The Thirty-First International Florida
Artificial Intelligence Research Society Conference (FLAIRS-31)

A Case-Based Reasoning Approach to Learning State-Based Behavior

Amrik Sacha Elapata Gunaratne, Babak Esfandiari, Ali Fawaz
Carleton University
1125 Colonel By Drive, Ottawa, ON
Canada, K1S 5B6

Abstract

Learning from Observation involves creating agents that observe experts performing tasks and imitate them. Case-Based Reasoning (CBR) is a tool that can be used for this purpose. Regular CBR can only learn memoryless behavior: behavior that doesn't rely on the past. Temporal Backtracking (TB) is an approach to learning state-based behavior that uses recency as its inductive bias, which may or may not be relevant to the agent behavior. We show how TB can be viewed as a particular case of a more generalized case-based approach to learning state-based behavior that can accommodate other inductive biases. We then propose five alternative similarity metrics to learn three different state-based behaviors in a 2D vacuum cleaner domain, and compare their performance to the TB algorithm's performance. We show that none of the proposed metrics (nor TB) is a one-size-fits all algorithm for learning state-based behavior.

1 Introduction

Motivation

Learning from Observation (LfO) is an approach for creating software agents or robots by having them observe and imitate an expert that is performing a task (Argall et al. 2009). Case-Based Reasoning (CBR) has been successfully used in the past for learning to play soccer (Floyd, Esfandiari, and Lam 2008) or play various video games (Floyd 2013). Traces of user behavior are captured and used as training data. In the LfO context, a given case's problem is a snapshot of the current situation perceived by the agent (and the expert), and the case's solution is the user's action in that situation. During testing and deployment, the agent is presented with a new and potentially unseen problem, and it is supposed to solve it by retrieving the solution(s) to the closest problem(s) in the case base, and adapting it (them) to the current one.

Case-based reasoning as we know it, can only be used to learn behavior that is memoryless: types of behavior that don't rely on the past, and only react to the current state of the environment. This is because cases are considered independent from each other, and therefore have no connection to past or future cases. Behavior that relies on the past is called state-based behavior: the "state" in this context refers to the information that presumably the expert is retrieving from memory, which captures something in the past that helps distinguish two apparently identical situations. It is therefore important not to confuse the agent's or the expert's "state" with the environment "state", which is just the current snapshot as perceived by the agent or the expert. In order to learn state-based behavior, it is important for the agent to determine what are those salient past events that were stored in the expert's memory, and so the case must be redefined to contain all the past events up till the current time. This idea was introduced in (Floyd and Esfandiari 2011) and it proposed an algorithm called Temporal Backtracking (TB). TB gives more importance to more recent events when comparing two cases. This can be viewed as one particular similarity metric which captures a particular *inductive bias* (Mitchell 1980), but there may be other biases that may be more relevant and effective to address certain state-based behaviors, as certain events in the more distant past may have more importance, or certain behaviors might depend on the frequency of occurrence of certain events or actions in the past. Therefore, we propose other similarity metrics that capture different biases when comparing two cases. By viewing TB as just one particular similarity metric among others, we unify state-based and memoryless CBR, and we allow for alternative (to TB) biases for state-based CBR.

Contributions

The domain used for testing and validation is the vacuum cleaning domain introduced by (Ontañón, Montaña, and Gonzalez 2014). The domain is a discrete grid world in which a Vacuum cleaner moves around avoiding obstacles and cleaning dirt. The domain is further explained in section 5. We propose five different similarity metrics that capture different biases. To show that the similarity metrics can capture different biases, we came up with three different experts to learn from. One expert travels in a fixed sequence, one travels in a zig-zag and one which relies on the frequency that it saw dirt to decide in which direction to turn when it reaches a wall. Each similarity metric is used when learning the expert behavior, and the results show how each metric captures different biases. These are also compared to the Temporal Backtracking algorithm mentioned earlier. Our main argument is that there doesn't seem to be a silver bullet or a one-size-fits-all bias for learning all state-based behavior at this point. However, we can hope to capture each bias as a particular similarity metric within the CBR framework.

Structure

Section 2 will discuss the formalisms required to understand the contributions and the state of the art. Section 3 will give a brief account of the state of art of case-based reasoning in its application to learning memory-based behavior. Section 4 will present the methodology, and section 5 will describe the experiments performed to test the similarity metrics. Section 6 will present and discuss the experimental results, and determine the validity of the hypothesis.

2 Background

Learning from Observation

Learning from observation entails learning behavior through observation of or demonstration by an expert. This expert is situated in an environment E which is a finite set of discrete, instantaneous states and is able to perform a set of actions Ac which transforms the state of environment (we will use here the definitions and notations of (Woolridge 2002)):

$$E = \{e_0, e_1, e_2, .., e_n\}$$

$$Ac = \{a_1, a_2, a_3 ...\}$$

An agent's goal, within learning from observation is to reproduce the expert's behavior, which is in the form of sequences of environment state-action pairs.

Runs

A sequence of environment state-action pairs is built in the following manner: The environment is in some initial state, and the expert chooses an action which transforms the environment. As a result, the environment will be in a new state, and the expert will choose an action based on this new state. This cycle continues, until a stopping condition is met. This sequence of environment state-action pairs is called a *run* (Woolridge 2002) and a given run *r* can be represented in the following manner:

$$r : e_0 \xrightarrow{a_0} e_1 \xrightarrow{a_1} ... \xrightarrow{a_{n_1}} e_n$$

Let $\mathcal{R}$ be the set of all possible finite runs and $\mathcal{R}^E \subseteq \mathcal{R}$ that contains runs that ends in an environment state.

Agents

An agent can also be formalized as a function *Ag*:

$$Ag : \mathcal{R}^E \rightarrow Ac$$

The agent requires the run to end in an environment state so that it can choose an action in response. This definition captures the fact that an agent's past actions and environment states may affect its current action. Woolridge defines this as a standard agent (Woolridge 2002).

A reactive agent is a type of standard agent that only depends on the current environment state to make a decision about its current action and can be defined in a simpler way as such:

$$Ag : E \rightarrow Ac$$

The standard agent model allows us to represent agents that are influenced by the past, through the runs. An agent that relies on its past events to make its current decisions can also be called a *state-based* agent. Woolridge provides a definition for the state-based agent as well.

$$action : I \times E \rightarrow Ac$$

$$next : I \times E \rightarrow I$$

Where I is an internal state that is updated each time the agent receives a new environmental state. This internal state captures the salient past events that are stored in the expert memory. Woolridge proposes that the standard agent and the state-based agent are equivalent in expressive power, and that one can always be converted into the other. An example of this can be seen in section 4.

Case-Based Reasoning

When applying case-based reasoning to LfO, A case $c \in C$ is an instance of an environment state-action pair from the set $C = E \times Ac$. When a case-based reasoning agent receives an environment state as a query, it searches through the casebase, and returns the action from the case that has an environment state that is most similar to the query. This retrieval method considers each case independently from each other. Therefore, an agent's action is only dependent on its current environment state, which is the definition of a reactive agent. An application of this can be seen in (Floyd, Esfandiari, and Lam 2008), in a 2D simulated soccer domain called RoboCup. Since there is no temporal link between the cases, and this retrieval method is not conducive to learning state-based behavior.

3 State of the Art

(Floyd and Esfandiari 2011) proposes that in order to learn more general behavior we need to redefine the set of cases C to contain all the past history until that point in time:

$$C = \mathcal{R}^E \times Ac$$

Now, each case $c \in C$ is temporally linked with its past. In this instance, the query should be a run that ends in an environment state, and the similarity metric should compare runs. There may be many possible similarity metrics, each capturing different inductive biases. If for the given environment state there are many differing actions suggested by the casebase, the Temporal Backtracking algorithm (TB) (Floyd and Esfandiari 2011) considers that the reason for the discrepancy is situated somewhere earlier in the run, and so it dynamically backtracks through the runs until it has found consensus on the action it should perform. The inductive bias of TB is therefore to give more weight to recent events and actions. In the paper, the algorithm was used to learn the behavior of an agent that toggles whether it turns left or right each time it hits an obstacle. The results showed that

TB was able to outperform the reactive retrieval. In section 4 we apply the TB algorithm and a frequency bias metric to a small example.

(Lora Ariza, Sánchez-Ruiz, and González-Calero 2017) use time series based CBR to determine the skill level of a player in a game of Tetris and adjust the level of help the player is provided during the game. Depending on the type of time-based case base created by the author, each case will have three time series of varying size, depicting the change of the parameters that describe the state of the game. The similarity between two cases is calculated as a linear combination of the similarities between their time series. The time series are compared using a similarity metric based on Euclidean distance, which gives equal weight to each point in the time series - an implicit inductive bias. The results showed that the user satisfaction, as well as average score improved when CBR was used to predict the player profile and adjust the game.

Another method that has been used for learning state-based behavior is probabilistic graphical models. (Ontañón, Montaña, and Gonzalez 2014) implemented three probabilistic graphical models to learn different types of expert behavior in a discrete 2D Vacuum cleaner domain (which we re-use as a benchmark for our experiments). The models were Bayesian networks, input-output HMMs and Dynamic Bayesian networks. They were used to learn reactive, state-based and stochastic behavior. The models learned each type of behavior and were compared against one another and it was shown that the Dynamic Bayesian networks were capable of capturing state-based behavior.

4 Methodology

Building state machines from runs

In Section 2, it was stated that one can theoretically build a state-based agent from the standard agent. Practically, this means that one should be able to build a state machine from a run and vice versa. However, the caveat is that the set $\mathcal{R}$ should contain all the possible runs of the agent. If this condition is not met, there are many state machines that can be built that will give rise to the runs in $\mathcal{R}$. Let us consider a simple vacuum cleaner domain. This domain has 4 environment states E = {NoWall (N), Dirt (D), Object (O), Human (H)} and 3 actions Ac = {Left (L), Right (RT), Straight (S)}. A run generated by an agent in this domain is shown below:

$$run : O \xrightarrow{RT} H \xrightarrow{RT} D \xrightarrow{L} H \xrightarrow{L}$$

We can see in the above run that given the environmental state H, two differing actions, namely L and RT, are observed. Therefore, it can be said that the latest environment state alone is not sufficient to explain the behavior (if we assume that the agent's behavior is not stochastic, that there are no features we are not capturing, and that there is no noise in the observed run) and so the expert's memory (i.e., its state), which cannot be directly observed, was also a factor in the decision. The following Finite State Machines (FSM) can be built from this run:

The state machine shown in Figure 1 is compact, and has made a determination about there being two states. The second FSM (Figure 2) is a simpler, less compact version. Both FSMs will generate the run (assuming the initial state is A). This is small example to prove that there exist many different state machines that can generate the same run.

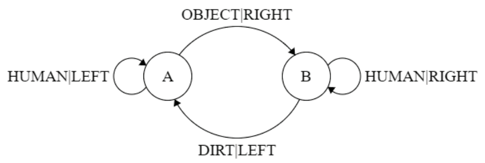

Figure 1: Compact State Machine

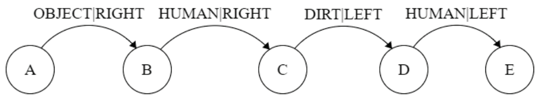

Figure 2: Simple State Machine

Since we generally only deal with a small subset of runs, we can only approximate the state machine that generated the subset of runs. Machine learning algorithms will create different state machines based on the biases they are looking for and use it to predict outputs given an input. In a CBR approach, the state machine is not directly inferred; instead, the similarity metric captures the salient inductive bias that allows to select a certain subset of the run as being relevant to the formation of the memory of the expert.

An example A small example of the generalized case-based approach to state-aware CBR is shown below, using the vacuum cleaning domain explained above. Consider an expert that has a behavior captured by the state-machine shown in Figure 3:

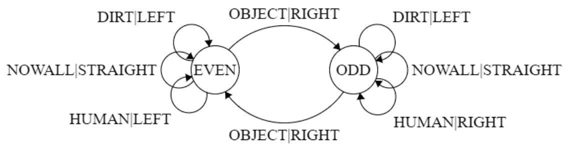

Figure 3: Expert in vacuum cleaner domain

Now consider the following two cases and query that are generated by this state machine (assuming the start state is Even). The correct action to be performed at environment state H of the Query is L. This can be seen by running the query through the state machine.

$$\begin{aligned} Case1 &: D \xrightarrow{L} N \xrightarrow{S} O \xrightarrow{RT} D \xrightarrow{L} H \xrightarrow{RT} \\ Case2 &: O \xrightarrow{RT} O \xrightarrow{RT} N \xrightarrow{S} D \xrightarrow{L} H \xrightarrow{L} \\ Query &: O \xrightarrow{RT} N \xrightarrow{S} O \xrightarrow{RT} D \xrightarrow{L} H \xrightarrow{?} \end{aligned}$$

We can apply the TB algorithm to the query and it would predict RT as the action.If we apply a frequency based metric that chooses the case with most similar probability distribution to the Query, the predicted action would be L. It can be seen that there are the same frequency (count) of environment state and actions in the Query and Case 2, which is not the case between the Query and Case 1. Algorithms 4 and 5 are the formal definitions of this metric. This shows that we can design counter examples of behavior that will cause certain algorithms to perform poorly. Therefore, we shouldn't rely on one type of algorithm to learn state-based behavior. In the next section we provide different similarity metrics that can be used to learn the different biases present in the runs.

Similarity Metrics

Similarity function Each of the algorithms below will have a similarity function - Similarity(element,element) that takes two elements, which can be environment states or actions and determines their similarity. In the framework[1] used for testing, the user predetermines a similarity function for each element before run time.

K-ordered This metric only considers the most recent k elements of the runs as important: it is biased towards the more recent elements. It performs a one-to-one comparison of each element in two runs from most recent element to the k'th element going back in time. The pseudo-code is shown in algorithm 1. The notation $|r|$ is used here to show the number of elements (actions, environment states) in the run r. Run[i] refers to the i'th element of a run, where the elements alternate between environment states and actions.

Algorithm 1 K-ordered-Similarity(Run1, Run2, k):Float

```
1: minSize ← min($|Run1|$,$|Run2|$)
2: for i = 0:min($k$,minSize) do
3:     sim += Similarity($Run1$[i],$Run2$[i])
4: return $\frac{sim}{k}$
```

Ordered This metric considers the entire run important and gives equal weight to all the elements. It performs a one-to-one comparison of each element in two runs taking into consideration the difference in their sizes. The pseudo-code is shown in algorithm 2:

Algorithm 2 Ordered-Similarity(Run1,Run2):Float

```
1: maxSize ← min($|Run1|$,$|Run2|$)
2: return K-ordered-Similarity($Run1$,$Run2$,maxSize)
```

K-unordered This metric captures a bias based on frequency. It compares the frequency of elements in two runs from the most recent element to the k'th element. It does also have a bias towards a more recent subset of the run and only considers the frequency of that subset.

In the following pseudo code for the k-unordered and unordered similarity, the function getV is used to compare frequencies of an element (el) in the two runs. The freq(element, Run) function is used in getV to get the frequency (count) of an element (el) in a run.

[1]available at https://github.com/sachag678/jLOAF

The pseudo-code for getV is shown in algorithm 3:

Algorithm 3 getV(Run1, Run2, el): Float

```
1: minFreq = min(freq($el$,$Run1$),freq($el$,$Run2$))
2: maxFreq = max(freq($el$,$Run1$),freq($el$,$Run2$))
3: return sim = $\frac{minFreq}{maxFreq}$
```

The pseudo-code for the k-unordered similarity metric is shown in algorithm 4. The runs are converted into sets in order to get the unique elements.

Algorithm 4 K-unordered-Similarity(Run1,Run2,k):Float

```
1: R1 ← $k$ most recent elements of $Run1$
2: R2 ← $k$ most recent elements of $Run2$
3: SetOne← R1 as a Set
4: SetTwo← R2 as a Set
5: for i= 0: |SetOne| do
6:     sim += getV(R1,R2,SetOne[i])
7: return $\frac{sim}{|SetOne \cup SetTwo|}$
```

Unordered This metric compares the frequency of elements in two runs. It considers the entire run to be important. The pseudo-code is shown in algorithm 5.

Algorithm 5 Unordered-Similarity(Run1,Run2):Float

```
1: maxSize ← min($|Run1|$,$|Run2|$)
2: return K-unordered-Similarity($Run1$,$Run2$,maxSize)
```

Weighted This metric compares each element in two runs from the most recent element to the last element and weights the more recent elements more heavily. This algorithm is biased towards the more recent elements but still considers all the elements in the run. The pseudo-code is shown in algorithm 6. The weight is arbitrarily assigned to 20 so that the first 10 elements are weighted in an exponentially decreasing manner. The rest of the elements in the run are weighted by 1.

Algorithm 6 Weighted-Similarity(Run1,Run2):Float

```
1: weight← 20
2: minSize ← min($|Run1|$,$|Run2|$)
3: for i = 0: minSize do
4:     $\omega$ ← (1+ $e^{-i}$× weight)
5:     sim += Similarity($Run1$[i],$Run2$[i]) $\times\omega$
6: return $\frac{sim}{minSize}$
```

5 Experiments

Vacuum cleaner domain

The vacuum cleaner domain consists of a grid world which contains obstacles, dirt and walls. For the purposes of our agents, the walls and obstacles are indistinguishable. An agent that performs in this world will receive 8 binary perceptions. The odd perceptions represent whether the agent sees dirt(1) or obstacle/wall(0) in NORTH, EAST, SOUTH, WEST directions. The even perceptions represent whether the object is near(0) or far (1) with near being right next to the agent. The agent also has access to five actions: Up, Down, Left, Right and Stand. The following agents are created within the constraints of this domain. An example expert (in green) can be seen in Map 1 in Figure 4. The perception it receives from the environment is 10101010.

Designed Agents

Frequency Agent The agent moves down and counts the number of times it sees dirt to the left and the number of times it is next to a wall on its right or left. When the agent collides with a wall or obstacle below it, if the dirt count is greater than the wall count it moves left, otherwise it moves right. Then the agent repeats this behavior but in the up direction.

Fixed Sequence Agent An agent that follows a fixed sequence of 21 actions and repeats it when it is over. We used the same agent defined in (Ontañón, Montaña, and Gonzalez 2014) because we had prior knowledge of TB underperforming when imitating this behavior (Gunaratne, Esfandiari, and Chan 2017).

Zig Zag Agent An agent that traverses the environment in a zig zag. It will travel to the left until it collides with a wall or obstacle, then moves up and moves to the right until colliding with a wall or obstacle. When it can go up no further it repeats the behavior but going down (Ontañón, Montaña, and Gonzalez 2014).

Experiment Design

Each experiment was performed using leave-one-out testing, with 3 traces containing a 100 cases each. The training set consists of 200 cases and the test set consists of 100 cases for each iteration of the leave-one-out test. Each trace contains the expert behavior on a different map. The maps were specifically designed to limit the TB algorithm's ability to predict the behavior of the frequency agent, and are shown in Figures 4 and 5. This was done by varying the number of dirt (dark gray) and nearby obstacles (light gray) to cause the agent (white square with arrow) to differ on whether it moves left or right when it comes into contact with a wall or obstacle above or below it[2]. For example, if the agent is going down on Map 1, it will move left when it comes into contact with the obstacle, while in Map 2 and 3 it will move right. This is because in Map 1 it saw dirt to its left, but it didn't in Map 2 and 3. We used the same maps for the sequence and zig zag agent. The value of k in the k-unordered and k-ordered algorithms was chosen to be 11. The results from the experiments are shown in the next section.

6 Results

The results of our experiments are summarized in Tables 1, 2 and 3. Each table contains the Accuracy (Acc) and Global F1 (F1) score of predicting the expert behavior on maps 1,2 and 3.

[2]https://github.com/sachag678/VacuumCleaner

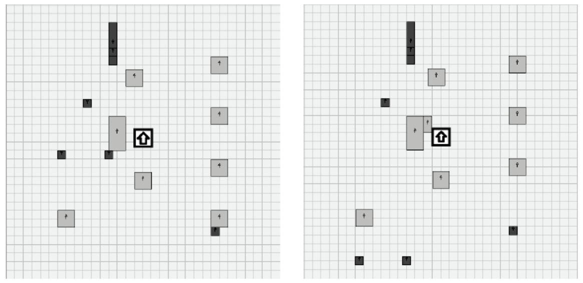

Figure 4: Map 1 and Map 2

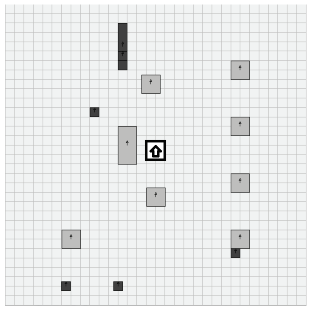

Figure 5: Map 3

Frequency Agent

The results for the Frequency agent are shown in Table 1. The reason the **TB algorithm** doesn't perform poorly is because for the most part the actions of the frequency agent are not based on frequency. The TB algorithm should have trouble predicting actions when the agent hits a wall above or below it. The maps were designed to increase the amount of times this occurred, however it was limited by the fact that in between two obstacles above or below, there had to be dirt or a wall to increase the counters of the agent. The **k-ordered** and **k-unordered** metrics outperform the TB algorithm, with the exception of Global F1 in Map 3.

Fixed Sequence Agent

The **TB algorithm** has a relatively low accuracy in comparison to the other algorithms for Map 3 which is shown in Table 2. Since TB dynamically backtracks one time step at a time, and gives equal weight to environment states and actions, it is unable to imitate a sequence of actions that is not dependent on the environment. The **k-ordered**, **ordered**, **k-unordered** and **unordered** algorithms are able to perform better for different reasons. The ordered algorithm backtracks to the beginning of the run and therefore will be able to know exactly at which point in the sequence the query sequence is at. The k-ordered algorithm is able to capture the behavior because of the cyclic nature of the sequence. If the k-ordered algorithm is able to have a k value that is half the size of the sequence then it will be able to capture the sequence behavior. The unordered and k-unordered metrics function based on the frequency of elements in the run. In this case the unique number of actions that occur before another action allow the two metrics to capture the behavior as

	Map 1		Map 2		Map 3	
Similarity	Acc	F1	Acc	F1	Acc	F1
TB	0.88	0.6	0.92	0.75	0.79	**0.87**
k-ordered	**1.00**	**1.00**	**0.94**	**0.8**	0.88	0.48
k-unordered	0.93	0.71	**0.94**	**0.8**	**0.93**	0.66
ordered	0.89	0.48	0.89	0.47	0.89	0.49
unordered	0.83	0.62	0.89	0.47	0.88	0.47
weighted	0.99	0.95	0.94	0.73	0.88	0.49

Table 1: Results for Frequency Agent (Bold indicates the best performance measure for each map)

	Map 1		Map 2		Map 3	
Similarity	Acc	F1	Acc	F1	Acc	F1
TB	**1.00**	**1.00**	**1.00**	**1.00**	0.85	0.83
k-ordered	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
k-unordered	0.99	0.99	0.99	0.99	**1.00**	**1.00**
ordered	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
unordered	0.97	0.97	0.97	0.97	0.94	0.94
weighted	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**

Table 2: Results for Fixed Sequence Agent

well.

Zig Zag Agent

The results for the Zig Zag agent are shown in Table 3. This agent was used to showcase the limitations of the frequency-based similarity metrics. The behavior of the zigzag agent is dependent on the action it performed the last time it collided with an obstacle. This action occurs approximately 10 time steps into the past. The cases before that time don't affect the current decisions. In Map 3, **k-unordered**, **unordered** and **ordered** under-perform relative to the other algorithms. The frequency of the actions or the environment states are not important factors in this behavior and this could be the reason for the poor performance of k-unordered and unordered. Since ordered gives equal weight to all elements, the correct subset of important elements isn't captured, and this could be the reason for its lower performance.

7 Conclusion

In this paper we have shown that we can come up with counter examples which limit TB's ability to predict the correct behavior, specifically with the frequency agent. We have also shown that there are agents, such as the zig zag agent, that theoretically limit some of our proposed similarity metrics' ability to predict the correct behavior. However, based on our results, it is difficult to conclusively say that a specific type of behavior will always cause a certain metric to under-perform. This reiterates our hypothesis that there isn't a universal one-size-fits-all bias for learning state-based behavior at this point.

One area for future work is to be able to characterize types of state-based behavior using the run. We can already differentiate between reactive and state-based behavior. Therefore, there is a possibility that it can be done between different types of state-based behavior. We could then test metrics on specific characterizations and determine if we are able to suggest that a specific metric will perform better based on the characterization. We would also like to implement metrics such as Levenshtein and Compression distance and further test these on agent behavior in this domain, as well as in a more real world domain, such as 2D simulated soccer - RoboCup. Another area to investigate is the use of Long Short-Term Memory (LSTM) based recurrent neural networks. LSTMs have been used to classify and predict time series and speech-based data and should be well suited for learning state-based behavior.

	Map 1		Map 2		Map 3	
Similarity	Acc	F1	Acc	F1	Acc	F1
TB	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**0.93**
k-ordered	0.98	0.96	0.98	0.96	0.98	0.88
k-unordered	0.98	0.97	0.98	0.97	0.88	0.88
ordered	0.96	0.92	0.96	0.92	0.68	0.57
unordered	0.88	0.67	0.88	0.67	0.69	0.48
weighted	0.97	0.96	0.97	0.96	0.86	0.77

Table 3: Results for ZigZag Agent

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*.

Floyd, M. W., and Esfandiari, B. 2011. Learning state-based behaviour using temporally related cases. *Proceedings of the Sixteenth UK Workshop on Case-Based Reasoning, Cambrdige, United Kingdom, December 13, 2011* 829:9–11.

Floyd, M. W.; Esfandiari, B.; and Lam, K. 2008. A Case-Based Reasoning Approach to Imitating RoboCup Players. *Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, {USA}* 251–256.

Floyd, M. W. 2013. A Comparison of Case Acquisition Strategies for Learning from Observations of State-Based Experts. *FLAIRS 2013 - Proceedings of the 26th International Florida Artificial Intelligence Research Society Conference* 387–392.

Gunaratne, S.; Esfandiari, B.; and Chan, C. 2017. Towards a Framework for Testing Learning from Observation of State-Based Agents. *AAAI Spring Symposium 2017*.

Lora Ariza, D. S.; Sánchez-Ruiz, A. A.; and González-Calero, P. A. 2017. Time series and case-based reasoning for an intelligent tetris game. In Aha, D. W., and Lieber, J., eds., *Case-Based Reasoning Research and Development*, 185–199. Cham: Springer International Publishing.

Mitchell, T. M. 1980. The need for biases in learning generalizations. Technical report.

Ontañón, S.; Montaña, J. L.; and Gonzalez, A. J. 2014. A Dynamic-Bayesian Network framework for modeling and evaluating learning from observation. *Expert Systems with Applications* 41(11):5212–5226.

Woolridge, M. 2002. *An Introduction to Multiagent Systems*. West Sussex, England: John Wiley and Sons, LTD.