ARO: A Memory-Based Approach to Environments with State Aliasing

Author's Names Omitted for Peer Review

Abstract

Episodic memory, or the ability to remember specific events from your past, is an essential part of human cognition. While most machine learning tasks do not require this capability, episodic memory is essential is learning in environments with severe state aliasing (a.k.a., perceptual aliasing). In this paper we define two specific environments with severe state aliasing and then present ARO, an algorithm for learning in such an environment. Our results show that ARO is effective where traditional algorithms would fail.

The foundation of this work is in research towards an artificial, general episodic memory. Episodic memory is one of three, long term memories commonly recognized in humans (Tulving 1983). Evidence from psychology research indicates that an impaired ability to form new episodic memories (i.e., anterograde amnesia) also impairs learning (Brooks and Baddeley 1976). Furthermore, common sense tells us that possesing a memory of past experiences is essential to a number of fundamental cognitive capabilities including our own sense of identity (Holland and Marques 2010; Nuxoll and Laird 2012).

Thus, we believe it behooves those who wish to create an artificial general intelligence to seek to create algorithms that possess and leverage an artificial, general episodic memory. An artificial episodic memory has been applied to a number of specific environments. ee (Tecuci and W. Porter 2007; Brom and Lukavsky 2008; Castro and Gudwin 2010; Chaudhry et al. 2018) for a sample. We know of only a few existing approaches to a general artificial episodic memory in part of a broad, general architecture (Bölöni 2011; Nuxoll and Laird 2012; Ménager and Choi 2016).

Extreme State Aliasing

To learn about effective episodic learning, we seek to create environments where an agent's episodic memory is essential for success. One such environment is the Blind Finite State Machine environment. In this environment, the agent navigates a finite state machine (FSM) (Hopcroft, Motwani, and Ullman 2006) to reach a goal state. The machine is designed so that the agent can reach the goal from any state (no dead ends) but otherwise the transition table is randomly generated. Upon reaching the goal the agent is immediately moved to a randomly selected, non-goal state and informed of its success. Thus the agent knows when it reaches the goal but can never actually take an action in the goal state.

The task seems trivial at first. However, it is greatly complicated because the agent is only aware of the following:

- the FSM alphabet (available actions)
- a goal sensor indicating when it reaches the goal

Specifically, the agent is <u>not</u> aware of the following:

- the number of states
- what state it currently is in
- the transition table of the FSM

The agent repeatedly performs the task in a given FSM and its success is measured by how many states it takes to reach the goal at each iteration. Experiments with human subjects (not published) indicate that this task is exceptionally difficult for humans.

In such an environment, a traditional machine learning agent is unable to learn effective behavior. Such an agent attempts to learn the action that yields the maximum expected utility for each possible sensory input. However, in this environment, all states (except the goal state) appear identical to the agent. So, the agent can only learn a single "best" action to take in any non-goal state.

To be successful in the Blind FSM environment, an agent must map *sequences of episodes* to actions where an episode is in this case as the agent's sensory input and the action it selected. For example, the agent can learn if it takes actions $a_1, a_2, ..., a_n$ in sequence and does not reach then goal that it can subsequently reach the goal in a single step by taking action a^*. Furthermore, the agent can learn that certain sequences of actions have more utility than others regardless of what non-goal state it is currently in.

To learn effective sequences of episodes, an agent must have an episodic memory. Specifically, a memory of previous episodes allows the agent to identify its current state and/or successful sequences of actions.

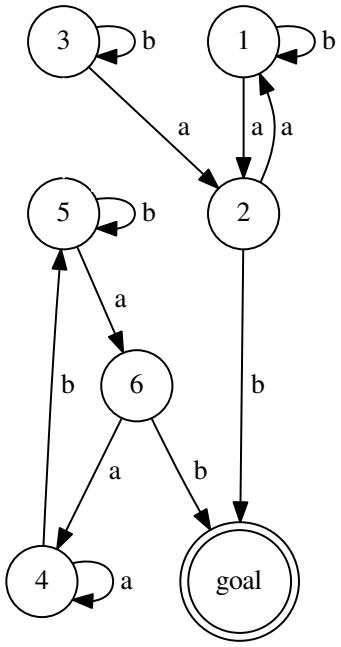

Figure 1: An example Blind FSM

In previous research, an algorithm called MaRz (Rodriguez et al. 2017) has shown the best success in Blind FSM environments. MaRz performs an A*-Search (Russell and Norvig 2009)over possible sequences of actions to find the shortest *universal sequence*. A universal sequence is a sequence of actions that causes the agent to reach the goal from any non-goal state. For example, consider the state machine in Figure 1.

The agent begins in a random, non-goal state. It first takes action 'b' and does not reach the goal as a result of this action. While the agent does not know the transition table of the FSM, we can look at the diagram and know that the agent must have begun in state 1, 3, 4 or 5 and now must be in state 1, 3 or 5. As a result, we know that the agent is guaranteed to reach the goal in two steps if it takes the action *a* followed by *b*. This makes *bab* a universal sequence for this particular FSM.

The proofs below show that all Blind FSM environments must have have a universal sequence and that no agent can guarantee it will reach the goal state with fewer actions by not following the actions in the shortest universal sequence.

Theorem 1 Consider a deterministic finite state machine M that consists of a finite set of states S, a finite alphabet α, a transition function $T : S \times \alpha \rightarrow S$ and a single goal state $g \in S$. Furthermore, T is defined so that there exists a sequence of actions from any non-goal state $s \in S$ to g.

There is a finite sequence of actions $U : u_1, u_2, ..., u_n$ such that an agent operating in M will always reach g from s by following this sequence, though it may reach g before completing the entire sequence.

Proof Consider some action $a_1 \in \alpha$, that allows the agent to reach g from one or more non-goal states. Such an action must exist, otherwise there would be no path to the goal from any state which is in contradiction to the definition of M. Therefore there is a non-empty $G_1 \subseteq S$ from which the goal can be reached by taking action a_1.

If the agent were to take this action and not reach the goal, then there is some subset, $R_1 \subset S$ which consists of the states the agent could currently be in after taking action a_1 and not reaching the goal. This must be a proper subset as each state has one outbound transition for a_1 and thus the number of such transitions is equal to the size of S and, as defined above, at least one such transition would take the agent to the goal.

If R_1 is the empty set, then all states can reach the goal by taking action a_1. In this case, $U = a_1$.

If R_1 is not empty, consider some action, $a_2 \in \alpha$ that defines a set $G_2 \subseteq R_1$ for which action a_2 allows the agent to either reach g, or reach a state in G_1. Again, such an action must exist to avoid contradiction with the original definition. For any state in G_2, the agent can reach the goal by taking action a_1 and then, if not already at the goal, taking action a_2.

Now define a set R_2 of states that consists of all the states in R_1 that are not in G_2. If R_2 is not the empty set, repeat the process above defining additional actions $a_3, a4, ..., a_n$ as needed that, in turn define sets $G_3, G_4, ..., G_n$ and sets $R_3, R_4, ..., R_n$ until such time as R_n is the empty set. This must occur as S is finite and each successive R is smaller than its predecessor. At this point, the sequence $a_1, a_2, ..., a_n$ will allow the agent to reach the goal from any state in S.

Theorem 2 Consider a agent A that is operating in a finite state machine M, as specified in *Theorem 1*. A is currently in a starting state $z \neq g$. A is unable to sense what state it is currently in but will sense if it takes an action that causes it to reach g. A has knowledge of S, α, and T.

Agent A can, in finite time, find a universal sequence U^* that is as short or shorter than any other universal sequence for M.

Proof Before selecting any action, agent A can consider all possible sequences of actions in order from shortest to longest. For each sequence it considers, it can use its knowledge of T and S to determine if it is a universal sequence. While there are an infinite set of possible sequences, Theorem 1 tells us that M must have a finite, universal sequence. This means that A must find a universal sequence in a finite time. Furthermore, given that the agent is searching from smallest to largest, the universal sequence it finds will be as short or shorter than any other universal sequence for M.

Theorem 3 Consider two agents A_1 and A_2 that are operating in a finite state machine M and beginning in an unknown state z as specified in *Theorem 2*. Agent A_1 has knowledge of S, α, T and U^*. Agent A_2 is only aware of α and U^*.

If A_2 selects its actions from and U^*. A_1 will be unable to ensure it will reach g in fewer steps than agent A_2.

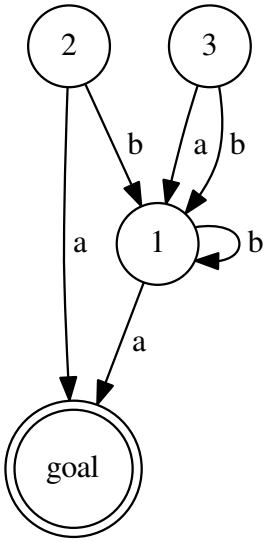

Figure 2: An Unusual Blind FSM

Proof Consider a scenario where agent A_1 reaches g via a sequence of actions G that is shorter than U^*. Since U^* is the shortest universal sequence, G can not itself be a universal sequence. Therefore, while A_1 did reach g in this scenario, it could not guarantee to reach g in fewer steps for all possible z.

Note

There are situations there are some FSMs where you can get to the goal in fewer steps *on average* with a universal sequence that is longer than the shortest universal sequence. Consider the FSM in Figure 2.

The shortest universal sequence for this machine is *ba*. However, if you use the longer universal sequence, *aba*, you will reach the goal in fewer steps on average. However, in randomly generated FSMs, such situations are unusual and, in practice, the difference is small.

A Little Less State Aliasing

The situation changes once the agent is given additional sensors beyond the goal sensor. Consider an environment like Blind FSM but the agent also has a binary sensor that tells it when it is currently in an odd state or an even state. We shall henceforth refer to this as the OSFSM (Odd-Sensor Finite State Machine) environment.

For an OSFSM environment using the FSM in Figure 1, the universal sequence (*bab*) is sub-optimal. The agent can consistently reach the goal faster (or at least as fast) by always taking action *b* in even numbered states and action *a* in odd numbered states. Thus, the MaRz algorithm needs to be enhanced or replaced in such situations.

An episodic memory remains essential to be successful in the OSFSM environment. Traditional machine learning algorithms still perform poorly in OSFSM since they can still only ever distinguish three states. However, as more and more sensors are provided a traditional machine learning algorithm becomes more and more effective. Thus, there is a continuum from complete state aliasing (e.g., the Blind FSM environment) to a fully observable environment.

The goal becomes to create an algorithm that can function well in both. This research presents ARO, an initial step toward such an algorithm.

Ryan Environment

We formally classify the environment for ARO. We are given a tuple (S, α, T, O, f, G), where S is a set of states, α is a set of actions valid in any state, $T : S \times \alpha \to S$ is a transition function, O is a set of possible observations (different sensor values) where $G \in O$ denotes the observation of the goal state, $f : S \to O$ computes what is observed in any state where $f(G) = G$ and $f(s) \neq G$ when $s \neq G$, and $G \in S$ is the unique goal state. An agent in this environment will know α, but is otherwise provided no information about the environment. At any time-step t when the agent is in state s_t, the agent will be told $f(s_t)$. If s_t is not a goal state, the agent must provide an action $a_t \in \alpha$. The agent will then transition to $s_{t+1} = T(s_t, a_t)$. If s_t is a goal state, the agent will be moved to non-goal state s_{t+1} at random. This will continue until an arbitrary but unknown number of goals are reached. The goal of the agent is to minimize the total number of time steps.

In the Blind FSM environment, $O = \{\epsilon, G\}$ and $f(s) = \epsilon$ when $s \neq G$. In the OSFSM environment, $O = \{0, 1, G\}$, with 0 and 1 denoting whether the odd sensor is off or on, respectively, and $f(s)$ returns the parity of s (the state of the odd sensor).

Ryan ARO Overview

Data Structures

We begin with data structures used for ARO. The agent stores previous events in atomic units called "episodes." Each episode consists of the tuple (o, a), where o is the observation ($f(s)$), and a is the action taken after the observation. In the OSFSM environment, an example episode may be $(1, b)$. When the agent reaches the goal, no action is performed, so we denote such an episode simply with "G." In non-goal episodes, we concatenate the tuple for brevity, so the above episode would be denoted "1b". In this episode, the agent was in an odd state and took action b.

Using these episodic memories, the agent records patterns of cause and effect in a structure called a "rule." Rules are denoted like this example:

$$1a \to 1 : 23.$$

This rule states that being in an odd state and taking action a "caused" an odd state 23 times. The effect part of a rule is always in this same style, but causes may be more complex. For example, consider this rule:

$$0b, 1a \to 1 : 6.$$

This rules says that the agent taking action a while being in an odd state that was reached by taking a b from an even

state has caused an odd state 6 times. In general, a rule consists of a sequence of episodes as the "cause", an observation $o \in O$ as an "effect," and the frequency that this cause-effect relationship has occurred. The cause sequence of the rule may be empty, which can be interpreted as transitioning to a new state causing an observation, or equivalently as the frequency of each observation. The number of episodes in the cause sequence of a rule is called the "depth" of a rule.

There are multiple rules with the same cause sequence. For instance, both $1a \rightarrow 0 : 3$ and $1a \rightarrow G : 2$ may be rules. By comparing all rules with the same cause sequence, the agent can compute the probability that any particular effect will occur. We will call this set of rules a "rule group." We define the function called GETRULEGROUP(episodes), which takes an array of episodes and returns the corresponding rule group.

Now that we have a method for storing patterns in cause and effect, we can use this information in the agent's memory to predict future outcomes and find the optimal move.

Expected Value

Given a sequence of previous episodes that have occurred and an observation, we can recursively compute the expected number of moves it will take to reach the goal. This is computed under the assumption that at each choice the agent may have in the future, it will make the best possible move. The algorithm is below.

1: **function** EXPECTEDVALUE(Episode[] $episodes$, Observation o)
2: $BestSum = \infty$
3: $BestMove = \epsilon$
4: **for** a in α **do**
5: $Sum = 0$
6: Let $next = (o, a)$ be a new episode.
7: Let $nextSequence$ be $episodes$ appended by $next$
8: $nextGroup$ = GETRULEGROUP($nextSequence$)
9: **if** $nextGroup$ is empty **then**
10: Sum = GETHEURISTIC( )
11: **end if**
12: **for** $rule$ in $nextGroup$ **do**
13: Let e be the effect of $rule$.
14: Let p be the probability of $rule$ occurring.
15: **if** $e = G$ **then**
16: $Sum = Sum + p$
17: **else**
18: $Sum = Sum + p *$ (EXPECTEDVALUE($nextSequence$, e) $+1$)
19: **end if**
20: **end for**
21: **if** $Sum < BestSum$ **then**
22: $BestSum = Sum$
23: $BestMove = a$
24: **end if**
25: **end for**
26: **return** $BestSum$
27: **end function**

This algorithm also computes the best move to make given a sequence of episodes and an observation in the $BestMove$ variable. The agent may call this function, supplying an episode sequence observation from memory. Thus, performing the action $BestMove$ can be used as a policy for the agent. In fact, this is exactly what ARO does. However, there are two aspects of this policy that are not yet well-defined. First, we cannot compute an expected value for a move we have never done before. This case is dealt with in line 10 of the code with the GETHEURISTIC function that we have yet to describe. Second, the agent may choose how big of a sequence to pass into EXPECTEDVALUE . It may pass the empty episode sequence, its entirety of its episodic memories, or any size in between. Each call may produce a different expected value and recommended move to make, and the agent must choose which recommendation to follow. These are the corresponding topics of the next two sections.

Explore Heuristic

Let us first deal with the case where we have an episode sequence, an observation, and an action for which there are no rules, and we must compute the expected value. Let seq be this sequence, o be this observation, and a be this action. This situation is then equivalently described as follows: the agent has never recorded episodes in the sequence seq, then observed o and performed action a. Thus, taking action a can be considered an "exploration" by the agent. We therefore need to decide the value of exploring in this environment in the units of expected value. While the explore-exploit tradeoff is well studied under the context of the Mulit-Armed Bandit problem and has several known solutions, we may use existing knowledge of this scenario to make a more specific solution.

It is possible to find an estimate of the expected value by using the rule group corresponding to removing the first episode from seq. However, repeating this process can lead to infinite depth recursion. Rather than creating a suitable base case to make the recursion finite, we instead attempted to estimate the value that this recursion will return. Let p denote the probability of reaching the goal according to the 0-deep rule group (that is, the probability corresponding to the rule $\rightarrow G$). We note that if we were to continue recursion infinitely, the probability of reaching the goal in any step should converge on p. If we estimate that this probability holds constant on every step, then the number of steps to goal follows a Geometric distribution with probability p. This gives an expected value of $\frac{1}{p}$. Hence, we can now define the heuristic function:

function GETHEURISTIC( )
return $1/p$
end function

Sequence Selection

The agent may now calculate the best move given a sequence of recent episodes. However, different sized sequences may produce different results. With a large sequence, the episode sequence may be unique or rare in memory, producing poor results if the agent lacks knowledge of any fast paths to the

goal given that context. With a small sequence, the episode sequence may be too common for the agent to know where it is in the machine or if it is going in a loop, which could cause undesirable behavior such as infinite looping. To resolve this, ARO uses the episode sequence that produces the smallest expected value. Hence, if the agent does have knowledge of a fast path to the goal from a large sequence, it will take it. If, on the other hand, the agent lacks useful knowledge from a large sequence, it can fall back on a more reliable and average short sequence. This alone, however, is not sufficient to prevent loops.

At timestep t, say the sequence seq produced the lowest expected value x for observation o with move a. Let $nextSeq$ be seq appended by the new episode (o, a). We found that the expected value of $nextSeq$ in timestep $t + 1$ can be higher than x. When this occurred, the agent would switch to a shorter sequence for a move recommendation, which effectively "forgets" that this bad event occurred. This, in turn, caused the agent to get stuck in a loop. To deal with this issue, we force the agent to deal with the bad event by not letting the agent switch to a different sequence for a move recommendation. In other words, if seq was used to find the best move at timestep t, then $nextSeq$ must be used to find the best move at timestep $t + 1$. There are two exceptions to this rule. First, if the agent observes a goal. Because the agent is moved randomly upon reaching the goal state, there is no information to be gained by using a sequence with a goal observation. Second, if the agent discovers a new rule for which the current sequence is the cause sequence. This implies the agent is in a novel scenario and the current sequence can provide no data to guide the agent, so a new sequence must be selected to choose how to act.

Using these rules, we can now fully define the policy of ARO. We define the global variable $prevSequence$. Let the function GETBESTMOVE take an episode sequence and return the best move calculated in EXPECTEDVALUE .

Initialize $prevSequence$ to the empty sequence.
function POLICY(Observation o)
  if $o = G$ **then**
    Set $prevSequence$ to the empty sequence.
    return
  end if
  Let e be the last episode.
  Let seq be $prevSeq$ appended by e.
  Let f be the frequency of the rule $seq \rightarrow o$
  if $f = 1$ or seq contains the episode G **then**
    Set $prevSeq$ be the episode sequence such that EXPECTEDVALUE($prevSeq$, o) is minimized.
    return GETBESTMOVE($prevSeq$)
  else
    $prevSeq = seq$
    return GETBESTMOVE($prevSeq$)
  end if
end function

ARO Overview

ARO was developed for the OSFSM environment defined above. In ARO, the episodic memory consists of a sequence of episodes that each consist of the value of both sensors (odd sensor and goal sensor) and the action taken by the agent. Here is an example shorthand representation of an episodic memory: 01a,01b,00b,01b,11a,00b

In this example, each episode consists of three characters

- the value of the goal sensor
- the value of the odd-state sensor
- the action selected

Another foundational concept in ARO is a rule, a common paradigm that defines cause and effect. Nominally a rule in ARO consists of:

sensors + action $\rightarrow$ resulting sensors : frequency

Specifically, for ARO, the sensor value is the value of the odd-state sensor (1=odd; 0=even) and the action is selected from the FSM's alphabet. For example, a rule may look like this:

$$1a \rightarrow 1 : 23$$

This is shorthand for "The odd-state sensor was on and I selected action a. As a result, the odd-state sensor was on. This has occurred 23 times in the past." Note that the goal sensor, which is almost always zero, is omitted for brevity.

The example rule above is what we will refer to as a *1-deep rule*. A rule in ARO can be extended to include multiple steps. Consider this example of a *2-deep rule*:

$$0b,1a \rightarrow 1 : 7$$

Here the rule states that "the odd sensor was off and the agent selected action b. Then, the odd sensor was on and it selected action a. As a result, the odd-state sensor turned on. This has happened 7 times."

A rule can also be shortened to contain only the overall frequency with which a sensor has had a certain value. For example, this 0-deep rule states the agent has seen an odd state 45 times.

$$1 : 45$$

For efficiency, ARO stores the rules in a tree with a limited depth. For example, say that the agent's current episodic memory ends with the following subsequence:

...,00b,01a,01?

The question mark indicates the episode is the present state and the agent has not yet selected an action. Note that all three of the example rules given above would 'match' the current situation. ARO tracks the currently matching rules by selecting particular nodes in its tree (see Figure 3).

*** TODO: Figure 3 will need to be modified as follows:
*** a) add the frequencies
*** b) complete the tree to include all nodes at depths 0-2
*** c) put all three sensor values in each node
*** d) generate as a high quality image, preferably in PDF format

Each time a node is added to current, its frequency is increased appropriately. Some example frequencies are shown in figure 3. Note that all three possible sensor readings (odd, even or goal) are grouped together. (While there are actually four possible readings, the odd-state sensor is ignored

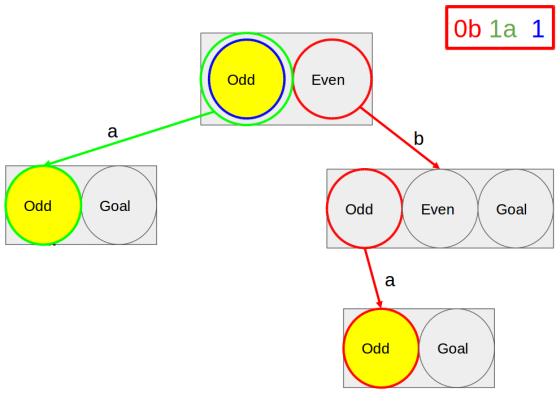

Figure 3: An example of ARO's rule tree

Table 1: Calculating expected value

action a	**probability**	**est. steps to goal**
odd	$\frac{2}{5}$	3.1
even	$\frac{1}{5}$	2.4
goal	$\frac{2}{5}$	1.0
expected value	$\frac{2}{5} \cdot 3.1$ $+\frac{1}{5} \cdot 2.4$ $+\frac{2}{5} \cdot 1.0$	2.1

by ARO when the goal sensor is on.) The sum of the frequency of occurrence of each node can be used to calculate the probably of that node occurring based upon the agent's experience.

Using these probabilities, the agent can estimate the number of steps that will be required to reach the goal via each possible action. From these values, an expected value of the agent's current state can also be calculated (see Figure 1).

ARO Explore vs. Exploit

As with many learning situations, ARO faces a choice here of whether to exploit the knowledge it already has or explore new knowledge. A traditional approach is an ϵ-Greedy algorithm where each state has an associated value that indicates the probability that the agent will take a random action (explore) rather than take the action that will take it most efficiently to the goal given the agent's current knowledge (exploit). The value of epsilon declines (at an exponential rate) each time the state is reached.

An approach like ϵ-Greedy can not be applied to OSFSM as it depends upon the agent for the same reason that traditional machine learning algorithms wile fail: state aliasing. Furthermore, the rate of declines of epsilon must be tuned to each particular environment. Since, this research aims to create a *general* articial episodic memory, environment-specific tuning is not desirable.

More sophisticated explore-exploit approaches exist (Kearns and Koller 1999; Brafman and Tennenholtz 2002) but they also suffer from a similar inapplicability.

To overcome this, ARO estimates the scope of its environment based upon the overall likelihood p of the agent reaching the goal on its next step regardless of the action taken. The term scope in this context does not refer to the size of the FSM. Certainly, the more states an FSM has the larger p will *tend* to be but this is not guaranteed.

This value of p can be estimated based upon the agent's experience to date and grows increasingly accurate as the agent gains experience. Thus it does not need to be tuned a priori on a per-environment basis.

Using p, the agent can estimate the number of steps it will take to reach the goal:

$$steps = \sum_{1..n} p \cdot n \cdot (1-p)^{n-1} = \frac{1}{p}$$

To avoid a divide-by-zero error when calculating $\frac{1}{p}$, the agent behaves as if it has already found the goal once. As ARO executes, the agent always chooses to explore when $\frac{1}{p}$ is less than the expected value of the current state. Thus, the exploration is frequent at first but gradually declines as the agent approaches optimal behavior.

Results

To assess ARO, we compared its performance in the OSFSM environment to that of two other algorithms: MaRz (Rodriguez et al. 2017) and Nearest Sequence Memory (McCallum 1995).

MaRz, as described above, an is algorithm that searches for the shortest universal sequence of its environment.

Nearest Sequence Memory (NSM) a reinforcement learning algorithm paired with an episodic memory and, thus, is suited to environments with extreme state aliasing like the OSFSM. In brief, NSM treats sequences of episodes as "states" for the purpose of calculating Q-values. Unlike MaRz and ARO, NSM is not a general algorithm and must be tuned to a particular environment in order to guarantee optimal performance.

Figure 4 below shows the performance of all three agents in the Blind FSM environment. These results are shown for a FSM with ***50 states and a three-letter alphabet (three possible actions per state). The ordinal measures successive trips the the goal. The abscissa measures the number of actions required to reach the goal on that trip. As the agent repeats its task, it gains more and more experiences (episodes) that it can draw upon to improve its future behavior.

*** TODO: Figure 4 will need to be modified as follows:

*** a) we need to put data from MaRz in there.

*** b) generate more data so we have a smoother curve

*** c) we can't rely on colors as this will be printed in b/w

*** d) generate as a high quality image, preferably in PDF format

Predictably, the number of steps to reach the goal declines exponential on successive trips the goal indicating that the

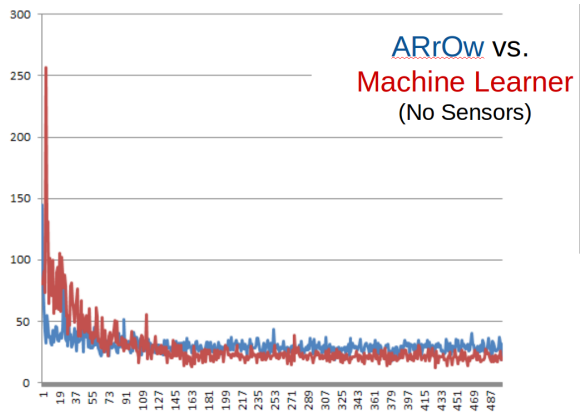

Figure 4: Agents in the Blind FSM Environment

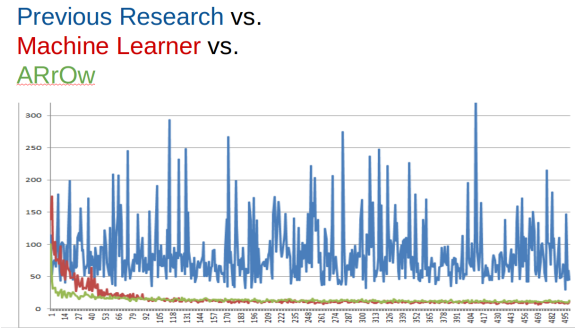

Figure 5: Agents in the Odd-Sensor FSM Environment

agent is learning to improve its behavior. This is true for all three algorithms.

Figure 5 shows the results for the OSFSM environment. The scope of the machine is the same: 50 states, 3 letter alphabet.

*** TODO: Figure 5 will need to be modified as follows:

*** a) we can't rely on colors as this will be printed in b/w

*** b) generate more data so we have a smoother curve

*** c) generate as a high quality image, preferably in PDF format

As shown, ARO is able to learn near optimal behavior much more rapidly than the other algorithms. These results were consistent across a variety of different machines ranging from [min] to [max] states and alphabets of [min] to [max] letters.

***TODO: fill in [min] and [max] above.

Future Work

This work demonstrates that ARO is effective in two specific environments with extreme state aliasing. Future work will be required to test the algorithm in other environments.

The most pressing concern is that it is not obvious how best to expand ARO's rule tree to accommodate additional sensors. The apparent, possibly naive, approach is to simply increase the branching factor of the rule tree. However, this factor increases exponentially as the number of sensors increases. It might be preferable to only include nodes on an as-needed basis or seek ways to discover sensors or sensor combinations that can be ignored without unduly impairing the agent's performance.

ARO may also need to be adjusted to address environment with these properties:

- a reward function rather than a single goal state
- a non-deterministic FSM
- a dynamic transition table

References

Bölöni, L. 2011. An investigation into the utility of episodic memory for cognitive architectures. *AAAI Fall Symposium - Technical Report*.

Brafman, R. I., and Tennenholtz, M. 2002. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research* 3(Oct):213–231.

Brom, C., and Lukavsky, J. 2008. Episodic memory for human-like agents and human-like agents for episodic memory. *AAAI Fall Symposium - Technical Report* 42–47.

Brooks, D. N., and Baddeley, A. 1976. What can amnesic patients learn? *Neuropsychologia* 14(1):111–122.

Castro, E., and Gudwin, R. 2010. *An Episodic Memory Implementation for a Virtual Creature*, volume 314. Springer. 393–406.

Chaudhry, A.; Ranzato, M.; Rohrbach, M.; and Elhoseiny, M. 2018. Efficient lifelong learning with A-GEM. *CoRR* abs/1812.00420.

Holland, O., and Marques, H. 2010. Functional embodied imagination and episodic memory. *International Journal of Machine Consciousness* 02.

Hopcroft, J. E.; Motwani, R.; and Ullman, J. D. 2006. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Kearns, M., and Koller, D. 1999. Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, 740–747.

McCallum, R. A. 1995. Instance-based state identification for reinforcement learning. In *Advances in Neural Information Processing Systems*, 377–384.

Ménager, D., and Choi, D. 2016. A robust implementation of episodic memory for a cognitive architecture. In *CogSci*.

Nuxoll, A. M., and Laird, J. E. 2012. Enhancing intelligent agents with episodic memory. *Cognitive Systems Research* 17:34–48.

Rodriguez, C.; Marston, G.; Goolkasian, W.; Rosenberg, A.; and Nuxoll, A. 2017. The marz algorithm: Towards an artificial general episodic learner. In *International Conference on Artificial General Intelligence*, 154–163. Springer.

Russell, S., and Norvig, P. 2009. *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice Hall Press, 3rd edition.

Tecuci, D., and W. Porter, B. 2007. A generic memory module for events. In *Twentieth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2007*, 152–157. AAAI Press.

Tulving, E. 1983. *Elements of Episodic Memory*. Clarendon Press.