Toward a Domain-independent Case-based Reasoning Approach for Imitation: Three Case Studies in Gaming

Michael W. Floyd and Babak Esfandiari

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

Abstract. We describe a domain-independent approach to learning by observation that uses case-based reasoning to allow a software agent to behave similarly to an expert when presented with a similar set of input stimuli. Case studies have been performed in two simulated domains, soccer and space combat, and a physical robotics domain and our experimental results show that successful imitation is possible in each of those domains.

1 Introduction

Transferring an expert's knowledge to a software agent can be a difficult task. This is especially true if the expert has difficulty modelling its knowledge, possibly if they lack computer programming skills, or they are not fully aware of all details related to how they perform a task. *Learning by observation* attempts to shift the burden of knowledge transfer from the expert to the software agent. The agent learns by watching the expert perform a task and, when faced with the same task, aims to behave in a similar manner.

Ideally, such an agent should operate completely autonomously from the expert and not require the expert to provide any prior knowledge. Existing approaches to learning by observation, however, require some level of background knowledge. This background knowledge can include information about the tasks or goals of the expert [1, 2], which sensory stimuli the expert reasons with [3] or how specific actions influence the environment [4]. While all of this information helps when learning a specific task, bias is added which makes it difficult to learn other unrelated tasks. Ideally we want to be able to imitate unrelated behaviours without adding extra expert knowledge so that the learning system can be deployed in a variety of domains. The primary contribution of our own work [5–7] is a learning by observation system that makes minimal prior assumptions about the expert although we have only experimented in a simulated soccer domain.

The interactions between the expert and its environment can be thought of as a series of environment states and actions by the expert [8]. Both the possible environments states, $\mathcal{S}$, and possible actions, A, are finite sets containing all environment states and actions that may be encountered:

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

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

When an agent is observed for a period of time, a *run*, R, of environment states and actions will be observed:

$$R : S_0 \overset{A_0}{\rightarrow} S_1 \overset{A_1}{\rightarrow} S_2 \overset{A_2}{\rightarrow} \dots S_{u-1} \overset{A_{u-1}}{\longrightarrow} S_u \tag{3}$$

The goal, when learning by observation, is to use one or more runs of an expert to approximate how the expert selects which actions to perform based on the state of the environment:

$$A_i = f(S_i, A_{i-1}, S_{i-1}, A_{i-2}, \dots) \tag{4}$$

The remainder of this paper will detail our domain-independent approach to learning by observing agents and examine what types of agents it is suitable for imitating. Section 2 will provide the model of our agent and case structure. An analysis of several properties of software agents, and their environments, are described in Section 3. Three case studies in the domains of robotic soccer, space combat, and physical robotics will be described in Section 4. These case studies will be used to examine the relation between an agent's properties and the ability of our system to imitate that agent. A summary of related work will be presented in Section 5, followed by conclusions and areas for future work in Section 6.

2 Agent Model

When looking at the ability of a software agent to reason, we can think of the current run as the *problem* that is to be solved and the performed action as the *solution*. This leads us to define a case, C, as a pair containing the current run, R, and the performed action, A.

$$C = \{R, A\} \tag{5}$$

The run is, as shown in Equation 3, a sequence of past environment states and actions. The state of the environment can be decomposed into the sensory stimuli that can be observed. If, in a particular environment, an agent can observe m *types* of stimuli then the environment state can be written as:

$$S = \{V_1, \dots, V_m\} \tag{6}$$

Each type of stimulus, V_i, is represented by a multi-valued attribute that contains all n_i *instances*, o, of that type of stimulus ($V_i = \{o_1, \dots, o_{n_i}\}$). This implies that the number of stimuli of a particular type can vary in different cases. The need to allow for a varying number of instances of each stimulus type is related to the fact that the agent may only have a partial view of its environment at any given

time. For example, in a soccer domain the number of teammates an agent can observe in its field of vision may change depending on where the agent is looking. At different times the agent might see no teammates, all of the teammates, or any number in between. Even in domains where an agent has a complete world view it might still be necessary to treat each stimulus as a multi-valued attribute. This is particularly true if objects can be added or removed from the environment, or if the agent is unable to uniquely identify similar objects.

Using such an approach, cases can be generated in an automated manner through passive observation. Using a passive approach, the *expert*, as shown in Figure 1, interacts with the *environment* and may not be aware it is being observed. The *observer* is able to view, and record, both the state of the environment and the actions performed by the expert so that they can later be used to construct cases.

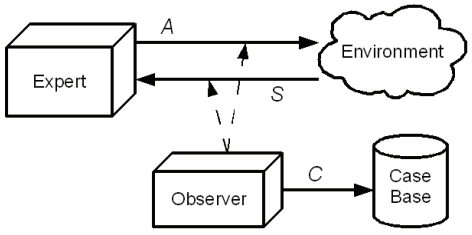

Fig. 1. Passive observation of an expert interacting with the environment

After automatically building a case base by observing an expert, the learning agent will then be able to use that case base when it interacts with the environment. When the agent observes a new environment state, it should ideally perform the same action the expert would have performed had the expert encountered a similar run. This requires using the current run as a query to the case base. The case base is searched using a standard k-nearest neighbour search to find a case with a run that is a minimal distance from the current run. Once the nearest neighbour search has found the case that is most similar to the current state of the environment, the action associated with the retrieved case is used. Our approach does not perform any adaptation since the agent is attempting to behave like the expert and has no idea what goal it is trying to achieve (a more detailed description of our case description and retrieval can be found in [5]).

The data flow of our approach can be thought of as a three step process: acquisition, preprocessing and deployment. As we mentioned previously, the cases are *acquired* in a completely automated manner by observing the interactions between the expert and the environment. While this automatic case acquisition allows for quick and inexpensive case generation it make no guarantees as to what cases will be acquired or what information is contained in those cases. For example, if the expert never performs a certain action or encounters a certain

environment state then those actions and states will not be available in the case base. Likewise, since no domain knowledge is provided by a domain expert such knowledge will need to be extracted from the cases. This is addressed through *preprocessing* the raw case base by performing automated tasks like feature selection, removal of redundant cases and ensuring the case base can be searched within real-time limits (a more detailed account of preprocessing can be found in [6]). Once an acceptable case base has been created it can then be *deployed* in order to allow the imitative agent to attempt to imitate the behaviour of the expert.

3 Agent and Environment Properties

Now that we have described how case-based reasoning can be used to imitate the behaviour of agents, we will turn our attention to the agents and their environments. The following lists several properties that will be examined and used to classify agents:

Goals: Each agent will perform a behaviour that attempts to achieve certain goals. The imitation system should ideally be able to imitate agents regardless of the goals they try to achieve and should require no knowledge of what those goals are.

Observability: The agent can have either a complete or partial view of the environment. With a complete view the agent is able to observe the entire environment at once, however this is often not possible due to limitations on the available sensors. Instead, most realistic agents can only view a portion of the environment at once. Partial observability can result in a variable number of objects visible to the agent at any given time which may result in the need to model sensory inputs as multi-valued attributes (it may also be necessary in fully observable environments if objects can be added or removed from the environment).

Noise: The sensory observations received by the agent may be subject to noise. This noise can be small, resulting in values being different from their true values, or large making some sensory information completely unknown to the agent. For example, if an object was located far from an agent the agent might be able to detect what type of object it is observing but not finer details about the object.

State: The agent may not reason exclusively with its current sensory inputs but may maintain an internal state. Since this state information is not directly visible to an observer it makes the imitation task significantly more complicated.

Environment Physicality: The agent could either be in a simulated environment or a physical environment (like controlling a robot).

4 Case Studies

We look to examine several agents with different goals and behaviours. Two simulated domains, soccer and space combat, and one physical domain will be used. A summary of the agents and their properties can be seen in Table 1.

Domain	Agent	Noisy	Fully Observable	State	Physical
Soccer	Krislet	Yes	No	No	No
Soccer	Sprinter	Yes	No	No	No
Combat	Shooter	No	No	No	No
Robot	Avoid	Yes	No	Yes	Yes
Robot	Arm	Yes	No	No	Yes

Table 1. Summary of agents and their properties

4.1 Simulated Robotic Soccer

The first domain we examine is RoboCup [9] simulated robotic soccer (this has been the focus of our previous research [5–7]) . The following types of objects exist in RoboCup: soccer balls, goal nets, boundary lines, boundary flags, teammates, opponents, unknown players[1]. At any given time the agent may see between *0* and *80* total objects, each of which is represented by its continuous-valued distance and direction (relative to the agent). The state of the environment in a simulated RoboCup game can be represented as:

$$S_{SOCCER} = \{ball, goal, line, flag, teammate, opponent, unknown\} \quad (7)$$

Each item in S_{SOCCER} represents a multi-valued attribute containing the objects of that type that are currently visible in the agent's field of vision. The agent can kick, move forward (dash) or turn its body. This makes the set of possible agent actions:

$$A_{SOCCER} = \{kick, dash, turn\} \quad (8)$$

In our experimentation we use two different agents as experts[2]. The first agent we use, Krislet, actively attempts to achieve the goals of soccer. Krislet agents turn until they can see the soccer ball, run toward the ball and then try to kick the ball toward the opponent's goal net. The second agent we use, Sprinter, does not perform typical soccer behaviour. Sprinter agents repeatedly run from one goal net to the other and make no attempt at scoring goals. These agents were selected to show that even in the same domain agents can have very different goals and behaviour, so any background knowledge about the goals of a particular expert may not be transferable to other experts.

For each expert a case base containing *5000* cases was used (this size of a case base was selected so that the case base could be searched within the real-time limits of the agent). A series of *1000* testing cases, or more specifically the

[1] Players can be of an unknown type if the agent can not tell what team they are on due to noise. Although RoboCup is a simulated domain there is a built in noise model to add realism.

[2] While we refer to the agents as experts we do not imply they are complex in nature. We use expert to refer to an agent with some amount of knowledge we wish to attain.

environment states of the cases, were used as input to the imitating agent. The action outputted by the imitating agent, who only considered the most recent sensory input (a run of length 1), was then compared to the known action from the case. The overall performance was measured using the f-measure statistic with values ranging from 0 (low) to 1 (high).

The results are shown in Table 2. In general, the f-measure values are fairly high for both agents (it should be noted that previous work [6] has shown that these results can be significantly improved by automatically preprocessing the case base, although no preprocessing was performed in these experiments). The exception is the results associated with *kick* in Krislet and *turn* in Sprinter, which lower the overall f-measure values. The reasons these values are lower is because the data sets are severely imbalanced, with less than 1% of Krislet cases having the *kick* action and less than 5% of Sprinter cases having the *turn* action (Sprinter never kicks, which is why there is no f-measure value for the kick action). However, even though the f-measure values are low for those actions when the case-based reasoning system is deployed in a game of soccer the actions appear to be performed properly. When watching the CBR agent play soccer, it is difficult to differentiate between that agent and the original expert agent.

	f-measure	f_{dash}	f_{turn}	f_{kick}
Krislet	0.59 (+/- 0.005)	0.71	0.80	0.25
Sprinter	0.69 (+/- 0.005)	0.91	0.48	—

Table 2. Results when learning from RoboCup agents

4.2 Simulated Space Combat

Thus far we have shown the ability of our system to imitate agents, with a variety of different behaviours, in the RoboCup soccer domain but we now look to extend our experiments into another simulated domain. The domain we use is XPilot [10], a simulated space combat game, where an agent controls a space ship and attempts to destroy enemy ships. In this domain the agent is able to detect the location of enemy ships (between *0* and approximately *10* other ships) and can move, turn or shoot:

$$S_{XPILOT} = \{ships\} \tag{9}$$

$$A_{XPILOT} = \{move, turn, shoot\} \tag{10}$$

The agent we imitate, which we will call *Shooter*, continuously turns until it can see at least one enemy ship and then shoots at the nearest ship. The results, when using a case base of *1000* cases and *1500* test cases, are shown in Table 3 (the Shooter agent does not perform the *move* action so that action is not listed in the table). These results are similar to what we got in the RoboCup

experiments and is what we would have expected given the similar properties of the agents and the environments. Like with the RoboCup experiments the poorest performance was related to rare actions, in this case the *shoot* action, although this could likely be improved by preprocessing the case base to balance the number of cases related to each action.

	f-measure	f_{turn}	f_{shoot}
Shooter	0.64	0.77	0.50

Table 3. Results when learning from XPilot agent

4.3 Physical Robots

Our experiments up to now have exclusively examined simulated domains, but now we turn our attention to two agents that control physical robots. The first agent, which we will call *Avoid*, controls a small wheeled robot and attempts to avoid collisions with obstacles. The robot has two sensors, sonar and touch, and can perform four movement actions:

$$S_{AVOID} = \{sonar, touch\} \tag{11}$$

$$A_{AVOID} = \{forward, backward, left, right\} \tag{12}$$

If an object is detected as being close to the robot, using the sonar sensor, the robot will turn left or right to avoid a collision. If it collides with an object, as indicated by the touch sensor, it will move backward and then turn. Otherwise it will just move forward. While this behaviour is mostly reactive in nature, the agent does have a simple internal state that determines which direction it should turn. The internal state ensures that the robot toggles between turning left and right.

The second agent, called *Arm*, controls a robotic arm robot. This robot has three sensors and can perform five actions:

$$S_{ARM} = \{sound, touch, colour\} \tag{13}$$

$$A_{ARM} = \{armForward, armReverse, armStop, closeClaw, stopClaw\} \tag{14}$$

Upon detecting a significantly loud sound, on the sound sensor, the arm begins moving forward until the touch sensor signals it has come in contact with an object. If the colour sensor ever determines a red object is within the claw's grasp the claw will be closed around the object and the arm will move in reverse. However, if it ever determines a blue object is within the claws grasp it will not close but instead move the arm in reverse.

There are three primary differences, not including that they control physical robots, between these agents and the ones we have previously examined. First,

these agents perform sequences of actions in response to each input instead of a single action. Secondly, since a sequence of actions is performed each action in that sequence is performed for a set amount of time. This requires each action to have an associated parameter related to how long it should be performed. Lastly, the sensory attributes are single-valued, with one attribute per sensor, rather than multi-valued.

For both agents, our system was able to imitate the behaviour very well. For each agent a case base was created by giving the agent *500* randomly created sensory inputs and recording the resulting action sequence. Using a separately generated set of *1000* testing cases each agent was able to perfectly reproduce, with 100% accuracy, the expected behaviour of the agent. This includes selecting the correct action sequence and executing each action for the correct duration[3]. However, for the Avoid agent, this perfect accuracy is only achieve by considering *left* and *right* turns identical. This occurs because two cases can have identical sensory information but different actions since the cases do not contain the internal state of the agent when it selected which actions to perform. This demonstrates that the presence of an internal state, even in a domain that can be imitated with high accuracy, can significantly complicate the imitative process.

5 Related Work

Case-based reasoning has been successfully applied to a variety of games and simulations including robotic soccer [11–13], American football [14], real-time strategy [15] and first-person shooters [16]. One promising use of CBR in games has been to create cases by observing experts. Examples of this include Tetris [17], real-time strategy [1], poker [4], chess [18] and space invaders [19].

The primary difference between these works and our own is that the processes for case generation and case retrieval are optimized for the specific domains and therefore are not suitable for general purpose learning by observation. The work most similar to our own involves using case-based planning in real-time strategy games [2]. They show their approach to be applicable to several different games, however domain knowledge is required for each new game. This knowledge involves having an expert define the goals of the game and the state of the environment associated with each goal.

Learning by observation has also been explored using other machine learning approaches. In robotics, learning from observation has been used in tasks such as controlling a robotic arm [20] and teaching a robotic helicopter aerial manoeuvrers [21]. Both of these approaches use a model for the movements of the robots, with the model parameters learnt through observation. This requires prior knowledge of an appropriate model to use and does not take into account any external stimuli. Grollman and Jenkins [22] have developed a system that allows a robotic soccer player to be simultaneously controlled by a human and

[3] The agent only uses a finite set of discrete action durations. If there was more variability in these durations then we would likely see error in the durations produced by the imitating agent.

an autonomous system. When the human is actively controlling the robot, the autonomous system learns from observing the human. This approach has been successful for learning simple behaviours, like moving or kicking, but has difficulty identifying when transitions between simple behaviours should occur.

Multi-layered perceptrons have been used to control the behaviour of agents in a first-person shooter game [3]. This approach requires a fixed set of input features, so it is not suited for situations where the inputs are multi-valued attributes. Similarly, learning by observation has been used to train virtual agents to move in life-like ways [23]. The movements of the agents are rated based on a fitness function, so each novel behaviour requires a user to define an appropriate fitness function.

6 Conclusions and Future Work

The technique for learning by observation using case-based reasoning described in this paper allows an agent to learn without any knowledge of the goals of the expert being observed. Three domains with different environments and goals have been studied. Software agents, with a variety of properties, were able to be successfully used as experts. The primary contribution of this approach is that a single case-based reasoning system is used in all domains. The only difference is the data contained in the cases, with the types of sensory stimuli and actions varying in the different domains.

While we have shown this approach to be applicable it three different domains, there still exist several limitations that provide areas for future work. The experts that have been observed have been largely reactive in nature, however more complex experts will likely have behaviour that is strongly related to their internal state. Even for the obstacle avoidance robot, which only had a simple internal state, it was not possible to fully account for the agent's state. We will look at how the entire run, instead of just the last environment state, can be used to better imitate stateful behaviour.

References

1. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: Case-based planning and execution for real-time strategy games. In: 7th International Conference on Case-Based Reasoning. (2007) 164–178
2. Mehta, M., Ontañón, S., Amundsen, T., Ram, A.: Authoring behaviors for games using learning from demonstration. In: Workshop on CBR for Computer Games at the 8th International Conference on Case-Based Reasoning. (2009)
3. Thurau, C., Bauckhage, C.: Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In: Proceedings of the GAME-ON Conference. (2003)
4. Rubin, J., Watson, I.: SARTRE: System overview. A case-based agent for two-player texas hold'em. In: Workshop on CBR for Computer Games at the 8th International Conference on Case-Based Reasoning. (2009)

5. 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. (2008) 251–256
6. 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. (2008) 195–209
7. Floyd, M.W., Esfandiari, B.: An active approach to automatic case generation. In: 8th International Conference on Case-Based Reasoning. (2009) 150–164
8. Wooldridge, M.: An introduction to multiagent systems. John Wiley and Sons (2002)
9. RoboCup: Robocup official site. http://www.robocup.org (2010)
10. XPilot-AI: Xpilot-AI A library for writing XPilot bots. http://www.xpilot-ai.org (2010)
11. Wendler, J., Lenz, M.: CBR for dynamic situation assessment in an agent-oriented setting. In: AAAI Workshop on Case-Based Reasoning Integrations. (1998)
12. Steffens, T.: Adapting similarity measures to agent types in opponent modeling. In: AAMAS Workshop on Modelling Other Agents from Observations. (2004) 125–128
13. Ros, R., Veloso, M., de Mántaras, R.L., Sierra, C., Arcos, J.L.: Retrieving and reusing game plays for robot soccer. In: 8th European Conference on Case-Based Reasoning. (2006) 47–61
14. Aha, D.W., Molineaux, M., Sukthankar, G.: Case-based reasoning in transfer learning. In: 8th International Conference on Case-Based Reasoning. (2009) 29–44
15. Molineaux, M., Aha, D.W., Moore, P.: Learning continuous action models in a real-time strategy environment. In: 21st International Florida Artificial Intelligence Research Society Conference. (2008) 257–262
16. Auslander, B., Lee-Urban, S., Hogg, C., Muñoz-Avila, H.: Recognizing the enemy: Combining reinforcement learning with strategy selection using case-based reasoning. In: 9th European Conference on Case-Based Reasoning. (2008) 59–73
17. Romdhane, H., Lamontagne, L.: Reinforcement of local pattern cases for playing Tetris. In: 21st International Florida Artificial Intelligence Research Society Conference. (2008) 263–268
18. Flinter, S., Keane, M.T.: On the automatic generation of cases libraries by chunking chess games. In: 1st International Conference on Case-Based Reasoning. (1995) 421–430
19. Fagan, M., Cunningham, P.: Case-based plan recognition in computer games. In: 5th International Conference on Case-Based Reasoning. (2003) 161–170
20. Atkeson, C.G., Schaal, S.: Robot learning from demonstration. In: Fourteenth International Conference on Machine Learning. (1997) 12–20
21. Coates, A., Abbeel, P., Ng, A.Y.: Learning for control from multiple demonstrations. In: 25th International Conference on Machine Learning. (2008) 144–151
22. Grollman, D.H., Jenkins, O.C.: Learning robot soccer skills from demonstration. In: IEEE International Conference on Development and Learning. (2007) 276–281
23. Dinerstein, J., Egbert, P.K., Ventura, D., Goodrich, M.: Demonstration-based behavior programming for embodied virtual agents. Computational Intelligence **24**(4) (2008) 235–256