results

The experimental results in this section demonstrate how the Distance-and-Frequency technique helps the agent avoid dead ends and infinite circles and how the CLO and LP method enhance this technique and leads the agent to find the optimal path in the second physical path.

As stated earlier, the filled barrier states are the ones discovered by the agent.

3.5.1 Dead End with CLO

The following experiment shows how an agent backs up from a dead end using the Distance-and-Frequency technique and how the CLO method results in the optimal path. Figure 10.a shows how the agent is stuck in a dead end when using the Distance-Only technique. Figure 10.b shows how the agent leaves the dead end using the Distance-and-Frequency technique, and Figure 10.c shows how the optimal path is reached using the CLO method.
3.5.2 Infinite Circle with CLO and LP

The following experiment shows how an agent avoids being stuck in an infinite circle using the Distance-and-Frequency based technique and how the LP method results in the optimal path. Figure 11.a shows how the agent leaves the circle using the Distance-and-Frequency based technique; Figure 11.b shows how applying the CLO method and how it did not change because there were no closed loops; Figure 11.c shows the path after applying the CLO method and how it did not change because there were no closed loops; Figure 11.d shows how applying the LP method results in the optimal path.
4 Unreachable Goal Detection

If the goal is unreachable, the agent will reach a deadlock trying to reach it. Another contribution in this paper is a method that helps the agent detect if the goal is unreachable.

Lemma 2: If the agent re-visits a state $s_{k+1}$ taking the same action $a_k$ it used when it visited that state before, then the goal is unreachable.

Proof: Assume the goal is reachable and the agent re-visits a state $s_{k+1}$ taking the same action $a_k$ it used when it visited that state before. Basing the agent’s action on both the Euclidean distance to the goal and state occurrence frequency, if the goal was reachable, the agent would have reached it after taking action $a_k$ to state $s_{k+1}$, and it is absurd that agent did not reach it the first time. Therefore, the goal is unreachable. Q.E.D.

To guarantee that this method works in all environments, the No-More flag is added to give certain states less weight than other possible ones but higher weight than barriers. A state is flagged with No-More if all possible states from it are either visited or barriers.

Running the MLProg software utility with this method being enabled in the environment in Figure 12.a leads the agent to decide and display that the “Goal is Unreachable!” Figure 12.b shows the agent path before deciding that the goal is unreachable. In the path, the agent moves UP from the starting state S at $[x=2, y=3]$ to $[2,2]$ then DOWN to $[2,3]$ then RIGHT to $[3,3]$ then RIGHT to $[4,3]$ then UP to $[4,2]$ then RIGHT to $[5,2]$ then UP to $[5,1]$ then UP to $[5,0]$ then RIGHT to $[6,0]$ then DOWN to $[6,1]$ then DOWN to $[6,2]$ then DOWN to $[6,3]$ then LEFT to $[5,3]$ then UP to $[5,2]$ then UP to $[5,1]$. Moving UP to $[5,1]$ is an action the agent took before when it first visited $[5,1]$; hence, the goal is unreachable.

5 Computation Speed Performance Analysis

What is also very important is that these techniques result in fast learning and optimal-result finding processes. The plots in Figure 13.a and their exponential curve fits in Figure 13.b illustrate a comparison in CPU run time between the proposed Distance-and-Frequency based technique and traditional Q-Learning technique for different environment areas where the area is the number of rows multiplied by the number of columns.

Analysis was done using a statistical analysis measure [6] that calculates the area under the curve for each technique and compares it with the other(s). The exponential curve-fit version of the plots $f(x)$ and its area are illustrated in (10). Since all curves have a fixed range of environment areas (minimum environment area $\text{Min} = 56$, and maximum environment area $\text{Max} = 169$ in this experiment), the area represents the overall CPU time for each curve. Hence, a smaller area means a faster process overall.
In our future work, we will incorporate these techniques in multi-agent environments and analyze the effect of a number of characteristics on knowledge acquisition among different agents.

7 References


