# Learning State-Based Behaviour using Temporally Related Cases

Michael W. Floyd and Babak Esfandiari

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, Ontario, Canada

**Abstract.** Learning by observation allows a software agent to learn an expert's behaviour, by examining the actions the expert performs in response to inputs, without the expert having to explicitly program the agent. Most learning by observation approaches only make use of the current inputs and actions of the expert and ignore any past inputs or actions. This limits the agents to only being able to learn reactive behaviour. We present an approach to case retrieval that uses the expert's past inputs and actions in order to allow for learning state-based behaviour. We demonstrate our approach by learning from a simulated obstacle avoidance robot that reasons using internal state information. Our results show a significant accuracy improvement over retrieval that does not take into account any past information.

**Keywords:** learning by observation, state-based agents, temporal cases

## 1 Introduction

Programming a software agent or a robot can be a difficult and time-consuming task as it requires the expert to have programming skills and a clear domain model to transfer to the agent. One approach to alleviate the knowledge transfer bottleneck is to have an agent that *learns by observation*. Such an agent is not explicitly trained by an expert but, instead, observes how the expert behaves when presented with sensory stimuli. The agent is then able to use those observations to train itself.

Case-based reasoning has been a popular approach to learning by observation with applications in a variety of domains [6, 9–11]. While learning by observation can be beneficial, it does have several areas that may negatively influence the agent's performance. Some of these items have been addressed by previous research [5, 6, 9–11] while others remain open problems. In particular, learning by observation treats the expert as a black box and examines the outputs, in the form of actions, produced after receiving inputs, in the form of sensory stimuli. Any internal information, like the expert's *state*, that influences reasoning will be unobservable. This can result in the observing agent missing key information that it would need to accurately learn the expert's behaviour.

Our idea is that if an observing agent only uses the latest sensory stimuli as a case description it can only, at best, learn an expert's reactive behaviour. However, if the case description captures the expert's behaviour over a period of time it should be possible to extract temporal relationships between sensory stimuli and actions. For example, past stimuli and actions may still influence the current behaviour. This temporal relationship can then be exploited in order to approximate the state of the expert.

The remainder of this paper will detail a case retrieval algorithm that exploits the temporal link between observed cases to allow imitation of state-based experts. Section 2 defines the cases that will be generated when observing a state-based expert and Section 3 describes how those cases will be retrieved. The experiments in Section 4, which involve learning state-based behaviour from a simulated obstacle avoidance robot, demonstrate how our approach is a significant improvement over retrieval that does not make use of temporal sequences of cases. Our work is compared to related work in Section 5 and conclusions are presented in Section 6.

## 2 Case Definition

The *interactions* between an expert, or any agent, and its environment can be thought of as a series of environment states and actions by the expert [15]. During each interaction, the expert will sense the current state of the environment and use the state, and possibly other information, to reason about what actions it should perform. Both the possible environments states, $\mathcal{S}$, and possible actions, $\mathcal{A}$, are sets containing all environment states and actions that may be encountered:

$$\mathcal{S} = \{S', S'', \dots\} \tag{1}$$

$$\mathcal{A} = \{A', A'', \dots\} \tag{2}$$

An observing agent is able to record each interaction and create a case, $C_i$, that contains the environment state encountered by the expert and the resulting action. In this case definition the environment state is the *problem* and the action is the *solution*.

$$\mathcal{C} = \mathcal{S} \times \mathcal{A} \tag{3}$$

$$C_i = < S_i, A_i > \tag{4}$$

If the expert is purely reactive then it will select an action to perform based solely on the current state of the environment:

$$A_i = f(S_i) \tag{5}$$

A case-based reasoning system that attempts to imitate the behaviour of a reactive expert, by determining which action to perform, can do so by retrieving cases that are similar to the currently observed environment state. However, experts that are not purely reactive may reason using information from their

internal state. The difficulty in this is that the internal state of the expert is not directly observable by an outside viewer and can not be added as a feature of the cases.

While not directly observable, internal state can be inferred by examining the complete series of past environment states and actions [15] (assuming the state is never changed stochastically). A state-based expert will not only reason using the current environment state but also using state information that was determined using past environment states and actions. This requires thinking of cases not as unrelated observations but as a temporally linked series of observations. When an expert is observed for a period of time, a *run* $R$ of environment states and actions will be observed:

$$R : S_0 \overset{A_0}{\rightarrow} S_1 \overset{A_1}{\rightarrow} S_2 \overset{A_2}{\rightarrow} \dots S_{u-1} \overset{A_{u-1}}{\longrightarrow} S_u \tag{6}$$

When learning by observing a state-based expert it then becomes necessary to use the run of the expert to approximate how the expert selects which actions to perform:

$$A_i = f(S_i, A_{i-1}, S_{i-1}, A_{i-2}, S_{i-2}, \dots) \tag{7}$$

In order to allow learning from a state-based expert, we extend our definition of the problem to the current run rather than just the current environment state. This leads us to redefine a case $C_i$ as a pair containing the current run $R_i$ and the performed action $A_i$.

$$C_i =< R_i, A_i > \tag{8}$$

This definition of a case is appropriate because it allows for approximating the action selection of both a reactive expert (using the environment state) and a state-based expert (using the entire run) since the run contains the environment state. However, since each run is simply a sequence of cases it still requires an analysis of the run to infer what state the expert is currently in.

## 3 Temporal Backtracking

The goal of the observing agent is to observe the expert and build a case base so that it can behave similarly to the expert when presented with similar sensory input. Ideally, the agent should be *behaviourally equivalent* to the expert. If the expert and agent are behaviourally equivalent then after encountering the same run they will produce the same action [15]. When performing case retrieval the objective of the agent will be to compare its current run to runs of the expert, which are stored as the problem portion of cases, in order to find the most similar expert run and reuse the associated action.

A run is composed of both environment states and actions. Therefore, when determining the similarity of two runs of equal length, $R^a$ and $R^b$, it is necessary to determine the similarity of the elements of the runs:

$$sim(R^a, R^b) = f(sim(S_i^a, S_i^b), sim(A_{i-1}^a, A_{i-1}^b), sim(S_{i-1}^a, S_{i-1}^b), \dots) \quad (9)$$

This requires defining two similarity metrics: the *problem similarity metric* and *solution similarity metric*. The problem similarity metric is used to calculate the similarity between two environment states and the solution similarity metric is used to calculate the similarity between actions. It should be noted that actions can, at different times, be part of both a case's problem portion and solution portion. The current action is the solution portion of a case since the case-base reasoning cycle is used to retrieve an appropriate action to perform. However, after an action is performed it is appended to a run, which is the problem portion of a case, so the action then becomes part of the problem.

The agent should, ideally, compare its entire run to runs in the case base but this may not be computationally feasible. As an agent interacts with the environment over time, the length of the run it has encountered will grow and so too will the amount of computation required to compare it to other runs. If the agent has received its $n$th environment state $S_n$ and is attempting to select its $n$th action to perform, $A_n$, then the run will contain $n$ environment states and $n-1$ actions. Given the previous definition of run similarity, this would make the similarity complexity $O(n)$. The value of $n$ could potentially be very large for a typical real-time system.

An alternative approach would be to only consider a fixed-sized run, of length $l$, for the agent. This would have the benefit of reducing the computational complexity of the similarity calculation to constant time, $O(1)$, but can result in a loss of information if an incorrect run size is selected. For example, in the situation where $l = 1$ the agent is ignoring all past information and behaving in a purely reactive manner. It may not even be possible to select an ideal value of $l$ to use if the necessary run length is time-varying or context-dependent. For example, the run length might be time-varying if a single environment state influences the expert's state. As the run grows, the influential environment state will move further and further into the past.

We propose an approach to run retrieval that starts with an initial run length of 1, only taking into account the current environment state, but can dynamically increase the run length if more information is necessary. Looking again at the definition of a run, we can see that the current run at time $t$, $R_t$, is composed of the run at $t-1$, $R_{t-1}$, along with the action performed in response to $R_{t-1}$, $A_{t-1}$, and the current environment state $S_t$.

$$R_t : R_{t-1} \xrightarrow{A_{t-1}} S_t \quad (10)$$

Each case, which is composed of a run and the associated action, can be rewritten as a tuple containing the current environment state, the action associated with the current run, the previous run and the action of the previous run:

$$C_t =< R_{t-1}, A_{t-1}, S_t, A_t > \quad (11)$$

Which can be further simplified as:

$$C_t =< C_{t-1}, S_t, A_t > \tag{12}$$

This simplification is beneficial because each case no longer needs to store the entire run but can instead store the most recent environment state and a link to the previous case.

The pseudocode for our retrieval approach is shown in Algorithm 1. The functions used by the algorithm each take three parameters: the current run of the agent (*run*), a collection of past runs (*pastRuns*), and a time offset that tells it how far back in the run to examine (*time*). Initially, the *stateRetrieve*(...) function is called and is given the current run of the agent, the entire case base, and a time offset of 0. This causes the function to only consider the most recent environment state (0 time units in the past) when comparing runs.

Two threshold values are used by the algorithm: the *problem threshold (PT)* and *solution threshold (ST)*. The problem threshold is used when comparing the environment state portions of runs (in the *stateRetrieve*(...) function) whereas the solution threshold is used when comparing the action portions of runs (in the *actionRetrieve*(...) function). Both functions retrieve the appropriate piece of the current run and past runs (using the *state*(...) and *action*(...) functions[1]) and compare those run pieces using either the problem similarity metric or solution similarity metric (line 3). If a run does not have an environment state or action at a specific time, because the run does not go that far back in the past, a null value will be returned. The similarity metrics will compute a similarity of 0 if one or both of the parameters are null. If the similarity is above the necessary threshold value (line 6), either the problem threshold or solution threshold, then the past run is added to the set of nearest neighbours ($NN$ on line 7).

After the current run has been compared to all past runs the nearest neighbour set is examined. If the nearest neighbour set is empty then the action associated with the most similar run (*bestRun*) is returned (line 10). In this situation there were no cases that were similar enough, based on the thresholds, but a best guess at the action is made. However, if there were one or more cases that were similar enough to the current run then it becomes necessary to see if they agree on which action should be performed. If all of the nearest neighbours had the same associated action ($NNactions$ has only one item) then that action is returned (line 11). If the nearest neighbours disagree on the action, more information is necessary to make the decision. The functions then recursively call each other using the current run, the nearest neighbours, and an updated time to examine (line 12). For example, initially the *stateRetrieve*(...) function is used to compare the runs using the current environment states (equivalent to a time offset of 0). If the function resulted in a non-agreeing nearest neighbour set then the algorithm will look further back in time by examine the actions at time offset 1 using the *actionRetrieve*(...) function. If examining the actions at

[1] When the *action*(...) function takes two parameters it returns the action that occured at a specified point in the run whereas when it only takes one parameter it returns the most recent action performed during the run.

**Algorithm 1:** Action Selection using Temporal Backtracking

**Input**: current run (*run*), candidate runs (*pastRuns*), time offset (*time*)
**Output**: action to perform (*action*)

---

**Function:** *stateRetrieve(run, pastRuns, time)* ***returns*** *action*

```
1  NN = ∅; NNactions = ∅; bestSim = −1; bestRun = NULL
2  foreach past ∈ pastRuns do
3      similarity = sim(state(run, time), state(past, time))
4      if similarity > bestSim then
5          bestSim = similarity; bestRun = past
6      if similarity > PT then
7          NN ← NN ∪ past
8          if action(past) ∉ NNactions then
9              NNactions ← NNactions ∪ action(past)
10 if NN == ∅ then return action(bestRun)
11 else if |NNactions| == 1 then return {NNactions}
12 else return actionRetrieve(run, NN, time + 1)
```

---

**Function:** *actionRetrieve(run, pastRuns, time)* ***returns*** *action*

```
1  NN = ∅; NNactions = ∅; bestSim = −1; bestRun = NULL
2  foreach past ∈ pastRuns do
3      similarity = sim(action(run, time), action(past, time))
4      if similarity > bestSim then
5          bestSim = similarity; bestRun = past
6      if similarity > ST then
7          NN ← NN ∪ past
8          if action(past) ∉ NNactions then
9              NNactions ← NNactions ∪ action(past)
10 if NN == ∅ then return action(bestRun)
11 else if |NNactions| == 1 then return {NNactions}
12 else return stateRetrieve(run, NN, time)
```

time offset 1 is still unable to produce an agreeing nearest neighbour set (or an empty nearest neighbour set) the algorithm will then look at the environment states at time offset 1 by calling the *stateRetrieve*(...) function.

The primary benefit of this approach is that the algorithm is able to dynamically backtrack, starting with the current environment state, until it has enough information (an agreeing set of nearest neighbours) to select an action. The algorithm recursively attempts to eliminate nearest neighbour cases by comparing portions of the runs that occur further in the past. In some situations the algorithm only needs to compare the current environment state while in other situations it may be necessary to compare a much longer portion of the runs. This is beneficial because even if an expert is state-based it may not perform all

of its reasoning using state information. There may be times when it behaves reactively and so performing a full run comparison would be unnecessary.

## 4 Experimental Results

Our experiments look to demonstrate the benefits of our retrieval approach when learning by observing a simulated obstacle avoidance robot. The robot moves around a *50 unit* $\times$ *50 unit* environment which contains a number of obstacles scattered throughout. There are 5 possible actions the robot can perform: move forward, move backward, turn left, turn right and reverse direction (turn 180 degrees).

$$\mathcal{A}_{robot} = \{Forward, Backward, Left, Right, Reverse\} \tag{13}$$

In order to sense the environment the robot has two sensors: touch and sonar. The touch sensor produces a binary value. When the robot comes into contact with something, like an obstacle, the touch sensor produces a value of 1. A value of 0 is produced if nothing is being touched. The sonar sensor produces a continuous value indicating the approximate distance to the nearest obstacle. Each environment state is then a pair containing the feature values from both sensors:

$$S_{robot} =< f_{tch}, f_{snr} > \tag{14}$$

Using the definition of the robot's action set and environment state we can now describe the problem similarity metric (Equation 15) and solution similarity metric (Equation 17) that will be used during retrieval. The problem similarity metric is calculated by taking the average similarity of each of the environment features (using Equation 16).

$$sim(S^a, S^b) = \frac{1}{2}\left(sim(f^a_{tch}, f^b_{tch}) + sim(f^a_{snr}, f^b_{snr})\right) \tag{15}$$

$$sim(f_i, f_j) = \begin{cases} 1 & ,iff_i = f_j \\ 1 - \frac{|f_i - f_j|}{f_i + f_j} & ,iff_i \neq f_j \end{cases} \tag{16}$$

The problem similarity metric returns continuous similarity values in the range $[0, 1]$ whereas the solution similarity metric produces a binary value depending on whether the actions are the same or not:

$$sim(A_a, A_b) = \begin{cases} 1 \ ,ifA_a = A_b \\ 0 \ ,ifA_a \neq A_b \end{cases} \tag{17}$$

An expert agent will be observed and learnt from in our experiments. If its touch sensor indicates it has come in contact with an obstacle (a sensor value of 1), the robot will be moved backward. Otherwise, the expert will base its action selection on the value of the sonar sensor. If the sonar value is less than

2 the robot will reverse direction, if the value is between 2 and 3 the robot will turn, and if the value is greater than 3 it will move forward. The direction the robot turns, when the sonar value is between 2 and 3, depends on what state the expert is in. The state of the agent is changed based on a previous action it has perform. This expert toggles its turning direction after each turn. In order to know which direction the expert will turn it is necessary to examine the run to see the last direction the expert turned.

The expert was observed interacting with the environment over a period of time. This resulted in a case base containing $50,000$ cases. Additionally, the expert agent was also observed in order to create testing case bases for use during evaluation. There were 25 testing case bases created and each case base contained $2,500$ cases. The placement of obstacles and the initial starting position of the robot was different when creating each of the 26 case bases (the main case base and the 25 testing case bases).

### 4.1 Class Separation

The selection of the expert was based on the assumption that its behaviour could not be learnt using only the current environment state. Using only the current environment state as inputs, inputs would appear similar to each other but result in different actions. The internal state of the agent would be a necessary feature in order to properly separate different classes in the problem space.

In order to test this assumption, the similarity of the *nearest like neighbour (NLN)* and the *nearest unlike neighbour (NUN)* for each case was calculated. The nearest like neighbour is the most similar case with the same associated action and the nearest unlike neighbour is the most similar case with a different associated action. Each case in the case base was compared to the remaining $49,999$ cases to find the NLN and NUN similarity values. The similarity when comparing cases is performed using the problem similarity metric and only takes into account the current environment state (the values of the touch and sonar sensors).

The mean similarity[2] of the nearest like neighbours and nearest unlike neighbours is shown in Table 1. All of the actions had a mean nearest like neighbour similarity that was nearly identical (a similarity of approximately 1.00). This should be expected since the size of the case base is much larger than the number of possible environment states (approximately 200 states if we discretize the sonar value to the nearest integer). For most actions, the mean NUN similarity was significantly lower (using a paired t-test with $p < 0.01$) than the mean NLN similarity. However, for both the *left* and *right* actions there was no significant difference between the mean NLN and NUN similarities. The cases with these actions, on average, had nearly identical like neighbours and unlike neighbours. Therefore, in many situations it would be impossible to determine the correct action to perform since there would be multiple identical cases in the case base but the cases would not all have the same action.

[2] The confidence intervals are not show in the table since, when rounded to two decimal places, they are all +/- 0.00.

| | Forward | Reverse | Backward | Left | Right |
|---|---|---|---|---|---|
| **NLN** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **NUN** | 0.86 | 0.92 | 0.67 | 1.00 | 1.00 |

**Table 1.** Mean similarity of nearest like neighbours and nearest unlike neighbours

### 4.2 Retrieval Results

The results from the previous section indicate that the left and right actions appear to be difficult to separate when using only the current environment state as the problem features. In order to test this, the accuracy of reactive retrieval (RR) was compared to the accuracy of temporal backtracking retrieval (TB). Reactive retrieval only uses the current environment state as features and performs a 1-nearest neighbour search to find the most similar case in the case base. The action associated with the nearest neighbour is returned. Temporal backtracking retrieval uses the approach described in Section 3. Each retrieval approach used the large cases base, containing $50,000$ cases, as the training case base. Each testing trial, 25 trials in total, used one of the testing case bases and used each case in the test case base as input to the retrieval algorithms. The tests looked to see how accurately the retrieval algorithms returned an action that matched the known action of the test case.

One modification to the temporal backtracking approach is that the problem threshold was split into two: the *current problem threshold (CPT)* and *past problem threshold(PPT)*. This split was done to allow the algorithm to be more lenient on the similarities of past environment states. For the temporal backtracking approach three similarity settings were tested: $\{CPT = 0.99, PPT = 0.90, ST = 0.90\}$, $\{CPT = 0.99, PPT = 0.90, ST = 0.00\}$. $\{CPT = 0.99, PPT = 0.00, ST = 0.90\}$. The first set of thresholds takes into account both past actions and past environments, the second only takes into account past environments and the third only takes into account past actions.

The results, in Table 2, show the accuracy results (and 95% confidence intervals) of the reactive retrieval approach and the results of temporal backtracking retrieval using the best threshold set. For the expert, the best threshold values were $\{CPT = 0.99, PPT = 0.00, ST = 0.90\}$. The best threshold values found confirm our assumptions about what information in the run is necessary to imitate the expert. There was a significant increase (using a paired t-test with $p < 0.01$) in the overall retrieval accuracy, the left action accuracy and the right action accuracy. However, there were small, yet statistically significant, decreases in the accuracy of the forward[3] and reverse actions. It should also be noted that, while not shown in the table, the temporal backtracking results using the other threshold values also significantly increased the overall accuracy. This shows us that while the settings for the threshold values are important it is possible to improve over reactive retrieval using non-optimal thresholds.

[3] The accuracy values are rounded to 1.00, but if no rounding is performed there is a small, statistically significant decrease to the temporal backtracking values.

| | Forward | Reverse | Backward | Left | Right | Overall |
|---|---|---|---|---|---|---|
| **RR** | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 0.48±0.02 | 0.50±0.02 | 0.80±0.01 |
| **TB** | 1.00±0.00 | 0.97±0.01 | 1.00±0.00 | 0.74±0.02 | 0.73±0.02 | 0.89±0.01 |

**Table 2.** Retrieval accuracy of standard reactive retrieval and temporal backtracking

One other area of note is how far back in the run the temporal backtracking approach needed to examine. In many cases, the algorithm only examined the current state of the environment. However, in some situations the algorithm needed to go as far as 58 time steps in the past. If it would not have had those past cases to examine it would have been unable to successfully select which action to perform. Even though there are only approximately 200 environment states and 5 actions, a run of length $n$ would have approximately $200^n \times 5^{n-1}$ states. When comparing runs, a relatively simple and small state space becomes much larger. Having a case base that only contained all runs of length 2 would have required approximately $200,000$ cases (four times larger than the case base we used) and would have been insufficient when longer runs were required.

## 5 Related Work

Most previous works that have made use of learning by observation, including case-based reasoning [4, 10, 11] as well as other techniques [2, 7], have only used the agent's current sensory information as inputs during action selection. This limits these approaches to only learning the behaviour of reactive experts. However, approaches that use planning [9] are able to learn from state-based experts since the plan nodes and transitions implicitly contain any necessary state information. These planning approaches require the goals of the expert to be manually defined beforehand, which may not be possible if the expert's behaviour is not known. Additionally, these planning approaches have been found to have difficulty learning reactive behaviour [9].

Previous work has examined using sequences of input data during reasoning [8, 13]. These approaches, unlike our own, only make use of past inputs and not past actions. Approaches that do reason with entire episodes of behaviour have been called trace-based reasoning [1] or episode-based reasoning [12]. These approaches have treated past episodes both as sequences of actions [3] and as sequences of actions and stimuli [1, 12]. The primary difference from our own work is that these techniques require a known start and end of each episode. If the optimal episode length was unknown, or there was no optimal episode length, it would be necessary to have a case base with all possible episodes that could be generated from a run. This would result in a space complexity that is $O(n^2)$, where $n$ is the length of the run, compared to $O(n)$ using our approach. Also, they use similarity calculations that involves comparing entire episodes to each other. Both their similarity measures and ours would have a computational complexity of $O(n)$ worst case but ours could be $O(1)$ in the best case (when only the most recent sensory input is relevant). Using an entire episode during

similarity calculation could also potentially introduce irrelevant information if only the last part of the episode is relevant.

This work can be thought of as a sequential decision making task. Many other approaches to sequential decision making assume Markovian sequence transitions (the probability of reaching the next item in the sequence is only dependant on the current item), require discretized sequence elements, or have difficulty when important state information occurred far in the past [14]. These limitations are not an issue with our approach since data can be examined far in the past (in our experiments the algorithm went as far as 58 time steps in the past) and the similarity calculations allow for continuous sensory inputs and actions.

## 6 Conclusions

In this paper we have described an approach to case retrieval for use in learning by observation systems. Unlike past approaches, our approach takes into account the entire run of an expert rather than just the current environment state. This is beneficial because it allows learning from experts who reason using internal state information. Additionally, the amount of past information required does not need to be defined a priori as our approach is able to dynamically look further back in the past when necessary.

Our experiments examined the ability to learn from a simulated obstacle avoidance robot. While the behaviour of this agent was simple, we showed that retrieval using only the current environment state as input was not sufficient to predict actions that were dependant on the agent's internal state. However, our temporal backtracking approach, which used past environment states and actions during retrieval, was able to significantly improve the retrieval accuracy. In some situations no past information was necessary during retrieval but in many situations information from many past environment states and actions was needed. We found selecting appropriate threshold values for use in the temporal backtracking algorithm is able to improve results but even using non-optimal thresholds significantly improves the results compared to standard retrieval. In this paper, we presented the results from a single expert agent. Further results, omitted due to space limitations, showed similar performance improvements when learning from two other experts: one agent that bases state-transitions on past environment states and another that bases state-transitions on both past environment states and past actions. While our approach did show significant improvements when learning from a simulated obstacle avoidance robot, future work will look at the applicability of our algorithm in other domains and with agents that have more than two internal states. Also, our work assumes that the expert does not switch between behaviours in a way that can not be identified by examining the run. If the expert did unexpectedly switch behaviours, we would need to extend our approach to separate unrelated behaviours that are part of the same run.

## References


1. Champin, P.A., Prié, Y., Mille, A.: MUSETTE: Modeling USEs and tasks for tracing experience. In: Workshop From Structured Cases to Unstructured Problem Solving Episodes For Experience-Based Assistance at the 5th International Conference on Case-Based Reasoning. pp. 279–286 (2003)
2. Coates, A., Abbeel, P., Ng, A.Y.: Learning for control from multiple demonstrations. In: 25th International Conference on Machine Learning. pp. 144–151 (2008)
3. Doumat, R., Egyed-Zsigmond, E., Pinon, J.M.: User trace-based recommendation system for a digital archive. In: 18th International Conference on Case-Based Reasoning. pp. 360–374 (2010)
4. Flinter, S., Keane, M.T.: On the automatic generation of cases libraries by chunking chess games. In: 1st International Conference on Case-Based Reasoning. pp. 421–430 (1995)
5. Floyd, M.W., Davoust, A., Esfandiari, B.: Considerations for real-time spatially-aware case-based reasoning: A case study in robotic soccer imitation. In: 9th European Conference on Case-Based Reasoning. pp. 195–209 (2008)
6. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating RoboCup players. In: 21st International Florida Artificial Intelligence Research Society Conference. pp. 251–256 (2008)
7. Grollman, D.H., Jenkins, O.C.: Learning robot soccer skills from demonstration. In: IEEE International Conference on Development and Learning. pp. 276–281 (2007)
8. Martín, F.J., Plaza, E.: Ceaseless case-based reasoning. In: 7th European Conference on Case-Based Reasoning. pp. 287–301 (2004)
9. Ontañón, S., Ram, A.: Case-based reasoning and user-generated AI for real-time strategy games. In: Gonzáles-Calero, P.A., Gomez-Martín, M.A. (eds.) AI for Games: State of the Practice (2011)
10. Romdhane, H., Lamontagne, L.: Forgetting reinforced cases. In: 9th European Conference on Case-Based Reasoning. pp. 474–486 (2008)
11. Rubin, J., Watson, I.: Similarity-based retrieval and solution re-use policies in the game of Texas Hold'em. In: 18th International Conference on Case-Based Reasoning. pp. 465–479 (2010)
12. Sànchez-Marrè, M., Cortés, U., Martínez, M., Comas, J., Rodríguez-Roda, I.: An approach for temporal case-based reasoning: Episode-based reasoning. In: 6th International Conference on Case-Based Reasoning. pp. 465–476 (2005)
13. Shih, J.: Sequential instance-based learning for planning in the context of an imperfect information game. In: 4th International Conference on Case-Based Reasoning. pp. 483–501 (2001)
14. Sun, R., Giles, C.L.: Sequence learning: From recognition and prediction to sequential decision making. IEEE Intelligent Systems 16(4), 67–70 (2001)
15. Wooldridge, M.: An introduction to multiagent systems. John Wiley and Sons (2002)