# A General-Purpose Framework for Learning by Observation

by

**Michael W. Floyd, B.Eng., M.A.Sc.**

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of
the requirements for the degree of
**Doctor of Philosophy**
in
Electrical and Computer Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario, Canada
May 2013

The undersigned recommend to

the Faculty of Graduate and Postdoctoral Affairs

acceptance of the thesis

# A General-Purpose Framework for Learning by Observation

Submitted by **Michael W. Floyd, B.Eng., M.A.Sc.**

in partial fulfilment of the requirements for the degree of

**Doctor of Philosophy**

in

Electrical and Computer Engineering

---

Chair
Howard Schwartz, Department of Systems and Computer Engineering
Carleton University

---

Thesis Supervisor
Babak Esfandiari, Department of Systems and Computer Engineering
Carleton University

---

External Examiner
Luc Lamontagne, Department of Computer Science and Software Engineering
Laval University

Carleton University

2013

# Abstract

Learning by observation allows domain experts with no programming skills to train a software agent or robot by moving the burden of knowledge transfer from the expert to the agent itself. Instead of being explicitly programmed, the agent learns a behaviour by watching the expert perform it. This thesis examines the areas that are specific to learning by observation in order to come up with a general-purpose framework for learning by observation. The framework is represented as a cyclical workflow containing the four major tasks of learning by observation: *modelling*, *observation*, *preprocessing* and *deployment*. Each of these tasks are examined in order to identify how the task should be performed in a general-purpose learning system so that an agent can learn a variety of behaviours from a variety of experts in a variety of environments. In effect, the *same agent* can be retrained for each new task. Techniques to perform each of these tasks are provided.

A general-purpose modelling framework is proposed, along with an agent design that uses the framework, in order to minimize the number of changes necessary when an agent changes environments. The agent design is further enhanced to allow hardware to be added or removed from a robot without any reprogramming of the agent. For the observation task, two observation acquisition strategies are provided that take advantage of the availability of the expert and they allow the agent to make observations it would not have been able to normally. The preprocessing task is enhanced by allowing the agent to analyze the expert's behaviour in order to remove

errors and characterize how the expert reasons. Finally, an approach is presented, in the deployment task, that allows an agent to learn from experts who reason with internal information. A variety of domains are used for case studies and evaluations. Each technique is used on a selection of the following domains: a physical obstacle avoidance robot, a robotic arm, simulated soccer, Tetris and a simulated obstacle avoidance robot.

# Acknowledgments

I would like to thank my family for their support and patience throughout this process. I would also like to thank Professor Babak Esfandiari for his valuable guidance, feedback and encouragement.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

A search and rescue robot is designed to go into hazardous disaster zones and assist in saving human lives. However, there is no way to know in advance where a disaster will occur or the specific properties of that disaster. Different disaster zones may require different hardware or behaviours, so programming the robot to handle all possible situations may not be possible. This would require regularly reprogramming the robot as the environment changes or behaviours are modified. The human members of a search and rescue team may have the expertise to deal with the changing requirements but not the technical skills necessary to reprogram the robot. Instead, it may be desirable for them to just demonstrate how the robot should behave and have the robot learn from those demonstrations.

Similarly, a robot that assists elderly patients would need to be flexible enough to handle multiple patients and tasks. Each patient would have different needs and preferences, so a single task might need to be performed in several ways. A health care professional would have the expertise to identify how a patient should be assisted but, like the search and rescue expert, might not have the technical skills necessary to train the robot. If the robot could just observe the health care worker assisting patients, it could learn from these observations and continue helping the patients even when the worker is gone.

The expert could train a learning by observation agent to control the robot by taking control of the robot, possibly with a remote control or other user interface, and instructing the agent it should begin observing (for example, pressing a *record* button). The learning agent would observe how the expert controls the robot and learn from those observations until the expert indicates it has finished demonstrating (pressing a *stop* button). Finally, the learning agent could be instructed to perform the behaviour itself (pressing a *play* button).

This type of learning, called *learning by observation*, only requires the expert to be able to perform the task and makes no assumptions about the technical skills of the expert, the goals of the expert or the task being performed. Additionally, it requires no extra time from the expert (other than the time to actually demonstrate the behaviour). The burden of knowledge transfer is moved from the expert to the software agent or robot. Even for an expert that has significant technical skills and agent training experience, learning by observation should ideally allow an agent to be trained much faster than traditional methods.

## 1.1 Motivation

As the name implies, a learning by observation agent learns to perform a task by observing that task being performed by an expert. Since the agent learns by observing the expert, it is only able to use what is visible to an outside observer: the *inputs* received by the expert and the *outputs* the expert produces. The goal of learning by observation is to use the observations so that, when presented with similar inputs, the learning agent will produce similar outputs. This is in contrast to many methods that learn from passive and static sources of data.

In order to learn a variety of tasks from a variety of experts and in a variety

of situations, a learning by observation agent should be general-purpose and task-independent in nature. This prevents the agent from being biased toward a specific expert, domain or behaviour. Perhaps more importantly, it allows for an agent that can be quickly repurposed if its task or environment changes. Furthermore, existing learning by observation systems are developed in an ad hoc manner and require significant human intervention if the domain or behaviour being learnt are changed. Ideally, an agent should not need to be reprogrammed by a human every time it is required to learn a new task.

There are several key opportunities and difficulties in creating such a system:

- **Means of Observation:** The agent learns by observing an expert rather than learning from a properly labelled training file. This requires the agent to be able to observe and record the expert's behaviour, and learn from those observations. This also means the expert can be part of a learning interaction and be actively involved in the learning task. However, situations may occur where the agent makes many similar observations of a certain aspect of the expert's behaviour while not making any observations related to other aspects. It would be beneficial if the agent was aware of what information it had learnt well and, more importantly, what it had not learnt.

- **Quality of Observations:** There are no guarantees of the quality of the observations. The learning agent is only able to observe the expert's inputs and outputs, so the agent may not be successfully attributing an output to the input (or inputs) that caused it. Similarly, the agent is unable to observe if any non-visible information, like the expert's beliefs or intentions, factored into the reasoning process. Expert error, when performing an output, or observer error, when recording inputs and outputs, can also degrade the quality of the observed traces.

- **Domain Knowledge:** The learning agent does not have an explicit representation of the task it is learning since it can be trained on a variety of tasks. This means the learning agent will not be aware of the goals of the task or have a utility function to maximize. Without this knowledge, it is more difficult for the learning agent to know if it is learning the behaviour properly or to verify the quality of the observations it collects.

## 1.2 Objectives

The goal of this research is to formalize the processes performed by learning by observation systems and develop a framework that allows learning by observation agents to be developed in a modular and task-independent manner. This leads to the following research question:

**Research Question**: What are the issues specific to learning by observation? Is it possible to come up with a general-purpose and task-independent framework for learning by observation and, if so, what are the components of such a framework?

In order to address the primary research question, the following additional questions will also be asked (how each question was answered will be discussed in the following section):

1. **Versatility**: Can a single learning by observation system be used to learn in a variety of domains without making any major modifications to the learning agent?

2. **Observation**: Are there certain input problems that the learning agent may encounter when deployed but never encounter by passively observing the expert?

If so, can alternate observation approaches be developed that allow those input problems to be solved by the expert?

3. **Analysis**: Can traces of an expert's behaviour be analyzed to determine if errors were observed or if the expert's behaviour is not purely reactive?

4. **Learning**: Can learning by observation be improved so that experts that maintain internal states (beliefs, goals and/or intentions), which can not be observed by a learning agent, can be successfully learnt from?

## 1.3 Contributions

The key contributions of this work are:

- A definition of the major steps that are performed in learning by observation and how these steps can be represented as a cyclical process (Chapter 3). These steps are modelling, observation, preprocessing and deployment.

- An approach to designing learning by observation agents that allows the same agent, with no changes, to learn a variety of behaviours from a variety of experts and, with minimal changes, to be deployed in new domains (Section 4.1).

- An extension of our agent design that allows sensors and effectors to be dynamically added to a robot (Section 4.1.3). Instead of requiring the agent to be modified every time hardware is changed, the agent can dynamically change the input and output model it uses.

- Two novel methods to acquire cases by observation (Section 4.2). One approach, mixed-initiative case acquisition, allows the agent to request help from the expert to solve difficult problems as the problems occur. The other approach,

active case acquisition, logs difficult problems and presents the logged problems to the expert in bulk at a later time.

- An approach to analyzing an expert's behaviour that allows detection of state-based or non-deterministic behaviour by the expert and that can be used to identify and remove observation errors (Section 4.3).

- An algorithm for case retrieval that allows a learning by observation agent to learn from state-based experts (Section 4.4).

- Conclusions that, in the domains tested, the following results were observed (Chapter 4):

  - *Modelling*: Using the learning by observation design framework, a single agent was able to learn behaviours in a variety of domains with no changes to how it reasons and minimal changes to how it interacts with the environment. This was demonstrated in case studies involving physical robots, simulated soccer and Tetris. Using our extension that allows hardware to dynamically register with the agent, the agent did not need to be modified at all when the hardware of the robot it controlled was changed.

  - *Observation*: Cases that would rarely, or in some situations never, be observed using passive observation can be observed using mixed-initiative or active case acquisition. This was empirically evaluated in the domain of Tetris.

  - *Preprocessing*: The analysis method was able to correctly identify if state-based or non-deterministic behaviour was exhibited by the expert. Additionally, a majority of errors were able to be detected and removed. This resulted in higher-quality cases and guided the selection of retrieval algorithms for the agent to use, which lead to improved learning performance.

This was empirically evaluated in a simulated obstacle avoidance domain.

  - *Deployment*: Using a retrieval algorithm that takes into account an expert's previous sensory inputs and actions, not just the current sensory input, allows an agent to learn from state-based experts. Like with preprocessing, this was empirically evaluated in a simulated obstacle avoidance domain (Section 4.4). An empirical evaluation was also performed to examine the use of mixed-initiative case acquisition and preprocessing when learning from a state-based expert (Section 4.5).
  - *Combination of Subtasks*: The contributed techniques can benefit the learning agent when used alone or in combination. This was demonstrated by examining the influence of combining techniques on the performance of a learning agent. An empirical evaluation was performed in a simulated soccer domain and a case study was performed in a physical robotics domain (Section 4.6).

- For each developed technique, a discussion of any encountered limitations and a characterization of when the approach should be utilized.

The techniques described for modelling, observation, preprocessing and deployment have been shown, experimentally, to simplify the process of developing learning by observation agents, increase their learning performance and allow them to learn more complex behaviours. These techniques have been examined in a physical obstacle avoidance robot, a robotic arm, simulated soccer, Tetris and a simulated obstacle avoidance domain (Table 1.1).

While these contributions advance the state of the art in learning by observation they do not completely solve the problem. Each contribution has certain limitations that were identified and overcoming those limitations will need to be the focus of future work (outside the scope of this thesis). Several of the key limitations include:

| | Physical Robots | RoboCup | Tetris | Simulated Robots |
|---|---|---|---|---|
| **Modelling** | ✓ | ✓ | ✓ | ✓ |
| **Observation** | ✓ | ✓ | ✓ | ✓ |
| **Preprocessing** | ✓ | ✓ | | ✓ |
| **Deployment** | ✓ | ✓ | | ✓ |

Table 1.1: The domains each technique were tested in

- **Modelling**: The modelling approach removes much of the need for domain experts but does not completely remove the need. There are still some things, like initially programming the sensors and effectors with their information, that can not be performed by the agent itself.

- **Observation**: The observation techniques allow the learning agent to improve learning performance but requires the expert to be available at a later time to solve additional problems. If the expert will not be available or is unwilling to assist the agent then the case acquisition approaches can not be used.

- **Preprocessing**: The preprocessing technique requires the expert to be present and solve additional problems. It would be preferable if this was not necessary and the agent could analyze an expert's behaviour with a single trace.

- **Deployment**: An algorithm was presented for learning state-based behaviours but it requires inferring the expert's state each time case retrieval is performed. A more efficient approach would involve storing the current state in order to guide subsequent retrievals.

## 1.4 Organization

Following the introduction, Chapter 2 provides an overview of the current state of research in learning by observation and discusses the issues that are specific to learning from a software agent or human. In Chapter 3, learning by observation is formalized. Additionally, the chapter presents the major steps involved in learning by observation and how those steps are connected.

The major contributions of this thesis are presented in Chapter 4. These contributions cover all four steps in learning by observation and include a general-purpose framework for modelling the inputs and outputs of learning by observation agents, alternate approaches for case acquisition, techniques for cleaning and analyzing user traces and a method for learning from state-based experts. An evaluation for each of these contributions is also presented in Chapter 4. The modelling technique is examined in two physical robotics domains, a simulated soccer domain and Tetris. The case acquisition techniques are compared to passive observation in the domain of Tetris. Both trace analysis and state-based learning are evaluated in the simulated obstacle avoidance domain. Simulated soccer and an obstacle avoidance robot are used to demonstrate the benefit of combining the various subtasks and show the performance benefit of each subtask. A summary of the work performed and concluding remarks are discussed in Chapter 5.

# Chapter 2

# State of the Art

This chapter will examine existing research related to learning by observation. Initially, we will look at how agents are defined in existing research and how learning from agents differs from traditional machine learning. We will then turn our attention to existing learning by observation systems. Specific attention will be placed on how those systems learn and what they are able to learn. While common learning by observation tasks are often performed in other machine learning and artificial intelligence fields, we will defer these discussions to subsequent chapters in order to place them in the proper context.

## 2.1 Learning from Agents

In this section, we will explore how a learning by observation system can learn from agents. We will discuss how agents are defined and then examine how machine learning can be used to learn from agents.

### 2.1.1 Definition of Agents

A learning by observation system will learn from either a human agent or a software agent. We can think of an agent, either human or software, an an entity that is

*situated* in an *environment* and autonomously acts in order to achieve its objectives [58]. The environment, which may change over time, can be modelled as a finite set $\mathcal{E}$ of instantaneous states:

$$\mathcal{E} = \{e, e', \dots\}$$

The agent *senses* the environment, using its sensors, as a single sensory input from the set of possible sensory inputs $\mathcal{S}$:

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

However, the way the agent senses the environment may be imperfect, due to the complexity of the environment or errors, so the set of sensory inputs will likely be different than the set of environment states. This requires the agent to map each environment state to a corresponding sensory input:

$$sense : \mathcal{E} \to \mathcal{S}$$

In response, the agent performs actions from a set of possible actions $\mathcal{A}$:

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

These actions, which are performed so the agent can work toward achieving its objectives, can be influenced by two primary factors: the current sensory input and the agent's internal state. The sensory input, as we have seen, is external and obtained from the environment. The state, however, is internal to the agent and will likely not be directly accessible to an outside entity. This internal information could include things like the agent's *beliefs*, *desires* or *intentions* [58]. The beliefs are knowledge that the agent has of the environment. These can be updated with each new perception of the environment but may also include beliefs related to parts of the environment that

are no longer visible to the agent. For example, if the agent moves to a different room in a house it may still have beliefs related to the contents of the previous room. The desires are a set of beliefs that the agent wants to achieve. This can be thought of as the goals the agent wants to achieve or the tasks it wants to complete. The intentions are a subset of the agent's desires that it has currently committed to achieving. While some of the desires of the agent may be long-term, the intentions can be thought of as the short-term goals and tasks.

### 2.1.2 Machine Learning

We have examined the agents that a learning by observation system will learn from and will now turn our attention to what is required of the system in order to learn. Mitchell [37] has formally defined machine learning as:

> *"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."*

This definition of traditional machine learning presents several issues when learning from an agent. The primary issue is related to the task, or class of tasks, that will be learnt. Recall from the definition of an agent that the task and goals of the agent are represented by its desires and intentions. If a learning by observation system is task-independent and general-purpose, it will not have any prior knowledge about the agent's desires or intentions nor will it be able to directly observe them since they are internal to the agent. Similarly, without knowledge about the class of tasks it will not be possible to measure the performance on those tasks. Instead, any evaluation will need to be general and not related to performing any particular behaviour.

Another issue is related to the experiences that a learning by observation system will use to learn. The learning system can use what it can observe, the agent's sensory

inputs and actions, as experiences but that may not provide the same information that the agent uses to reason. This is because an agent uses its beliefs, representing what it knows about the environment, and its intentions, related to its current goals or tasks, to reason. The beliefs can, to some extent, be directly observed by examining the sensory inputs the agent receives. However, the agent's beliefs can also contain information related to aspects of the environment it can no longer directly observe so in order to have complete knowledge about the agent's beliefs it would be necessary to know how it internally models the world and how that model is updated over time.

We have discussed the ways in which traditional machine learning is different from learning by observation but that does not mean that machine learning algorithms can not be used by general-purpose learning by observation systems. We can think of a machine learning tool like WEKA [25] as an attempt at general-purpose learning. In WEKA, a variety of machine learning algorithms are available that can, for the most part, be used interchangeably on datasets (as long as the data is stored in a known format). This allows different algorithms to be tested and compared to each other. While these algorithms do not perform all necessary functions of a general-purpose learning system, they could be used by a system to perform some of the learning tasks like preprocessing, classification or evaluation.

MOSCA [9] is a general abstract learning framework devised specifically for learning from agents, and involves interactions between five agents: Apprentice, Oracle, Client, Probe and Master. In this model, the Apprentice is the learner and the others assist or evaluate the learning. The Oracle provides the Apprentice, upon request, with the correct solution to a problem. For testing purposes, the Probe can request the solution to a problem from the Apprentice, based on what it has learnt, and the Master can provide feedback to the Apprentice about its performance. The Client also requests solutions from the Apprentice but does so to actually get the solution, not to test the Apprentice.

It is possible to see how each role of MOSCA would be occupied in learning by observation. The learning agent would take the Apprentice role, since it is doing the learning, and the Probe role, since it is responsible for evaluating its own performance. The environment would take the Client role since it is providing the learning agent with novel problems to solve. The final two roles, Master and Oracle, would be the responsibility of the expert. However, this makes the assumptions that the expert provides correct solutions during observation and is able to provide direct feedback to the learning agent. In practice, these assumptions might not hold true.

An area of artificial intelligence that has similar goals to learning by observation is general game playing [20]. The goal of general game playing is to design an agent that is able to play any type of game. This requires the agent to use algorithms that are not biased to any particular game or genre of games. At runtime, the agent receives a formal description of the game it will play and uses that description to attempt to learn a strategy. These descriptions can include information about possible game states, the goals of the game, start and end conditions, rules, and supporting concepts. These formal game descriptions are authored by domain experts and can often be quite complex. Also, the ability of the agent to learn is dependant on the quality of these descriptions. If they are incomplete or contain errors then the agent might not be able to learn the correct strategies.

Plan recognition involves observing an agent and attempting to identify what behaviour the agent is performing. This is similar to learning by observation but the key difference is that the behaviour is identified rather than learnt. Also, plan recognition systems typically have knowledge about what goals the agent may attempt to achieve and a library of plans that the agent may use to achieve those goals [5]. Neither the goals nor a plan library are available to a learning by observation agent. The lack of knowledge about goals also differentiates learning by observation from reinforcement learning. Reinforcement learning [30] requires knowledge about

the goals in order to reward behaviour that helps achieve those goals and penalize behaviour that does not.

Hidden Markov models (HMMs) can be used to represent systems where there are hidden, unobservable states. This is related to learning by observation since learning by observation systems can only learn using data that is directly observable from the expert and environment. However, HMMs require prior knowledge of the states, the number of states and how those states relate to each other [53]. For a general-purpose learning system, this knowledge would not be available in advance.

### 2.1.3 Lessons Learnt

The exploration of issues related to learning from agents has identified the following key ideas:

- In addition to information related to the state of the environment, an agent may reason with internal information like its beliefs, desires and intentions. This information will not be directly visible to an observer.

- A system that can learn from a variety of agents regardless of their behaviour or environment will not be able to explicitly encode information related to the agent, task or environment since those can change.

- Traditional machine learning is defined as having a task, or class of tasks, to learn. In general-purpose learning by observation those tasks will not be known in advance. This can restrict how the learner evaluates its performance since it does not have an explicit representation of what it is learning.

- Traditional machine learning algorithms can still be used in general-purpose learning by observation systems. However, they may need to be supplemented

in some way in order to handle the conditions that are specific to learning from an agent.

- The MOSCA learning framework is designed for learning from agents in a tutoring environment. However, it is possible to map the various entities in MOSCA to the entities in learning by observation. The main difference is that there is not a Master who can provide critiques or feedback on performed behaviour. The closest the expert can come to providing feedback is to provide additional observations for the system to learn from.

- General game playing is another type of general learning approach. The primary difference is that a general game playing system is provided with a formal description of the game whereas a learning by observation system does not have any such domain knowledge. However, unlike learning by observation systems that get to observe an expert perform a behaviour, a general game playing system must learn a strategy on its own.

## 2.2 Learning by Observation

Learning by observation is often referred to in the literature by names including *imitation learning*, *learning by demonstration*, *programming by demonstration*, and others. We can think of learning by observation as any technique where an agent learns by observing an expert perform a behaviour. In this section, several existing areas of learning by observation research will be examined.

### 2.2.1 General-purpose Learning by Observation

Ontañón, Montaña and Gonzalez provide a framework for characterizing the different types of learning by observation and the properties of each type [40]. They identify

four levels of increasing difficulty: *strict imitation*, *reactive skills*, *tactical behaviour in known environments*, and *tactical behaviour in unknown environments*. Strict imitation does not require any knowledge of the environments, since it only performs a fixed sequence of actions, and is therefore not able to generalize its knowledge in any way. Reactive learning by observation requires interaction with the environment in order to map inputs to outputs and would cover most existing learning by observation systems, whereas tactical learning by observation is able to learn from experts who reason with internal information. Their most difficult category, tactical behaviour in unknown environments, involves an agent who does not have predetermined knowledge about where it will be deployed. A general-purpose learning by observation agent, like the type being developed in this thesis, would fall under that category.

Learning by observation systems tend to be domain-dependant but there have been attempts to create general-purpose learning by observation systems. The Darmok system has been used to generate plans by watching an expert play a real-time strategy game [39]. Each observation that is collected contains the current environment state, the goal of the expert and a plan to achieve that goal. The learning agent can reproduce the expert's behaviour by retrieving observations that have similar environment states and goals, and executing the associated plans. This work has been extended to work in multiple domains [22], including interactive dramas [35], however it still requires a definition of the goals and subgoals the expert will attempt to achieve. This also requires the learning agent to be aware of its own goals while reproducing the expert's behaviour. This approach to learning by observation has been found to work well for behaviours that require planning but has been less successful for reactive behaviours [41] because the generated plans tend to be more complex than necessary. The authors have also examined the influence of observation acquisition on learning by observation performance [38]. All of the examined acquisition approaches were passive in nature and focused on the structure of observations and what information

is contained in them.

Rubin and Watson have used learning by observation to train a poker playing agent [48]. The learning agent observes games involving other software agents and is often able to outperform the agents it learns from [49]. While their initial work was not general-purpose, they have subsequently examined how observations can be generalized so that a case base collected by watching one variant of poker can be used to play another variant of poker [50]. This allows their approach to move toward general-purpose learning by transferring knowledge between highly related domains.

### 2.2.2 Lead-through Programming

Lead-through programming (also called walk-through programming) [10] is similar to learning by observation in that it allows robots to be trained using expert demonstrations. It is used primarily when the robot is required to move in a fixed path of motion. However, the learner in lead-through programming only memorizes a sequence of actions and then performs the identical sequence of actions during deployment. The agent is not situated in its environment because it only performs actions and does not receive any external inputs. The fact that the agent does not reason using sensory inputs makes it impossible to determine if the actions are having the desired influence or to recover from errors.

In shipbuilding, lead-through programming has been used to show a robot how to weld [2]. The human welder controls the robot and performs the welding while the robot tracks the precise path that its welding arm is being moved. When the human is no longer controlling, the robot imitates the behaviour by moving its welding arm in exactly the same way as when the human controlled. The robot does not pay attention to any sensory inputs, like if the welding was successful, so its behaviour is only a fixed sequence of actions.

### 2.2.3 Learning Interface Agents

Learning interface agents [31] are a form of learning by observation. An agent acts as a user's personal assistant, for tasks like sorting e-mail messages, and learns by observing the user. The agent stores instance data, containing information about the email (the sensory input) and where the user moved it (the action), and then compares new e-mail messages to these instances in order to determine where to recommend they should be moved. This work is meant to be assistive to a user rather than replace a user entirely (as is the case in many learning by observation systems). Many of the features that are used are related to how the user initially interacts with a new e-mail, like reading it or replying to it, so the agent would be unable to operate without the user's involvement.

A similar work, related to scheduling and meeting management, has examined the ability of the interface agent to take automated actions [27]. They provide an interface design that attempts to reason about the goals of the user to determine if an action should be performed automatically or if further user input is needed. Since the actions of an interface agent might have significant impact on the user, like if a meeting was scheduled or cancelled, it should be certain it is taking actions that meet the user's goals. If it is not certain, it can reduce the uncertainty by providing the user with possible solutions or getting more information from the user.

In both of these systems, the interface agents are designed to assist the user in a specific task (email sorting and meeting scheduling) so the task is known in advance. What the systems learn is a user's behaviour and preferences when performing the tasks.

Learning interface agents have also been used to predict how a user will behave when note-taking [52]. The interface agent learns by watching the user enter textual notes into a computer program. It can then assist the user by providing suggestions

for how to complete notes based on what has already been typed. In this work, the agent is not necessarily observing the environment in the same way as the expert. The user likely senses the environment by encountering the situation it is taking notes about, like a school lecture, whereas the interface agent only senses the text the user types. The examples the system uses to train itself will be significantly different from the beliefs of the user so the system could never replace the user.

### 2.2.4 Learning by Observation using Case-based Reasoning

Case-based reasoning (CBR) is a problem solving approach that uses knowledge about previous problem solving experiences to solve novel problems [1]. It is based on the idea that similar problems tend to have similar solutions. This idea fits well with the goals of learning by observation where a learner should produce the same outputs (the *solutions*) as the expert in response to the same inputs (the *problems*).

Unlike many artificial intelligence approaches, CBR does not rely solely on general knowledge about how problems should be solved or generalize previous problem solving experiences but instead stores concrete examples of past problems and how they were solved. For a general-purpose learning by observation system, it is beneficial to not have to rely on domain knowledge or require input problems to take a predefined format since this type of information might not be known in advance. Instead, knowledge can be learnt from the concrete problem examples. Also, since CBR does not generalize the problem solving examples, it allows new examples to easily be added at any time.

We define the following case-based reasoning terms:

**Case**: An instance of a previously encountered problem solving situation. A case will generally contain the *problem*, the *solution* to the problem, and possibly information about how the solution was derived.

**Case base**: A collection of cases that is used by a case-based reasoning system.

Case-based reasoning systems generally follow a cycle [1]. The CBR cycle is composed of four primary stages (Figure 2.1): *retrieve*, *reuse*, *revise* and *retain*.

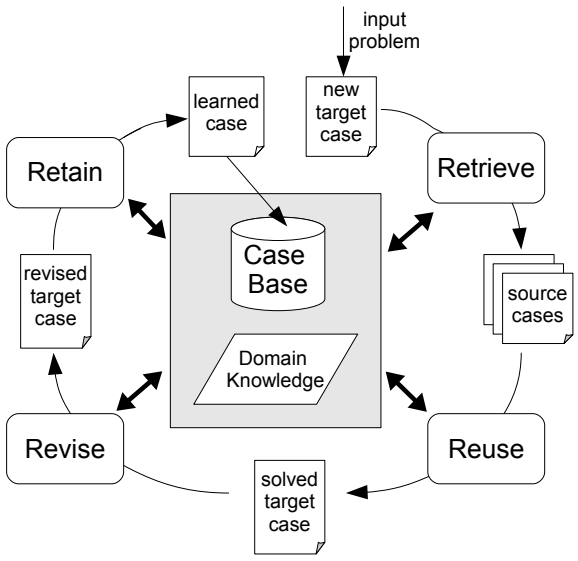



Figure 2.1: The case-based reasoning cycle

1. **Retrieve**: Prior to the retrieval stage, the input problem that is to be solved is used to create a new case called the *target case*. The target case is initially an incomplete case since it contains a problem but no solution. During retrieval, the target case is compared to the cases in the case base (the way in which cases are compared will depend on the domain and how the data in cases is represented). The most similar cases, called the *source cases*, are returned by the retrieval stage. The number of source cases returned could be a constant or vary each time retrieval is performed, depending on the specific retrieval approach that is used.

2. **Reuse**: This stage uses the solutions of the source cases to find a solution for the target case. This could involve directly reusing one of the solutions, modifying

one of the solutions to fit the input problem, or combining parts from several solutions to create a novel solution. This solution can then be added to the target case and used to attempt to solve the problem.

3. **Revise**: If the CBR system is able to get direct feedback on the success or failure of a solution, it can then revise the solution if necessary. A successful solution would not require any revision but an unsuccessful, or partially successful, solution may need to be modified. This modified solution, whether it is a slight modification or a drastic change, can then replace the existing solution of the target case. If possible, the CBR system can then attempt to apply the modified solution to the problem and see if it is now successful (possibly leading to further revision).

4. **Retain**: If the solution in the target case was successfully able to solve the input problem, the target case can then be retained by adding it to the case base (in some systems, it may even be possible to store target cases with unsuccessful solutions as negative cases). This allows the case base to grow over time and cover a wider subspace of possible problems. Retained cases can then be used, just like any other cases in the case base, to help solve input problems.

While most CBR systems use these four stages, the CBR cycle only provides a rough framework for system design. The method by which each stage is performed can vary greatly and some systems may even omit certain stages.

The discussion of case-based reasoning is important because many learning by observation systems use it to perform learning. This includes existing learning by observation systems (discussed in Section 2.2.4.2) as well as our own research.

#### 2.2.4.1 Case-based Reasoning Frameworks

There have been several efforts at creating frameworks for case-based reasoning systems. The two CBR frameworks that are currently being the most actively developed are jCOLIBRI [11] and myCBR [55].

jCOLIBRI is a general-purpose, feature-rich CBR framework that allows the development of a wide variety of stand-alone CBR applications. The framework was developed with an emphasis on separating the algorithms used in the CBR systems from the domain models. Unlike jCOLIBRI, myCBR focuses on similarity based retrieval and does not place as much emphasis on other parts of the case-based reasoning cycle. It is particularly suited for prototyping purposes as it allows for retrieval algorithms to be easily created, tested and compared.

Both myCBR and jCOLIBRI look to provide general frameworks that can be applied in a variety of CBR areas such as textual CBR, recommender systems or conversational CBR systems. We can think of them as providing building blocks that allow existing case-based reasoning components to be interconnected. A designer can use an application programming interface or graphical user interface to build and test the systems. However, due to this general nature they are not optimized for the specific needs of learning by observation systems. These needs can include things like partial observability, non-determinism, complex environmental inputs and real-time constraints. For example, since a learning by observation system might not have predefined knowledge about the sensory inputs it will receive, it would not be possible to define the format of the input data or the similarity functions in advance. Similarly, their frameworks are designed to be used on desktop computers or as web-based applications whereas a learning by observation system is often deployed in an embedded system.

#### 2.2.4.2 Learning by Observation Systems

Case-based reasoning has been a popular technique for learning by observation. We have previously discussed two case-based learning by observation systems in the context of general-purpose learning by observation, the Darmok system [22, 35, 38, 39, 41] and the work of Rubin and Watson [48, 49, 50], but will now turn our attention to ad hoc systems. The earliest work looked at learning to play chess by observation [14]. In their approach, cases are created by examining logs of grandmaster games and finding common substructures of the chess board. Each move in the logged games is used to create a case, composed of the chess board (represented by the substructures it contains) and the resulting move. They found this approach, since it could be automated, significantly reduced the time required to build a case base.

Romdhane and Lamontagne [47] have used a combination of case-based reasoning and reinforcement learning to train a Tetris playing agent. The agent observes an expert play Tetris in order to create a case base where each case contains the current state of the game and the resulting action. As the agent plays Tetris on its own, it uses reinforcement learning to identify which cases result in good game play and which do not. They also examined which of these cases can be removed from the case base when storage or computational resources are limited [46]. The main limitation of this approach is that it requires a way to determine if a particular case helped the agent play well. If no such measurement is possible, then reinforcement of cases can not be performed.

In the previously mentioned CBR systems, the current environment state is used to determine a single action that will be performed. In order to account for the fact that a single environment state might result in one of several actions, Gillespie et al. [21] use stochastic policies to represent, for a particular environment state, the probability for each possible action. The expert demonstrations are analyzed in

order to create these policies and store them in cases. During runtime, when presented with an environment state the agent can retrieve a similar case and use the associated stochastic policy to select an action. Since this approach uses stochastic policies to account for the same environment state having different possible actions, it does not take into account that these actions may be dependant on internal state information.

### 2.2.5 Other Learning by Observation Approaches

Learning by observation has not been limited to robots that only perform sequences of actions. One such work was used to teach a robotic helicopter how to perform acrobatic aerial manoeuvres [7]. The primary contribution of this work is that it learns, using a Kalman smoother, manoeuvres from multiple demonstrations rather than a single demonstration. The performance was improved because errors that might have been present in a single demonstration were unlikely to occur in all demonstrations. However, programmed expert knowledge was required, along with the expert demonstrations, to successfully learn most manoeuvres. For example, restrictions on the central position of the helicopter were required in order to learn how to flip the helicopter. While this added knowledge makes the manoeuvres learnable, it requires knowledge engineering and fine-tuning from a domain expert.

The previously described techniques all learn by observing the expert or examining game logs. One approach that attempts to improve on how observation occurs has been performed in the domain of soccer playing robotic dogs [23]. This work uses mixed-initiative control so that a robot can either be controlled by a software agent or a human expert. A human expert is able to observe the software agent control the robot and seize control of the robot when it determines the agent has made a mistake. The agent can then record the expert's behaviour and use those observations to further train itself. This type of control is beneficial because it allows the expert to fine-tune the agent's behaviour. However, identifying problems and determining when

further observation is necessary is entirely controlled by the expert. It would be more advantageous if the learning agent could examine its own behaviour and determine its limitations. In their experiments, they found that simple behaviours like tracking the soccer ball and moving could be successfully learnt but difficulties occurred when sequences of actions were necessary. For example, walking toward the ball and then kicking it. In order to overcome this they found it necessary to artificially add an input that represented the agent's current internal state [24].

Learning by observation has been used to control a software agent in a first-person shooter video game [57]. Game logs, containing input-action pairs, are collected by observing a hard-coded agent play the game and then used to train a neural network. After training, the neural network is used by the software agent to control its behaviour. This technique only uses an expertly selected subset of the sensory inputs (related to the agent's position and the nearest opponent's position) and only controls the movement of the agent. This selection of features was performed to overcome the limited observability of the agent and ensure that there were always known values for each sensory input.

Learning by observation has been used to create realistic virtual characters [12]. This work focuses on creating characters that appear to be human-like in their movements. When an expert demonstrates human behaviour, only the sequence of actions is recorded. These action sequences are then used to produce agent behaviour that is both novel and realistic. While the agents learn by observation, they require fitness functions to measure the quality of generated behaviours. This requires pre-programmed knowledge about what a correct behaviour should look like so the fitness function will need to be designed by an expert for each behaviour.

### 2.2.6 Lessons Learnt

After examining the work in learning by observation, the following insights were identified:

- Most existing learning by observation systems are designed to learn in a specific domain. If there are any changes in the domain, or in some situations even a change in the expert or behaviour, the agents need to be significantly modified. This requirement is counter to learning by observation since it does not allow an agent to learn on its own. Instead, the agent is heavily reliant on a programmer to make modifications.

- The one learning by observation approach that does work in multiple unrelated domains still requires pre-programmed expert knowledge since the goals of the expert must be defined. Every time the domain, expert or behaviour is changed the goals will also need to be re-programmed.

- The ability to transfer knowledge that is learnt by observation requires domains that are highly similar to each other. While it is possible to transfer cases between variants of the same game, it might not be applicable to our goal of learning in substantially different environments.

- All existing learning by observation work performs the observation passively. This can include directly observing the expert perform the behaviour or examining behaviour logs. This means that the agents have no control over what they observe so they may be missing important data. For example, if an expert never demonstrates a specific part of its behaviour the learning agent will never be able to observe and learn it. One approach allows the expert to take control and provide further demonstrations but this is still passive observation since the agent is not in control of what is shown.

- Learning by observation approaches that do not take into account the environment, like lead-through learning, can only perform fixed sequences of actions. Since they do not examine the success of their actions or look for any unexpected environment states, they are not able to respond to changes or generalize their behaviour.

- Expert observation analysis would be important for a learner to perform and could include identifying and cleaning noise, and determining properties related to the expert's behaviour. Since the observed data is what an agent uses to learn, it is important that it is of a high quality.

- The existing learning by observation approaches are not able to learn behaviours that are dependant on an expert's internal state. The internal state is not directly observable to an agent, since a learning by observation agent can only examine an expert's inputs and actions, so it must be inferred from observations. To some extent, approaches that use planning can perform multi-state behaviours if the state transitions are encoded in the plans. However, even if a single plan contains behaviours related to two (or more) internal states, the current state is not explicitly identified or stored so when the agent switches plans there is no guarantee the current state of the agent will be the same as the required starting state of the plan. Since many complex experts, both humans and software agents, maintain internal states, a robust learning by observation system should be able to learn from multi-state experts.

- Based on the levels of difficulty, proposed by Ontañón, Montaña and Gonzalez, a general-purpose learning by observation agent would be at the most difficult level (tactical behaviour in unknown environments). The type of system we look to develop in this thesis would be at that level because it would need to learn from experts who reason with internal information while interacting with

their environment and have no predefined knowledge of their environment.

- Case-based reasoning would be a good artificial intelligence technique to use for a general-purpose learning by observation system because it does not generalize the training examples but stores them as concrete problem-solution instances. This allows new training examples to be added at any time without needing to retrain the system. Also, it does not constrain the examples to a fixed predefined format, like a feature vector, and allows different examples to have different structures.

- There are two major frameworks for developing case-based reasoning systems: jCOLIBRI and myCBR. While both of these frameworks are designed to allow the creation of a variety of CBR applications, they are not designed to handle many of the needs of learning by observation systems. Similarly, the resulting applications are not designed to be deployed on embedded systems, or other systems with limited computational resources, so they may not meet the needs of learning by observation systems.

## 2.3 Discussion

This chapter has examined related work in learning by observation and the issues specific to learning from agents. Using the insights that were gained in this chapter it is possible to identify the commonalities between the existing learning by observation systems. More specifically, we can see what common tasks are performed in learning by observation systems and how they learn. The remainder of this thesis will formalize the workflow of learning by observation systems and address the technological requirements necessary to make them general-purpose in nature.

# Chapter 3

# Learning by Observation Cycle

Learning by observation agents attempt to reproduce the behaviour of an expert. The expert is treated like a black box that receives inputs and in turn produces outputs (Figure 3.1). The goal of the learning agent is to approximate how the expert generates outputs in response to inputs.

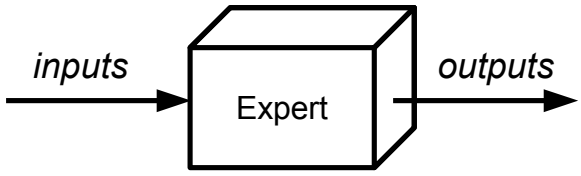



Figure 3.1: An expert receiving inputs and producing outputs

For an expert that is *situated* in an environment, the inputs represent the sensory information perceived by the expert and the outputs are the actions the expert performs. At time $t$, the expert can interact with the environment by receiving sensory information $S_t$ and performing an action $A_t$ (Figure 3.2). These expert-environment interactions can be observed by a learning agent and used to determine how the expert reasons.

Learning by observation agents do not train themselves with existing data but instead collect and label the data themselves. After seeing the expert presented with a problem, the sensory input, the learning agent infers what the solution was based

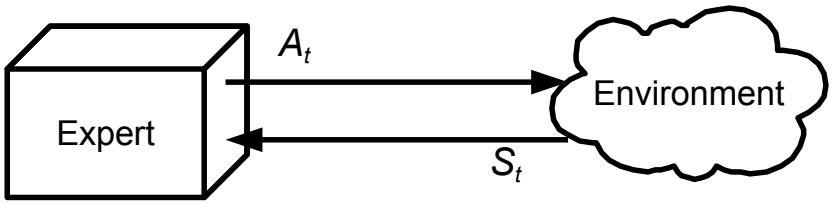



Figure 3.2: An expert interacting with the environment

on the actions the expert performed. Once the data has been collected, traditional machine learning algorithms can then be used.

This chapter will examine the major steps involved in learning by observation and how these steps fit together in a cyclical workflow. Initially, we will define how the agent stores observations and how individual observations relate to each other. A general learning by observation cycle will be presented that defines the steps that are necessary for learning. Each of these steps will be examined to determine how they should be performed.

## 3.1 Terminology

Each of the observed interactions between the expert and the environment, which are composed of the sensory inputs and actions, can be stored by the learning agent. Since the agent uses case-based reasoning, these interactions will be stored as *cases*. A case $C_t$, observed at time $t$, contains the sensory input $S_t$ and the action $A_t$ that were observed:

$$C_t = \langle S_t, A_t \rangle$$

Using case-based reasoning terminology, the sensory input is the problem and the action is the solution to that problem.

Over a period of time, the learning agent is able to view an entire *run* [58] of

sensory inputs and actions. A run of length $u$, $R_u$, would contain $u$ sensory inputs and $u-1$ actions:

$$R_u : S_1 \overset{A_1}{\rightarrow} S_2 \overset{A_2}{\rightarrow} S_3 \overset{A_3}{\rightarrow} \ldots S_{u-1} \overset{A_{u-1}}{\longrightarrow} S_u$$

Each run represents an entire sequence of the expert's behaviour that was observed by the learning agent.

## 3.2 Learning Cycle

Learning by observation has been used in numerous works but is generally looked at as a single task to be performed. However, upon closer inspection, it can be seen that learning by observation is actually composed of several subtasks. These subtasks are interconnected, as we will show below, but not necessarily tightly coupled. This means that each subtask of learning by observation can potentially be solved independently and later combined to make a complete system. This section will identify what the subtasks are, how they are connected, and which are crucial to making general-purpose learning agents.

### 3.2.1 Tasks of the Cycle

The learning by observation cycle (Figure 3.3) contains four main tasks: *modelling*, *observation*, *preprocessing* and *deployment*.

#### 3.2.1.1 Modelling

The *modelling* task in learning by observation involves two steps: modelling inputs and modelling outputs. Ideally, the way the observing agent models inputs and outputs should be identical to how they are represented by the expert. This ensures

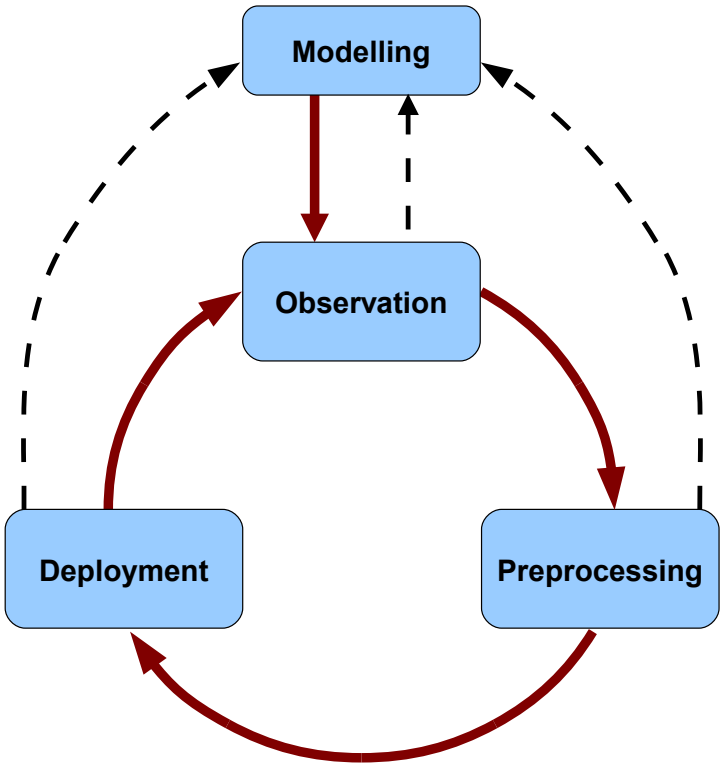



Figure 3.3: The learning by observation cycle

that the information the agent uses to learn the expert's behaviour is the same as the information the expert uses to reason. The modelling would involve defining the format for the sensory inputs received by the expert ($S_t$) and for the actions performed by the expert ($A_t$).

If the input and output models used by the learning agent are not the same as the inputs and outputs used by the expert, it may not be possible to successfully learn the expert's behaviour. For example, if the expert reasons about its actions based on both what it can see and what it can hear, the learning agent should include both of those pieces of sensory information in its input model. Failing to include either of those items would leave out important information from the model. However, if the model included extra sensory information, like what the expert can smell, this would not have a significant negative impact on the agent's ability to learn since the extra

information could be removed later (during the *preprocessing* task).

The modelling does not only involve what is contained in the inputs and outputs but also how they are structured. This can involve modelling how the expert is thought to observe its environment (a complete view of the environment or only a limited area), how it represents objects it can view (are all objects unique or can some be interchanged with each other), how objects are structured (are they atomic or can they be composed of other objects), or how it represents missing sensory information (such as objects outside its field of vision).

Let us consider an example of an agent learning to play the game of soccer. The expert senses its environment visually and can see five types of objects on the field: soccer balls, teammates, opposing players, boundary flags and goal nets. The expert can also perform three actions: moving, turning and kicking. Therefore, the modelling task should model the expert's sensory inputs as a collection of visible objects, with each object being of one of the five types, and the expert's actions as being one of the three possible action types. The more difficult task is modelling how the visual objects are represented. For example, should the expert be assumed to have a complete view of the field at all times or a limited view? This will influence whether the sensory inputs will contain information about all objects on the field (if the expert can view the entire field) or only a subset of objects (if the expert has a limited field of vision). Also, it may be necessary to model if the expert views different objects as interchangeable. For example, does the expert treat all opponents as equal or behave differently depending on which particular opponent it is? The correct answer for these modelling decisions may vary depending on the expert being observed so these decisions can influence an agent's ability to learn from a specific expert.

The modelling task is the first learning by observation task that is performed since the developed model is used by all other tasks. Once the modelling stage is

complete, the learning by observation agent can then proceed to the *observation* task (Figure 3.3). In general, the modelling task will only be performed once. However, if any of the other tasks determines that the input or output models are incorrect they can return to the modelling task. Such a transition back to the modelling task can occur as a result of an error in the initial modelling or due to changes in how the expert interacts with the environment (the introduction of new sensory information or ability of the expert to perform new actions).

#### 3.2.1.2 Observation

The *observation* task involves watching the expert interacting with the environment. Over a period of time, the observing agent is able to examine the run of the expert and record observations related to the expert's behaviour. Going back to our soccer example, during each interaction the learning agent would record observations containing what the expert could currently see on the soccer field (the collection of visible objects) and the action the expert performed (move, turn or kick).

The format of the observed items, the actions and sensory inputs, are defined in the modelling task stage. Therefore, the recorded observations are heavily influenced by the modelling. Likewise, since the data that was collected is used in the remainder of the cycle (*preprocessing* and *deployment* tasks), it has a significant influence on the remaining tasks. If the agent did not observe the expert for a long enough period of time or did not observe the expert's inputs and outputs correctly (or accurately due to incorrect modelling), it may not have enough information to correctly learn the expert's behaviour. The observation task can, however, be repeated later if the agent needs to perform additional observation or to restart the observation from the beginning.

#### 3.2.1.3 Preprocessing

The observations that were recorded during the *observation* task are raw in nature. The goal of the *preprocessing* task is to examine the raw observations in order to extract any important information or to convert the observations in some way. The manner in which the preprocessing is performed will be guided by one or more performance metrics. These metrics could include the agent's estimated response time, estimated ability to perform the learnt behaviour, or any other desired metric.

Much like the metrics used to guide preprocessing can vary depending on the particular goals of the learning agent, so too can the preprocessing methods that are used. The preprocessing methods can include techniques to extract information from the observations, like what information the expert uses during reasoning, or to modify the collection of observations, like removing duplicate observations. The information extracted, or modifications made to the observations, during preprocessing will later be used during the *deployment* task.

In the soccer example, the sensory inputs can contain five different object types (soccer balls, teammates, opposing players, boundary lines and goal nets). However, the expert might not use information from all of those types of objects during reasoning. Let us consider an expert that only uses the soccer ball and goal nets during reasoning. Keeping the other objects in the sensory inputs may be unnecessary or even detrimental if it increases the computational requirements needed to perform reasoning. Using preprocessing, the agent can determine which parts of the sensory information are necessary (in this case, the ball and goal nets) and remove the others. This will not only reduce the storage requirements for the observations but will improve real-time performance since fewer pieces of information are examined during reasoning [15].

#### 3.2.1.4 Deployment

The final task in the cycle is *deployment*. The deployment task is where the agent will use the observations it has collected, along with any information it extracted during preprocessing, to train itself and perform the learnt behaviour. Various learning methods can be used to train the agent so the choice of how the agent learns will need to be selected, in advance, by the agent's designer (a description of the learning method used in this thesis will be provided in Chapter 4). This selection may be guided by the nature of the task being learnt or by properties of the environment the agent will operate in.

Once the agent has learnt the behaviour, it can then be deployed in the environment and attempt to perform the behaviour it has learnt from the expert. Returning again to the soccer example, the learning agent would be placed on the soccer field and attempt to play soccer like the expert did. If the learning was completely successful, the behaviour of the agent should be exactly the same as the expert's behaviour would have been. However, in practice it may not be possible to replicate the expert's behaviour perfectly. Reasons for suboptimal learning can include the expert having highly complex behaviour, the agent not observing enough of the expert's behaviour, insufficient preprocessing, errors in modelling or noise. If the agent did a good job learning the behaviour, based on some performance criteria or a qualitative analysis by a human, it can continue to perform the behaviour in the environment. However, if the learning is deemed to be poor, the agent can change the learning algorithm used or continue the learning cycle by performing additional observation (or remodelling the inputs and outputs).

### 3.2.2 Case-based Reasoning Cycle within the Learning by Observation Cycle

In the learning by observation cycle, reasoning occurs during deployment and so that is where, for our system, case-based reasoning will be performed. As was described in the previous chapter, the case-based reasoning cycle contains four stages (retrieve, reuse, revise and retain). We will examine how each of these stages will be performed in our learning by observation system and any limitations on how those stages can be performed.

#### 3.2.2.1 Retrieve

During the *retrieve* stage of the case-based reasoning cycle, input problems are compared to cases in the case base in order to retrieve similar cases (details on specific retrieval algorithms will be discussed in Section 4.4). In order to calculate how similar cases are to the input problem, each case's sensory input component (the case problem) is compared to the input problem using a similarity function. One solution would be to hard-code the similarity functions that are used, but that would require prior knowledge of the sensory inputs and how they should be compared. Instead, our approach will look to incorporate similarity functions in the modelling so that each sensory input is responsible for how it should be compared (discussed in Section 4.1).

Consider an agent that has learnt to play soccer by observation. If it can currently see a soccer ball directly in front of it (and nothing else), that will be its current input problem and it will retrieve cases from its case base that are similar. Ideally, the retrieved cases should all have a similar number of visible objects, a single soccer ball, and the objects should be at a similar location, in front of the agent.

#### 3.2.2.2 Reuse

The *reuse* phase of the case-based reasoning cycle in our framework was designed to contain no domain information. This limits the reuse algorithms to those that directly copy solutions from retrieved cases or perform knowledge-poor adaptation. For direct solution copying, this could involve using the solution, unchanged, from the most similar case in the case base or the most common solution found among similar cases.

If any adaptation was performed, in order to tailor the solution to the specific problem, it would need to be knowledge-poor adaptation since no domain knowledge would be available. Instead, adaptation rules would need to be learnt from the observed cases. Although such adaptation is possible, we have not currently investigated adaptive reuse and only use solution copying.

In the soccer example, the retrieved cases would be examined to determine what action to perform. Since the soccer ball was directly in front of the agent it is likely that the retrieved cases would have a *kick* action. The agent could then reuse that action and attempt to kick the soccer ball.

#### 3.2.2.3 Revision and Retention

The remaining two parts of the case-based reasoning cycle, *revision* and *retention*, have mostly been excluded from our framework. Both of these processes require some knowledge about the task being performed. During revision, a case-based reasoning system might evaluate the proposed solution or repair the solution. Without any knowledge of the task that the expert has demonstrated or direct feedback from the expert it would not be possible to know the quality of the proposed solution. Similarly, since the quality of the solution can not be measured, the system would not know which problem-solving episodes should be retained as new cases.

## 3.3 Discussion

The cycle presented in this chapter has described the major tasks performed by learning by observation systems. Each task in the cycle can be implemented in a variety of ways (or in some cases may not be performed at all). While the descriptions of each task make it possible to see the common approaches of existing learning by observation systems, it also makes it possible to see the requirements of each task in order to allow for general-purpose learning by observation. In Chapter 4, solutions will be presented to each of the four subtasks. Each proposed technique will look to advance the state of the art in one of the learning by observation subtasks. As a result, this will lead to an improvement in the agent's learning performance or what the agent can learn.

# Chapter 4

# Case-based Learning by Observation

The previous chapter described the learning by observation cycle and the major subtasks that make up the cycle. This chapter will examine each of the subtasks and our approaches to address them.

## 4.1 Modelling: General-purpose Modelling

Learning by observation agents attempt to shift the burden of knowledge transfer from the expert, who has the knowledge, to the agent itself, who wishes to learn that knowledge. However, the area of the learning by observation cycle that is not performed by the learning agent and requires human intervention is the modelling task. Ideally, the learning agents should be able to learn a variety of behaviours from a variety of experts in a variety of environments. However, due to the human intervention in the modelling task, a change in the behaviour, expert or environment often requires a significant amount of time before the agent can begin learning.

The remainder of this section will describe an approach to learning by observation agent design that is able to reason on a variety of inputs and outputs. Initially, we will describe how the expert's inputs and outputs are modelled. We will then describe how agents can be designed in a way that decouples the parts of the agent that perform

reasoning from those that interact with the environment. A description of how the agent can reason using the inputs and outputs will be presented. Particular attention will be given to detailing how the agent is able to use these inputs and outputs, without any prior knowledge of their exact formatting, as long as they follow the modelling framework. Finally, case studies in several domains will be presented.

### 4.1.1 Input and Output Modelling

The sensory inputs received by the expert can come in many forms. If the expert is a robot with simple sensors, the inputs would likely be in the form of numeric readings from the sensors. An expert who plays a board game would likely receive inputs regarding the current configuration of the board whereas an expert with object recognition capabilities might get inputs in the form of a set of visible objects. In order to account for the variability in what constitutes an expert's sensory inputs, the inputs can be viewed as coming in two forms: *atomic inputs* and *complex inputs*. An atomic input is used to model simple *feature* values. Each complex input is composed of a collection of other inputs, either atomic inputs or other complex inputs:

```
        <input> ::= <atomic-input> | <complex-input>
 <atomic-input> ::= <name> <feature>
      <feature> ::= <value>
<complex-input> ::= <name> { <input> }
```

We take a similar approach when modelling *actions*. Actions can be atomic if they represent a single action or complex if they represent a sequence of actions. Each atomic action contains a collection of action features that represent the parameters of the action. This allows the modelling of very specific actions that have no parameters, like moving forward, or more general parameterized actions, like moving with a given direction and velocity.

```
        <action> ::= <atomic-action> | <complex-action>
 <atomic-action> ::= <name> { <feature> }
<complex-action> ::= <name> { <action> }
```

It should be noted that nowhere in our models is there any information about what the meaning of each input or action is. For example, no background information is provided to tell the observer that the touch sensor of a robot indicates the robot has come in contact with an obstacle or that a *forward* action is supposed to move the robot. More importantly, nowhere in these models, or anywhere in our design, is there a need to explicitly represent the goals of the expert or add non-observable features related to those goals. This allows the same input and action models to have task-dependent meaning and to be reused for different experts even if those experts have different behaviours or work toward different goals.

Different domains can potentially have very different sensory inputs and will therefore require different approaches to calculate similarity. Instead of encoding the specific similarity metrics in the retrieval algorithms, we make use of the *strategy design pattern* [19] (Figure 4.1). This allows each sensory input to be aware of its own strategy for comparing itself to other inputs.

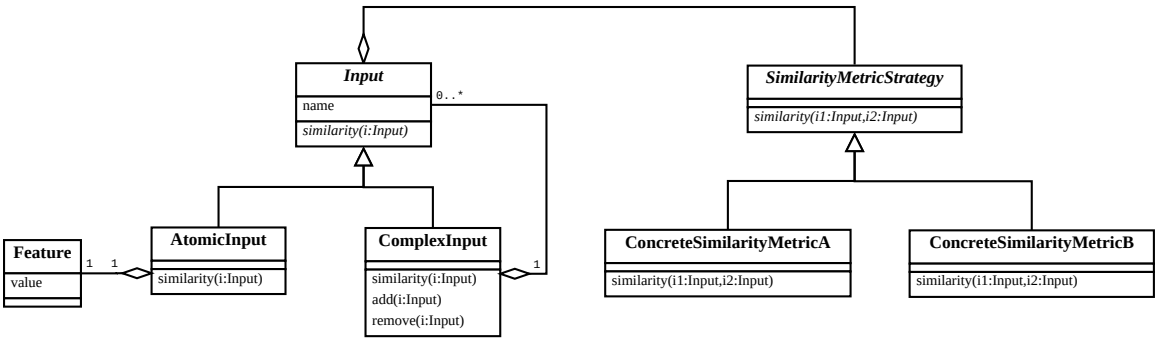



Figure 4.1: Each input has an associated similarity metric that will be used when its similarity method is called

Each type of input can have its own method of similarity calculation, but the similarity calculation strategy is *decoupled* from the input to allow different strategies to be used, or for different features to (re)use the same strategy. This allows the various retrieval algorithms to be developed independently of the input model. The retrieval algorithm can call the similarity method of the input which will then delegate the calculation to the associated similarity metric.

Actions, like inputs, have associated similarity metrics. This may seem unintuitive since actions constitute the *solution* portion of the cases (what the reasoning system attempts to predict). However, it may be necessary to compare actions if they can be part of the case *problem*, like when the expert's entire run is compared during retrieval, or when comparing actions during solution *adaptation* (during the reuse part of the CBR cycle).

### 4.1.2 Agent Design

The modelling approach that has been presented allows for a variety of different inputs and actions to be modelled, but no mention has been made of how it will be used by the learning agent. In our agent design (Figure 4.2), we look to decouple the part of the agent that performs reasoning (the *Reasoning* module) from the parts that interact with the environment (the *Perception* and *Motor Control* modules). The agent is assumed to have *sensors* and *effectors* that allow it to interact with the environment. If the agent is situated in a physical robot, the sensors and effectors could be physical sensors, like a sonar sensor, and physical effectors, like a robotic arm. In a virtual environment, those sensors and effectors would instead be methods by which the agent can send or receive information from the software controller.

The Perception module is responsible for reading the sensors and converting their readings into a form that can be understood by the Reasoning module (based on the input model that was developed). This sensory information is used by the Reasoning

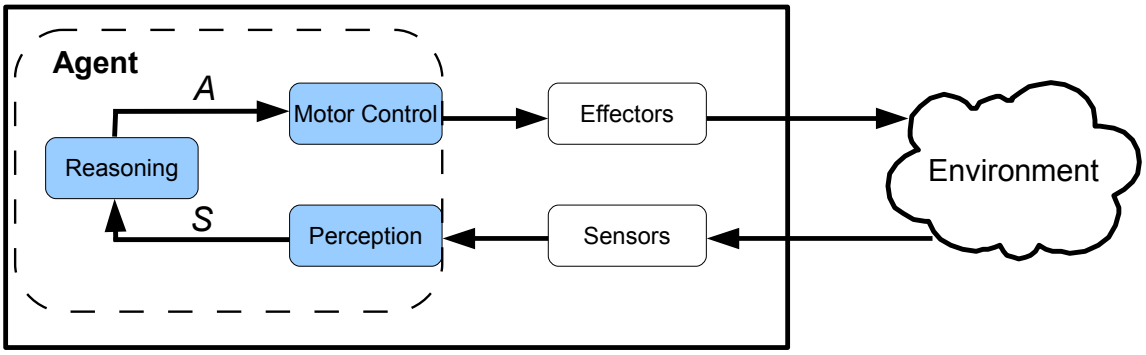



Figure 4.2: Learning by observation agent design

module to determine what action to perform. This action is then sent to the Motor Control module which causes the appropriate effector to perform the action (either physically or in a virtual way). These two modules, which could be implemented by a human expert or provided as loadable modules by each sensor/effector, will generally be quite simple since they only perform mappings. The Perception module maps from raw sensor inputs into structured sensor inputs whereas the Motor Control module maps from structured actions to raw effector actions.

The Reasoning module does not contain any prior knowledge about the set of sensors and effectors that will be used, or the input and output models that they use, so an identical Reasoning module can be used for a variety of environments. However, both the Perception and Motor Control modules must be modified if the sensors or effectors are changed. By using this design, the portion of the agent that performs reasoning is completely separate from the parts that interact with the environment. As long as the sensory information that is sent by the Perception module to the Reasoning module follows the modelling framework, the Reasoning module is able to reason with it.

### 4.1.3 Dynamic Case Representation

One method to create the Perception and Motor Control modules is for a human expert to hard-code then. However, this human intervention is not ideal for a general-purpose learning agent. Instead, we will examine an approach that allows an agent to dynamically change how it senses and acts on the environment.

#### 4.1.3.1 Changing a Robot's Sensors and Effectors

For a multi-purpose robot, using a fixed hardware configuration might not be an ideal design since the necessary sensors and effectors would be dependant on the particular task. Instead, it might be more effective, from both a design and cost standpoint, to make the robot *modular* so that hardware can be added or removed as necessary. Even if the robot could be built initially with all possible sensors and effectors, a modular design would still be beneficial since it would allow the addition of state of the art hardware or the upgrade of old hardware.

The ability to dynamically change what hardware a robot has also requires a controlling agent that is able to respond to the changes and make use of the new hardware. This idea is largely motivated by how animal brains deal with newly added sensors. It has been shown that frogs can incorporate sensory information from surgically implanted extra eyes [8] and that humans are able to adapt when existing sensors are substituted for new ones [4]. These results seem to indicate that animal brains are not hard-coded for a fixed set of sensors but can use and learn from any sensors that are wired to the brain.

#### 4.1.3.2 Sensor and Effector Registration

We propose passing the burden of handling new sensors and effectors onto the hardware itself instead of requiring a human expert to modify the agent. We extend the

agent design so that a change in the sensors or effectors no longer requires a redesign of the Perception or Motor Control modules. To achieve this, each hardware component becomes a smart-component that contains the necessary information about itself. Each component has information $\mathcal{I}_k$ that contains its name $n_k$, value type $t_k$, and similarity function $f_k$.

$$\mathcal{I}_k = \langle n_k, t_k, f_k \rangle$$

The component name is a unique identifier that distinguishes between the various components. The value type defines what kind of value a sensor produces or an effector accepts and is selected from a set of predetermined types. These value types can be simple (boolean, integer, continuous, etc.) or more complex (matrix, vector, etc.) and are used to perform simple error checking on data sent to or received from the component. The component information also includes a similarity function. This similarity function would be the similarity strategy that the input uses to compare itself to other inputs.

When a component is connected to the system it registers by providing this information to the agent. The Perception module maintains a list of all registered sensors and the Motor Control module keeps a list of registered effectors. This means that the sensory information received, or observed, by the agent is no longer composed of a constant number of predetermined sensory values. Instead, the number of sensory values received by the agent is dependant on the number of registered sensors and can therefore change over time. In Figure 4.3, there are initially no sensors registered until a sensor (*Sensor* 1) registers with the Perception module. Once the sensor registers, data from that sensor can be sent to the Reasoning module. Later, when an additional sensor registers (*Sensor* 2) the Reasoning module will get sensor data from both sensors. This requires having a case definition that is not static but can be modified as new sensors and effectors register with the system.

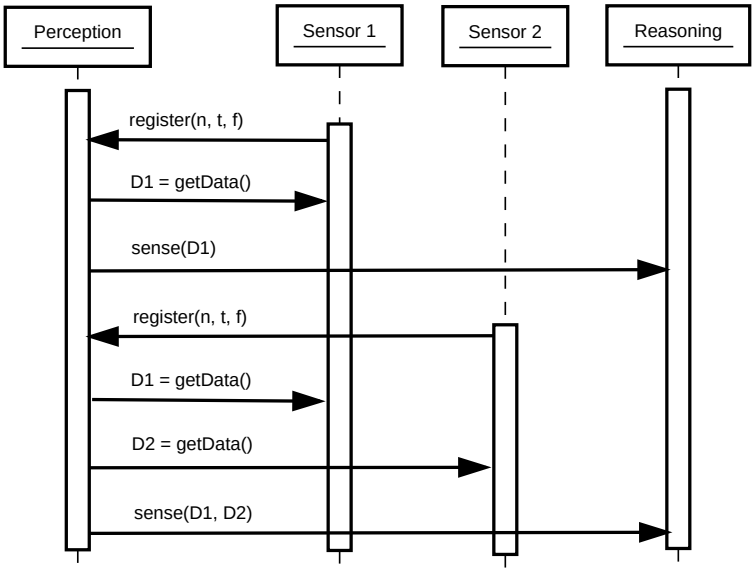



Figure 4.3: Sensors registering with the learning by observation agent

To allow for a dynamic case definition, the sensory input created by the Perception module is initially empty. As new sensors register with the agent, the sensory information will contain values from those new sensors. If there are currently $REG_t$ registered sensors at time $t$, the sensory input $S_t$ will contain data from each of those sensors:

$$S_t = \langle D_1, D_2, \ldots, D_{REG_t} \rangle$$

Where the data $D_j$ from each sensor contains both the sensor value $v_j$ and the sensor information $\mathcal{I}_j$:

$$D_j = \langle v_j, \mathcal{I}_j \rangle$$

### 4.1.4 Case Studies

Case studies in five domains will be used to demonstrate how our modelling approach can be utilized. We will demonstrate how agents can be created in each of these

domains and examine their performance when learning by observation.

#### 4.1.4.1 Sensor-Based Agents

The first two domains we will examine involve controlling physical robots. The first is an obstacle avoidance robot (Figure 4.4) that has a touch and sonar sensor. The learning task is as follows: the robot should move forward until it detects an obstacle in front of it, using the sonar sensor, or determines it has come into contact with something, using the touch sensor. In situations where an obstacle is detected, it turns either left or right (it toggles its turn direction after every turn). If it comes into contact with something it moves backwards. We will model the sensory input $S_{AVOID}$ as follows:

$$\begin{aligned} S_{AVOID} &= \langle S_{touch}, S_{sonar} \rangle \\ S_{touch} &= \langle f_{touch} \rangle \\ S_{sonar} &= \langle f_{sonar} \rangle \end{aligned}$$

The robot has a set $\mathcal{A}_{AVOID}$ of four possible atomic actions it can perform: *move forward* ($A_F$), *move backwards* ($A_B$), *move left* ($A_L$), and *move right* ($A_R$).

$$\mathcal{A}_{AVOID} = \{A_F, A_B, A_L, A_R\}$$

The following sample code shows how the sensory model for this robot can be manually implemented by a domain expert using our framework:

```
1  Input avoid = new ComplexInput("avoid");
2  Input touch = new AtomicInput("touch");
3  Input sonar = new AtomicInput("sonar");
4  SimilarityMetricStrategy s1;
5  s1 = new Mean();
6  SimilarityMetricStrategy s2;
7  s2 = new NormalizedDifference();
8  avoid.setStrategy(s1);
9  touch.setStrategy(s2);
10 sonar.setStrategy(s2);
11 avoid.add(touch);
12 avoid.add(sonar);
```

Figure 4.4: Obstacle avoidance robot

Initially, each of the sensory inputs are created (lines 1-3). Two similarity metrics are then created: *Mean* and *NormalizedDifference* (lines 4-7). As the names imply, the Mean similarity metric calculates the mean similarity of all child inputs and the NormalizedDifference similarity metric calculates the normalized difference between inputs. The Mean strategy is used by the complex input (line 8) and the NormalizedDifference strategy is used by both atomic inputs (lines 9 and 10). The complex input then adds both atomic elements as children (lines 11 and 12). It should be noted that in this example, for illustrative purposes, the similarity metrics and relationships between inputs need to be set each time the inputs are created. Instead,

subclasses of ComplexInput and AtomicInput could be created for each input type in order to encapsulate these settings.

The second robot we examine is a robotic arm (Figure 4.5). This robot has three sensors: a colour sensor, touch sensor and sound sensor. The problem-space of this domain is slightly larger than that of the obstacle avoidance robot but still relatively small (approximately 100 states for the obstacle avoidance robot and 200 for the robotic arm). It can perform five atomic actions: *move the arm forward* ($A_{armF}$), *move the arm backwards* ($A_{armB}$), *stop the arm* ($A_{armS}$), *close the claw* ($A_{clawC}$), and *stop the claw* ($A_{clawS}$). 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. We model the sensory inputs $S_{ARM}$ and action set $\mathcal{A}_{ARM}$ of the robotic arm similarly to those of the obstacle avoidance robot:

$$
\begin{aligned}
S_{ARM} &= \langle S_{colour}, S_{touch}, S_{sound} \rangle \\
S_{colour} &= \langle f_{colour} \rangle \\
S_{sound} &= \langle f_{sound} \rangle \\
\mathcal{A}_{ARM} &= \{A_{armF}, A_{armB}, A_{armS}, A_{clawC}, A_{clawS}\}
\end{aligned}
$$

There are two things that should be noted from our modelling of the sensory inputs and actions of these two robots. First, we see that although the robots were quite different there was still an opportunity to reuse a small portion of the model related to the touch sensor both robots had. Robots that make use of the same, or highly similar, sensors can therefore be modelled in similar ways. Both models could

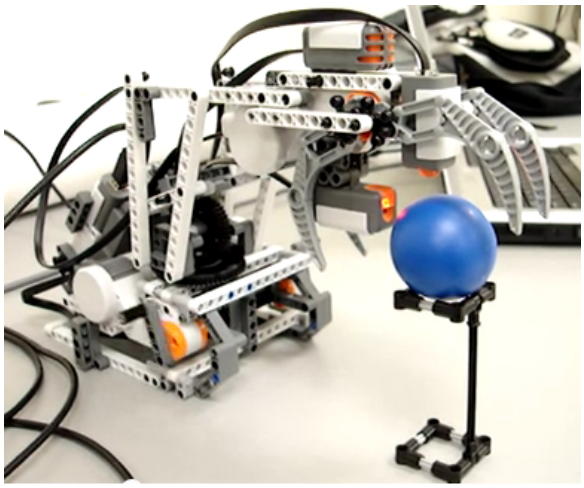


Figure 4.5: Robotic arm

use the same similarity strategy, for their complex inputs, that calculates the mean similarity of the atomic inputs. Secondly, although we described the behaviour of the obstacle avoidance robot and robotic arm, the models we created are applicable to any robots that have the same sensors and effectors. If the obstacle avoidance robot was reprogrammed to follow objects it could still be observed and learnt from using the same model.

**Evaluation**

While we have shown how the expert's inputs and actions can be modelled, we will now turn our attention to the learning agent's ability to learn from each of the experts. In these experiments, both agents will use an identical Reasoning module but will use individual Perception and Motor Control modules. During case retrieval, the Reasoning modules retrieve the source case, from the case base $CB$, that is most similar to the target case and directly reuse the solution:

$$C_{source} = \arg\max_{C_i \in CB} sim(C_i, C_{target})$$

The case similarity is calculated by calculating the similarity of the sensory input

components of the cases (either $S_{AVOID}$ or $S_{ARM}$). If you recall from the modelling example presented earlier, the model defined that these sensory inputs (which are complex) would use a similarly measure that calculated the mean similarity of all atomic inputs contained in them.

A learning agent, using our agent design, passively observed each of these robot control programs (performing the previously described behaviours) and generated 500 cases for each robot for use as a case base[1]. Additionally, each robot had 1000 extra cases observed for testing purposes. During the deployment, the learning agent used the case base to try and replicate the behaviour of the robots. Each of the testing cases, which had a known action, were given as input to the learning agent. By comparing the action selected by the learning agent to the known action of the test cases the accuracy was measured.

For the robotic arm, the learning agent achieved 100% accuracy performing each type of action (Table 4.1). For the obstacle avoidance robot, the learning agent achieved 100% accuracy for forward and backwards actions, but a lower accuracy for the left and right actions (row *Avoid (toggle turn)* in Table 4.2). This occurred because the obstacle avoidance robot would toggle its turn direction, so the turn direction was related to its internal state and not any external information that could be observed.

| | | **Arm** | | | **Claw** | |
|---|---|---|---|---|---|---|
| | **Overall** | **Forward** | **Backward** | **Stop** | **Close** | **Stop** |
| **Robotic Arm** | 100% | 100% | 100% | 100% | 100% | 100% |

Table 4.1: Accuracy of the agent when learning to control the robotic arm

[1]Throughout this thesis, different domains will use different sized case bases. The sizes were selected so that the learning agent would be able to search the case base and perform an action within the real-time limits of the particular domain. Both these limits and the computational complexity of the similarity calculations can be different for different domains.

| | **Overall** | **Forward** | **Backward** | **Left** | **Right** |
|---|---|---|---|---|---|
| **Avoid (toggle turn)** | 75% | 100% | 100% | 50% | 50% |
| **Avoid (constant turn)** | 100% | 100% | 100% | 100% | 100% |

Table 4.2: Accuracy of the agent when learning to control the obstacle avoidance robot

As a modification, the obstacle avoidance robot control program was changed to only turn in one direction. This removed the need for the control program to maintain an internal state and made it purely reactive. A new training case base of 500 cases and a testing case base of 1000 cases were generated by observing the modified robot. When the evaluation was performed[2], the learning agent was able to achieve an accuracy of 100% (row *Avoid (constant turn)* in Table 4.2). This demonstrates that, even for simple behaviours where it is possible to achieve perfect accuracy, state information can significantly hinder an agent's ability to learn if it assumes the expert will behave in a purely reactive manner. We will revisit dealing with internal states in Section 4.4.

#### 4.1.4.2 Object Inputs

We turn our attention to agents that have more complex sensory capabilities that allow them to observe the environment at a higher level of abstraction. Instead of receiving inputs in the form of sensor values, these agents are able to sense objects and their locations. In simulated soccer (Figure 4.6), like the RoboCup Simulation League [45], an agent's sensory input $S_{SOCCER}$ contains the visible *balls* ($S_{ball}$), *teammates* ($S_{team}$), *opponents* ($S_{opp}$), *goal nets* ($S_{net}$), *boundary lines* ($S_{line}$), and *flags* ($S_{flag}$). Each of these objects has sub-inputs related to their distance and direction relative

[2]A video of the learning agent performing the obstacle avoidance behaviour: `http://sce.carleton.ca/~mfloyd/LearningVideos/ObstacleAvoidanceRobotLearning.mp4`

to the agent.

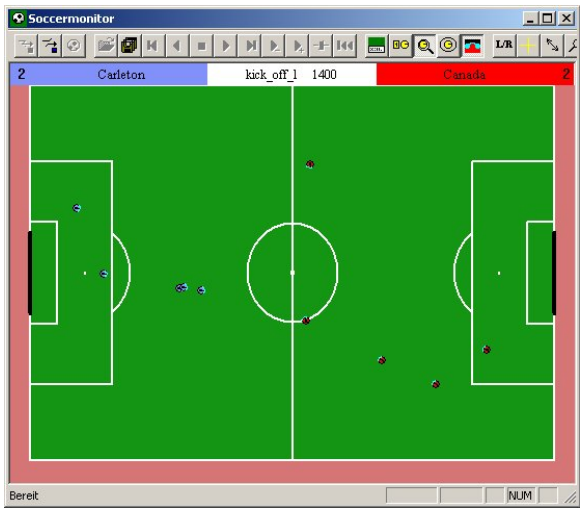



Figure 4.6: RoboCup Simulation League

$$
\begin{aligned}
S_{SOCCER} &= \langle S_{ball}, S_{team}, S_{opp}, S_{net}, S_{line}, S_{flag} \rangle \\
S_{ball} &= \{S^1_{object}, \ldots, S^a_{object}\} \\
S_{team} &= \{S^1_{object}, \ldots, S^b_{object}\} \\
S_{opp} &= \{S^1_{object}, \ldots, S^c_{object}\} \\
S_{net} &= \{S^1_{object}, \ldots, S^d_{object}\} \\
S_{line} &= \{S^1_{object}, \ldots, S^e_{object}\} \\
S_{flag} &= \{S^1_{object}, \ldots, S^f_{object}\} \\
S_{object} &= \langle S_{distance}, S_{direction} \rangle
\end{aligned}
$$

Since objects can move in or out of the player's field of vision, the number of objects of each type can change over time. Therefore each object type is viewed not as a single-valued feature but instead as a multi-valued feature. For example, if the player is facing away from the field of play it might not see any opponents ($|S_{opp}| = 0$). Later, when it turns back to the field it may see a number of opponents ($|S_{opp}| > 0$).

If the player can not uniquely identify the opponents, then when comparing two sensory inputs any sub-input in the first (any individual opponent) could potentially be compared to any sub-input in the second. The difficulty becomes determining which sub-input in the second sensory input would be optimum (or near optimum) to compare to. This requires the use of a similarity strategy that can handle multi-valued features, such as the one described in [18] that uses bipartite set-matching algorithms.

A soccer playing agent has a set $\mathcal{A}_{SOCCER}$ of three possible atomic actions: *kick* ($A_{kick}$), *dash* ($A_{dash}$) and *turn* ($A_{turn}$).

$$\begin{aligned} \mathcal{A}_{SOCCER} &= \{A_{kick}, A_{dash}, A_{turn}\} \\ A_{kick} &= \langle f_{power}, f_{direction} \rangle \\ A_{dash} &= \langle f_{velocity} \rangle \\ A_{turn} &= \langle f_{angle} \rangle \end{aligned}$$

Unlike in the physical robotic domains, where the actions did not have parameters, the soccer actions all have associated parameter features. The *kick* action shows an example of an action that can have multiple parameters since it has both a kick power ($f_{power}$) and direction ($f_{direction}$).

The observing agent learnt from a computer controlled expert that turns until it can see the soccer ball, runs toward the ball and kicks the ball toward the opponent's goal net. A case base of 5000 cases was used along with 3000 test cases. The accuracy results, shown in Table 4.3, may seem low but other work has shown these results can be drastically improved by preprocessing the case base [15]. Since the goal of this case study was to highlight the modelling[3] and not to optimize learning performance, no preprocessing was performed. Also, this domain had a significantly larger problem-space than the two robotics domains we examined previously (approximately $2^{9600}$

[3]A video of the learning agent performing the soccer behaviour: `http://sce.carleton.ca/~mfloyd/LearningVideos/RoboCupLearning.mp4`

states if the environment is discretized) and the distribution of actions is severely imbalanced (fewer than 1% of observed cases have kick actions). We will revisit this domain in Section 4.6 and show how the other subtasks (observation, preprocessing and deployment) can improve these result.

| | **Overall** | **Kick** | **Dash** | **Turn** |
|---|---|---|---|---|
| **RoboCup** | 61% | 27% | 71% | 83% |

Table 4.3: Accuracy of the agent when learning the RoboCup simulated soccer behaviour

#### 4.1.4.3 Tetris

Our next case study will involve the game of Tetris. In Tetris (Figure 4.7), there is a rectangular game region where the player stacks incoming game pieces of varying shapes. When the pieces form a horizontal line from one side of the game region to the other, the entire line is removed from the game region thereby freeing space. The player must strategically place pieces in order to ensure the stacked pieces do not reach the top of the game region.

The sensory input in Tetris $S_{TETRIS}$ contains two sub-inputs: the current state of the game region ($S_{region}$) and the current piece that needs to be placed ($S_{piece}$).

$$S_{TETRIS} = \langle S_{region}, S_{piece} \rangle$$

Each of these sub-inputs is represented by a matrix containing information about which squares are currently occupied. The game region is a $20 \times 10$ rectangle that contains 200 squares and the piece is a $4 \times 4$ rectangle that is composed of 16 squares. Each square $S_{cell}$ contains a feature ($f_{occupied}$) representing if the square is occupied

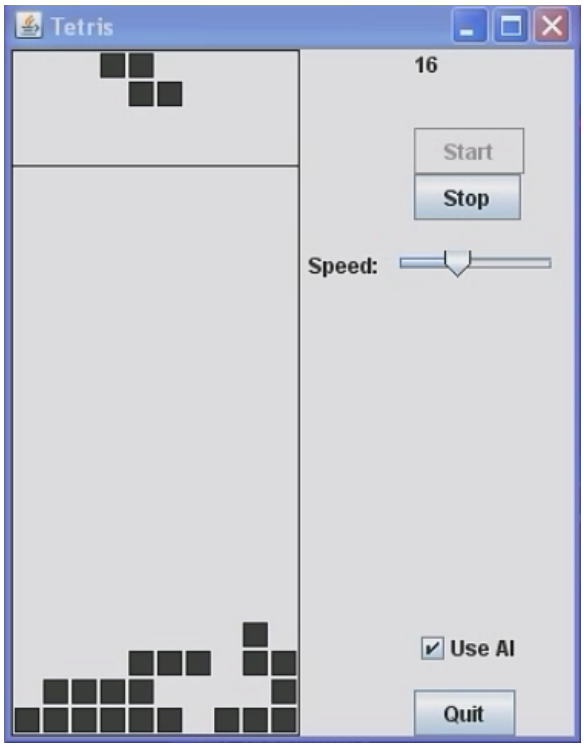



Figure 4.7: Tetris

or not.

$$
\begin{aligned}
S_{region} &= \langle S_{cell,1}, \ldots, S_{cell,200} \rangle \\
S_{piece} &= \langle S_{cell,1}, \ldots, S_{cell,16} \rangle \\
S_{cell} &= \langle f_{occupied} \rangle
\end{aligned}
$$

Unlike the soccer domain, where there was partial observability, the Tetris agent is always able to see the entire game region and game piece. Due to the full observability of these inputs it was possible to model them as being ordered and of a fixed-length (length 200 and 16 respectively) rather than as multi-valued inputs with a time-varying number of elements. This is important because similarity calculations become more computationally expensive with multi-valued inputs [18] and should therefore be avoided, if possible, when the agent has real-time constraints. Here, the similarity strategy of the complex inputs just calculates the mean of the similarities of the

atomic inputs (like was done in the robot case study). The designer could also create a custom similarity strategy that takes inspiration from human players [46].

The similarity strategy for the complex input calculates the mean similarity of sub-inputs:

$$sim(S^a_{TETRIS}, S^b_{TETRIS}) = mean\left(sim(S^a_{region}, S^b_{region}) + sim(S^a_{piece}, S^b_{piece})\right)$$

Similarly, both the region sub-input and piece sub-input use a similarity metric that calculates the mean similarity of their own sub-inputs.

$$sim(S^a_{region}, S^b_{region}) = mean\left(\sum_{i=1}^{200} sim(S^a_{cell,i}, S^b_{cell,i})\right)$$

$$sim(S^a_{piece}, S^b_{piece}) = mean\left(\sum_{i=1}^{16} sim(S^a_{cell,i}, S^b_{cell,i})\right)$$

This shows that all three types of inputs can make use of identical similarity functions. The similarity strategy for individual cells just looks to see if they contain the same value:

$$sim(S^a_{cell}, S^b_{cell}) = \begin{cases} 1 & , if S^a_{cell} = S^b_{cell} \\ 0 & , if S^a_{cell} \neq S^b_{cell} \end{cases}$$

In Tetris, there is only one action to perform. However, unlike the previous domains, this action $A_{movepiece}$ is a complex action that contains two atomic actions. The first sub-action $A_{slide}$ is related to how many squares the game piece should be moved horizontally (with positive values representing sliding right and negative left) and the other $A_{rotate}$ is related to how many times the game piece should be rotated

by 90 degrees clockwise.

$$\begin{aligned} \mathcal{A}_{TETRIS} &= \{A_{movepiece}\} \\ A_{movepiece} &= \langle A_{slide}, A_{rotate} \rangle \\ A_{slide} &= \langle f_{slide} \rangle \\ A_{rotate} &= \langle f_{rotate} \rangle \end{aligned}$$

Since $A_{movepiece}$ is a complex action, both of the sub-actions will be performed. If the player did not want to perform one, or both, of the sub-actions they could set their parameters, $f_{slide}$ or $f_{rotate}$, to be zero.

In our experiments, the observing agent learnt by passively watching a Tetris player, controlled by a computer program, and generated $100,000$ cases for the case base. When playing Tetris by itself[4], the agent was able to complete an average of approximately 2 lines per game[5] (over 350 test games). While this performance is far worse than the expert, it does show that the agent is able to reproduce the behaviour of the expert (completing lines in Tetris) to some extent. Comparatively, when the agent just placed the pieces at random, it completed an average of approximately 0.1 lines per game (over 350 games). One reason for the poor performance of the agent is the large state-space of Tetris (approximately $2^{216}$ using our representation). This is compounded by the fact that the expert plays near-optimally and therefore rarely puts itself in disadvantageous positions. However, the learning agent will make mistakes which can cause the environment state to be significantly different from any it has encountered during observation. This domain shows an example of where alternative case acquisition approaches might be beneficial and we will revisit Tetris in Section 4.2.

[4]A video of the learning agent playing Tetris: `http://sce.carleton.ca/~mfloyd/LearningVideos/TetrisLearning.mp4`

[5]These results are comparable to the results obtained by Romdhane and Lamontagne [46] when using a similar case representation for a case-based learning by observation agent in Tetris. Their system completed an average of 1.8 lines per game.

### 4.1.5 Sensor Registration

The previous case studies have made use of Perception and Motor Control modules that were hard-coded by a domain expert. For this case study, we will use the extension of our modelling approach that allows hardware to register with the learning agent (as described in Section 4.1.3). Initially, the learning agent is not aware of what type of hardware it will be connected to and therefore has no registered sensors or effectors. At this point, the agent is unable to perform any observation or learning because it has no way to interact with the environment. Later, a robot (Figure 4.8) is connected to the agent. The robot that is connected is the commercially available iRobot Create [29].


Figure 4.8: The iRobot Create robot

When the robot is connected to the agent, each of the robot's sensors and effectors register with the agent. There are six sensors, all of which produce binary values, that register with the agent: an infrared detector, left and right bumper sensors (used to detect when the robot runs into something), and three wheel sensors (used to determine if the wheels have pressure against them). The two effectors that register are the drive system and the buzzer. The drive system can be controlled by sending

it one of five possible directional values: forward, reverse, left, right and stop. The buzzer can be used to make a beeping sound.

The agent observes and learns a simple obstacle tracking behaviour. A power charging station is placed in the robot's environment and produces an infrared signal. If the infrared signal can be detected, using the infrared sensor, the robot will drive forward. However, if the infrared signal can not be detected the robot turns in a clockwise direction until it can detect the signal. When the robot reaches the power station, as indicated by either of its bumper sensors, the robot will stop.

A human expert was used to demonstrate the behaviour and provided one demonstration of the described behaviour. This resulted in 14 cases being observed. After observing the expert, the agent was able to accurately reproduce the behaviour. To examine the accuracy of the learning agent, an additional 100 cases were observed. These cases were used as testing cases and each case had its sensor information given as input to the agent. The action performed by the agent was then compared to the action portion of the case to see if they matched. Our results showed the agent was able to select the proper action 100% of the time. The ability of the agent to learn this behaviour is what we would expect given the small problem space (only $2^6$ states). While the behaviour learnt in this case study was simple, it did show that the agent was able to learn without any predetermined knowledge about the task or what robotic hardware it would be controlling. Our goal was not to learn difficult tasks but instead to show the adaptive nature of our learning by observation agent. We will revisit sensor and effector registration, and show how the agent can be easily retrained, in Section 4.6.

### 4.1.6 jLOAF Framework

This section has described our approach for modelling the inputs and outputs of a case-based learning by observation agent. Additionally, a method for agent design

was provided that decouples the reasoning of the agent from the parts of the agent that interact with the environment. We have created a reference implementation[6] of the modelling and design framework described in this this section, called the *Java Learning by Observation Framework (jLOAF) [16, 17].* We have used the jLOAF implementation for the case studies described in this thesis.

In addition to the work in this thesis, jLOAF has also been used for other learning by observation research:

1. Glen Robertson and Ian Watson. Case-Based Learning by Observation: Preliminary Work. In *Proceedings of The 8th Australasian Conference on Interactive Entertainment*, 24 , ACM Press, 2012.

2. Glen Robertson. Applying Learning by Observation and Case-Based Reasoning to Improve Commercial RTS Game AI. In *Proceedings of the Doctoral Consortium at the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 31-33, AAAI Press, 2012.

3. Vivian Andreeva, Sabrina Gaudreau and Jordan Beland. Using Player Imitation to Create Non-Player Characters in a FPS Game. *4th Year Engineering Project Report*, Carleton University, 2013.

These works use jLOAF to create agents in gaming domains, but the domains are different than those examined in this thesis. The work by Robertson and Watson involves learning in real-time strategy games whereas the work of Andreeva, Gaudreau and Beland involves a first-person shooter game.

### 4.1.7 Discussion

In this section we have described an approach for modelling the inputs and outputs of a learning by observation agent and provided a design for such agents that uses

[6]Available at `http://www.nmai.ca`

case-based reasoning. We looked to provide an approach that did not require any knowledge about the behaviour or goals of the expert being observed. Instead, a learning agent only needs a definition of its input and action models along with interfaces to allow it to properly observe and interact with the environment. By separating the core reasoning algorithms from any domain specific knowledge, the same reasoning system can be utilized in a variety of environments.

Additionally, we have described an approach for creating case-based learning by observation agents that does not require reprogramming the agent when the available sensors or effectors are changed. This approach, which was motivated by how animal brains respond to changing sensors, does not use a fixed case representation but instead dynamically modifies the structure of cases as sensors and effectors are added. When a new piece of hardware is added to a robot, the hardware registers with the case-based reasoning agent and provides any necessary information about itself. This is particularly beneficial in domains where the necessary hardware configuration of the robot can not be anticipated in advance.

While we have focused on a domain independent approach that is not biased toward any specific task, there is nothing limiting an agent designer from introducing such bias in order to optimize performance. For example, the sensory input models could be designed to only contain inputs that the expert uses during reasoning or use similarity metrics that are tailored to a specific behaviour. Ideally, our approach aims to learn such optimization information during the preprocessing steps but it could also be hard-coded in order to save time or to deliver improved performance.

We presented five case studies, in both simulated and physical environments, that show how inputs and actions can be modelled using our framework. Our examples

included domains with simple sensor systems, high-level object detection, partial observability and full observability. Additionally, we described domains with parameter-free actions, multi-parameter actions and complex sequences of actions. Our experimental evaluation shows that the same agent, without changing the reasoning module, can successfully learn a variety of behaviours in several different domains. Not only do these agents perform well in experimental evaluations but they are also able to perform the learnt behaviour after learning[7].

There are several limitations of this modelling framework. If an agent is only going to be used for a single task and domain knowledge can be obtained inexpensively, using our framework may not be appropriate. The preprocessing approaches would likely not learn the domain knowledge as precisely as if a domain expert provided it. Additionally, since no information about the goals or behaviours of the expert are provided the agent can not make use of techniques like reinforcement learning since it does not know when it has done something correctly or incorrectly.

While the extension of our approach that allows hardware to register with the agent removes the need to reprogram the agent, it does not completely remove the programming requirement. Each sensor and effector needs to be programmed with information about itself so it can register with the case-based reasoning agent. If a single sensor is used in multiple robot configurations, the sensor only needs to be programmed once whereas traditional learning by observation systems would requiring modifying the agent for every configuration. Even if each sensor or effector is only used once, the effort required to program each sensor would likely be less than modifying the agent itself. An example of this would be adding a sensor and later removing it. In our design, the registration and deregistration would handle modifying the agent whereas traditional learning by observation systems would require

[7]The video "Case-Based Imitation: A Sequel" from the 2010 AAAI Artificial Intelligence Video Competition shows demonstrations of agents created using this framework: `http://www.videolectures.net/aaai2010_floyd_cbi/`

modifying the agent twice (once for addition and once for deletion). While the case study that used this approach looked exclusively at robotic domains, our techniques could also be applied to simulated environments (although repurposing a simulated agent might not have many practical applications).

The case studies allowed us to identify many possible areas for improving the performance of the learning agent. In the subsequent sections, we will revisit many of these case studies and examine how the other learning by observation subtasks can improve the agent's performance.

## 4.2 Observation: Alternate Approaches

Instead of making use of a case base where each case is manually authored by an expert, case-based reasoning systems that learn by observation automate (to varying degrees) case acquisition. Automatic case acquisition is highly desirable because it greatly reduces the cost of generating each case. However, the downside of current approaches for acquiring cases automatically is that there is no control over *what cases* will be acquired.

Using passive observation, where the agent observes the expert without directly interacting with it, the case generation process is completely dependant on the behaviour of the expert and the state of the environment. If the expert never performs certain actions or encounters certain sensory inputs it will be impossible for the learning agent to obtain cases related to those actions and states. To overcome the limitations of passive observation, two alternative observation approaches will be presented. One approach, mixed-initiative case acquisition, allows the agent to request help from the expert to solve difficult problems as the problems occur. The other approach, active case acquisition, logs difficult problems and presents the logged problems to the expert in bulk at a later time.

As we will see, while beneficial, these approaches are more invasive than purely passive approaches since they require directly interacting with the expert rather than passively observing. As such, they are designed to supplement passive observation, to obtain cases that are difficult to gather passively, rather than replace the passive approach.

### 4.2.1 Passive Case Acquisition

Learning by observation, by its nature, is a passive learning method. The observer watches an expert and attempts to learn the behaviour the expert demonstrates. Interaction occurs between the expert and the environment as the environment produces sensory inputs that are sensed by the expert and the expert performs actions that influence the environment. As shown in Figure 4.9, the observer can then view and learn from these interactions.

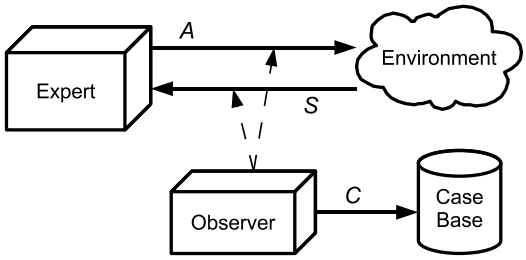



Figure 4.9: Observing an expert using passive case acquisition

There is no direct interaction between the observer and expert, and the expert may not even be aware it is being watched. Using passive observation, the observing agent has absolutely no control over what it sees (the environment controls which sensory inputs are produced[8] and the expert control the actions). This can be a problem if the expert repeatedly performs some behaviours or does not perform other behaviours. For example, consider in the domain of soccer if the expert is a very strong player and is playing against a weak player. The expert may only demonstrate

[8]The expert can also attempt to influence the environment through its actions.

offensive behaviours, since it is far superior to its opponents, and never demonstrate defensive behaviours. It may be impossible, no matter how long the expert was observed, to ever view the defensive behaviours. One solution would be to simply change the opponents the expert is competing against but it might not be possible for the agent to know, in advance, that there is a limit to what they are observing. Instead, the observing agent might not realize it is missing certain problem instances until it encounters them during deployment.

### 4.2.2 Mixed-initiative Case Acquisition

The first approach to observation we will present is designed for when the expert is willing to assist the learning agent at runtime. The expert can be thought of as a teacher or tutor who is available to provide assistance when required. The learning agent will, during the observation subtask, attempt to gauge how well it can replicate the expert's behaviour by solving problems on its own. However, if the agent feels it can not properly solve an input problem it can request that the expert solves the problem for it. Just like problems the expert solves during passive observation, problems that are solved after a request by the learning agent can also be stored as cases. Ideally, this should allow the learning agent to fine-tune its case base by only adding new cases that it was unable to solve on its own.

To achieve this type of observation, mixed-initiative control will be used [26]. Mixed-initiative systems allow for the control of a single entity, in our case a software agent or robot, by several controllers concurrently. At any time $t$, only one of the $n$ controllers has initiative over the agent and may control the actions the agent performs. In our discussion we will limit the number of controllers sharing initiative to two[9]: the case-based learning by observation system and the expert. It should be noted that the software agent (or robot) being controlled refers to a dummy agent

[9]Although the work could easily be extended to include multiple experts.

that is used to interact with the environment, not the learning agent. In order to avoid confusion, for the remained of this subsection we will refer to the dummy agent as the *agent* and the case-based learning by observation agent as the *CBR system*.

Since the motivation for using a mixed-initiative approach is case generation, under most circumstances the CBR system will control the agent. By giving the majority of control to the CBR system there will be two primary benefits. Firstly, the CBR system will handle the majority of the problem solving. This minimizes the amount of work the expert must perform since it will be passive most of the time. Secondly, and more importantly, the CBR system will be attempting to behave in a similar manner to the expert but will likely make errors that the expert would never have made. These errors in the ability of the CBR system to replicate the expert's behaviour can actually be advantageous because it allows for the exploration of previously inaccessible areas, due to the expert's optimum behaviour, of the problem space.

#### 4.2.2.1 Agent Control

Since the agent can only be controlled by either the case-based reasoning system or the expert at a particular moment in time, the mechanisms for transferring control are important. Each controller has two control actions: they can *seize* control of the agent or *cede* control to another controller. Therefore, the current controller may either *cede* or do nothing and the other may either *seize* or do nothing. The following details the situations in which the controllers will seize or cede control:

**Expert**

- **Seize:** The expert may seize control at any time (although this will likely occur rarely). Seizing would generally be performed if the expert noticed the CBR system to be performing poorly and wanted to provide unsolicited assistance.

- **Cede:** The expert will automatically cede control after controlling the agent for a single turn. This is done to give the majority of the control to the CBR system.

**Case-based Reasoning System**

- **Seize:** The CBR system will never attempt to seize control. Since the expert will automatically cede control back to the CBR system there is no need to seize control.

- **Cede:** The CBR system will cede control to the expert when it determines it is unable to successfully solve the input problem. The system could also cede if it required more information, however our implementation is currently only failure-driven.

The case-based reasoning system determines which problems it is unable to solve based on the similarity of a problem to the cases in its case base. The input problem will be compared to each case in the CBR system's case base. If none of the cases in the case base $CB$ have a similarity to the input problem $S_{target}$ greater than a threshold $\tau$, the CBR system will cede control of the agent to the expert ($\forall C_i = \langle S_i, A_i \rangle \in CB, sim(S_{target}, S_i) < \tau$). Otherwise, it will solve the problem itself. A flowchart of the process is shown in Figure 4.10. By setting an appropriate similarity threshold it is possible to control how often the CBR system will request help from the expert.

When the case-based reasoning system has ceded control (or the expert has seized control) of the agent it will then observe the expert and create a new case from the observation. We have described how the CBR system cedes control of the agent to the expert during runtime but even passive learning by observation can be thought of in a mixed-initiative context. During the initial training the system essentially sets the similarity threshold arbitrarily high ($\tau = \infty$) such that it is constantly

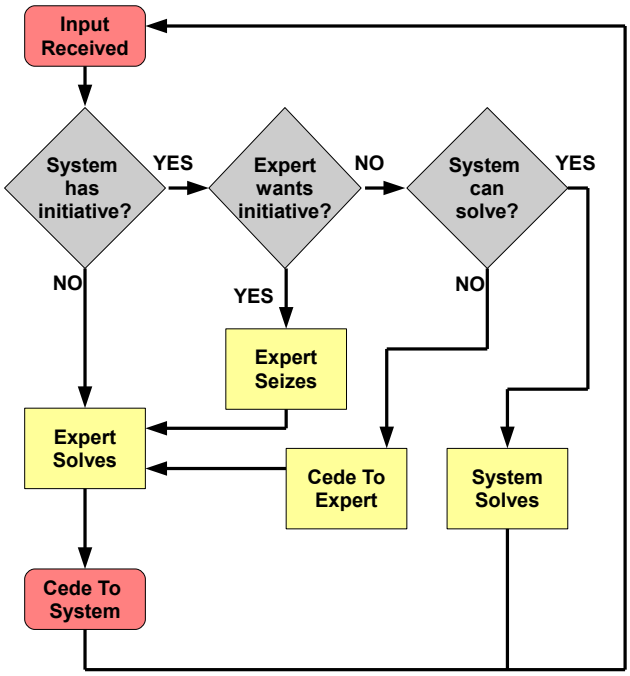



Figure 4.10: Flowchart of how initiative is seized and ceded by the expert and the CBR system

allowing the expert to solve the input problems. The CBR system has a limited case base so it is focused on letting the expert solve problems so that it can observe and obtain more cases. As the case base size grows, and the CBR system becomes better at solving problems on its own, the similarity threshold can decrease (either to a constant value or at a specific rate over time) to allow the CBR system to behave with more autonomy. Finally, when the expert is no longer available the CBR system will never ask for assistance ($\tau = 0$).

### 4.2.3 Active Case Acquisition

The primary limitation of mixed-initiative case acquisition is that it requires an expert that is available to assist the learning agent at runtime. If the expert is unavailable, or unwilling, to help at runtime then that approach can not be used. Instead, we present an approach, called active case acquisition, that does not require the expert

to solve problems as they occur but instead stores them for a later time.

Like with the mixed-initiative approach, active case acquisition requires the learning agent to be able to identify problems it is unable to solve. This allows the learning agent to identify areas in the problem space that are poorly represented in its case base and supplement those areas using further observation. We present two methods that can be used to identify potential problems to be solved using active learning. These methods are not mutually exclusive and can be used in combination or separately.

- **Runtime Identification:** This is the same approach used by mixed-initiative case acquisition. After the learning agent has learnt a number of cases, it can then use those cases to attempt to perform the behaviour of the expert. During runtime, the learning agent will receive sensory inputs from the environment and search its case base for cases with similar problem portions. If no cases in the case base are similar enough to the input problem (using the same threshold $\tau$ used for mixed-initiative acquisition), then the learning agent can log the input problem so it can be solved by the expert later using active learning. Like with mixed-initiative case acquisition, the threshold value will influence the number of input problems that are recorded for active learning, with higher threshold values resulting in more input problems being logged.

- **Secondary Case Base:** When learning by observation, different case bases can be created depending on the expert being observed. For each type of expert that is observed a separate case base is created that represents the behaviour of that expert. Two experts may perform the same task, like playing soccer, but may do so in different ways and with different levels of skill. The two experts may react differently when presented with the same sensory inputs, so it may not be appropriate to have cases from two different experts in a single case base. For example, one expert might be a defender on a soccer team and the other a

forward. The observer may only want to behave in an defensive manner. Even if these case bases can not be combined directly, it is still possible to extract information from other related case bases. Given two case bases, the learning agent's case base and a secondary case base, cases from the secondary case base can be compared to those in the agent's case base. Like with the runtime approach, any cases that have no similar cases in the agent's case base can be logged for active learning.

A third method that could be applied would be to randomly create problems. This approach, however, is limited in that there is no guarantee of the validity of these randomly created problems. In the previous two approaches, all of the problems have been encountered while observing an expert so these problems are known to be valid. There may be underlying constraints on problems, such as the acceptable values of sensory inputs, that need to be considered when creating problems. If these constraints were unknown, it would be possible to create problems that are impossible to actually encounter. For example, when observing a soccer playing expert there is a limit to the number of opponents that the expert could ever see in the environment due to the rules of soccer. If this limit was unknown to the observer, a randomly created problem could be created that contained more opponents than are allowed.

When a problem is identified for active learning using the methods described previously, it must be presented to the expert to be solved. Like with mixed-initiative case acquisition, this requires an expert that is willing to assist the learning system. It should be noted that although active case acquisition does not require the expert to be available at runtime to assist, it does require the expert to be available at some point in the future.

### 4.2.4 Automatic Case Acquisition in CBR

Active learning has been used in textual case-based reasoning domains in order to efficiently prioritize which unlabelled data items should be presented to an expert for labelling [28]. Their approach uses metrics of diversity (how dissimilar unlabelled data is from the labelled data) and density (how close data items, both labelled and unlabelled, are to each other). The primary difference between this work and our own is that our approaches do not have the complete set of unlabelled examples in advance. The mixed-initiative acquisition approach will only have a single unlabelled data item, the current problem, available whereas the active approach may have more available before they are presented to the expert but is not required to.

Active learning has also been used by case-based planning systems that reason about, and potentially change, the goals of the system [42]. Since the plan being executed is dependent on the goal of the system, it is important to know the correct goal so an acceptable plan can be selected. In situations where the system is unsure about the current goal, it can query an expert to get the correct goal. The active learning in such a system only looks to get a single piece of knowledge from the expert and does not actively learn other information (like the optimum plan to be executed).

Many case-based reasoning systems use cases that are created or collected by an expert. However, there are approaches, other than learning by observation, that allow for the automatic acquisition of cases. Asiimwe et al. [3] extract cases from reports about home upgrades that help people with disabilities more easily live in their homes. These reports are written by human experts but do not need to follow any specific formatting or content guidelines. Similarly, Yang et al. [59] extract information from aviation maintenance reports written by technicians. Data from these textual reports are combined with information from computer generated fault messages to create cases. In both approaches, cases are automatically acquired by

mining text documents. While these approaches remove the need for the expert to format their knowledge in a specific way, they still require the person to take time compiling a report. The expert is still manually providing the case data but is just not explicitly formatting it as a case.

Automatic Case Acquisition (ACE) is an approach for generating cases that has been used in the games of chess [43] and checkers [44]. When presented with a problem to solve, ACE randomly selects an action to perform and uses that problem-solution pair to create a new case. At the end of each game, cases are rewarded or penalized based on if they were part of a winning or a losing game. This approach is able to create a wide variety of cases but requires a method for evaluating how good each case is. If the success of a solution can not be immediately measured, which is generally the situation in learning by observation, then this approach can not be used. Also, since solutions are randomly generated the approach is highly exploratory and does not use any existing case knowledge to assist learning.

The previously mentioned approaches are able to automatically acquire data but they do not attempt to control what data is acquired. Metrics can be used to analyze existing data to find areas of the problem space that are poorly covered. This analysis can then be used to determine a set of novel problems that should be solved by an expert so that new cases can be collected. Two popular analysis metrics are the complexity of existing data [33] and the coverage of existing data [34]. However, these metrics require knowledge of both the size of the problem space and any restrictions on problems. For example, in the domain of Tetris these approaches would need to know which input problems can never occur (and that knowledge may not be available to the agent). Additionally, the metrics are calculated offline so it may not be feasible to determine if a problem encountered at runtime is of interest.

### 4.2.5 Evaluation

Our experiments will look to demonstrate the benefits of the two observation techniques, mixed-initiative acquisition and active acquisition. If you recall, in Section 4.1 when the case study of Tetris was performed it was found that the agent often did not have cases that were similar to its current sensory input. We will use the Tetris domain in order to evaluate if these approaches can generate cases that can not be obtained in a passive manner. The way in which the Tetris domain is modelled will be exactly as it was described previously and the same computer-controlled expert will be used.

#### 4.2.5.1 Experiment Parameters

Both algorithms used a similarity threshold of $\tau = 0.80$. This value causes the learning agent to determine it can not accurately solve the input problem if no case in its case base has a similarity above 0.80. Fifteen case bases that each contain $100,000$ cases, five for each acquisition technique (passive, active and mixed-initiative), were generated by observing a computer controlled expert. All case bases use the same set of $90,000$ initial cases, called the *seed cases*, that were generated passively. The first five case bases, which we will call the *passive case bases*, each contain an additional $10,000$ cases that were generated through passive observation (first column in Figure 4.11). The second five case bases, called the *mixed-initiative case bases*, each contain $10,000$ additional cases that were generated using our mixed-initiative approach (second column in Figure 4.11). To generate these additional cases, the agent used the seed case base during reasoning and added new cases, by having the expert solve the input problem, whenever it had no similar cases[10]. The final five case bases, the

[10]A video showing the learning agent playing Tetris and occasionally ceding to the expert when unable to retrieve a similar case: `http://sce.carleton.ca/~mfloyd/LearningVideos/MixedInitiativeTetris.mp4`

*active case bases*, each contain 10,000 cases generated through active case acquisition (third column in Figure 4.11). Like with the mixed-initiative case bases, the learning agent initially only used the cases in the seed case base. For each active case base, it continued to interact with the environment until it had logged 10,000 problems it could not solve. These 10,000 problems were then given to the expert to be solved.

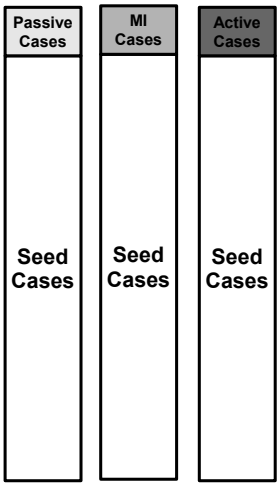



Figure 4.11: The source of cases in each type of case base

#### 4.2.5.2 Rarity of Cases

The first experiments look to examine whether mixed-initiative and active acquisition are able to generate cases that are unlikely to be obtainable through a purely passive approach. As was described previously, for each of the mixed-initiative and active case bases 10,000 cases were generated using the appropriate acquisition approach. It is possible that these cases were dissimilar to the 90,000 seed cases but would have occurred had more cases been generated passively.

To test this, a much larger case base of 2 million cases was generated passively. The 10,000 cases that were generated with either passive, active or mixed-initiative acquisition (from each of the fifteen case bases) were compared to the larger case base to find their most similar case. The results, shown in Table 4.4, show that while the

passively acquired cases frequently had identical or highly-similar cases in the larger case base, the mixed-initiative and active cases did not. None of the mixed-initiative or active cases had a case in the larger case base with a similarity of 0.85 or more, whereas 96.9% of the passive cases had a similarity at least that high. Looking at the mean similarity of each case, we can see that the passive cases are highly similar to cases in the larger case base (mean similarity of 0.94) whereas the mean similarities of the other approaches are much lower. The mean similarities for both the active and mixed-initiative cases, 0.78 and 0.79 respectively, are actually lower than the $\tau$ value of 0.80 that was used to generate the cases. It should be noted that while there appears to be more mixed-initative cases than active cases with similarities above 0.80, the majority of the cases are only slightly above that threshold which is why the mean similarities of both approaches are so close.

| | $\geq$ **0.80** | $\geq$ **0.85** | $\geq$ **0.90** | $\geq$ **0.95** | **= 1.00** | **mean** |
|---|---|---|---|---|---|---|
| **Passive** | 100% | 96.9% | 73.8% | 40.0% | 4.1% | 0.94 |
| **Active** | 22% | 0% | 0% | 0% | 0% | 0.78 |
| **MI** | 32.7% | 0% | 0% | 0% | 0% | 0.79 |

Table 4.4: Percentage of acquired cases with a maximum similarity, when compared to a larger passive case base, in various ranges

What these results demonstrate is that the vast majority of the cases generated using the mixed-initiative and active approaches would not have been obtained even if a much larger passive case base was generated. This confirms our hypothesis that there are certain input problems that would not be observed in a purely passive manner. The errors made by the CBR system lead to unexplored areas of the problem space that could then be solved by the expert.

#### 4.2.5.3 Game Performance

Having a case base with a more diverse collection of cases may be desirable but we also look to show that it is beneficial for Tetris-playing performance. In order to examine this we measured the average length, in number of game pieces played, of a Tetris game. A longer game length implies the agent was better able to manage the height of the stacked blocks by using strategic piece placement or completing lines.

The CBR system used each of the fifteen case bases to play 100 games of Tetris. In total, each type of case base (passive, active and mixed-initiative) was used to play 500 games. During these games the CBR system was not able to receive assistance from the expert and reasoned solely with the case base it was using. Table 4.5 shows the mean number of pieces played for each type of case base (and the 95% confidence interval).

| | **Passive** | **Active** | **Mixed-Initiative** |
|---|---|---|---|
| **pieces** | 25.49 ($\pm$ 0.55) | 26.68 ($\pm$ 0.54) | 27.65 ($\pm$ 0.63) |

Table 4.5: Mean pieces played using each type of case base

We can see that both the mixed-initiative and active case bases result in a significant increase (using a paired t-test with $p < 0.05$) in the number of pieces played per game compared to the passive case bases. The mixed-initiative results are also a significant increase over the active results. The increased number of pieces per game, while not a large increase, does show that using mixed-initiative or active case generation allows the case-based reasoning system to play the game of Tetris better. It should also be noted that all of the case bases contained 90% of the same information (the seed case base) so the improvement was a result of the remaining 10% of the cases. This is likely because the CBR system is better able to recover from errors it makes. Since the expert has solved problems that represent the game state after these

errors, the CBR system will possess cases that can help it during these situations. Since the mixed-initiative approach can immediately learn from the expert, instead of having the expert solve all $10,000$ problems at the end, it is able to collect cases that are different from all cases, both the seed cases and any acquired cases, rather than just the seed cases. This can make the acquired cases more diverse, compared to the actively acquired cases, which is likely why the mixed-initiative results outperformed the active ones.

#### 4.2.5.4 Case Generation Cost

One other area of interest is related to the number of problems, during case acquisition, that the case-based learning by observation agent was able to solve itself. When generating the $10,000$ mixed-initiative cases the CBR system solved a mean (and 95% confidence interval) of $108673 \pm 901$ problems without the assistance of the expert. Since it required the expert's assistance for $10,000$ problems (the portion of the case base generated using mixed-initiative control), the CBR system was able to solve approximately 91% of the problems itself. During active acquisition, the CBR system was able to solve $56414 \pm 1500$ problems by itself while generating the $10,000$ actively acquired cases (approximately 85% of the problems solved by itself).

The large number of problems the CBR system can solve itself is beneficial for two primary reasons. First, less of a burden is placed on the expert. The expert only needed to solve problems that the CBR system was unable to solve. Even if the initial case base was empty and all cases were generated using mixed-initiative or active acquisition, the expert would still only be required a portion of the time since the system would be able to solve more problems as more cases were added. Secondly, had the CBR system added the *first* $10,000$ cases it observed it likely would have added many cases that were highly similar to cases it already had in its case base.

Comparing the mixed-initiative and active approaches, we can see that the mixed-initiative approach allows the agent to solve many more problems on its own while generating the $10,000$ cases. This is because every time the agent encounters a dissimilar problem it immediately has the expert solve it and adds it to its case base. Any subsequent problems that are similar to that problem will be solvable by the agent. However, since the active acquisition approach does not learn from the expert until all $10,000$ problems have been logged there is the possibility that some of those problems are similar or identical.

#### 4.2.5.5 Mixed-initiative Learning Curves

The previous experiments identified mixed-initiative case acquisition as the top performing case acquisition strategy but no attempt was made to optimize the parameters used. In all previous experiments, a seed case base of $90,000$ cases and a similarity threshold of $\tau = 0.8$ were used. In these experiments we will examine the influence of changing these parameters on the performance of the learning agent.

Figure 4.12 shows the performance, measured by the number of pieces played per game, when a seed case base of $90,000$ cases is used with a variety of similarity thresholds (from 0.75 to 0.99). For all settings, $10,000$ additional cases were generated and used by the learning agent, along with the seed cases, to play 500 Tetris games. One important result from this figure is that, except for the ends of the curve ($\tau > 0.98$ and $\tau < 0.76$), all values were statistically significant improvements over the passive approach. This means that an ideal similarity threshold is not necessary in order for mixed-initiative case acquisition to perform well. The curve has a single optimum value that occurs at approximately $\tau = 0.87$.

A similar analysis was performed by using the optimum threshold of $\tau = 0.87$ and varying the number of seed cases used during mixed-initiative case acquisition (from 1 to $99,000$). The results, in Figure 4.13, show that most sizes of seed case bases will

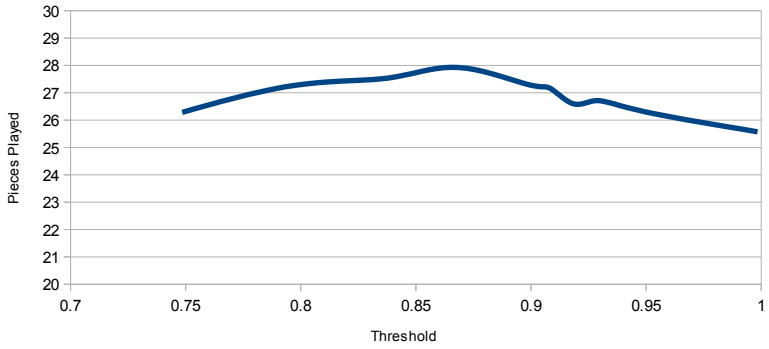



Figure 4.12: The performance of a learning agent using 90,000 seed cases and a variety of threshold values

result in improved performance over passive case acquisition. The only results that were not statistically significant improvements over the passive approach where when a very large seed case base is used (greater than $97,000$). This is to be expected since larger seed case bases result in fewer cases being discovered using mixed-initiative case acquisition. As the number of seed cases approach the maximum size of the case base, in our experiments $100,000$, the case acquisition becomes almost entirely passive.

The optimum size of the seed case base was found to be approximately $50,000$ seed cases (50% of the total case base size). This indicates that there is a noticable benefit in acquiring cases using the mixed-initiative approach. However, below $50,000$ seed cases the performance declined, so passive case acquisition is still beneficial. This is likely because it is beneficial to be able to retrieve exact solutions to some common problems, like the start of games, instead of solutions to problems that are only somewhat similar (a similarily of 0.87 in these experiments). It should be noted that while these parameter values, $\tau = 0.87$ and a seed case base of $50,000$ cases, were found to be optimal when learning from the Tetris expert, these values would likely be different in other domains.

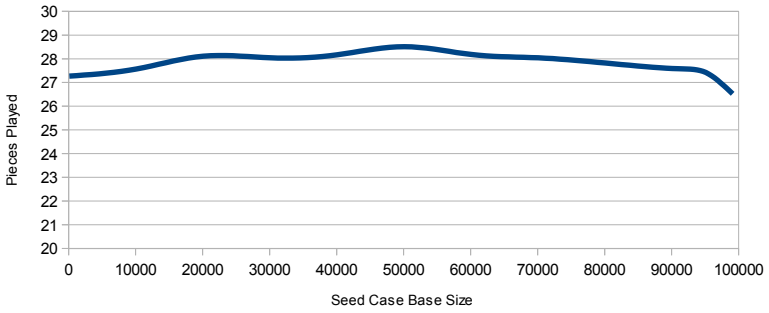



Figure 4.13: The performance of a learning agent using a threshold of 0.87 and a variety of seed case base sizes

#### 4.2.5.6 Random Case Generation

An alternative approach to passive, mixed-initiative and active case acquisition is to generate problems randomly. The limitation of random case generation is that not all randomly generated problems may actually occur in the problem domain. In a domain like Tetris, invalid board configurations can occur if the board contains a completely occupied row (since that line would be immediately removed once it was completed), an empty row with pieces above it, or pieces that are not touching any other pieces or borders (since the pieces would be shifted downward into the empty space). Without any prior knowledge of these restrictions, it is not possible to know if a given problem is valid or not.

In order to evaluate the quality of randomly generated cases, the initial seed case base of $90,000$ cases (the same seed case base used in previous experiments) was used to generate five case bases of $10,000$ randomly generated cases. Problems were generated randomly by uniformly selecting between 0 and 200 cells in the board to be set as occupied and randomly selecting a valid Tetris piece. If no cases within the seed case base had a similarity to the problem greater than 0.80 ($\tau = 0.8$), the problem was provided as an input to the expert and a case was stored. Otherwise, the problem was discarded.

The case bases that were generated in this manner had a mean of $3696 \pm 57$ cases with problems that could never be encountered in Tetris (36.96% of cases). It should be noted that these results are when a valid Tetris piece is used. If the squares representing the piece are also set randomly, in a similar way to how the board is randomly generated, then $9994 \pm 2$ cases have problems that can never be encountered (99.94% of cases). It should be noted that the domain knowledge we used to determine valid cases could also be used when generating random cases to ensure valid cases. However, a general purpose learning by observation agent would likely not have this information available for all domains it could be deployed in.

While the randomly generated cases are all different than the cases in the seed case base, many of the cases can never be encountered by the agent so they are not likely to be of any benefit. Each of the five random case bases was combined with the seed case base, for a total size of $100,000$ cases in each case base, and used by the learning agent to play 100 games of Tetris (for a total of 500 games played). The agent played a mean of $25.89 \pm 0.55$ pieces per game. This is an improvement over using cases that are only generated passively but the improvement is not statistically significant. This is likely because only 63.04% of the cases, the cases that can actually occur, are positively influencing the agent's performance.

### 4.2.6 Discussion

The alternative observation approaches, mixed-initiative and active case acquisition, were found to allow for acquisition of cases that could not be obtained through purely passive observation. This was a result of errors the learning agent would make during deployment leading to areas of the problem space the expert would never encounter due to optimum or near-optimum behaviour. Since it is not reasonable to assume a learning agent can learn every behaviour perfectly, it is important to have an observation strategy that can handle these situations.

In addition to acquiring cases that would be difficult or impossible to observe passively, using the alternative acquisition approaches resulted in increased Tetris game performance. Even though all three approaches (passive, active and mixed-initiative) used a large percentage of identical cases, the additional cases that were observed resulted in significantly improved performance for the mixed-initiative and active approaches over the passive approach. Mixed-initiative acquisition was found to be an improvement over active acquisition in both Tetris performance and how many problems the agent was able to solve itself during acquisition. These results indicate that, if possible, mixed-initiative case acquisition is preferable to active acquisition but either approach would be more beneficial than purely passive observation.

Both of the approaches presented address the limitations of passive observation. However, both approaches have limitations of their own. Both approaches require a willing expert that is available to assist the learning agent when necessary. Mixed-initiative case acquisition requires the expert to be available at runtime so that a problem can be solved by the expert when it is encountered. Active acquisition does not require the expert to always be available, since problems are logged for later, but this delays when the cases can be used by the agent.

The cases that are acquired using any of the three acquisition approaches can be used by the agent during deployment, as we have shown in our experiments, but they can also be used to examine the behaviour of the expert. In the following section we will turn our attention to preprocessing the case base, and show how the quality of the cases can be improved and important information can be extracted.

## 4.3 Preprocessing: Trace Cleaning and Analysis

The observations that are collected by a learning by observation agent are vitally important since they contain the information that the agent uses to learn. However, the

observations could be incorrectly recorded if the learning agent made an observation error or if the expert performed an error in manipulation or reasoning and produced an erroneous output. Analyzing the generated observations in order to verify and validate their quality becomes an important consideration since it can directly influence the agent's ability to learn.

In addition to analyzing the observations for errors, they can also be examined to get insight into the behaviour the expert has demonstrated. This can include information about how the expert reasons or how much information from the observations is used during reasoning. By analyzing how the expert reasons, it becomes possible to select an appropriate set of algorithms for the agent to learn and perform the expert's behaviour. This can potentially improve the performance of the learning agent or reduce the computational resources necessary for it to learn.

### 4.3.1 Expert Traces

The learning agent can generate a *trace* of the expert's behaviour during observation. An expert trace is simply a log of the entire run (which was defined in Section 3.1) of the expert that was observed by the agent over a period of time. If the agent observed the expert for time $u$ then the expert trace $T$ will contain $u$ sensory inputs and $u$ actions:

$$T : S_0 \overset{A_0}{\rightarrow} S_1 \overset{A_1}{\rightarrow} S_2 \overset{A_2}{\rightarrow} \ldots S_u \overset{A_u}{\rightarrow}$$

As can be seen from the definition, each trace alternates between sensory inputs and the resulting actions just like the runs that are used to create the traces. Previously, we have defined each input-action pair as a case. However, since the input-action pairs in a trace are part of the trace logs and not stored as cases in a case base we will refer to each pair as an *interaction* $I_t$:

$$I_t = \langle S_t, A_t \rangle$$

However, it is likely that these interactions will eventually become cases once trace analysis is complete. We can rewrite the expert's trace as a temporally linked series of observed interactions:

$$T : I_0 \rightarrow I_1 \rightarrow I_2 \rightarrow \cdots \rightarrow I_u$$

We wish to detect and analyze situations in a trace where the same sensory input ($S_1$) results in different actions ($A_1$ and $A_2$ where $A_1 \neq A_2$):

$$\exists A_1, A_2 \in \mathcal{A} \wedge \exists S_1 \in \mathcal{S} \text{ such that } \langle S_1, A_1 \rangle \in T \wedge \langle S_1, A_2 \rangle \in T$$

Initially, we will examine the different types of experts that can be learnt from and how the agent can learn. This will allow us to identify why similar sensory inputs may result in different actions and motivate how we will analyze traces.

### 4.3.2 Types of Learning

The method by which an agent learns by observation can be heavily influenced by the complexity of the expert that is being observed. This complexity relates to how the expert reasons about what actions it should perform and what information it uses during reasoning. We will introduce three types of learning by observation for experts of increasing complexity: rote, reactive and state-based.

#### 4.3.2.1 Rote Learning

The simplest method of learning by observation involves *rote learning*. A rote learning agent memorizes the sequence of actions demonstrated by the expert. An example of learning by observation using rote learning is lead-through learning [10] (which was

discussed in Chapter 2). However, a rote learning agent does not pay attention to the context in which those actions were performed. This means that such an agent will perform the same sequence of actions that were demonstrated by the expert in the exact order they were demonstrated. A rote learning agent assumes that the expert selects an action $A$ to perform without considering any outside information and simply replicates a fixed sequence of actions:

$$A_0 \dots A_t = constant$$

Even for a simple expert that does not reason using external inputs, there are still challenges that can make learning by observation difficult. The two primary difficulties are *noise* and *expert error*. If the learning agent does not observe the expert's actions perfectly and makes observation errors due to noise, the agent would be unable to replicate the expert's behaviour exactly as demonstrated. For example, consider an expert that performs a behaviour where it makes a series of movements. If the learning agent makes errors observing each of the movements, like observing how far the expert moved each time, the learning agent may end up in a different location after trying to perform the movements itself. Even if the agent can observe the expert perfectly, the expert might make an error. For example, the expert may accidentally move too far. Since the observing agent has no way of knowing this was done in error, it will assume that it was the correct action and attempt to perform it when replicating the expert's behaviour. In such a situation, even an agent that can learn perfectly will still end up learning the wrong behaviour.

This representation of an expert assumes that the expert does not reason with any external information, such as sensory information, and just performs a fixed sequence of actions. However, it is often unrealistic to make such an assumption. Many agents need sensory information to determine the current state of their environment, and therefore to decide on the next action to perform, or to verify that an action they

performed resulted in the desired outcome.

#### 4.3.2.2 Reactive Learning

Consider an example in the domain of soccer where the expert performs the following sequence of five actions: *move forward*, *move forward*, *move forward*, *move forward*, *kick ball*. If the agent uses rote learning, it will always attempt to replicate the behaviour by moving forward four times and then kicking the ball. The agent does not pay attention to the location of the ball so if the ball if not near the agent after moving forward four times the agent will be unable to successfully kick.

A method of learning by observation that takes into account external information is *reactive learning*. Reactive learning assumes that the expert selected an action to perform in response to the most recent sensory information $S_t$ it received:

$$A_t = f(S_t)$$

Unlike with rote learning, where the learning agent could simply memorize a sequence of actions, reactive learning requires approximating the function the expert uses to select an action to perform. In our system, we use case-based reasoning to approximate this function and determine what actions the expert would have performed when presented with sensory information. Looking back at the soccer example that was presented, if the agent knew what sensory information resulted in the expert kicking, like *the ball was less than 1 meter away*, it could avoid performing the kick action when the ball is not near.

The inclusion of sensory information in the reasoning process adds additional challenges for the observing agent. In addition to noisy observation of actions and expert error, which were also problems for rote learning, the observations of sensory information can also be noisy.

#### 4.3.2.3 State-based Learning

The major limitation of reactive learning is that it only considers the current sensory information and assumes the expert does as well. If the expert has multiple internal states, it may reason not only with the current sensory information but also its internal state information. For experts that reason with internal state information, the learning agent will need to use *state-based learning*. One difficulty of state-based learning is that the expert's internal state is not directly observable by the learning agent. However, internal state can be inferred by examining the complete series of past sensory information and actions the expert has encountered [58]. A state-based expert will not only reason using the current sensory information but also using state information that was determined using past sensory information and actions. We can model the action selection of the expert as a function of the current sensory information it perceives as well as all past sensory information it has received and all actions it has performed (its entire run):

$$A_t = f(S_t, A_{t-1}, S_{t-1}, A_{t-2}, S_{t-2}, \ldots, A_0, S_0)$$

As an example, consider an expert that controls a robot by navigating it toward a landmark and then, after it reaches the landmark, navigates it to another location. For this expert, the fact that the current sensory input indicated the landmark was visible would not be enough to differentiate between moving toward the landmark or toward the other location. Information contained in the run, related to if the robot had reached the landmark yet, would be needed.

Each method of learning by observation increases the amount of information it assumes the expert uses when reasoning. Both rote learning and reactive learning are actually special cases of state-based learning that only consider a subset of the information (rote learning only uses the past actions and reactive learning only uses

the most recent sensory information) and can therefore only be used to learn from a subset of possible experts. While we have described the types of learning, no mention has been made of how this learning will be performed (Section 4.4 will examine how to learn from state-based experts).

#### 4.3.2.4 Expert Action Selection

The simplest of the three experts, an expert that simply performs a fixed sequence of actions, does not use any external information when selecting an action to perform. Learning from a rote expert only involves keeping track of past actions so that the learning agent will know how far along the sequence it is.

A reactive expert selects an action to perform based on the most recent sensory input $S_t$ it received from the environment (where $S_t \in \mathcal{S}$). Such an expert will, unless it makes an error, always perform the same action for each sensory input:

$$Expert_{reactive} : \mathcal{S} \rightarrow \mathcal{A}$$

The sensory input contains all of the information that is necessary for the reactive expert to reason with and allows it to select a single action to perform for each sensory input. However, if the expert uses an internal state during reasoning then it may use any information from its run (any sensory inputs it has received or any actions it has performed) during reasoning. This can result in a single sensory input leading to a set of possible actions (where $\mathcal{P}(\mathcal{A})$ is the powerset of $\mathcal{A}$):

$$Expert_{state} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$$

This is because the sensory input, on its own, does not contain all of the information the expert uses during reasoning. The current sensory input is only the end of the run:

$$R_t : R_{t-1} \xrightarrow{A_{t-1}} S_t$$

On two runs of the expert that are each of length $t$, $R_t^a$ and $R_t^b$, the most recent sensory inputs might be identical ($S_t^a = S_t^b$) but the rest of the runs might have differences ($(A_{t-1}^a \neq A_{t-1}^b) \vee (R_{t-1}^a \neq R_{t-1}^b)$). The entire run $R_t$ (where $R_t \in \mathcal{R}$) of the expert might be necessary for a state based expert to always perform a single action (assuming the expert does not perform any errors):

$$Expert_{state} : \mathcal{R} \rightarrow \mathcal{A}$$

These three types of experts are all deterministic since they will perform the same sequence of actions when presented with the same sequence of inputs. However, an expert could also be non-deterministic in some or all of its behaviour. For example, the expert might randomly select actions to perform rather than performing any formal reasoning. Like with the state-based expert, a non-deterministic expert does not select a unique action to perform based on the current sensory input:

$$Expert_{non-deterministic} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$$

Unlike a state-based expert, an expert that displays non-deterministic behaviour will not always select a single action to perform even when presented with its entire run:

$$Expert_{non-deterministic} : \mathcal{R} \rightarrow \mathcal{P}(\mathcal{A})$$

These definitions of the different types of expert reasoning will be used to analyze runs and predict how a particular expert reasons.

### 4.3.3 Expert Trace Analysis

The discussion of the different types of experts that an agent can learn from allows us to identify three key causes of similar sensory inputs resulting in different actions:

- Reasoning with an internal state
- Non-deterministic behaviour
- Error

A fourth cause of similar sensory inputs having different actions is if the agent does not have some of the sensors that the expert has, and therefore it is missing some of the features to describe a given perceived situation. For example, the agent may not have a sound sensor but the expert reasons with sounds it can hear. Since the agent can not observe theses sounds they will never be recorded in the traces. In all of these situations, the agent is not viewing the interactions in the same way the expert is and may not get a clear indication of how the expert reasons. We will assume in what follows that the agent does have all the sensors needed to perceive the environment like the expert does.

Differentiating between the three properties might not be obvious when only examining a single trace of the expert's behaviour. For example, it would not be possible to tell with any degree of certainty that the expert performed an erroneous action. What was an expert error could also appear to be evidence of an internal state or non-deterministic behaviour. In order to help differentiate between the three properties, we will examine multiple traces that were derived from the original.

Two levels of analysis are performed: *single trace analysis* and *multi trace analysis*. Initially, we perform single trace analysis in order to determine if a trace is a candidate for multi trace analysis. Ideally, we would like to perform multi trace analysis on every user trace that is generated. That may not be possible, since, as we will see later,

multi trace analysis requires the expert to solve additional problems. If the amount of time the expert is available is limited, we would only want to make use of the expert when it is clearly necessary.

#### 4.3.3.1 Single Trace Analysis

Single trace analysis is used to identify traces that might have any of the three properties. Since, as we described earlier, each of the properties presents itself through interactions where similar sensory inputs result in different actions, we will look to measure how many of these differences occur in the trace. When looking for differences, each interaction in the trace is compared to all other interactions in the trace as well as all interactions that occur in cases that are already in the case base. Cases that are already in the case base are also used during single trace analysis because they contain knowledge collected during past observation sessions. These cases were likely collected as part of a previous user trace and then converted into cases and stored in the case base.

When comparing two interactions, $I_i$ and $I_j$, they are marked as *noteworthy* if they have sensory inputs that are sufficiently similar ($sim(S_i, S_j) > \tau$, where $\tau$ is a threshold used to determine if two sensory inputs are sufficiently similar) and they have different actions ($A_i \neq A_j$)[11]. If there are $N$ interactions in the trace and $n$ of those interactions are noteworthy, we calculate the ratio of noteworthy interactions ($\frac{n}{N}$).

#### 4.3.3.2 Multi Trace Analysis

In multi trace analysis, new traces are generated by having the expert replay the sensory inputs it received during the initial trace. If the initial trace contained $N$ interactions between the expert and the environment, the $N$ sensory inputs will be

[11] We could also use a threshold when comparing actions like we did with sensory inputs.

given to the expert, in their original order of occurrence, for the expert to reason with again. Each time the expert replays the initial trace, the entire run of the expert will be recorded as a new trace. The use of multi trace analysis makes the assumptions that the expert will be available to generate the new traces and that the agent can present the problems to the expert in a similar manner to how the expert receives inputs from the environment. These would be reasonable assumptions if the expert is willing to help the agent learn (and assumes the role of a tutor or teacher). Presenting the problems to the expert is far more feasible in simulated environments or when the expert interacts with the environment through a user interface.

The primary objective of multi trace analysis (Figure 4.14) is to identify the differences, if any, that exist between the initial trace and all replay traces. If the initial trace was used to generate $M-1$ additional traces (Generate Additional Traces in Figure 4.14), there will be $M$ total traces each containing $N$ interactions. The $i$th interaction in each of the $M$ traces will all have the same sensory input but may have different actions.

Each of the $n$ noteworthy interactions from the original trace will be grouped together with the corresponding interactions from the other generated traces, resulting in $n$ groups of interactions each containing $M$ interactions (Group Interactions in Figure 4.14). For example, if the 3rd interaction from the original trace was deemed noteworthy, it will be grouped together with the 3rd interactions from each of the other $M-1$ traces.

For each grouping of noteworthy interactions, we calculate the *agreement ratio* $AR_i$:

$$AR_i = \frac{s_i}{M}$$

Where $s_i$ is the number of interactions in the $i$th grouping that had the same action as the interaction from the initial trace. The agreement ratio can then be used to

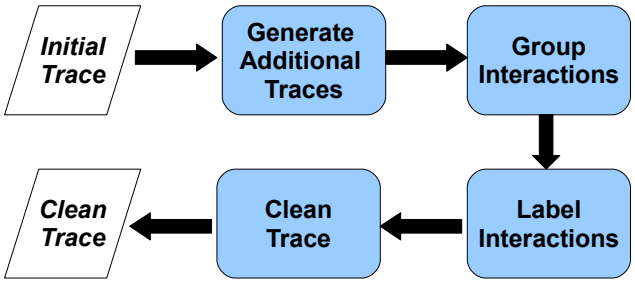



Figure 4.14: Flow chart of multi trace analysis

label (Label Interactions in Figure 4.14) each of the groupings:

$$label_i = \begin{cases} state-based & ,if \quad AR_i \geq \alpha \\ non-deterministic & ,if \quad \beta < AR_i < \alpha \\ error & ,if \quad AR_i \leq \beta \end{cases}$$

Interactions of a state-based expert are identified as noteworthy since a single sensory input can lead to multiple actions. However, during replay of those traces the expert will generally perform a single action since it is reasoning with its entire run. This will result in an agreement ratio that is greater than $\alpha$ for noteworthy interactions that are a result of internal state. If there was an interaction that only occasionally (less than a threshold $\beta$) resulted in one type of action being performed then it is labelled as being an error. This assumes that the expert generally performs the correct action in the traces and only rarely performs the erroneous action. A noteworthy interaction will be labelled as non-deterministic if it results in several different frequently occurring actions during replay (an agreement ratio between $\alpha$ and $\beta$). This assumes that since there was no clear correct action that the expert selected the action in a non-deterministic manner. One limitation of this labelling approach is when the expert selects an action in a non-deterministic way but the probability of selecting the action is low (less than $\beta$) or high (greater than $\alpha$). In these situations, the interactions will

incorrectly be labelled as an error or internal state instead of non-deterministic. This problem can be minimized by selecting an appropriate value of $\alpha$ and $\beta$ but can not be completely eliminated.

For any interactions that are labelled as having an error in the original trace, those interactions can be cleaned by replacing them with the correct versions of the interactions from one of the other traces (Clean Trace in Figure 4.14). The correct version of an interaction is selected by finding the most common action from the interactions in a grouping and using one of the interactions with that action (all interactions with the same action will be identical since they will always have the same sensory input). If the analysis was performed a second time, the number of noteworthy interactions due to error should ideally decrease while the number of noteworthy interactions due to state-based or non-deterministic behaviour should remain (although some may be removed if they were only labelled as noteworthy due to the erroneous interactions).

An example of trace analysis is given in Figure 4.15. Initially, only the original trace, which contains five interactions, is available to the learning agent (Figure 4.15a). Single trace analysis is then performed (Figure 4.15b) and four noteworthy interactions are found. The first and fifth interactions are noteworthy because they have the same sensory input ($S_a$) but different actions ($A_a$ and $A_c$). Similarly, the second and fourth interactions had the same sensory input ($S_b$) but different actions ($A_b$ and $A_a$). In order to perform multi trace analysis, the expert is made to encounter the same sequence of sensory inputs, contained in the original trace, to generate additional traces. In this example, two additional traces are generated (Figure 4.15c).

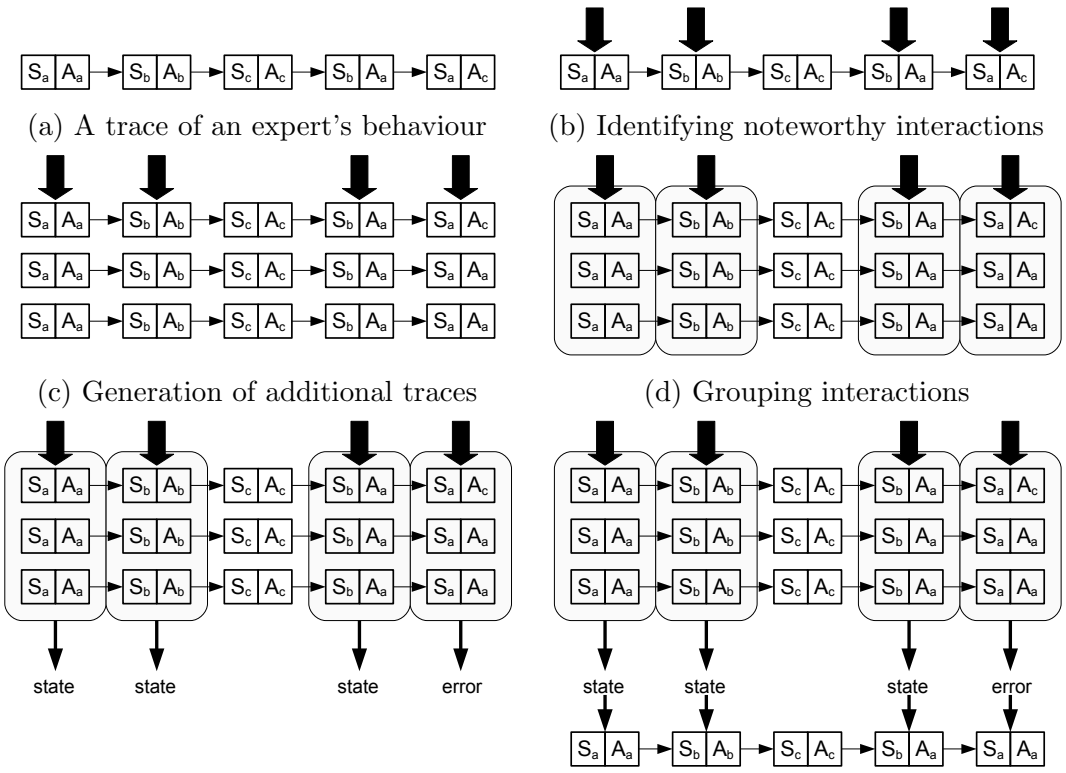
Four groupings, one for each noteworthy interaction, are then compiled (Figure 4.15d). Each grouping contains one interaction from each trace and all interactions in a group occur at the same position in their trace (and therefore have identical sensory inputs). Using the threshold approach we described previously, each grouping can be

labelled (Figure 4.15e). Since the groupings that contain the first, second and fourth interactions all have the same action ($A_a$, $A_b$ and $A_a$ respectively) they are labelled as being state-based. However, the grouping of the fifth interactions do not all have the same actions. The interaction in the original trace has one action ($A_c$) whereas the others have a different action ($A_a$) so this grouping is labelled as an error. A cleaned version of the original trace can now be generated (Figure 4.15f). Since there was only one noteworthy interaction labelled as erroneous, the fifth interaction, only one interaction is cleaned in the final trace. The most common action, from the entire grouping, is used as the correct action for that interaction ($A_a$). It should be noted that the first interaction in the original trace, which was labelled as state-based, was labelled that way because of the erroneous interaction (the fifth interaction). If single trace analysis was performed again, on the cleaned trace, it would no longer appear to be noteworthy since the error would be fixed.

After the trace analysis has been completed and any errors have been eliminated, the trace can be converted to cases and added to the case base. These cases will now be of a sufficiently high quality that they can be used by the learning agent when it attempts to perform the behaviour it learnt from the expert or they can be used to identify noteworthy interactions in newly recorded traces. The interactions that were not cleaned, those labelled as state-based or non-deterministic, provide quantifiable insight into the behaviour of the expert. If a trace was determined to contain non-deterministic behaviour or multi-state behaviour, appropriate learning algorithms could be selected so they can properly deal with these behaviours.

### 4.3.4 Evaluation

In order to evaluate the trace analysis technique that was presented, we will examine traces of a simulated obstacle avoidance robot. The simulated robot moves around a *50 unit* $\times$ *50 unit* environment which contains a number of obstacles scattered




(a) A trace of an expert's behaviour

(b) Identifying noteworthy interactions

(c) Generation of additional traces

(d) Grouping interactions

(e) Labelling each grouping

(f) Generating a clean trace

Figure 4.15: Example of trace analysis

throughout. The sensory inputs and actions of the simulated robot will be modelled similarly to the physical obstacle avoidance robot we looked at in Section 4.1. The only differences were introduced in order to allow for the creation of slight variations on the behaviour. There are 5 possible actions the simulated robot can perform: move forward ($A_F$), move backwards ($A_B$), turn left ($A_L$), turn right ($A_R$) and reverse direction (turn 180 degrees, $A_{REV}$).

$$\mathcal{A}_{robot} = \{A_F, A_B, A_L, A_R, A_{REV}\}$$

In order to sense the environment the simulated robot has three sensors: touch, sonar and sound. 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. The sound sensor converts the received audio signal into a discrete value, in our case a value from the set $\{0, 1, 2\}$, depending on what sound was heard. Each sensory input is then a triple containing the feature values from all three sensors:

$$\begin{aligned} S_{robot} &= \langle S_{touch}, S_{sonar}, S_{sound} \rangle \\ S_{touch} &= \langle f_{touch} \rangle \\ S_{sonar} &= \langle f_{sonar} \rangle \\ S_{sound} &= \langle f_{sound} \rangle \end{aligned}$$

Using the modelling of the robot's sensory inputs we can now describe the similarity metric that will be used. The similarity metric is calculated by taking the average similarity of each of the sub-input features:

$$sim(S^a_{robot}, S^b_{robot}) = mean\left(sim(f^a_{touch}, f^b_{touch}) + sim(f^a_{sonar}, f^b_{sonar}) + sim(f^a_{sound}, f^b_{sound})\right)$$

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

The robot will be controlled by a computer control program (although we will also use a human controller is Section 4.3.4.3). If the robot's touch sensor indicates it has come in contact with an obstacle, the control program will move the robot backwards. Otherwise, it 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, will be changed depending on which properties we want to be present in the trace:

- **No non-deterministic behaviour or internal state:** The control program will always turn the robot left.
- **Non-deterministic behaviour but no internal state:** The control program will turn the robot left 50% of the time and right 50% of the time.
- **Internal state but no non-deterministic behaviour:** The control program will toggle the direction it turns. For example, if it previously turned left it will always turn right next.

#### 4.3.4.1 Single-Property

To start, we will perform trace analysis on traces that only have one of the properties present. Each of the traces will be generated by observing the control program for $10,000$ interactions.

- **Base trace:** This trace will contain no errors, non-deterministic behaviour or reasoning with an internal state.

- **Error trace:** Errors will be added by incorrectly recording the action performed by the control program in 2% of the interactions. The control program will not exhibit any non-deterministic behaviour or use an internal state. A second version of this trace without the errors will also be stored. After the errorenous version of the trace has been analyzed and cleaned, it will be compared to the correct version to see how successful the cleaning was.

- **Non-deterministic trace:** The control program will perform some non-deterministic behaviour but will not use an internal state and no errors will be added. A copy of this trace will be kept to make sure that after analysis no errors have been introduced.

- **Internal state trace:** The control program will use an internal state when reasoning but no non-deterministic behaviour or errors will be added. Like with the non-deterministic trace, an identical second version of the trace will be used to ensure no errors are introduced during analysis.

For the analysis, the threshold for sensory inputs to be considered highly similar is $\tau = 0.9999$ (99.99% similarity) and labelling thresholds of $\alpha = 0.90$ and $\beta = 0.10$ were used. These thresholds were selected so that an erroneous interaction can occur in at most one other trace and a state-based interaction can have at most one other trace with a different action. In both situations, these thresholds allow for the possibility that replay traces will also contain some errors. During multi trace analysis, each initial trace was used to generate nine additional traces. For each of the four initial traces, the number of noteworthy interactions (coulmn NW in Table 4.6) that were identified, along with the percentage of noteworthy interactions that were labelled as errors (column Error), non-deterministic behaviour (column ND) and internal state (column State) were measured. Additionally, the number of errors that existed in the original traces (column Initial Errors) and final traces after cleaning (column Final

Errors) were also measured.

| | NW | Error | ND | State | Initial Errors | Final Errors |
|---|---|---|---|---|---|---|
| **Base** | 0 | 0% | 0% | 0% | 0 | 0 |
| **Error** | 835 | 19.8% | 1.2% | 79.0% | 200 | 35 |
| **Non-deterministic** | 445 | 0% | 97.8% | 2.2% | 0 | 0 |
| **State** | 458 | 0% | 0% | 100% | 0 | 0 |

Table 4.6: The analysis results and quality of each single-property trace after one round of analysis

For the trace without any of the properties present, we can see that the analysis does not identify any noteworthy interactions. The analysis of the internal state trace is able to detect numerous noteworthy interactions and successfully labels them all as a result of internal state. Similar results are displayed by the non-deterministic trace, with nearly all noteworthy interactions labelled as resulting from non-deterministic behaviour. However, several of the interactions are incorrectly labelled as resulting from internal state. This is because for these interactions, during replay, most of the generated traces had the same action randomly selected. This makes it appear as if the agent intentionally performed the same action each time even though it was purely coincidental.

The analysis of the trace with error added is less clear. We can see that some interactions are correctly labelled as resulting from error but the majority are labelled as resulting from internal state. This is because the erroneous interactions result in a number of correct interactions being identified as noteworthy. The correct interactions seem to be a result of state because the agent continues to respond the same way to them in each of the generated traces. However, the trace cleaning is able to remove a large number of the actual errors that exist in the trace (an 82.5% decrease from 200 to 35). After these errors have been removed, the correct interactions are no

longer identified as noteworthy. If three rounds of analysis are performed by further analyzing each of the cleaned traces (Table 4.7), we can see that no interactions are incorrectly labelled as resulting from non-deterministic behaviour or internal state. There are still some erroneous interactions in this trace, even after cleaning, but they are unable to be identified by the analysis because there were no similar interactions in the trace. The results show only one instance of an error being introduced by the analysis and cleaning. A single error was added to the Non-deterministic trace after three rounds of analysis. For all other experiments, the number of errors was either reduced or remained constant.

| | **NW** | **Error** | **ND** | **State** | **Initial Errors** | **Final Errors** |
|---|---|---|---|---|---|---|
| **Base** | 0 | 0% | 0% | 0% | 0 | 0 |
| **Error** | 0 | 0% | 0% | 0% | 29 | 29 |
| **Non-deterministic** | 445 | 0.2% | 98.2% | 1.6% | 0 | 1 |
| **State** | 458 | 0% | 0% | 100% | 0 | 0 |

Table 4.7: The analysis results and quality of each single-property trace after three rounds of analysis

#### 4.3.4.2 Multi-Property

The previous results evaluated the analysis technique when only one property was present in each trace and now we will examine when more than one property exists in a trace. Two additional traces were generated: one that has both errors and non-deterministic behaviour (referred to as the Error+ND trace in the results tables), and one that has both errors and internal state (Error+State in the results tables). Both of these traces will have errors inserted in 2% of the interactions and a correct, error-free version of the trace will also be stored. Like the previous evaluation, each initial trace had nine additional traces generated during multi trace analysis and the

same thresholds were used for analysis ($\tau = 0.9999$, $\alpha = 0.90$ and $\beta = 0.10$).

The results (Table 4.8) show a sizable decrease in the number of errors in the traces (80.5% decrease for Error+ND and 85.6% decrease for Error+State). This is similar to what was seen when the trace only had errors and shows that the trace cleaning still works even when other properties are also present. We also see that the traces with errors in them continue to result in correct interactions being labelled as resulting from internal states. As with the previous experiments, most of these incorrect labels disappeared after the trace was cleaned and re-analyzed (Table 4.9). After subsequent rounds of analysis and cleaning, nearly all of the noteworthy interactions were correctly labelled as resulting from non-deterministic behaviour (in the Error+ND trace) or internal state (in the Error+State trace). Several errors remained in the traces but, as with the error-only trace from the previous experiments, these erroneous interactions did not have any similar interactions in the trace so it was not possible to label them as noteworthy.

| | **NW** | **Error** | **ND** | **State** | **Initial Errors** | **Final Errors** |
|---|---|---|---|---|---|---|
| **Error+ND** | 1061 | 14.6% | 38.5% | 46.9% | 185 | 36 |
| **Error+State** | 1238 | 14.0% | 0.6% | 85.4% | 202 | 29 |

Table 4.8: The analysis results and quality of each multi-property trace after one round of analysis

| | **NW** | **Error** | **ND** | **State** | **Initial Errors** | **Final Errors** |
|---|---|---|---|---|---|---|
| **Error+ND** | 417 | 0% | 98.1% | 1.9% | 35 | 35 |
| **Error+State** | 493 | 0% | 0.2% | 99.8% | 24 | 24 |

Table 4.9: The analysis results and quality of each multi-property trace after three rounds of analysis

#### 4.3.4.3 Human Expert

The previous evaluations of the trace analysis technique used a computer program as the expert and errors were artificially inserted. We look to extend our evaluation by observing from a human expert. The human will likely perform errors by accident due to reasoning errors, fatigue, lack of precision, or pressing an incorrect button. The expert will attempt to control the robot in the same way as the control program that used an internal state and will be observed for 250 interactions. While a trace of the human expert is being generated, the control program will also select actions to perform in order to generate a correct trace of what the human should have done in each interaction.

Upon initial examination of the human trace, it is clear that the human expert made significantly more errors than any of the previous experiments. The trace had an error rate of 16.8% (42 erroneous interactions out of 250) compared to the 2% error rate that was artificially added to other traces. The two primary sources of error were maintaining the internal state and switching to different actions. The human expert often forgot its internal state (which direction it had previously turned) so numerous errors were a result of turning the incorrect direction. The second source of error occured when the expert had been repeatedly performing one type of action, like moving forward, and continued doing that action even after it should have switched to another action, like turning. This is likely because the expert was quickly clicking one button to repeatedly perform the first action and was not able to react in time to the change in sensory inputs.

In the previous experiments, when identifying noteworthy interactions each interaction was only compared to other interactions in the trace. For this round of analysis, cases from an existing case base are also used to identify noteworthy interactions. This is done to show the value of the previously observed cases during

trace analysis, especially when the newly generated trace is relatively short (only 250 interactions). The case base was built from the cleaned trace of the control program when it used an internal state and had error artificially added (Error+State trace in Table 4.8). Other than using the case base during analysis, all other parameters remained the same (9 additional traces generated, $\tau = 0.9999$, $\alpha = 0.90$ and $\beta = 0.10$).

The results (Table 4.10) show that if only the human expert trace is used to detect noteworthy interactions (row Human) then very few noteworthy interactions are detected. The cleaned trace removes only one error (a 2.4% decrease). This is what we might expect since the trace only contains 250 interactions so it is unlikely an erroneous interaction will be highly similar to another interaction in the trace. If the expert was available for significantly longer periods of time, so longer traces could be generated, then it might be possible to identify more noteworthy interactions in the trace. However, if the demonstrated behaviour is relatively short or the expert is only willing to demonstrate for short periods of time then only using a single trace will not be sufficient. The results are improved when an existing case base is also used to detect noteworthy interactions (row Human+Casebase). The number of detected noteworthy interactions increases and, more importantly, the number of errors in the trace decreases (a decrease of 35.7%). While the decrease in errors is not as large as it was in the previous experiments, the results show that cleaning can reduce the number of errors even when the error rate is high (much higher than the artificially added 2% error rate and even higher than the $\beta$ parameter). Even through the generated replay traces also had high levels of noise, since they too were generated by observing the human expert, the analysis was able to detect that the noteworthy intereactions were largely due to errors and internal state.

| | NW | Error | ND | State | Initial Errors | Final Errors |
|---|---|---|---|---|---|---|
| **Human** | 2 | 50% | 0% | 50% | 42 | 41 |
| **Human+Casebase** | 27 | 55.6% | 3.7% | 40.7% | 42 | 27 |

Table 4.10: The analysis results and quality of a human expert trace

### 4.3.5 Discussion

This section has described an approach to analyze and clean traces of an expert's behaviour. The analysis identifies when a single sensory input can, at different times, result in different actions being performed. The expert is made to replay the original trace in order to generate several new versions of the trace and those traces are used to determine if the expert reasoned with an internal state, performed non-deterministic behaviour or performed any errors. The trace can then be cleaned in order to remove any detected errors.

Our experiments demonstrated the applicability of the analysis in an obstacle avoidance domain. The results showed that the analysis was able to correctly detect which of the three properties were present in the trace and cleaning was able to remove many of the errors. This was true when only one property was present in each trace and when multiple properties were present, and for both computer and human experts.

The major assumptions of this approach are that the expert is available to generate new traces and that the agent is able to present the inputs in a realistic manner. If the expert is not available, this approach can not be used since it relies on the generated traces. If the agent does not present inputs to the expert in a way that is similar to how the environment presents them, the expert may behave differently which can compromise the quality of the generated traces.

In this section we have shown how state-based behaviour can be identified, but we

have not described how we can use this information during learning. The following section will describe how a case-based learning by observation agent can use cases collected from experts that reason with internal state information and perform the demonstrated behaviours.

## 4.4 Deployment: Learning from State-based Experts

Any information that is internal to an expert, like the expert's state, is not directly observable by a learning agent. This can result in the observing agent missing key information that it would need to accurately learn the expert's behaviour.

This section will examine the idea that if an observing agent only uses the latest sensory input 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 inputs and actions. Indeed, past inputs and actions may still influence the current behaviour. This temporal relationship can then be exploited in order to approximate the state of the expert.

### 4.4.1 Temporal Case Definition

For a state-based expert, the previous definition of a case (in Section 3.1) would not be appropriate since it only contains the sensory input and action. Key information is missing from the case as it does not contain any portions of the expert's run related to past sensory inputs or actions. As we discussed in the previous section, this can result in cases that have identical problem portions but different solutions.

The entire run of an expert may be necessary in order to infer what the expert's

internal state is [58] and successfully select which action to perform (as we described in the previous section). This leads us to redefine a case $C_t$ as a pair containing the current run $R_t$ and the performed action $A_t$:

$$C_t = \langle R_t, A_t \rangle$$

This definition of a case is appropriate because it allows for approximating the action selection of a rote expert (using past actions), a reactive expert (using the current sensory input) and a state-based expert (using the entire run).

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 sensory inputs and actions. Therefore, when determining the similarity of two runs of equal length, $R_t^a$ and $R_t^b$, it is necessary to determine the similarity of the elements of the runs:

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

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 sensory inputs and the solution similarity metric is used to calculate the similarity between actions (recall that in Section 4.1 both sensory inputs and actions were modelled as having associated similarity functions). 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 if the agent has been interacting with the environment for a significant amount of time. Since agents often operate under real-time constraints, having a similarity function that becomes more computationally expensive over time would not be a suitable option. This issue would be compounded if the sensory inputs or actions had complex structures and required computationally expensive similarity measures.

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 sensory input or action influences the expert's internal state. As the run grows, the influential sensory input will move further and further into the past. In the domain of soccer, if a player toggles between kicking the ball and passing the ball, the information that influences the internal state (whether the player kicked last or passed last) will move further in the past if the player performs other actions like moving around the field. There is no way to know, without examining the run, where in the run the last kick or pass actions occurred.

Previous case-based reasoning work has examined using sequences of input data during reasoning. Martin and Plaza [32] note that reasoning often can not be performed at a single point in time but requires reasoning over a period of time. In the domain of intrusion detection, they show the benefit of reasoning with a series of inputs rather than just the currently received input. Similarly, it was found that reasoning with a sequence of past inputs allows inputs that are otherwise similar to be differentiated [54]. One limitation of these approaches is that they only consider the influence of past inputs and do not consider past sequences of actions during reasoning.

Approaches that do take into account sequences of past actions have been called trace-based reasoning [36] and episode-based reasoning [51]. These approaches, while they have different names, both follow a similar approach. Instead of reasoning with a single input they use either a fixed-length sequence of past actions [13] or a fixed-length sequence of both inputs and actions [6, 51]. This allows the past behaviour of the reasoner, in the form of the previous actions it has performed, to influence its future reasoning. These approaches are limited since they require defining, in advance, the number of past inputs and actions to use. This can result in important information being excluded or large amounts of extra information being included. One alternative would be to store sequences of all possible lengths but that would increase both the space complexity for storing sequences and computational complexity for comparing sequences.

We propose an approach to run retrieval that starts with an initial run length of 1, only taking into account the current sensory input, 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 sensory input $S_t$.

$$R_t : R_{t-1} \xrightarrow{A_{t-1}} S_t$$

Each case, which is composed of a run and the associated action, can be rewritten as a 4-tuple containing the current sensory input, the action associated with the current run, the previous run and the action of the previous run:

$$C_t = \langle R_{t-1}, A_{t-1}, S_t, A_t \rangle$$

Which can be further simplified as:

$$C_t = \langle C_{t-1}, S_t, A_t \rangle$$

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

### 4.4.2 Temporal Backtracking

One approach to finding cases that are similar to an input problem, which is key to case retrieval in CBR, is to compare the entire input problem to the entire problem portions of the cases. However, as was discussed previously, when the problems can grow over time, like with runs, this can cause the similarity calculations to become increasingly computationally expensive. Instead, our case retrieval algorithm will exploit the recursive nature of our case definition.

The algorithm, shown in Algorithm 1, is able to dynamically backtrack, starting with the current sensory input, 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, either sensory inputs (retrieved using the *state(..)* function) or actions (retrieved using the *action(..)*

function), that occur further in the past. In order to eliminate potential cases, two threshold values are used: the *problem threshold (PT)* and the *solution threshold (ST)*. The problem threshold is used when comparing sensory inputs and the solution threshold is used when comparing actions. In some situations the algorithm only needs to compare the current sensory input 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.

### 4.4.3 Evaluation

In Section 4.1, we saw that even a simple internal state, like toggling the direction in which an obstacle avoidance robot turns, can have a significant impact on learning performance. In those experiments, reactive retrieval was used to control a physical obstacle avoidance robot. In this section we will look to demonstrate the benefits of our temporal backtracking retrieval approach when learning by observing a simulated version of the obstacle avoidance robot.

The simulated robot will use the same input and output models that were used in Section 4.3. The problem similarity metric is the same metric that was used in that section and the solution similarity metric produces a binary value depending on if the actions are the same or not:

$$sim(A_a, A_b) = \begin{cases} 1 & , if A_a = A_b \\ 0 & , if A_a \neq A_b \end{cases}$$

**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)
```

#### 4.4.3.1 Expert Agents

Three different expert agents will be passively observed and learnt from in our experiments. All experts share the same general obstacle avoidance behaviour. If their touch sensor indicates they have come in contact with an obstacle (a sensor value of 1), the robot will be moved backwards. 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 three experts were selected because they have different causes of their state changes and the internal states will need to be identified using different information in the runs. The causes of state change that we examine are:

- **Action-based state change (ABSC)** - The state of the agent is changed based on an action they perform. This expert toggles its turning direction after each turn (Figure 4.16). 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.

- **Input-based state change (IBSC)** - The state of the agent is changed based on an external input. The expert (Figure 4.17) turns the same direction until it receives an input on the sound sensor telling it what direction it should now turn (a 1 value representing left and a 2 value representing right). The run must be examined to find the previous turn-indicating value that appeared on the sound sensor.

- **Input and action-based state change (IABSC)** - The state of the agent is changed based on both past actions and external inputs. Like the IBSC agent, the agent only switches its turn direction when it receives an input on

the sound sensor (Figure 4.18). However, the sound sensor does not explicitly indicate which direction to turn but instead just informs the agent to switch its turn direction. When examining the agent's run, if there was a sound sensor value indicating the turn direction should be changed (and the agent did not turn afterwards) it is also necessary to see what the last turn direction of the agent was.

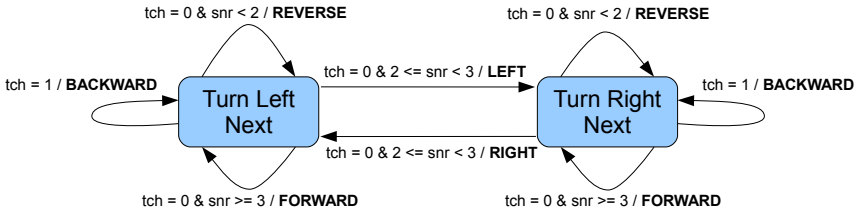



Figure 4.16: State machine of expert that has an action-based state change

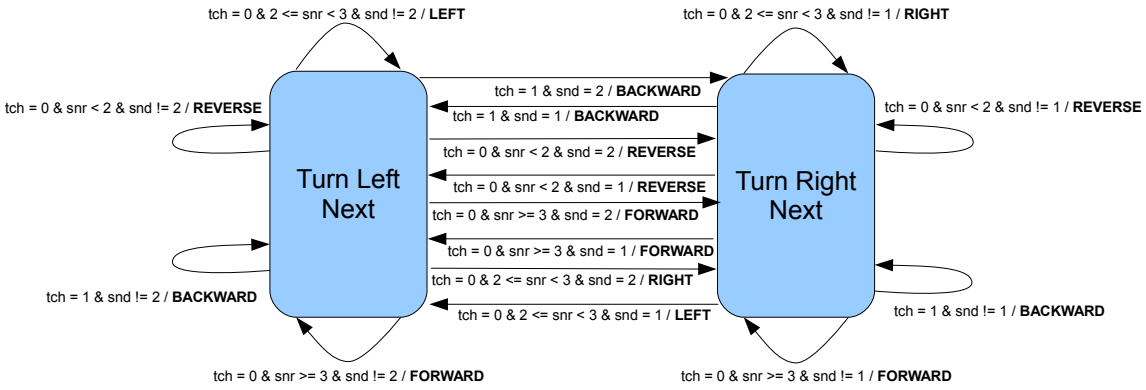



Figure 4.17: State machine of expert that has an input-based state change

An example of the state changes, from the *turn left next* (TLN) and *turn right next* (TRN) states[12], is shown in Figure 4.19. This example demonstrates that even though presented with identical inputs and starting from the same initial state (TLN) the agents have different state transitions and outputs. The state of the ABSC agent

[12]It should be noted that these state names can be slightly misleading since, in Figure 4.17 and Figure 4.18, there are state transitions where the turn direction is different than the name implies. However, in order to use the same state names for all three agents, these names are used to indicate what the most common turn direction will be.

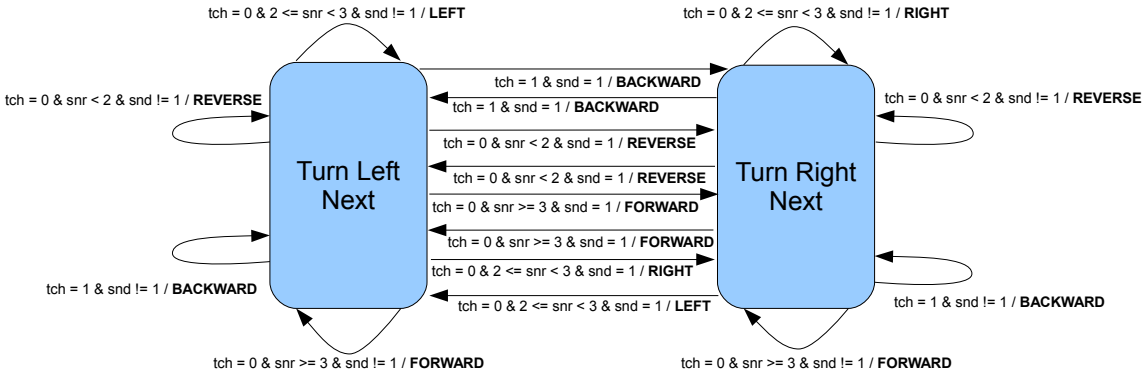



Figure 4.18: State machine of expert that has an input and action-based state change

is dependant on its past turn direction whereas the IBSC agent's state is dependant on the most recent non-zero value on the sound sensor. The IABSC agent's state is dependant on both its previous state and the value on the sound sensor.

| | | | | | |
|---|---|---|---|---|---|
| **touch** | 0 | 0 | 0 | 1 | |
| **sonar** | 2.5 | 7.1 | 2.1 | 0.1 | |
| **sound** | 0 | 1 | 0 | 2 | |
| ABSC | TLN —LEFT→ | TRN —FORWARD→ | TRN —RIGHT→ | TLN —BACKWARD→ | TLN |
| IBSC | TLN —LEFT→ | TLN —FORWARD→ | TLN —LEFT→ | TLN —BACKWARD→ | TRN |
| IABSC | TLN —LEFT→ | TLN —FORWARD→ | TRN —RIGHT→ | TRN —BACKWARD→ | TRN |

Figure 4.19: The state changes of the three experts in response to sensory inputs

Each of the experts was passively observed interacting with the environment over a period of time. This resulted in a case base, for each expert, containing $50,000$ cases. Additionally, each expert agent was also observed in order to create testing case bases for use during evaluation. There were 25 testing case bases created for each expert 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).

#### 4.4.3.2 Class Separation

The selection of the three experts was based on the assumption that their behaviour could not be learnt using only the current sensory input during retrieval. 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 sensory input (the values of the touch, sonar and sound sensors).

The mean similarity[13] of the nearest like neighbour and nearest unlike neighbour, for each expert, is shown in Table 4.11. All of the actions, for each expert, 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 300 states if we discretized 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

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

would not all have the same action.

| | ABSC | | IBSC | | IABSC | |
|---|---|---|---|---|---|---|
| | **NLN** | **NUN** | **NLN** | **NUN** | **NLN** | **NUN** |
| **Forward** | 1.00 | 0.86 | 1.00 | 0.86 | 1.00 | 0.86 |
| **Reverse** | 1.00 | 0.92 | 1.00 | 0.92 | 1.00 | 0.92 |
| **Backwards** | 1.00 | 0.67 | 1.00 | 0.67 | 1.00 | 0.67 |
| **Left** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| **Right** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |

Table 4.11: Mean similarity of nearest like neighbour and nearest unlike neighbour

#### 4.4.3.3 Retrieval Results

The previous results indicate that the left and right actions appear to be difficult to separate when using only the current sensory input during retrieval. 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 input as a feature 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 Algorithm 1. Each retrieval approach used the large cases base, containing $50,000$ cases, as the training case base. Each testing trial, 25 trials per expert, 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 sensory inputs. For the temporal backtracking approach three similarity settings were tested (ST refers to the *solution threshold*): $\{CPT = 0.99, PPT = 0.90, ST = 0.90\}$, $\{CPT = 0.99, PPT = 0.90, ST = 0.00\}$, $\{CPT = 0.99, PPT = 0.00, ST = 0.90\}$. 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 4.12, show the accuracy results (and 95% confidence intervals) of the reactive retrieval approach and of temporal backtracking retrieval using the best threshold set. For the action-based state change (ABSC) expert the best threshold value was $\{CPT = 0.99, PPT = 0.00, ST = 0.90\}$. For the input-based state change (IBSC) expert the best threshold was $\{CPT = 0.99, PPT = 0.90, ST = 0.00\}$, and the input and action-based state change (IABSC) expert was $\{CPT = 0.99, PPT = 0.90, ST = 0.90\}$. The best threshold values found confirm our assumptions about what information in the run is necessary to imitate each of the experts. For all three experts, 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[14] 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.

[14]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.

| | ABSC | | IBSC | | IABSC | |
|---|---|---|---|---|---|---|
| | **RR** | **TB** | **RR** | **TB** | **RR** | **TB** |
| **Forward** | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 |
| **Reverse** | 1.00±0.00 | 0.97±0.01 | 1.00±0.00 | 0.97±0.00 | 1.00±0.00 | 0.96±0.01 |
| **Backwards** | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 | 1.00±0.00 |
| **Left** | 0.48±0.02 | 0.74±0.02 | 0.61±0.02 | 0.85±0.01 | 0.52±0.01 | 0.62±0.02 |
| **Right** | 0.50±0.02 | 0.73±0.02 | 0.59±0.02 | 0.86±0.01 | 0.51±0.02 | 0.62±0.02 |
| **Overall** | 0.80±0.01 | 0.89±0.01 | 0.84±0.01 | 0.93±0.00 | 0.81±0.00 | 0.84±0.01 |

Table 4.12: Retrieval accuracy of standard reactive retrieval and temporal backtracking

### 4.4.4 Discussion

In this section 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 three different simulated obstacle avoidance robots. While the behaviour of these three agents 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 sensory inputs and actions during retrieval, was able to significantly improve the retrieval accuracy. We found selecting appropriate threshold values for use in the temporal backtracking algorithm

was able to improve results but even using non-optimal thresholds significantly improves the results compared to reactive retrieval. While we were able to improve the retrieval accuracy, the accuracy was not perfect on the testing data. This is because there were situations where none of the cases had highly similar runs to the input problem.

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 sensory input. However, in some situations the algorithm needed to go as far as 58 time steps in the past in order to select the correct action to perform. 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 300 environment states and 5 actions, a run of length $n$ would have approximately $300^n \times 5^{n-1}$ states. When comparing runs, a relatively simple and small state space becomes much larger. Even if the maximum run length was 2, it would have been necessary to store $450,000$ unique runs (nine times larger than the case base we used) and these would have been insufficient when longer runs were required.

It should be noted that, although this approach has been referred to as a case-based reasoning retrieval algorithm, it actually performs both retrieval and reuse. The algorithm returns an action, which is the solution to the input problem, instead of returning one or more source cases. This could easily be changed by having the algorithm return the set of nearest neighbour cases in order to make it only a retrieval algorithm. This would allow a more complex reuse algorithm to be used instead of directly copying the solution. However, since we reuse the solutions directly, the retrieval and reuse algorithms were combined for simplicity.

The focus of this section has been on state-based learning, but all previous sections only made use of reactive retrieval. This means that no attempt was made to examine the benefit of the previous techniques when learning from a state-based

expert. We will now further our evaluation by combining state-based learning with case acquisition and trace analysis.

## 4.5 Combining Subtasks

The previous sections of this chapter have presented the techniques for each learning by observation subtask but have examined those subtasks largely in isolation. Only the modelling subtask was necessary for each other subtask since every subtask makes use of the input and output models. In this section, the ability of subtasks to work together will be examined. The applicability of mixed-initiative case acquisition for a state-based expert will be examined as well as the benefit of using analysis to select an appropriate retrieval algorithm to use.

### 4.5.1 Mixed-initiative Case Acquisition from a State-based Expert

Mixed-initiative case acquisition was found to improve the performance of a learning by observation agent, as shown in Section 4.2, but the experiments only involved a reactive expert. We will now examine the benefits of mixed-initiative case acquisition when learning from a state-based expert.

Recall that when mixed-initiative case acquisition was used the learning agent would cede control to the expert when it was unable to solve an input problem. This occurred when there were no problems in the case base with a similarity to the input problem $S_t$ above a threshold $\tau$. However, when temporal backtracking retrieval (Algorithm 1 in the previous section) is used to learn from state-based experts there is a second type of retrieval failure that can occur. If, during backtracking, the retrieval algorithm eliminates all possible candidate solutions then it will simply return the action of the most similar case. In this situation, the learning agent will also cede

to the expert and the expert will provide both the correct action $A_t$ as well as the portion of its run it used to determine that action. For example, if the previous turn direction influenced the expert's current action, and the previous turn action occurred three time steps in the past ($A_{t-3}$), then the expert will provide a portion of its run that includes that information. The case base $CB$ will have a case related to the current sensory input and action along with cases from the other run information provided by the expert ($CB = CB \cup \{C_t, C_{t-1}, C_{t-2}, C_{t-3}\}$ where $C_t = \langle C_{t-1}, S_t, A_t \rangle$, $C_{t-1} = \langle C_{t-2}, S_{t-1}, A_{t-1} \rangle$, $C_{t-2} = \langle C_{t-3}, S_{t-2}, A_{t-2} \rangle$ and $C_{t-3} = \langle null, S_{t-3}, A_{t-3} \rangle$).

This results in a varying number of cases being provided by the expert each time it assists the learning agent and also makes the assumption that the expert can identify what portion of its run should be provided to the agent. However, if the expert can not identify what portion of the run should be provided then a fixed-length portion of the run can be provided. For example, the expert could always provide the action to perform and the five case from its run. However, if a fixed-length number of cases are provided there is no guarantee that all necessary run information is included (or that unnecessary information is not included).

The state-based expert we will use for experimentation is the action-based state change agent that controls a simulated obstacle avoidance robot (from Section 4.4). Recall that this expert toggles the direction it turns so it never turns in the same direction twice in a row. In Section 4.4, this expert was observed passively in order to create a case base of $50,000$ cases and 25 test case bases of $2,500$ cases each. To generate the mixed-initiative case base, the learning agent initially started with an empty case base. Whenever the agent ceded control to the expert and observed the expert's behaviour, those cases (the case related to the current sensory input and action, along with the cases provided from the expert's run) were added to the case base until $50,000$ cases were generated using mixed-initiative case acquisition. The thresholds used for temporal backtracking retrieval were identical to those used

previously (CPT = 0.99, PPT = 0.00; ST = 0.90) and the mixed-initiative similarity threshold was $\tau = 0.99$.

The accuracy of the learning agent when using the mixed-initiative case base was measured by providing each case from each of the 25 testing case bases to the learning agent and measuring how often the correct action was selected. Table 4.13 show the results as well as the results that were previously found, in Section 4.4, when using the passively acquired case base. These results show that even though both case bases used by the agent were the same length, $50,000$ cases, the overall accuracy when using the mixed-initiative case base was a statistically significant improvement (using a paired t-test with $p < 0.01$). The accuracy when performing two actions, turning left and right, were statistically significant improvements while the remaining actions had no statistically significant differences.

| | **Passive** | **Mixed-initiative** |
|---|---|---|
| **Forward** | 1.00±0.00 | 1.00±0.00 |
| **Reverse** | 0.97±0.01 | 0.97±0.01 |
| **Backwards** | 1.00±0.00 | 1.00±0.00 |
| **Left** | 0.74±0.02 | 0.78±0.02 |
| **Right** | 0.73±0.02 | 0.78±0.02 |
| **Overall** | 0.89±0.01 | 0.91±0.01 |

Table 4.13: Retrieval accuracy when using a passive and mixed-initiative case base

### 4.5.2 Using Trace Analysis to Guide Algorithm Selection

The experiments in Section 4.3 have shown the results of trace analysis and now we will see if performing trace analysis is beneficial for the performance of a learning by observation agent (recall that the previous experiments only identified state-based behaviour and never made use of that information during learning). We look to see if

the learning agent can benefit by knowing that the expert maintains an internal state by selecting an appropriate case retrieval algorithm (such as the one described in Section 4.4). The agent observed the computer controlled obstacle avoidance expert that had an internal state and generated a trace of 10,000 interactions (this corresponds to the Error+State trace that we previously showed in Table 4.9). The original version of the trace was used to create one case base and the cleaned version of the trace was used to create a second case base.

For testing purposes, 25 extra traces of 1,000 interactions each were created by observing the same expert. For each of the 25 test traces, the sensory input from each of the 1,000 interactions was given as input to the agent and the agent's accuracy was calculated by measuring the percentage of times the agent was able to select the correct action (since each interaction in the trace also contained the correct action). The average accuracy of all test runs and the 95% confidence interval were calculated. Using reactive retrieval, the agent had a slightly higher accuracy when using the cleaned case base ($79.7\% \pm 0.9\%$) compared to using the original case base ($79.0\% \pm 0.9\%$) although the difference was not statistically signficant (using a t-test with $p < 0.05$). When the agent used state-based retrieval with the cleaned case base it achieved its highest accuracy ($87.9\% \pm 0.5\%$). This was significantly higher than the accuracy using reactive retrieval and the accuracy using state-based retrieval with the original case base ($86.4\% \pm 0.6\%$).

These results show that cleaning errors out of the traces can improve the performance of a case-based learning by observation agent. Perhaps more importantly, the ability of the trace analysis to identify state-based behaviour allows the agent to select an appropriate case retrieval algorithm to use. The agent had the largest accuracy improvement by selecting the state-based retrieval algorithm instead of the reactive retrieval algorithm. The agent could choose to use state-based retrieval in all situations, without performing analysis, but using state-based retrieval is more

computationally expensive than reactive retrieval. Using the state-based retrieval unnecessarily could be detrimental to the agent's performance if the agent operates under real-time constraints.

### 4.5.3 Discussion

This section has examined the ability to combine techniques together so that multiple learning by observation subtasks are used by the learning agent. For both situations, combining mixed-initiative case acquisition with state-based learning and combining trace analysis with state-based learning, there was a quantifiable performance benefit to using the subtasks together. These results help to illustrate that the subtasks in the learning by observation cycle and the techniques developed in this thesis are not stand-alone items but are part of a larger workflow.

## 4.6 Complete Cycle

This section will examine how all of the subtasks in the learning by observation cycle can be used together to improve the performance of a learning agent. Simulated soccer agents and an agent controlling a physical robot will be examined.

### 4.6.1 Simulated Soccer

The input and action models for simulated soccer were previously presented, in Section 4.1, but we did not examine how the other learning by observation subtasks influence the ability of an agent to learn soccer behaviour. Two expert agents will be used for this evaluation:

- *Krislet*: Krislet[15] agents behave in a simple reactive manner. They turn until

[15] `http://www.ida.liu.se/~frehe/RoboCup/Libs/libsv5xx.html`

they can see the soccer ball and then run toward the ball. When they get to the ball they attempt to kick it toward their opponent's goal.

- *CMUnited*: CMUnited [56] is far more complex and were the former champions of the RoboCup Simulation League. They use a layered learning architecture and a number of strategies including formation strategies and agent communication. CMUnited players can have multiple states of behaviour and maintain internal models of the world, so their behaviour is significantly more complex than anything we have examined and likely more similar to that of a human expert.

Each of these agents was passively observed in order to collect a case base of $5,000$ training cases and 25 testing case bases of $3,000$ cases each. Each case base was generated from a single, unique trace of the expert. The mean accuracy of the learning agent, when attempting to predict the action of each test case using reactive retrieval, was used to determine a baseline performance when learning from each expert agent (the *passive/reactive* rows in Tables 4.14 and 4.15) .

| **Acquisition** | **Retrieval** | **Kick** | **Dash** | **Turn** | **Overall** |
|---|---|---|---|---|---|
| *Passive* | *Reactive* | 0.27±0.07 | 0.71±0.02 | 0.83±0.01 | 0.61±0.02 |
| *Mixed-initiative* | *Reactive* | 0.48±0.11 | 0.65±0.02 | 0.86±0.00 | 0.66±0.02 |
| *Mixed-initiative* | *State-based* | 0.48±0.11 | 0.65±0.02 | 0.86±0.00 | 0.66±0.02 |

Table 4.14: The accuracy of the learning agent when learning from Krislet

The baseline results demonstrate the use of the modelling and observation subtasks. Those two training case bases were then used during preprocessing in order to analyze the observations. The analysis performed single trace analysis by comparing each training case to the remaining $4,999$ training cases in order to identify cases that were highly similar ($\tau = 0.99$) but resulted in different actions. As we might expect

| Acquisition | Retrieval | Kick | Dash | Turn | Overall |
|---|---|---|---|---|---|
| *Passive* | *Reactive* | 0.37±0.04 | 0.58±0.02 | 0.55±0.03 | 0.50±0.02 |
| *Mixed-initiative* | *Reactive* | 0.45±0.03 | 0.57±0.02 | 0.57±0.03 | 0.53±0.02 |
| *Mixed-initiative* | *State-based* | 0.48±0.03 | 0.62±0.02 | 0.61±0.04 | 0.57±0.02 |

Table 4.15: The accuracy of the learning agent when learning from CMUnited

from these experts, the reactive Krislet agent had no noteworthy interactions identified whereas the more complex CMUnited agent did (Table 4.16). Multiple trace analysis was then performed on the trace used to generate the CMUnited training case base (labelling thresholds of $\alpha = 0.90$ and $\beta = 0.10$). The majority of CMUnited's noteworthy interactions were labelled as being a result of internal state, but there was also a presence of non-deterministic behaviour and error. It should be noted that after cleaning the identified errors there was no noticeable difference in the learning agent's performance when using reactive retrieval. This lack of improvement is to be expected given the low error rate in the trace.

| | Noteworthy | Error | ND | State |
|---|---|---|---|---|
| **Krislet** | 0 | - | - | - |
| **CMUnited** | 113 | 6.2% | 33.6% | 60.2% |

Table 4.16: The analysis results on traces of Krislet and CMUnited

The information gained from analysis, that Krislet is reactive and CMUnited is largely state-based, can be used to guide further observation and deployment. As we discussed in the previous section, when mixed-initiative case acquisition is performed on a reactive expert the solution to a single problem is provided whereas for a state-based expert a portion of its run is also provided. This makes it important to know if the expert is reactive or state-based since retaining information about the expert's

run will cause the case base to grow at a faster rate. If the learning agent can only use a case base of a fixed size, then it may not be beneficial to acquire cases related to problems the agent was able to solve itself (the learning agent would have already asked the expert for assistance if it could not solve any previously encountered problems). However, a state-based expert will need the other cases if state-based retrieval is used.

Using the trace analysis results, when performing mixed-initiative case acquisition the Krislet agent will only provide the solution to the current problem whereas the CMUnited agent will provide the solution to the current problem and the nine previous problem-solution pairs in its run. This causes the learning agent to add a single case every time it cedes to the Krislet agent and at most 10 cases every time it cedes to the CMUnited agent. For each expert, the learning agent only used the first $2,500$ training cases (representing the first $2,500$ interactions in the training trace) and was deployed in the environment until it had acquired an additional $2,500$ cases using mixed-initiative case acquisition (using $\tau = 0.90$). This resulted in a total case base size of $5,000$ cases (the same size as the passively acquired case base). Each of these case bases was then used by the learning agent and the accuracy on the 25 test case bases was measured.

When reactive retrieval was used (*mixed-initiative/reactive* rows in Tables 4.14 and 4.15) the overall accuracy when learning from both experts was improved. The Krislet agent had statistically significant (using a paired t-test with $p < 0.05$) improvements in the overall accuracy, the accuracy of the kick action and the accuracy of the turn action. There was also a significant decrease in the accuracy of the dash action. This is likely because the action distribution is severely imbalanced with far more dash actions than either kick or turn actions. Mixed-initiative case acquisition helps to improve this imbalance by adding more kick and turn actions. Similarly, when learning from the CMUnited agent there were statistically significant improvements

to the kick, turn and overall accuracy. While there was also a decrease in the dash accuracy, the difference was not statistically significant.

While the results using mixed-initiative acquisition are an improvement over the baseline, we will now use state-based retrieval in an attempt to improve the performance further. When performing state-based retrieval (using a current problem threshold of 0.95, a past problem threshold of 0.00, and a solution threshold of 0.90) the agent will use the mixed-initiative case base for training. The results (*mixed-initiative/state-based* rows in Tables 4.14 and 4.15) showed that there was no change in the performance when learning from the Krislet agent. This is what we would expect since, based on the analysis performed previously, there were no similar problems with different solutions so the state-based retrieval would never need to backtrack. When no backtracking is performed, the state-based retrieval is essentially reactive retrieval. However, there are statistically significant improvements (over both passive/reactive and mixed-initiative/reactive) for all accuracy values when using state-based retrieval on the CMUnited case base. These results show that even for a highly complex agent like CMUnited there is still a benefit of using state-based retrieval.

These experiments have shown how each subtask in the learning by observation cycle can be used in combination and the relative performance improvement of each technique. While there was a benefit, when learning from both the Krislet and CMUnited agents, using individual techniques the best performance was found when using the techniques together. When learning from the CMUnited agent, the best performance was a result of using mixed-initiative case acquisition and state-based retrieval. Learning from the Krislet agent was not improved when using state-based retrieval but there was also no decrease in performance (compared to using reactive retrieval and the mixed-initiative case base). This shows that even if the Krislet agent was incorrectly thought to be state-based there would not be a performance decrease in using state-based retrieval. It should also be noted that while analysis did not provide

any quantifiable performance benefits it did provide important information that was used to guide other parts of the learning by observation cycle.

While there were performance improvements, the ability to learn from complex agents is still difficult and not fully achieved. When the learning agent uses the mixed-initiative training data it is able to do a reasonable job of playing soccer like Krislet (when examined qualitatively). However, the agent is still unable to perform the CMUnited behaviour as expected (even when using mixed-initiative case acquisition and state-based retrieval). The learning agent still performs many errors and often times appears to be behaving significantly different than the original expert.

There are several key limitations of our approach that we feel result in the behaviour of highly complex agents like CMUnited to be difficult to fully learn. Firstly, the CMUnited agent relies on inter-agent communication and the soccer model used by the learning agent does not capture that. Since the learning agent can not observe how the expert communicates with its teammates, it will not be able to reason with that information. Secondly, the CMUnited agents maintain a world model and reason with a significant amount of internal state information. This results in a significant number of situations where the learning agent needs to look at past run information during temporal backtracking. In many situations, the learning agent is unable to select an action with a high degree of confidence because none of the similar cases have a run that is similar to the agent's. One solution would be to increase the case base size, but this is often not an acceptable solution since the size of the case base may be limited by real-time constraints (the number of cases that can be compared within a given time limit). Finally, no attempt is made to reproduce the non-deterministic behaviour of the expert. Although the behaviour is non-deterministic, an action selected non-deterministically may influence further action selection (as part of the expert's run). In such a situation, it would be important to perform non-deterministic actions with a similar probability distribution to that of the expert.

### 4.6.2 Physical Robot

We will now present a case study that shows how all tasks in the learning by observation cycle can be used when watching a human expert control a physical robot.

#### 4.6.2.1 Modelling

The case-based learning agent learns by observing a human expert performing different behaviours with different hardware configurations. As in Section 4.1.3, the sensors and effectors available to the learning agent are not defined in advance but instead register when they are connected.

For the first behaviour, the iRobot Create robot (described in Section 4.1) is connected to the learning agent and a human expert controls the robot while the learning agent observes. The demonstrated behaviour is a tracking behaviour where the robot is navigated toward its charging station (the same behaviour that was demonstrated in the initial iRobot Create case study). If the robot can not currently detect an infrared signal, which is produced by the charging station, it turns until the signal can be detected and then moves forward. When the robot comes into contact with the charging station (the infrared signal is present and the bumper sensor indicates the robot is in contact with something) it will stop. The learning agent collected 14 cases by observing the expert perform this behaviour.

The second behaviour also involves connecting the iRobot Create to the learning agent. However, instead of moving toward an object (the charging station) the expert demonstrates an obstacle avoidance behaviour. The expert moves the robot forward until it comes in contact with an obstacle, as indicated by the bumper sensor, and then turns until the robot is no longer in contact with the obstacle. Once the robot is free of the obstacle it is then moved forward again. The demonstration of this behaviour resulted in 15 cases being observed.

The final behaviour represents a hypothetical upgrade to the robot. The obstacle avoidance behaviour requires the robot to actually come in contact with the obstacle so the robot could potentially get stuck or be damaged. This behaviour has both the robot and a sonar sensor connected to the learning agent. Like the sensors on the robot, the sonar sensor registers with the learning agent when it is connected. Instead of relying on the bumper sensor to detect an obstacle, the expert uses the value of the sonar sensor. This allows the expert to detect and react to obstacles as the robot nears them. 12 cases were collected when observing this behaviour.

The learning agent was able to achieve 100% accuracy, for each of the three behaviours, when attempting to predict the associated action of 100 test cases. More importantly, when controlling the robot on its own the learning agent was able to successfully perform all three of the behaviours[16]. This shows that the learning agent is able to successfully learn and perform behaviours without any predefined knowledge of what it will observe or what hardware it will have available. The robot can rapidly be repurposed as its task or hardware change.

#### 4.6.2.2 Mixed-initiative Case Acquisition

The learning agent was able to learn the obstacle tracking behaviour but did not necessarily do it in the most efficient manner. Recall that the learning agent observed the expert perform the entire behaviour and collected 14 cases. We will now examine the ability of the agent to learn the obstacle tracking behaviour using mixed-initiative case acquisition. Initially, the learning agent will start with an empty case base. If the agent is unable to retrieve a case with a similarity to the current sensory input above a threshold of $\tau = 0.90$, the agent will prompt the user for an action to perform, create a new case from the resulting interaction, and add the case to the case base.

[16]A video of the learning agent observing and performing all three behaviours: `http://sce.carleton.ca/~mfloyd/LearningVideos/LearningThreeBehaviours.mp4`

If the agent can retrieve a case with a similarity above the threshold, it will perform the associated action.

Using this approach, the agent collects four cases. Although this case base is much smaller than the case base of 14 cases when cases are acquired passively, when the learning agent uses the case base it still achieves the same accuracy on the test cases (100%) and is able to perform the behaviour[17]. This is because the learning agent stores many identical cases during passive observation and there is no benefit in keeping numerous instances of the same case. Additionally, using mixed-initiative case acquisition is beneficial because the expert only needs to demonstrate four actions instead of demonstrating the entire behaviour (14 actions).

#### 4.6.2.3 Preprocessing

The observed behaviours thus far have been error free. However, when learning from a human expert there is the possibility that the expert might make an error when demonstrating the behaviour. For this part of the case study, the tracking behaviour was demonstrated again but the human expert performed errors. The learning agent observed the expert perform the behaviour and 30 cases were collected. During the demonstration, the human expert performed two errors (turning when the infrared signal could be detected and moving in reverse instead of forward) but recovered from the errors and fully demonstrated the behaviour.

The learning agent was unable to perform the tracking behaviour if it used the case base that contains errors. Even though there are only two erroneous cases, these cases caused the learning agent to behave incorrectly in many situations so it was never able to get to the charging station. In order to handle the errors, trace analysis was performed on the case base (using $\tau = 0.99$, $\alpha = 0.90$, $\beta = 0.10$ and 2 additional

[17] A video of the learning agent using mixed-initiative case acquisition to learn the behaviour: `http://sce.carleton.ca/~mfloyd/LearningVideos/MixedInitiativeRobotLearning.mp4`

traces were generated). Single trace analysis identified 8 noteworthy interactions and multi trace analysis labelled 2 of them as being a result of error and 6 of them as being a result of state-based behaviour. As was the situation in previous trace analysis experiments, the state-based interactions were a result of the erroneous interactions causing cases to appear noteworthy. However, when the erroneous cases were cleaned the resulting case base no longer had any noteworthy interactions when single trace analysis was performed again.

When the cleaned case base was used by the learning agent it was able to successfully perform the tracking behaviour[18]. This demonstrates that even a low error rate can significantly impact the ability of a learning agent to perform a behaviour. However, even if errors do exist they can be identified and removed so that the agent is able to successfully learn.

#### 4.6.2.4 State-based Learning

The previously demonstrated behaviours did not require reasoning with any internal state information. For the final part of the case study, the tracking behaviour was modified to require the use of an internal state. Instead of stopping when the robot reached the charging station, the expert now moved the robot in reverse. This requires the expert to maintain an internal state related to if the charging station has been reached yet. If it has not, the robot should be moved toward the charging station whereas if it has reached the charging station it should be moved away from the charging station. The learning agent acquired 24 cases when observing this behaviour.

When the learning agent attempted to perform this behaviour using reactive retrieval it was unable to successfully perform the behaviour. The learning agent was able to navigate the robot to the charging station but immediately stopped. This is

[18]A video of the influence of errors, and how trace analysis can remove errors and improve performance: `http://sce.carleton.ca/~mfloyd/LearningVideos/RobotTraceAnalysis.mp4`

because the agent only retrieved cases with sensory inputs that were similar to the current sensory input and did not take into account past sensory inputs or actions. However, if the learning agent used temporal backtracking retrieval (using a current problem threshold of 0.95, a past problem threshold of 0.95, and a solution threshold of 0.90) it was able to perform the entire behaviour[19]. This shows that reactive retrieval can be used to learn portions of a state-based behaviour but in order to fully learn the behaviour a retrieval algorithm that takes into account internal state must be used.

#### 4.6.2.5 Discussion

The physical robot case study has shown that even if behaviours can be learnt with a high degree of accuracy, there is still a benefit in using the techniques presented in this thesis. The modelling approach allows both the demonstrated behaviour and the robotic hardware to be rapidly changed. Using mixed-initiative case acquisition, the agent reduces the number of problems the expert needs to solve and avoids storing redundant cases. If demonstrated behaviours contain errors or state-based behaviour, the learning agent is unable to successfully perform the behaviours. However, using trace analysis allows the agent to identify and remove errors and state-based retrieval allows the agent to perform behaviours that require internal state information.

### 4.6.3 Discussion

In this section we have examined how the techniques developed in this thesis can be used together in various domains. The learning agent was able to significantly improve its ability to learn simulated soccer behaviours by combining the techniques together. For both soccer experts, the expert initially had difficulty learning the behaviours

[19] A video of the expert demonstrating the state-based behaviour and the learning agent attempting to perform the behaviour using both reactive and temporal backtracking retrieval: `http://sce.carleton.ca/~mfloyd/LearningVideos/StateBased.mp4`

correctly but was able to show noticeable improvements as each new technique was used.

The learning agent was initially able to learn the robot control behaviours well, unlike the soccer behaviours, but the learning techniques in this thesis also showed benefits in this domain. Mixed-initiative case acquisition allowed more efficient learning, trace analysis allowed errors to be identified and cleaned, and temporal backtracking retrieval allowed a state-based behaviour to be learnt.

However, even though we have shown two examples of successful uses of the various techniques, there are still several key limitations that must be taken into account:

- Mixed-initiative case acquisition from a state-based expert requires the learning agent to add additional cases, related to the expert's run, not just a single case. In situations where the expert knows what extra information it reasoned with, the expert can provide the correct cases to the learning agent. However, in many situations this is not possible. As we saw in the simulated soccer domain, the learning agent always received a fixed-length number of previous cases. There is no guarantee that these cases contained the correct information needed to infer the expert's state or, if the correct information was included, that no unnecessary information was also provided.

- In domains where the learning agent has real-time constraints on the case base size, there is a limit to the amount of previous run information that can be added to the case base. If too much information from the run is added, the size of the case base will quickly reach the size limit. However, if not enough run information is added the learning agent will have difficulty inferring the internal state.

- Case acquisition, trace analysis and temporal backtracking retrieval all require threshold values to be set. Although we were able to show the benefits of the

techniques using largely similar threshold values in each domain, there is no guarantee those threshold were the optimum values or that they would work in every domain. The learning agent could attempt to optimize these threshold values but, in situations where it needs to be deployed quickly, it might not have the time necessary to compute the optimum values.

- The learning agent is able to use information contained in the run to infer the expert's internal state but it does not try to learn what information influences the internal state. If this information was learnt, the agent could attempt to maintain the state rather than infer it on every retrieval.

- The temporal backtracking retrieval algorithm works well for reactive and state-based behaviour but does not attempt to perform non-deterministic behaviour. For some experts, like CMUnited, it might be useful to identify when the agent should be behaving non-deterministically and attempt to approximate this behaviour. Additionally, the evaluation of the agent would be improved if it could be determined an error was due to incorrectly selecting an action in a non-deterministic situation instead of a reactive or state-based situation.

## 4.7 Discussion

This chapter has presented improvements to each of the four learning by observation subtasks: modelling, observation, preprocessing and deployment. Each of these approaches attempts to address a limitation of existing techniques and improve the performance of learning by observation systems. The motivation for these improvements was to address issues related to general-purpose learning by observation so a variety of case studies and evaluation domains were used. In the examined domains, each of these techniques was found to significantly improve the ability of the agent

to learn or increase what the agent can learn.

# Chapter 5

# Conclusions and Future Work

This thesis has addressed the research question *"What are the issues specific to learning by observation? Is it possible to come up with a general-purpose and task-independent framework for learning by observation and, if so, what are the components of such a framework?"* by examining the state of the art in general-purpose learning and learning by observation systems. The major subtasks performed by learning by observation systems were identified and used to define a cyclical workflow that formalizes how these subtasks are interconnected. Additionally, since learning by observation systems were decomposed into their major subtasks it was possible to examine how each of these subtasks would need to be improved in order to allow for general-purpose learning by observation.

This chapter will provide a summary of the contributions and results, describe known limitations of the work and identify directions for long-term future work.

## 5.1 Summary of Contributions and Results

The following are a summary of the key contributions of this work:

1. **Literature Review** (Chapter 2): The state of the art in learning by observation was examined as well as the issues that are specific to learning from an

agent. It was determined that, although existing learning by observation work is largely ad hoc, there exist a number of common subtasks that each of these systems perform. These subtasks relate to how the expert's inputs and outputs are modelled, how the agent observes the expert, how the agent processes those observations and how the agent performs the behaviour it has learnt.

2. **Learning by Observation Cycle** (Chapter 3): Four common subtasks in learning by observation systems were identified and defined: modelling, observation, preprocessing and deployment. These subtasks were arranged into a cyclical workflow, called the learning by observation cycle, that represents the standard transitions between them. Additionally, four key requirements of these subtasks were identified so that learning by observation could be used for general-purpose learning:

   - *Modelling*: The task of a learning by observation agent can be changed so it is important not to hard-code a definition of the task. Similarly, since the agent can be reconfigured to perform a new task it should not hard-code the features it uses to reason or the actions it can perform.
   - *Observation*: The way in which learning by observation systems observe an expert is passive. This can result in parts of the behaviour that may never be observed since the expert never fully demonstrates them. However, since these systems learn by observing an expert it may be possibly to more actively involve the expert in the learning.
   - *Preprocessing*: Learning by observation systems observe expert agents or human experts, both of which might make errors, reason using internal state information or behave non-deterministically. The learning agent should be able to examine the observations in order to characterize the behaviour of the expert, and identify and clean errors.

- *Deployment*: Existing learning by observation systems can only learn from reactive experts. If an expert has multiple internal states and uses that state information to reason then agents will not be able to successfully learn from it or reproduce the expert's behaviour during deployment.

3. **Modelling** (Section 4.1): A framework for modelling the inputs and outputs of a learning by observation system was presented. This framework was used to design a learning by observation agent that separates the agent's reasoning capabilities, in its Reasoning module, from how it interacts with its environment, the Perception and Motor Control modules. Instead of having to modify the entire agent when the environment is changed, only the Perception and Motor Control modules need to be changed. Case studies in obstacle avoidance, robotic arm control, simulated soccer and Tetris were described in order to show how a single Reasoning module can be used, in all four domains, by only changing the Perception and Motor Control modules. This design was extended to allow sensors and effectors to be dynamically added or removed from the agent without needing to reprogram the agent.

4. **Observation** (Section 4.2): Two observation strategies, active case acquisition and mixed-initiative case acquisition, were presented that can be used as alternatives to passive observation. Unlike the passive approach, these approaches allow the agent to identify areas of the problem space that are poorly covered and attempt to have the expert solve problems in those areas. Mixed-initiative case acquisition allows the agent to ask an expert to solve a specific problem if the agent can not solve it itself. Active case acquisition differs in that it does not have the expert solve difficult problems when they occur but logs them for later. Both of these techniques were able to make observations that would rarely, or

in some instances never, be observable using a passive approach. Additionally, when the agent had the additional observations it was able to improve its learning performance.

5. **Preprocessing** (Section 4.3): The preprocessing technique that was developed involves analyzing traces of the expert's behaviour. Initially, the trace is examined in order to identify situations, called noteworthy interactions, where similar sensory inputs result in different actions. These noteworthy interactions can be a result of four properties: errors, non-deterministic behaviour, unobservable features or multi-state behaviour. In order to differentiate between the properties, the expert is made to encounter the same sequence of sensory inputs several times. The results showed that this analysis was able to successfully measure which properties were present in each trace and remove a majority of the errors from the traces.

6. **Deployment** (Section 4.4): A retrieval algorithm was developed that allows learning from experts with multiple internal states. Instead of using only the expert's current sensory inputs during retrieval, the algorithm also uses past sensory inputs and past actions. In order to make the retrieval computational feasible, the algorithm only uses information when necessary. It starts by using only the current sensory inputs, making it appropriate for reactive experts, but if multiple actions are returned as solutions it begins adding extra information in order to discriminate between the possible actions. The results showed that reactive retrieval algorithms were not able to successfully learn from multi-state experts. However, the novel retrieval algorithm was able to significantly improve the learning performance. The algorithm was able to handle state transitions that occurred due to past actions of the expert, past sensory inputs or a combination of past sensory inputs and actions. Also, the algorithm was

able to handle both when the information that caused the state change was relatively recent and when it was far in the past.

7. **Combination of Techniques** (Section 4.6): An evaluation of how the various techniques in this thesis work together was performed in a simulated soccer domain and a robotics domain. This demonstrated the relative benefit of each technique and how the best learning performance was achieved when techniques from all four learning by observation subtasks were used.

## 5.2 Derived Publications

The following peer-reviewed publications were derived from the results of this thesis:

1. M.W. Floyd and B. Esfandiari. Analysis and Cleaning of User Traces Through Comparison of Multiple Traces. In *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*, AAAI Press, 2013.

2. S. Ontañón and M.W. Floyd. A Comparison of Case Acquisition Strategies for Learning from Observations of State-based Experts. In *Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference*, AAAI Press, 2013.

3. M.W. Floyd, M.V. Bicakci and B. Esfandiari. Case-Based Learning by Observation in Robotics Using a Dynamic Case Representation. In *Proceedings of the Twenty-Fifth International Florida Artificial Intelligence Research Society Conference*, 323-328, AAAI Press, 2012.

4. M.W. Floyd and B. Esfandiari. Learning State-Based Behaviour using Temporally Related Cases. In *Proceedings of the 16th United Kingdom Workshop on*

*Case-Based Reasoning*, 34-45, 2011.

5. M.W. Floyd and B. Esfandiari. Building Learning by Observation Agents Using jLOAF. In *Proceedings of the Workshop on Case-Based Reasoning for Computer Games at the 19th International Conference on Case-Based Reasoning*, 37-41, 2011.

6. M.W. Floyd and B. Esfandiari. Supplemental Case Acquisition using Mixed-Initiative Control. In *Proceedings of the 24th International Florida Artificial Intelligence Research Society Conference*, 395-400, AAAI Press, 2011.

7. M.W. Floyd and B. Esfandiari. A Case-Based Reasoning Framework for Developing Agents Using Learning by Observation. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence*, 531-538, IEEE Computer Society Press, 2011.

8. M.W. Floyd and B. Esfandiari. Toward a Domain-independent Case-based Reasoning Approach for Imitation: Three Case Studies in Gaming. In *Proceedings of the Workshop on Case-Based Reasoning for Computer Games at the 18th International Conference on Case-Based Reasoning*, 55-64, University of Piemonte Orientale Press, 2010.

9. E. Acosta, B. Esfandiari and M.W. Floyd. Feature Selection for CBR in Imitation of RoboCup Agents: A Comparative Study. In *Proceedings of the Workshop on Case-Based Reasoning for Computer Games at the 18th International Conference on Case-Based Reasoning*, 25-34, University of Piemonte Orientale Press, 2010.

10. M.W. Floyd and G.A. Wainer. Creation of DEVS Models using Imitation Learning. In *Proceedings of the 42nd Summer Computer Simulation Conference*, 334-341, SCS Press, 2010.

11. M.W. Floyd and B. Esfandiari. Comparison of Classifiers for use in a Learning by Demonstration System for a Situated Agent. In *Proceedings of the Workshop on Case-Based Reasoning for Computer Games at the 8th International Conference on Case-Based Reasoning*, 87-96, 2009.

12. M.W. Floyd and B. Esfandiari. An Active Approach to Automatic Case Generation. In *Proceedings of the 8th International Conference on Case-Based Reasoning*, 150-164, Springer, 2009.

In addition to published research papers, descriptions and demonstrations of this work has also been presented in video form. The video *"Case-Based Imitation: A Sequel"* won the *Best Video Award* at the Artificial Intelligence Video Competition held at the 24th Conference on Artificial Intelligence (AAAI 2010). Also, the video *"Case-based Reasoning in Games"* was nominated for the *Most Innovative Video Award* at the Artificial Intelligence Video Competition held at the 25th Conference on Artificial Intelligence (AAAI 2011).

## 5.3 Limitations and Future Work

This work has examined a cyclical learning by observation workflow and has made several improvements in the algorithms used for various subtasks of the cycle. However, there are still interesting open problems and limitations of this work that are long-term future work. Several of these limitations and areas of future work are:

- **Goal Inference**: Since this work has dealt with a knowledge-poor approach to learning by observation, no information about the expert's goals is known. It would be beneficial to be able to examine observations and learn the expert's goals, which goals the expert is trying to achieve and how those goals are achieved. This would allow other parts of the case-based reasoning cycle,

revision and retention, to be performed since the learning agent would have some idea of what it should be doing.

- **Less-expensive Trace Analysis**: The preprocessing algorithm presented in this thesis relies on having the expert replay traces of its behaviour. This can be expensive since it requires a significant amount of time from the expert. It would be beneficial to have an analysis technique that only requires the examination of a single trace but can achieve results that are similar to the multi trace approach.

- **Large State-space**: Experts with many internal states require a large case base in order for the learning agent to successfully differentiate between those states. However, as the case base grows so too does the computation resources required to search the case base. If the learning agent has fixed computational resources, it may not be able to store enough cases to successfully perform a complex behaviour.

- **Characterizing State Transitions**: The presented retrieval algorithm is able to learn from multi-state experts but it does not remember what information was necessary to determine the state the expert was in. This information would be useful to help characterize the behaviour of the expert and improve subsequent retrievals.

- **Mixed-initiative Case Acquisition from a State-based Expert**: When mixed-initiative case acquisition is used to learn from a state-based expert, the learning agent acquires a case related to the current input-action pair as well as cases from the expert's run. In some situations, the expert can provide the portion of its run that contains all necessary information to infer its state. However, most experts will be unable to provide such precise information and

the learning agent will only be given a predetermined number of cases. These cases may not contain all necessary information (too few cases) or they may contain unnecessary information (too many cases).

- **Learning Non-deterministic Behaviour**: The agent is able to learn state-based behaviour but does not attempt to replicate non-deterministic behaviour. As we have seen, some experts can perform both state-based and non-deterministic behaviour so ideally the learning agent should be able to handle both types of behaviours. This would involve learning probability distributions that approximate the non-deterministic behaviour of the expert and attempting to perform actions using those distributions when appropriate.

- **Life-long Learning**: Existing learning by observation research has looked at learning single behaviours or learning over a short interval of time. Future work should examine how an agent can learn over its entire lifetime. This would include how the agent responds to changing behavioural requirements, transfers observations of one task to another task, and adjusts to changes in its sensory and motor control abilities.

# List of References


[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59. IOS Press, 1994.

[2] M. H. Ang, W. Lin, and S.-Y. Lim. A walk-through programmed robot for welding in shipyards. *Industrial Robot: An International Journal*, 26(5):377–388. Emerald Group Publishing, 1999.

[3] S. Asiimwe, S. Craw, B. Taylor, and N. Wiratunga. Case authoring: From textual reports to knowledge-rich cases. In *7th International Conference on Case-Based Reasoning*, pages 179–193. Springer, 2007.

[4] P. Bach-y-Rita and S. W. Kercel. Sensory substitution and the human-machine interface. *Trends in Cognitive Sciences*, 7(12):541–546. Cell Press, 2003.

[5] S. Carberry. Techniques for plan recognition. *User Modeling and User-Adapted Interaction*, 11(1-2):31–48. Kluwer Academic Publishers, 2001.

[6] P.-A. Champin, Y. Prié, and A. Mille. MUSETTE: Modeling USEs and Tasks for Tracing Experience. In *Workshop From Structured Cases to Unstructured Problem Solving Episodes For Experience-Based Assistance: 5th International Conference on Case-Based Reasoning*, pages 279–286, 2003.

[7] A. Coates, P. Abbeel, and A. Y. Ng. Learning for control from multiple demonstrations. In *25th International Conference on Machine Learning*, pages 144–151. ACM Press, 2008.

[8] M. Constantine-Paton and M. I. Law. Eye-specific termination bands in tecta of three-eyed frogs. *Science*, 202(4368):639–641. AAAS, 1978.

[9] E. de Barros Costa, M. A. Lopes, and E. Ferneda. Mathema: A learning environment based on a multi-agent architecture. In *12th Brazilian Symposium on Artificial Intelligence*, pages 141–150. Springer, 1995.

[10] M. P. Deisenroth and K. K. Krishnan. On-line programming. In S. Y. Nof, editor, *Handbook of Industrial Robotics*, pages 337–352. John Wiley and Sons, 1999.

[11] B. Díaz-Agudo, P. A. González-Calero, J. A. Recio-García, and A. A. Sánchez-Ruiz-Granados. Building CBR systems with jCOLIBRI. *Science of Computer Programming*, 69(1-3):68–75. Elsevier, 2007.

[12] J. Dinerstein, P. K. Egbert, D. Ventura, and M. Goodrich. Demonstration-based behavior programming for embodied virtual agents. *Computational Intelligence*, 24(4):235–256. Wiley, 2008.

[13] R. Doumat, E. Egyed-Zsigmond, and J.-M. Pinon. User trace-based recommendation system for a digital archive. In *18th International Conference on Case-Based Reasoning*, pages 360–374. Springer, 2010.

[14] S. Flinter and M. T. Keane. On the automatic generation of cases libraries by chunking chess games. In *1st International Conference on Case-Based Reasoning*, pages 421–430. Springer, 1995.

[15] M. W. Floyd, A. Davoust, and B. Esfandiari. Considerations for real-time spatially-aware case-based reasoning: A case study in robotic soccer imitation. In *9th European Conference on Case-Based Reasoning*, pages 195–209. Springer, 2008.

[16] M. W. Floyd and B. Esfandiari. Building learning by observation agents using jLOAF. In *Workshop on Case-Based Reasoning for Computer Games: 19th International Conference on Case-Based Reasoning*, pages 37–41, 2011.

[17] M. W. Floyd and B. Esfandiari. A case-based reasoning framework for developing agents using learning by observation. In *23rd IEEE International Conference on Tools with Artificial Intelligence*, pages 531–538. IEEE Press, 2011.

[18] M. W. Floyd, B. Esfandiari, and K. Lam. A case-based reasoning approach to imitating RoboCup players. In *21st International Florida Artificial Intelligence Research Society Conference*, pages 251–256. AAAI Press, 2008.

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addision-Wesley, 1995.

[20] M. R. Genesereth, N. Love, and B. Pell. General Game Playing: Overview of the AAAI Competition. *AI Magazine*, 26(2):62–72. AAAI Press, 2005.

[21] K. Gillespie, J. Karneeb, S. Lee-Urban, and H. Muñoz-Avila. Imitating inscrutable enemies: Learning from stochastic policy observation, retrieval and reuse. In *18th International Conference on Case-Based Reasoning*, pages 126–140. Springer, 2010.

[22] P. P. Gómez-Martín, D. Llansó, M. A. Gómez-Martín, S. Ontañón, and A. Ram. MMPM: A generic platform for case-based planning research. In *Workshop on Case-Based Reasoning for Computer Games: 18th International Conference on Case-Based Reasoning*, pages 45–54, 2010.

[23] D. H. Grollman and O. C. Jenkins. Dogged learning for robots. In *24th IEEE International Conference on Robotics and Automation*, pages 2483–2488. IEEE Press, 2007.

[24] D. H. Grollman and O. C. Jenkins. Learning robot soccer skills from demonstration. In *6th IEEE International Conference on Development and Learning*. IEEE Press, 2007.

[25] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1):10–18. ACM Press, 2009.

[26] M. A. Hearst. Trends & controversies: Mixed-initiative interaction. *IEEE Intelligent Systems*, 14(5):14–23. IEEE Press, 1999.

[27] E. Horvitz. Principles of mixed-initiative user interfaces. In *18th Conference on Human Factors in Computing Systems*, pages 159–166. ACM Press, 1999.

[28] R. Hu, S. J. Delany, and B. M. Namee. EGAL: Exploration guided active learning for TCBR. In *18th International Conference on Case-Based Reasoning*, pages 156–170. Springer, 2010.

[29] iRobot Corporation. http://www.irobot.com, 2012.

[30] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4(1):237–285. 1996.

[31] P. Maes and R. Kozierok. Learning interface agents. In *11th National Conference on Artificial Intelligence*, pages 459–465. AAAI Press, 1993.

[32] F. J. Martín and E. Plaza. Ceaseless case-based reasoning. In *7th European Conference on Case-Based Reasoning*, pages 287–301. Springer, 2004.

[33] S. Massie, S. Craw, and N. Wiratunga. Complexity-guided case discovery for case based reasoning. In *20th National Conference on Artificial Intelligence*, pages 216–221. AAAI Press, 2005.

[34] D. McSherry. Automating case selection in the construction of a case library. *Knowledge-Based Systems*, 13(2-3):133–140. Elsevier, 2000.

[35] M. Mehta, S. Ontañón, T. Amundsen, and A. Ram. Authoring behaviors for games using learning from demonstration. In *Workshop on Case-Based Reasoning for Computer Games: 8th International Conference on Case-Based Reasoning*, 2009.

[36] A. Mille. From case-based reasoning to traces-based reasoning. *Annual Reviews in Control*, 30(2):223–232. Elsevier, 2006.

[37] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[38] S. Ontañón. Case acquisition strategies for case-based reasoning in real-time strategy games. In *25th Florida Artificial Intelligence Research Society Conference*, pages 335–340. AAAI Press, 2012.

[39] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram. Case-based planning and execution for real-time strategy games. In *7th International Conference on Case-Based Reasoning*, pages 164–178. Springer, 2007.

[40] S. Ontañón, J. L. Montaña, and A. Gonzalez. Towards a unified framework for learning from observation. In *Workshop on Agents Learning Interactively from Human Teachers: 22nd International Joint Conference on Artificial Intelligence*, 2011.

[41] S. Ontañón and A. Ram. Case-based reasoning and user-generated AI for real-time strategy games. In P. A. Gonzáles-Calero and M. A. Gomez-Martín, editors, *Artificial Intelligence for Computer Games*, pages 103–124. Springer-Verlag, 2011.

[42] J. Powell, M. Molineaux, and D. W. Aha. Active and interactive discovery of goal selection knowledge. In *24th International Florida Artificial Intelligence Research Society Conference*, pages 413–418. AAAI Press, 2011.

[43] J. H. Powell and J. D. Hastings. An empirical evaluation of automated knowledge discovery in a complex domain. In *Workshop on Heuristic Search, Memory Based Heuristics and their Applications: Twenty-First National Conference on Artificial Intelligence*, 2006.

[44] J. H. Powell, B. M. Hauff, and J. D. Hastings. Evaluating the effectiveness of exploration and accumulated experience in automatic case elicitation. In *6th International Conference on Case-Based Reasoning*, pages 397–407. Springer, 2005.

[45] RoboCup. Robocup official site. http://www.robocup.org, 2012.

[46] H. Romdhane and L. Lamontagne. Forgetting reinforced cases. In *9th European Conference on Case-Based Reasoning*, pages 474–486. Springer, 2008.

[47] H. Romdhane and L. Lamontagne. Reinforcement of local pattern cases for playing Tetris. In *21st Florida Artificial Intelligence Research Society Conference*, pages 263–268. AAAI Press, 2008.

[48] J. Rubin and I. Watson. Similarity-based retrieval and solution re-use policies in the game of Texas hold'em. In *18th International Conference on Case-Based Reasoning*, pages 465–479. Springer, 2010.

[49] J. Rubin and I. Watson. On combining decisions from multiple expert imitators for performance. In *22nd International Joint Conference on Artificial Intelligence*, pages 344–349. AAAI Press, 2011.

[50] J. Rubin and I. Watson. Successful performance via decision generalisation in no limit Texas hold'em. In *19th International Conference on Case-Based Reasoning*, pages 467–481. Springer, 2011.

[51] M. Sànchez-Marrè, U. Cortés, M. Martínez, J. Comas, and I. Rodríguez-Roda. An approach for temporal case-based reasoning: Episode-based reasoning. In *6th International Conference on Case-Based Reasoning*, pages 465–476. Springer, 2005.

[52] J. C. Schlimmer and L. A. Hermens. Software agents: Completing patterns and constructing user interfaces. *Journal of Artificial Intelligence Research*, 1:61–89. AAAI Press, 1993.

[53] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden markov model structure for information extraction. In *Workshop on Machine Learning for Information Extraction at the 16th National Conference on Artificial Intelligence*, pages 37–42, 1999.

[54] J. Shih. Sequential instance-based learning for planning in the context of an imperfect information game. In *4th International Conference on Case-Based Reasoning*, pages 483–501. Springer, 2001.

[55] A. Stahl and T. Roth-Berghofer. Rapid prototyping of CBR applications with the open source tool myCBR. In *9th European Conference on Case-Based Reasoning*, pages 615–629. Springer, 2008.

[56] P. Stone, P. Riley, and M. M. Veloso. The CMUnited-99 champion simulator team. In *RoboCup*, pages 35–48. Springer, 1999.

[57] C. Thurau and C. Bauckhage. Combining self organizing maps and multilayer perceptrons to learn bot-behavior for a commercial game. In *4th International Conference on Intelligent Games and Simulation*, pages 119–126. EUROSIS, 2003.

[58] M. Wooldridge. *An introduction to multiagent systems.* John Wiley and Sons, 2002.

[59] C. Yang, B. Farley, and R. Orchard. Automated case creation and management for diagnostic CBR systems. *Applied Intelligence*, 28(1):17–28. Springer, 2008.