Maximizing Local Outlier Factors As an Objective for Exploration in Reinforcement Learning

Dylan R. Ashley

2017-12-22

Abstract

We borrow the idea of Local Outlier Factors from Unsupervised Learning as a means of encouraging a Reinforcement Learning agent to explore uniformly. We show how this can be done and how it can be supplied to the agent as a simple reward signal. We also provide experimental results on a modified gridworld domain which gives strong evidence that this is a useful way of incentivizing uniform exploration. Finally, we discuss the key problems that have to be solved for this to be a practical method and how it can be further improved.

1 Introduction

Reinforcement Learning deals with an agent trying to maximize some reward signal by interacting with an environment. In most cases, the agent is given little to no knowledge about the environment and so must explore the environment to understand how to maximize this signal. The task of finding a good way of interacting with the environment is known as the *exploration problem*.

We propose a way to use Local Outlier Factors from Unsupervised Learning as a way of incentivizing an agent to explore the environment in a uniform manner. This uniform method of exploration is intrinsically different to exploring the environment randomly. Randomly exploring an environment is likely to entail spending much more time in easily accessible parts of the environment and much less time in remote areas of the environment whereas a uniform exploration strategy should spend roughly equal time in both. In domains where it essential that the agent visits remote parts of the environment, a uniform exploration policy will be significantly better than a random exploration policy.

Finding a good exploration policy has long been an important problem in Reinforcement Learning, so a decent amount of work has been done in this area. For example, Thrun considered the question of whether exploration should be directed or undirected [6]. Some time later Barto considered what the analogy of the psychological concept of intrinsic motivation, which is deeply related to the idea of exploration, would be in Reinforcement Learning [1]. More recently Bellemare et al. developed a novel method of guiding exploration using pseudo-counts [2]. However, to our knowledge, the idea of using outliers and Unsupervised Learning as a means of guiding exploration is as of yet unexplored.

This work deals with two intersection areas: Reinforcement Learning and Unsupervised Learning. This intersection means that to understand this work a reader first needs to understand both Local Outlier Factors and Reinforcement Learning. Therefore we go through the necessary background on what Reinforcement Learning is in Section 1.1 and then what Local Outlier Factors are in Section 1.2. After that, we elaborate on our proposed method in Section 2. Following that, we perform an empirical evaluation of our method in Section 3. Finally we conclude in Section 4.

1.1 Reinforcement Learning

Reinforcement Learning [5]is a branch of machine learning that considers the problem of an agent interacting with an environment to try and maximize some reward signal (see Figure 1). Specifically, at each discrete timestep t, when the environment is in state S_t, the agent takes an action A_t, and in response, the environment changes its state to a new state, S_{t+1}, and rewards the agent with some real number, R_{t+1}.

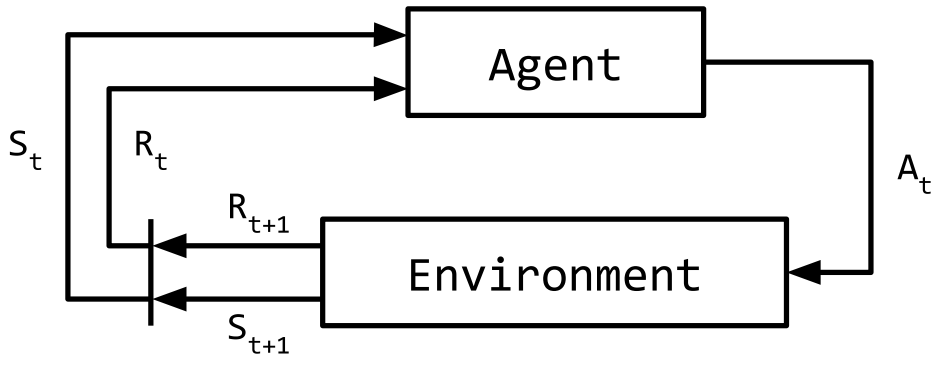

Figure 1: Visualization of the Reinforcement Learning Problem

We consider two kinds of domains in Reinforcement Learning: episodic and continuing domains. In episodic domains, the agent begins in some start state and continues interacting with the environment until it reaches some terminal state. The objective of the learning is then to find a set of rules, or a *policy*, by which it selects an action to take in the current state such that the expected total reward it receives over the course of an episode, or the expected value of the *return*, is maximized. We write the return for an episodic task with n total timesteps as follows:

$$G_t = \sum_{i=t}^{n} R_i \tag{1}$$

While this definition suffices for episodic domains, in continuing domains there are no terminal states, but instead, we consider rewards to be better when they are received sooner. Specifically, when the agent is picking an action for the timestep t, it considers a reward R_{t+k} to be worth $\gamma^{k-1}R_{t+k}$ where $\gamma \in [0,1)$ is known as the *discount factor*. In this domain we write the return as follows:

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t-1} R_i \tag{2}$$

The objective of the learning in a continuous domain is then to find a policy by which it selects an action to take in the current state such that the expected sum of discounted rewards, or the expected value of Equation 2, is maximized. The objective of learning in a continuous domain would be identical to the objective of learning in an episodic domain if the discount factor is equal to 1 and the episode had infinite length, both of which are not generally allowed in this dichotomy. Using the definitions of return as mentioned above, we can formalize what it means for a state, or the act of taking an action in a state, to be valuable.

We say the value of a state when the agent is following some policy is the expected return after entering that state. We write the value of a state S at time t while the agent is following a policy π as follows:

$$v_\pi(S_t) = \mathbb{E}_\pi\left[G_t\right] \tag{3}$$

The value of a state tells us how "good" it is to be in a state if we want to maximize the return. However, without a model of transitions between states, it does not give us any knowledge of how to pick a "good" action in a given state. Because of this, many learning algorithms instead learn the *action value* of states. The action value of a state S and action A at time t while the agent will follow a policy π for all actions taken after time t is denoted as follows:

$$q_\pi(S_t, A_t) = \mathbb{E}\left[R_{t+1}|S_t, A_t\right] + \gamma\mathbb{E}_\pi\left[G_{t+1}|S_t, A_t\right] \tag{4}$$

One intuitive way to use action values is to modify our policy when the action value under our policy of the action the policy would take is less than the action value under our policy of some other action. When we do this its called *policy iteration* and, in practice, it frequently converges to the optimal policy as long as there is no action in any state that we have a zero probability of selecting. If there is such an action, there's a chance that we could perhaps miss some rare but valuable reward. An easy way to prevent this is to pick the "best" action most of the time but take a random action on occasion. This method of picking actions is known as an *ϵ-greedy* policy, and many algorithms can compensate for it and learn the values of states or action values of state action pairs as if the agent were actually following a true greedy policy. One example of an algorithm that learns action values and can correct for an ϵ-greedy policy is Q-learning. While many algorithms can compensate for an ϵ-greedy policy, many other algorithms can compensate for the agent following some arbitrary policy while learning about some arbitrarily different policy.

Reinforcement Learning algorithms that learn about a different policy than the policy they use to pick actions are known as *offpolicy* algorithms. These are especially important algorithms as it is often essential to be able to explore intelligently to learn about an environment rapidly and these algorithms allow complicated exploration behavior while still being able to formulate an optimal policy as the agent explores. However, while exploration is an important aspect for rapid learning, it is important to note that good exploration of the environment and a good learning algorithm alone are often not sufficient for learning to progress quickly in any Reinforcement Learning domain.

In addition to having algorithms that compensate for some deviation in the policy of the agent, we also have algorithms where the agent need not be provided with the true state of the environment but can instead provide some vector which describes the current state. This vector is known as a *feature vector* and an example of an algorithm that can learn with feature vectors is, again, Q-learning. There are various methods to create a feature vector from the true state in a way that makes learning easier. However, the performance of most methods of these methods are heavily dependent on the domain. One simple example of such a method that works for cases where the state is described by a continuous-valued hyperplane is *tile coding*. The general idea of tile coding is to reduce the dimensionality of a problem by overlaying the hyperplane with a series of oversized grids. Each grid has the same dimensionality as the hyperplane, but each subsequent grid is shifted slightly in relation to the previous grid. The indices of the cells over the grids of which a state falls in are then used to describe the state to the learning algorithm. The purpose of having multiple grids is so that states that are somewhat close to one another will appear in the same tile on some grids but different tiles in other ones. For more information on tile coding or any of the other previously mentioned details about Reinforcement Learning see *Reinforcement learning - an introduction* by Sutton and Barto [5].

Before we conclude this brief overview of Reinforcement Learning, there are a few small but important additional but advanced details about the Reinforcement Learning problem and the prior description of it that have so far been omitted for clarity but which we include here for completeness. Most notably, in Reinforcement Learning, we enforce the constraint on domains that they must retain the *Markov Property.* That is, we require the following to hold in the domain:

$$\mathrm{P}\left(R_{t+1}, S_{t+1} | S_t, A_t, S_{t-1}, A_{t-1}, ...\right) = \mathrm{P}\left(R_{t+1}, S_{t+1} | S_t, A_t\right) \tag{5}$$

In other words the reward R_{t+1} and next state S_{t+1} is conditionally independent of all but the last state and action given the last state and action (S_t, A_t). Another interpretation of this is that knowing the full history of the interactions between the agent and environment does not give us any more relevant information we could use to predict the reward and next state than knowing only the last action taken and the last state. The enforcement of the Markov Property on the domain is the most significant detail that was omitted however there is one other important detail that was omitted.

Another detail that was previously omitted was that, in contrast to the Markov Property, it is not an explicit requirement of the Reinforcement Learning problem that timesteps be discrete. We do not handle the alternate case here however as most algorithms in Reinforcement Learning assume we are working with discrete timesteps.

1.2 Local Outlier Factor

Local Outlier Factors, or LOFs, are a metric used to find outliers in a dataset. First introduced by Breunig et al. [3], they estimate the density of the space around a data point and compare it to an equivalent estimate of the density of the space around nearby points. Comparing points to nearby points makes this a locality-based outlier metric and allows it to easily handle variable density clusters, unlike most global outlier metrics. To estimate the density of the space around a point, the LOF method simply calculates the average *reachability distance* to its k nearest neighbours (k-NN) where the reachability distance from a point a to b is simply the maximum of the distance between the points and the distance from point b to the most distant point in its k-NN, or its k*-distance*. The LOF of a data point is then just the ratio of the estimate of the density of the space around that data point, or the *local reachability density* of that point, to the average local reachability density of each of that point's k-NN. We formalize this using the following set of equations:

$$k\text{-distance}(p) = \max_{q \in k\text{NN}(p)} d(p, q) \tag{6}$$

$$\text{reach-dist}_k(p, q) = \max\{\text{d}(p, q), k\text{-distance}(q)\} \tag{7}$$

$$\text{lrd}(p) = \frac{1}{\frac{1}{k}\sum_{q \in k\text{NN}(p)} \text{reach-dist}_k(p, q)} \tag{8}$$

$$\text{LOF}(p) = \frac{\frac{1}{k}\sum_{q \in k\text{NN}(p)} \text{lrd}(q)}{\text{lrd}(p)} \tag{9}$$

Of crucial importance in the above equations is the fact that computing the LOF of a data point requires a combined total of $k + 1$ different k-NN searches. However, when computing the LOF for a data point, only the k-NN searches can be dependent on the number of data points in the dataset. This fact, along with the locality property of the LOF method, makes it suitable for data streams [4] which is the norm in Reinforcement Learning problems.

2 Methods

If we want a Reinforcement Learning to explore the environment uniformly, we ideally want to reward the agent for seeing parts of the environment it has not seen frequently. Local Outlier

Factors provide a scalar value for each data point that indicates to what degree each point is an outlier. However, an outlier is a rare data point. So it seems natural that we need only reward a Reinforcement Learning agent with the value provided by Local Outlier Factors to encourage good exploration. This idea is the intuition behind why we could use outliers as a way of guiding exploration. There are, however, a few questions that must be answered to determine how this can be done. Firstly, we must determine what represents parts of the environment so that we can determine what we can use as data points for the LOF method. Secondly, we must determine what constitutes similarity when referring to two parts of the environment so that there is a dissimilarity metric for the LOF method. Lastly, we must determine how we can adapt the values produced by the LOF method so that it can serve as a suitable reward for an Reinforcement Learning agent. We provide answers to these three questions in the following paragraphs.

The first problem is the easiest: to represent parts of the environment we need only use the feature vectors as data points. However, this raises the question of whether uniform exploration means that all states are visited roughly equally or whether it means that all transitions between two states are experienced roughly equally. Both are valid interpretations, and for the latter case we can simply append the feature vector for the next state to the feature vector for the current state and use this as coordinates for locations in the environment. One advantage of these as ways of representing parts of the environment is that both of these representations have an equally straightforward dissimilarity metric.

In Reinforcement Learning similar states should, in general, have similar feature vectors. Hence the term feature in feature vector. So using the simple Euclidean distance between the vectors is a good choice for a dissimilarity metric of two states. While this is not the only metric that could be used, it is the most straightforward one. It also works equally well if we opt to use transitions rather than states. While this resolves one problem, there is still the problem of how the LOF value must be adapted, so it is suitable as a reward for an Reinforcement Learning agent.

The reward signal in Reinforcement Learning is interpreted the same way we consider rewards in our lives. In other words, a positive reward indicates a good action, a negative reward indicates a bad action, and a reward of zero indicates a neutral action. Furthermore the value it outputs for each data point is a positive value where values around 1 indicate a data point is not an outlier and values much greater than one indicate a data point is an outlier. So to make LOF values suitable for an Reinforcement Learning agent, we begin by subtracting one so that no reward would be given to the agent if it visits an area of the environment it has seen frequently. To reduce the noise in the resulting signal, we opt instead to use the exponential of the absolute value of the new signal so that the reward given to the agent is roughly equal when exploring well-explored areas of the environment and the reward given to the agent is substantial when exploring badly explored areas. While using the absolute value treats different LOF values differently it is unimportant as the exponential effectively suppresses values affected by this. While using the exponential has benefits, it again rewarded the agent for visiting an area of the environment it has frequently seen so we must again subtract one from the result to produce our reward signal. We write the resulting reward signal with states as data points as follows:

$$R_t = e^{|LOF(S_t)-1|} - 1 \tag{10}$$

If we instead use transitions as data points we write the resulting reward signal as follows:

$$R_t = e^{|LOF([S_{t-1},S_t])-1|} - 1 \tag{11}$$

The most significant advantage of this method is that it merely produces a reward signal that helps to guide exploration. Since it merely produces a reward signal, it can be used with any algorithm in the Reinforcement Learning literature and can be easily combined with other methods of guiding exploration. For example we could consider a three-tiered system where the agent selects a greedy action with respect to the real expected reward with $1-\epsilon$ probability, a greedy action with respect to the LOF based reward signal with $\epsilon(1-\epsilon)$ probability, and a random action with ϵ^2 probability. However, as it modifies the reward signal, to learn an optimal policy with under the true reward signal it is necessary to use two learners. Here one learner learns the values of the states or state action pairs under the LOF based reward signal and one the other learner learns the values of the states or state action pairs under the true reward signal. Using two learners in this way implies that the learner that learns using the true reward signal must use an offpolicy algorithm. This is a requirement of most exploration strategies and such algorithms are abundant. Furthermore while this effectively double the amount of computation required, calculating the LOFs of states or transitions is significantly more expensive than all but the most expensive such algorithms. There remains a way to alleviate this computation overhead somewhat though.

While we do not explore it further, it is not essential that the LOF based reward we use to be calculated with respect to all previous states visited or transitions experienced. Indeed we could choose to calculate it using only the newest n data points. Using only the n newest data points would have two key advantages. Firstly, it would allow us to deal with environments that change as time progresses. Secondly, it would bound the amount of computation required for calculating the reward signal to give to the agent. Both of these remain important considerations in Reinforcement Learning. However, such an improvement comes at a price. In this case, a key disadvantage of this extension is that it introduces a new hyperparameter that must be tuned.

3 Experimental Evaluation

We begin by discussing the domain we experiment with in Section 3.1. We then describe the Reinforcement Learning algorithms we use with the LOF based reward signal in Section 3.2. After that, we discuss how we calculate the k-NN of states and transitions in our experiments in Section 3.3. Following that, we explain and justify the error metric we use to describe the deviation from uniform exploration in Section 3.4. Finally, we report and discuss our experimental results in Section 3.5.

3.1 Domain

For the experimental evaluation, we use the gridworld domain shown in Figure 2. The gridworld has 25 states arranged in a 5 x 5 grid. We label the states $(0, 0)$ through $(4, 4)$ based on their position in the grid. The gridworld has exactly one start state for each episode in the north-west corner (i.e., state $(0, 0)$) and one terminal state which is the center-most state (i.e., state $(2, 2)$). There is no discounting factor in this domain, and the reward at each timestep is always -1. In each state there are four actions: $North$, $South$, $East$, and $West$. Taking one of these four actions will deterministically move the agent one cell in that direction. For example taking the action $North$ in state $(1, 1)$ will result in the agent moving to state $(1, 0)$. If the agent attempts to move outside the gridworld, say by taking action $East$ in state $(4, 0)$, then the agent does not move at all as if the gridworld were surrounded by walls and the agent had simply walked into a wall.

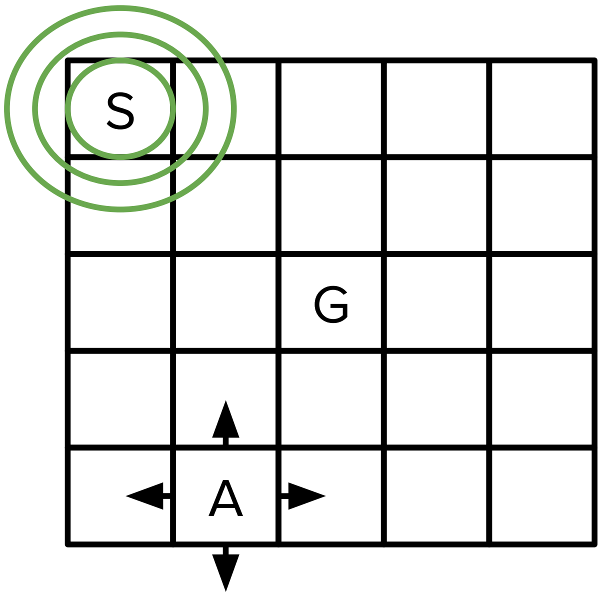

Figure 2: Visualization of the Domain Used for Experimental Evaluation

As an activity to further understand this domain, we consider how to construct an optimal policy. Assume, without loss of generality, that the agent is in state (x, y) with $x \leq 2$ and $y \leq 2$. Then an optimal policy takes action $East$ a total of $2 - x$ times and action $South$ a total of $2 - y$ times. The order of the actions does not matter which leads to there being exponentially many optimal polices. Note that if $x > 2$ then instead of taking the action $East$ a total of $2 - x$ times it takes the action $West$ a total of $x - 2$ times. Similarly, if $y > 2$ then instead of taking the action $South$ a total of $2 - y$ times it takes the action $North$ a total of $y - 2$ times. While constructing an optimal policy is straightforward there are some issues in using this domain as is with the proposed exploration method.

There are two critical issues in using this domain as a basis for evaluating the proposed exploration method. Firstly, it is a tabular domain and so would fail to demonstrate both the ability of the proposed method to handle the linear function approximation case and the ability of the proposed method to handle continuous values natively. Secondly, the local reachability density of a point diverges to infinity when it and its k-NN all have a pairwise distance of 0. Resolving this issue in the proposed exploration method is outside of the scope of this work and is brought up as a topic for future consideration in Section 4. Because of the issues mentioned above regarding reachability distances, we deny the agent access to the underlying state and instead provide it with a modified state. We call this modified state an observation and refer to the original state as the true state. It is important to note that, from the perspective of the learning algorithm, the observation is the state.

Instead of receiving the true state of the gridworld the agent receives an observation which, from its perspective, is the state of the environment. To accomplish this, we overlay the grid of true states with a 2D plane with a continuous-valued x and y axis both going from 0 to 1. Each cell of the original grid then occupies one-fifth of the x axis and one-fifth of the y axis. For example the cell $(0, 0)$ occupies both x values and y values in the range $(0, 0.2)$. The cell $(1, 3)$ occupies x values in the range $(0.2, 0.4)$ and y values in the range $(0.6, 0.8)$. The observation the agent receives from the domain upon entering a cell is drawn from a 2D truncated Gaussian distribution over the plane and centered at the middle of the cell the agent is in. All truncated Gaussian are bounded with x and y values both between 0 and 1. The truncated Gaussian distribution has a standard deviation equal to 0.2 or the amount of space in both the x and y axis a single cell maps to. Evidently, this adds a layer of noise to the observations the agent receives.

The form of the observations the agent receives causes the modified gridworld to be significantly more challenging to learn than the unmodified version. Notably, this creates ambiguity in the true location of the agent even with perfect function approximation. Perhaps more interesting than the ambiguity in the position of the agent, if one naïvely translates an optimal policy from the original domain and uses a single layer of 5 x 5 tile coding as function approximation then the resulting policy is not necessarily an optimal policy in the modified domain. For example consider Figure 3 where the green dot represents an observation. If the agent is indeed in cell $(1, 1)$ then both the *South* and *East* actions are optimal actions. However in cell $(2, 1)$ the only optimal action is *South*, and in cell $(1, 2)$ the only optimal action is *East*. As the agent is more likely to receive this specific observation if it is in cell $(2, 1)$ than if it is in cell $(1, 2)$ the only optimal action for the agent to take is *South*.

3.2 Learning Algorithms

We use the Q-learning algorithm in all of our experiments because it is a well known and common offpolicy Reinforcement Learning algorithm. When using it for exploration, we use a step size of

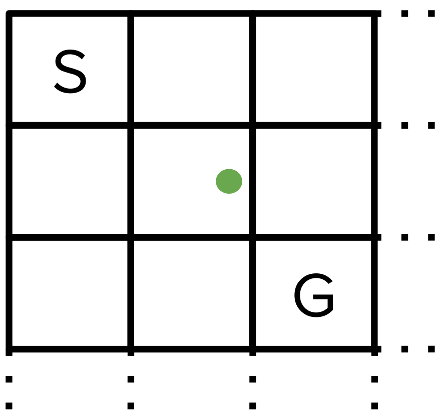

Figure 3: Visualization of the Potential for an Optimal Policy to Become Sub-optimal in the Modified Domain

0.5, a trace decay value of 0.9, and a discount factor of 0.9. We use one layer of 10 x 10 tile coding as function approximation. We also employ a form of ϵ-greedy exploration where, at each timestep, the optimal action with respect to our LOF based reward is taken with 90% probability, and a random action is taken instead with a 10% probability. All the above values are selected after trying a few values. While a more fine sweep may provide additional performance, we are limited by computational resources, and this is sufficient to achieve the desired results. We show that the above is adequate for use with the domain from Section 3.1 by showing that is learnable with this algorithm empirically.

To show that the domain is learnable, we use Q-learning under the same conditions as above except that we use the true reward signal, a step size of 0.01, and the true discount factor of the domain. We run this for 10000 episodes and note, for each episode, what percentage of all actions taken within and before this episode were optimal actions. The results are displayed in Figure 4. This environment has stochasticity introduced by the method in which observations are generated, so an optimal policy will not necessarily make the correct action at each timestep. However the fact that the percentage of correct actions taken increasing implies that the Q-learning agent is learning a better policy and thus we conclude this domain is learnable with the Q-learning algorithm.

3.3 Finding the k-NN

The most computationally expensive part of our experiments is finding the k-NN of data points. We use two alternate methods based on if we are using states or transitions. When working with transitions, we use the naïve method where we merely sort all data points and select the closest k for the k-NN of a data point. The naïve method has the advantage that it is only lightly affected by dimensionality and requires no preprocessing. However, this method takes a considerable amount of time as the number of data points increases as it is in O(nd) for n data points of d dimensionality. Because of this, we use a more efficient method when using the lower dimensional states.

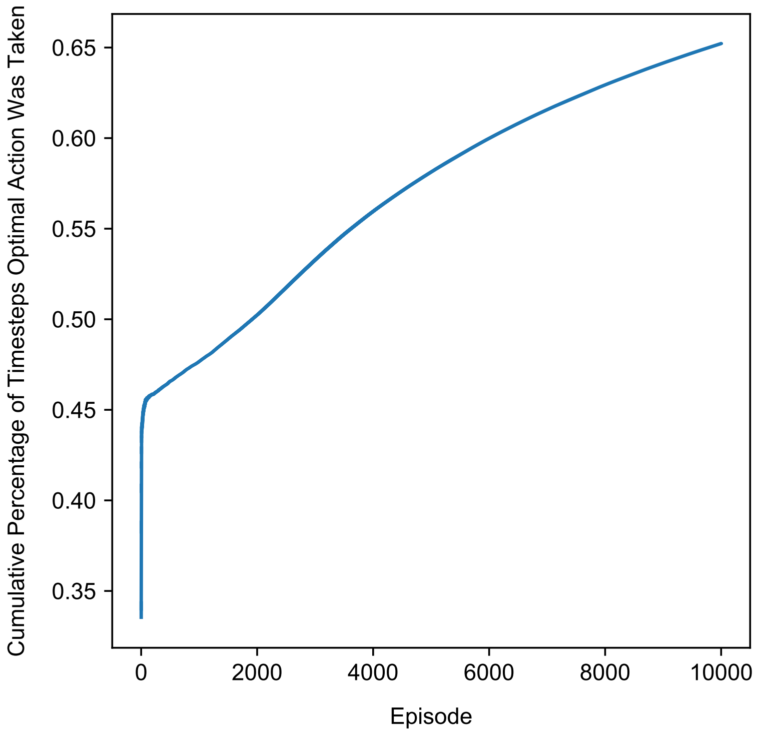

Figure 4: Percentage of Optimal Actions Taken Using Policy Iteration

When working with the lower dimensional states, we overlay the x and y axis in the domain with a 10 x 10 grid with equal sized cells similar to the one shown in Figure 5. We then precompute the distances between all cells in the grid. We define the distance from a cell to itself to be zero. For two different cells, we define the distance between the two cells that are dx cells apart in the x direction and dy cells apart in the y direction to be $dx+dy$ if $dx \neq dy$ and $dx+dy-1$ otherwise. This distance metric is equivalent to creating increasingly sized hypercubes around the cell of interest and stating that the distance to each other cell is the minimum size of the hypercube that contains the other cell. Computing the pairwise distances between all cells is only a part of the necessary precomputation to make this grid-based method efficient.

For both the naïve method and the grid-based method we must collect $k+1$ data points before we can calculate the LOF for any data point. In our experiments, we pick a k of 20 after trying a few values and then collect $k+1$ data points by randomly moving around the environment. While a more fine sweep over k values may provide additional performance, we are limited by computational resources, and this is sufficient to achieve the desired results. It is also important to note that we ignore this additional behavior when reporting our results in Section 3.5. After collecting enough data points, we can calculate the k-distance and k-NN of cells. We define the k-distance of a cell to be the upper bound for the distance to another cell that could contain a data point in the k-NN of a point in the original cell. We similarly define the k-NN of a cell to be the set of cells such that the distance between those cells and the original cell is no more than the k-distance of the original cell. As a simple example consider a cell near the center of our 10 x 10 grid where the cell

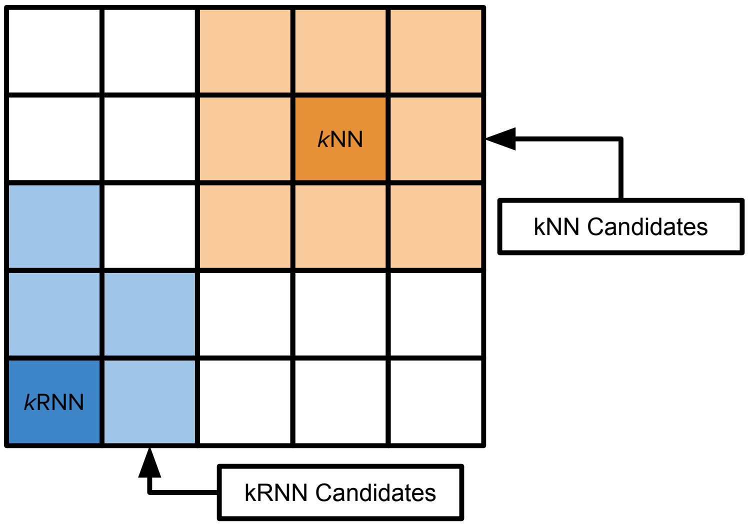

Figure 5: Visualization of the k-NN Grid

contains $k + 1$ data points. We would say the k-distance of that cell is 1 because the only cells that could contain data points in the k-NN of a point in the original cell is the original cell itself and the surrounding nine cells. The original cell and the surrounding nine cells would then form the k-NN of the original cell. The idea behind the grid is that when finding the k-NN of a data point we only need to search cells in the k-NN of the cell the data point appears in. Furthermore, we can precompute the k-distances and k-NN of cells and update them as new data points are added.

When adding a new data point some of the k-distances and, by extension, the k-NN for cells may change. Of crucial importance is that these k-distances can only decrease. We have two methods of determining which cells may have their k-distance changed when a new data point is inserted. We could iterate over cells and find those that have that cell within their k-distance or we can precompute the k-RNN of cells and update them as needed. Updating them entails storing, for each cell, a list of cells that have that cell in their k-NN. When a data point is added, we search through each of those cells and determine if the k-distance of any of them has changed. If the k-distance for a cell has changed, we have to go through and update the k-RNN for each cell that is no longer in the k-NN of that cell. This operation will be frequent with few data points but will become less frequent as the number of data points increases. So this method provides a trade-off where the computation time for the first few data points is significantly higher than the naïve method, but it becomes considerably lower than the naïve method as time goes on. This method has an additional advantage though in that it can help with the k-RNN of data points as well.

If we choose to precompute the k-RNN of cells, then to find the k-RNN for a data point in the grid one only has to look at the cells in the k-RNN of the cell the data point appears in. This abstraction to cells again exploits locality to reduce the number of data points that must be searched through to find the k-RNN of a data point. Figure 5 gives some visual intuition about the locality property.

3.4 Error Metric

As the objective is to achieve roughly uniform exploration of the state space, we use the deviation from uniform exploration as an error metric. Specifically, with V_{S_i} being the total number of visitations to the true state S_i, we use the standard deviation over all V_{S_i} as the error metric:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{|S|} \left(V_{S_i} - \overline{V}\right)^2}{|S| - 1}} \tag{12}$$

Using this as an error metric has a few advantages. Firstly it adequately captures when visitations to states are not equal. Secondly, it penalizes both a disproportionately large number of visitations to a state and a disproportionately small number of visitations to a state more severely than a minor difference in visitation between many states. This distribution of penalization is desirable as it captures the desire to maximize how often we see rare events which is precisely what we want the agent to seek.

3.5 Results

Using the above, we compare and contrast our method with random exploration over 25 episodes. We also compare our method which uses the Local Outlier Factors as an incentive with the similar method of using only the much less expensive computationally expensive k-distance (see Section 1.2) as an incentive. We compare both the LOF based method and the k-distance based method using states and transitions as data points. Due to the variable computation time for each of the methods we report the average of 1000 independent runs of random exploration, 500 independent runs of the state and transition variants of the k-distance based exploration, and 300 independent runs of the state and transition variants of the LOF based exploration. With the data from these runs, we now discuss our key results.

For our key results, we consider cumulative error after an episode to be the standard deviation of all state visitations before and within the episode. We further divide the cumulative error after an episode by the number of total visitations before and within the episode. By dividing by the total visitations, we get a metric for how evenly the agent visits the states over time. The result of this is shown in Figure 6. There are some important things to note here. Firstly, after a short amount of time has passed, every method we proposed, with the expectation of the k-distance based method when it uses transitions, appears to be outperforming random exploration. This increase in performance provides the proof of concept that this method works to some degree. Secondly, the LOF based methods are outperforming the k-distance based methods. This difference in performance is likely indicative that the richer information provided by the LOF method justifies the additional computation. Lastly, the methods that use states are regularly outperforming the methods that use transitions. This likely indicates that using transitions in the gridworld is not a vastly better representation of locations within the environment than states. If that were the case then using transitions rather than states would just reduce the rate in which the reward for visiting a certain state decays. While this analysis provides us with a proof of concept, it is worth looking at the data in a few different ways to better understand what exactly is the relationship between these methods and how accurate this analysis is.

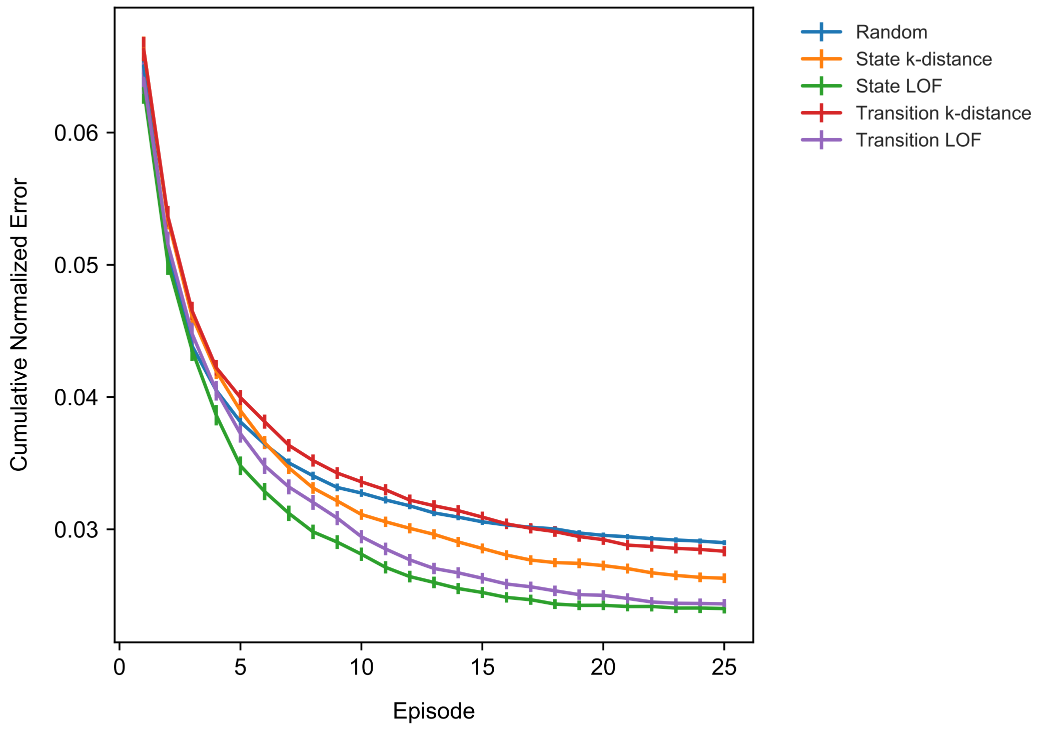

Figure 6: Cumulative Error After Each Episode Normalized by Number of Timesteps

We previously normalized the cumulative error by the number of timesteps in each episode. We now look at the cumulative error and the lengths of episodes separately. Figure 7 shows the cumulative error after each episode and Figure 8 shows the lengths of episodes. Looking at Figure 7, it now appears that the LOF using transitions is outperforming the LOF using states. In fact, it seems like the LOF using transitions is the only method outperforming random exploration. It becomes more evident why this figure appears to present this contradiction and why we were previously weighting by the total number of visitations when we look Figure 8. In Figure 8 we see that the methods using transitions tend to have shorter episodes as time goes on whereas the methods using states tends to have longer episodes. It makes sense that the methods using state would tend towards having longer episodes as this would help it to explore the more distant states such as the true state $(4, 4)$. It somewhat unclear why the methods using transitions seem to have shorter episodes as time goes on but one possible reasoning is that the increased dimensionality of the space is reducing the speed at which the agent becomes uninterested in areas of the environment. These seemingly conflicting results show one of the issues that arise using an analysis based on episodes. In light of this, we now consider how these methods compare timestep to timestep.

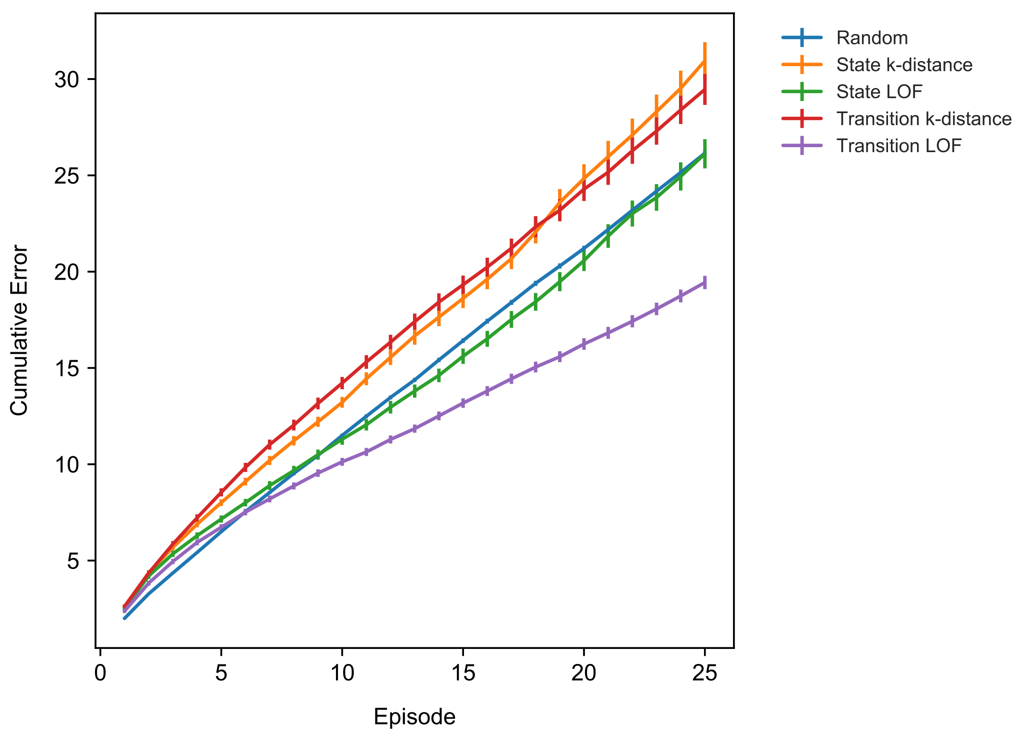

Figure 7: Cumulative Error After Each Episode

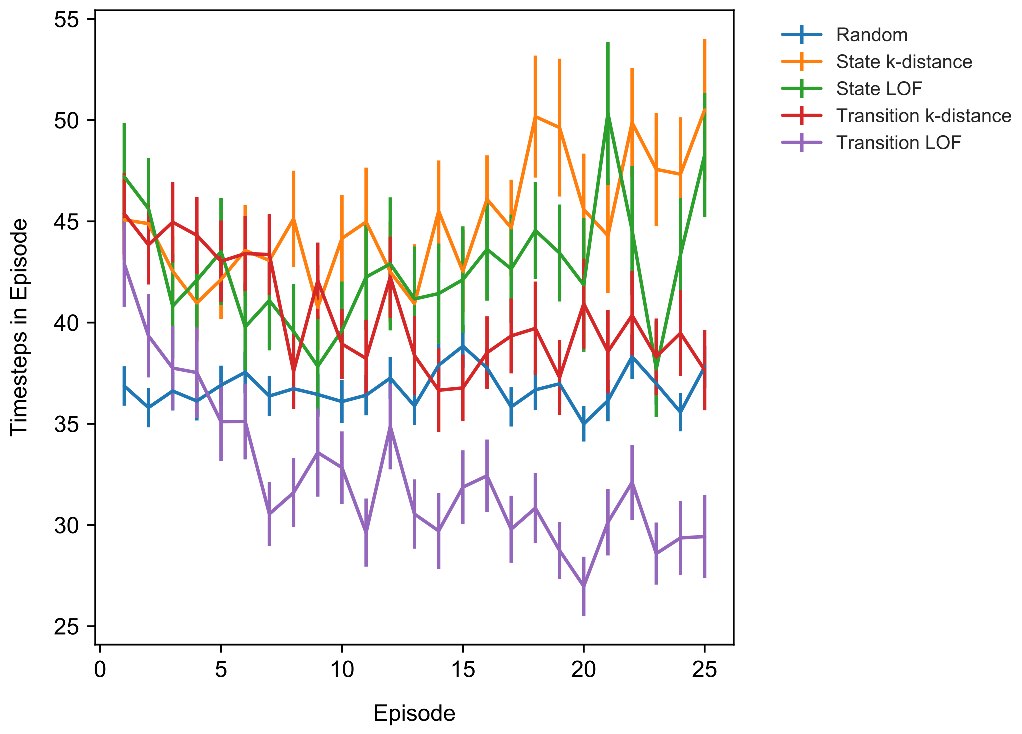

Figure 8: Length of Episodes

When comparing timesteps, we see similar trends to what we saw initially. Figure 9 shows the cumulative error at each timestep. The curves in Figure 9 continue until more than 75% of runs have terminated. By cutting the curves at this point, we prevent the error bars from growing problematically large. The flaring of the error bars near the end of the curves in Figure 9 is in part due to this as the values are being computed only over runs that have not yet terminated. Figure 9 confirms our earlier analysis that the LOF metric is regularly outperforming random exploration when using either states or transitions. It also confirms our earlier analysis that, when using states, the k-distance method often outperforms random exploration. This plot does however cast doubts on the performance of these methods in the long run. This doubt provides additional motivation to explore the idea of retaining only a limited number of states or transitions as was briefly discussed in Section 2. While this is an interesting contribution to our analysis, it is worth looking at the slopes of these curves to understand more fully why we have so far highlighted performance over episodes.

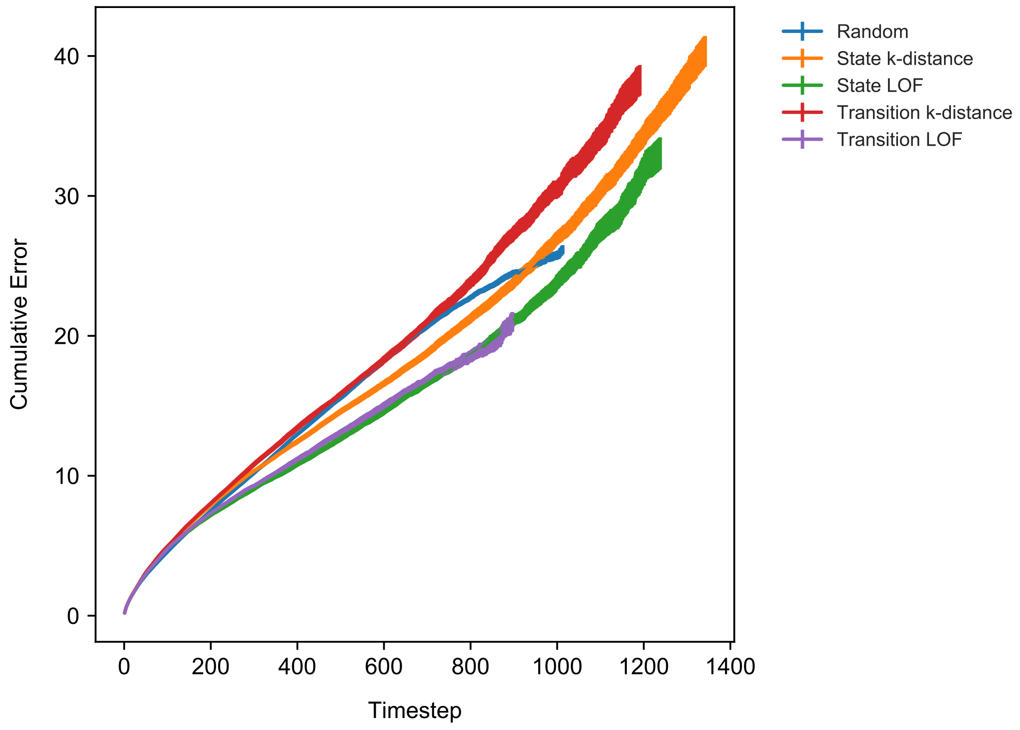

Figure 9: Cumulative Error at Each Timestep

Figure 10 shows the slope of this curve or the rate at which the cumulative error increase over timesteps. Like in Figure 9, the curves in Figure 10 continue until more than 75% of runs have terminated. There is little that can be derived from this figure given the size of the error bars. Our choice for the number of runs we do is based on reducing the error bars to the point where we can confidently discuss any improvements while still being efficient enough to work with the limited resources available. Reducing the error bars further would require significantly more computational power than was available. However, this is an interesting avenue of for future work.

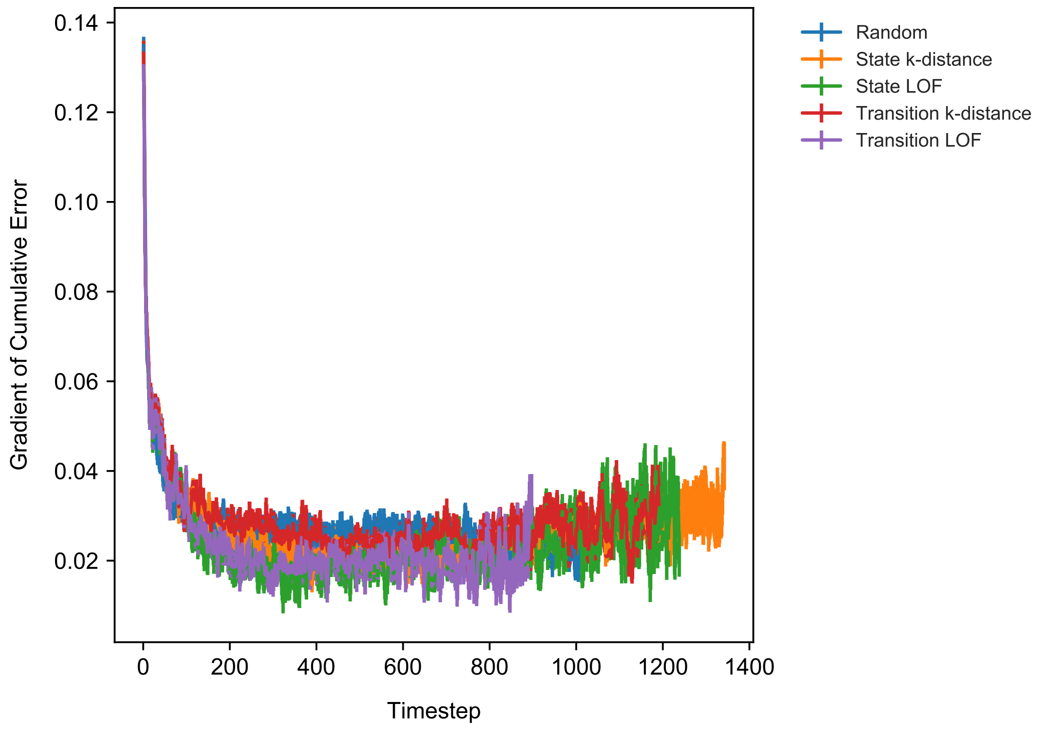

Figure 10: Gradient of the Cumulative Error at Each Timestep

4 Conclusions

We consider the problem of how to encourage a Reinforcement Learning agent to explore the environment in a uniform manner. To accomplish this, we borrow the method of Local Outlier Factors from Unsupervised Learning and adapt it to serve as a reward signal for an agent to incentivizes the agent to explore parts of the environment it has not heavily explored yet. We show that incentivizing the agent in this way causes it to explore more uniformly than random exploration in one modified gridworld domain. We also provide some evidence to show that using the k-distance instead of the LOF results in less uniform exploration in this domain. Altogether this result serves as a strong motivation for further work in this area.

There is a significant quantity of additional work required to make this method into a practical, usable method and to understand it better. Firstly, only one domain has been considered so far. While this is sufficient to show a proof of concept it does not give any firm intuition for what kinds of domains this is a useful method. Secondly, as Local Outlier Factors diverge to infinity when identical data points are allowed, work needs to be conducted to determine if values can be safely clipped or whether something more sophisticated is required to deal with this problem. Thirdly, a more efficient data structure that allows efficient insertion of data points and efficient k-NN queries needs to be found and implemented to understand precisely how expensive this method is. Fourthly, as mentioned briefly in Section 2, the idea of using a limited number of data points need further investigation. There may even be some form of geometric decay for the importance of data points that could provide a robust metric that is similar to the LOF metric but more able to deal with a changing environment. Lastly, similarly to the first consideration above, more work needs to be done to understand what domains a uniform exploration strategy works well. Perhaps more importantly, more work needs to be done to determine how this compares to the exploration policy induced by other state of the art methods such as pseudo-count based exploration [2].

References

[1] Barto, A. G. *Intrinsic Motivation and Reinforcement Learning.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 17–47.

[2] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain* (2016), D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., pp. 1471–1479.

[3] Breunig, M. M., Kriegel, H., Ng, R. T., and Sander, J. LOF: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA.* (2000), W. Chen, J. F. Naughton, and P. A. Bernstein, Eds., ACM, pp. 93–104.

[4] Pokrajac, D., Lazarevic, A., and Latecki, L. J. Incremental local outlier detection for data streams. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2007, part of the IEEE Symposium Series on Computational Intelligence 2007, Honolulu, Hawaii, USA, 1-5 April 2007* (2007), IEEE, pp. 504–515.

[5] Sutton, R. S., and Barto, A. G. *Reinforcement learning - an introduction.* Adaptive computation and machine learning. MIT Press, 1998.

[6] Thrun, S. B. Efficient exploration in reinforcement learning. Tech. rep., Pittsburgh, PA, USA, 1992.