An Active Approach to Automatic Case Generation

Michael W. Floyd and Babak Esfandiari

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

Abstract. When learning by observing an expert, cases can be automatically generated in an inexpensive manner. However, since this is a passive method of learning the observer has no control over which problems are solved and this can result in case bases that do not contain a representative distribution of the problem space. In order to overcome this we present a method to incorporate active learning with *learning by observation*. Problems that are not covered by the current case base are automatically detected, during runtime or by examining secondary case bases, and presented to an expert to be solved. However, we show that these problems can not be presented to the expert individually but need to be part of a sequence of problems. Creating this sequence of cases is non-trivial, and an approach to creating such sequences is described. Experimental results, in the domain of simulated soccer, show our approach to be useful not only for increasing the problem coverage of the case base but also in creating cases with rare solutions.

1 Introduction

In case-based reasoning (CBR), the solutions to novel problems are determined using the solutions of previously encountered problems. These previously encountered cases are crucial to the problem solving ability of CBR systems, so it is important that cases be of a high quality and representative of the entire problem space. The initial set of cases used by a CBR system is typically provided by an expert, either manually authored or transfered in another manner. However, having an expert manually author cases can be an expensive task and requires the expert to be able to encode their knowledge in case form. Another approach to transferring knowledge from an expert into cases is to have a system that *learns from observation*.

In existing CBR systems that learn from observing an expert [1,2,3] cases are generated by remembering how the expert behaves (its outputs) in response to sensory stimuli (its inputs). A limitation of such a passive learning approach is that the learnt cases are directly related to the observed behaviour of the expert. If the expert does not encounter specific problems while being observed

L. McGinty and D.C. Wilson (Eds.): ICCBR 2009, LNAI 5650, pp. 150–164, 2009.

then there will be no cases created related to those problems. Even if the expert is observed for an extended period of time, or on multiple occasions, there is no guarantee that a representative sample of the entire problem space will be encountered. If there is no way to directly interact with the expert, like asking for a specific problem to be solved, then there can exist areas of the problem space that are not represented in the case base.

We examine a hybrid approach that incorporates active learning in order to explore areas of the problem space that are not represented in a case base that was created using passive learning. When a problem is identified that is not sufficiently similar to any existing case in the case base, that problem is artificially[1] presented to the expert. The resulting actions of the expert can then be assumed to be the solution to that problem and a new case can be added to the case base.

A flaw with this simple approach to active case learning is that it assumes that the solution provided by the expert is only dependant on the currently encountered problem. If a temporal link exists between problems [4], such that a set of problems can be ordered based on the time they are encountered, then the solution to a problem can instead be a function of the current problem as well as several previous problems. Such a situation can occur when the expert maintains an internal model of the world. If problems are not presented to the expert in the proper order then the world model may not be properly built, resulting in the expert reacting differently than if the problems had been presented in the correct order.

In order to overcome this, we look to estimate a set of problems that were likely to have occurred before the problem of interest. Initially, the case from the case base that is most similar to the problem is found. A series of problems that connect the similar case to the problem of interest are then created. This is done by performing a series of alterations to the case such that each new intermediate problem is slightly more similar to the problem of interest, compared to the previously created intermediate problem. The entire series of problems can then be sequentially presented to the expert. As an added benefit, the solutions to these intermediate problems will also be determined and can therefore be added to the case base.

The remainder of this paper will present an approach for actively acquiring cases when cases are learnt by observing a teacher. In Section 2, work related to automatic case generation and learning from observation is presented. Section 3 deals with capturing an expert's behaviour in cases. Approaches for identifying problems that may be of interest and actively acquiring their solutions is discussed in Section 4. Next, Section 5 presents a method to generate a sequence of intermediate cases that link two cases. Section 6 details experimental results and Section 7 provides conclusions and directions for future work.

[1] In a simulated environment this would involve altering the input messages sent to the expert. In a physical environment it would require altering the sensory inputs of the expert, like with a virtual reality system.

2 Related Work

In our previous work, we demonstrated how a soccer playing agent can learn, using CBR, by observing the behaviour of another agent [1]. While we examined how the case base can be preprocessed to select representative cases [5], the cases were created in a passive manner. Similarly, Romdhane and Lamontagne [2] have used case-based reasoning to teach an agent how to play the game of Tetris by observing an experienced player. In a real-time strategy domain, Ontañón et al. [3] build cases for use in case-based planning by watching a human. Flinter and Keane [6] automatically extract cases from logs of grandmaster chess matches, so their CBR system attempts to play chess like the grandmaster would. Much like our work, these approaches all collect cases in a passive manner so the CBR systems have no control over the problem space covered by the resulting case base.

Learning from observation has also been explored using learning methods other than case-based reasoning. Atkeson and Schaal [7] present a method for teaching a robotic arm to rotate and balance a pendulum by watching a human perform the task. Similarly, Coates et al. [8] teach a robotic helicopter aerobatic manoeuvres by having a human control the robot during a series of demonstrations. A common experimental result in these works is that while the robots are able to perform parts of the learnt tasks well, there exist parts that are difficult to perform. If the learners were able to actively produce more data, in these problem areas, they might be able to improve their ability to perform those tasks. Grollman and Jenkins [9] attempt to overcome this by allowing a soccer playing robot to be simultaneously controlled by a human and an autonomous system. If the human is actively controlling the robot the autonomous system learns by comparing what it would have done to what the human actually did. The autonomous system controls the robot otherwise. While this allows for active learning, the autonomous system is not able to automatically detect areas that require further learning and requires an expert to initiate the learning process.

In Yang et al. [10], aviation maintenance cases are generated in an automated manner from a pair of data sources. Text reports, from technicians, as well as computer generated fault messages are mined for data and combined to create cases. Their method places a significant importance on automatically extracting information from text, as does Asiimwe et al. [11] where cases are extracted from reports about home upgrades that help people deal with disabilities. Automatic Case Elicitation (ACE), which has been applied to checkers [12] and chess [13], uses reinforcement learning to rate automatically created cases. Solutions, in the form of which game piece should be moved, are randomly generated and applied to the current game board and a case is created. Any cases created during a winning game gain positive reinforcement, whereas losing games have their cases reinforcement values decreased. This approach allows a measurement of the usefulness of generated cases but requires a way to determine if a case resulted in a positive outcome, which may not always be easily determined.

Active learning has been used in case-based reasoning using measures of complexity [14] and coverage [15]. These approaches are successful in identifying

areas of the case base that are poorly covered but only present individual problems to the expert to solve which may not be applicable if the expert reasons using information from a series of past problems. The method we use to create a series of connecting cases is similar to the idea of adaptation paths [16]. Adaptation paths are used to create a series of slightly different problems in order to transform the initial problem into a problem with a known solution. These adaptation paths are not presented to the expert to be solved, but are instead intermediate steps used to trace out the logic used during adaptation.

3 Modelling an Expert's Behaviour

When using a case-based reasoning system to learn from observation, the goal is to determine how the expert behaves in response to the state of the environment. A case, C, can then be defined as a tuple containing the sensory stimulus, S, received by the expert and the corresponding actions, A, performed by the expert.

$$C = (S, A)$$

During a period of observation a series of N cases will be learnt from the expert, with a temporal relationship existing between these cases. Since each case represents the expert's stimulus at a moment in time, and subsequently performed actions, the ith case will have been observed before the $(i+1)$th case.

Our existing representation of a case, however, assumes the expert behaves in a purely reactive manner. The actions of the expert are only considered to be a function of the current stimulus, $A_i = f(S_i)$, so no information about the preceding stimuli is included. If the expert does not simply react to the current stimulus but maintains an internal model of the world then the cases will not contain all of the information that the expert reasons with. Thus, the actions of the expert are actually a function of the current stimulus as well as the k previously encountered stimuli ($A_i = f(S_i, S_{i-1}, \ldots, S_{i-k})$). If these previously encountered stimuli are changed, but the current stimulus remains the same, then the actions performed by the expert may change.

This introduces the need to have the cases ordered and for a case to be aware of the cases that precede it. The definition of a case can then be extended to include a timestamp, T, that is used to provide a temporal ordering to the cases. This allows a case to identify and use information from preceding cases.

$$C = (S, A, T)$$

An alternate approach, which would remove the need for a timestamp, would be to have each case include the k preceding cases. While such an approach would encapsulate all of the information needed for reasoning in a single case, it requires knowing how many preceding stimuli are required. If the observer has no information about the expert being watched it will not know how many previous stimuli should be included in each case. For example, a purely reactive

expert would not require any previous stimuli whereas experts who maintain world models would. Using a timestamp allows for a more dynamic approach, and reduces duplication of information, since it is possible to go back (or forward) any number of cases.

4 Improving Passive Learning with Active Case Generation

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 stimuli that are sensed by the expert and the expert performs actions that influence the environment. As shown in Figure 1, the observer can then view and learn from these interactions. There is no direct interaction between the observer and expert and the expert may not even be aware it is being watched.

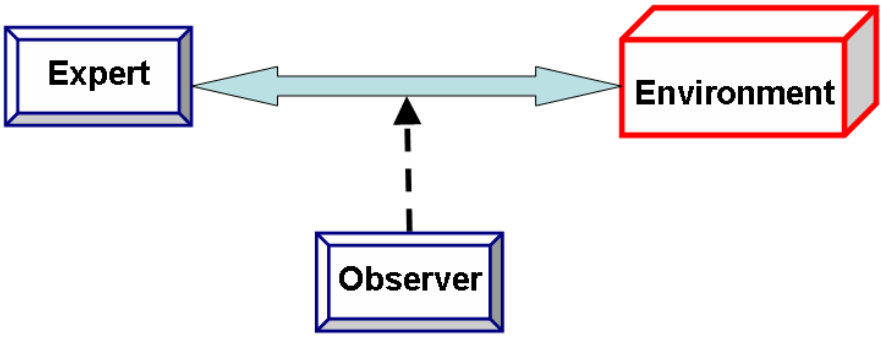

Fig. 1. Passive learning by observing an expert

As was mentioned in the previous section, the observer will create a series of cases while watching the expert. These cases will capture the interactions between the expert and the environment over a period of time. Thus, only the expert and the environment have any control over the cases that are produced. The environment controls which stimuli are contained in the cases and the expert controls the actions. In such a passive approach, even if the observer watches for an extended period of time there is no guarantee that every possible action will be performed or that a representative sample of the possible stimuli will have been sensed.

Ideally, we want the observer to be able to examine the cases it has learnt and identify areas of the problem space that it should explore further. When interesting problems that are not represented in the case base are identified, active learning can be used to complement the passive learning process. During

active learning, the observer can present problems to the expert to solve, thereby gaining a level of control over the contents of the cases. It should be noted that although active learning gives more control over the problems that are solved it is more invasive than passive learning and, as we will see in the next section, requires more computations. Active learning, therefore, will be used as a secondary learning method with the majority of the learning being done using passive learning.

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:** After the observer has learnt a number of cases, it can then use those cases to attempt to imitate the behaviour of the expert. During runtime, the observer will receive a stimulus from its own environment and search its case base for cases with similar stimuli. The actions from these cases are then used to determine an action for the observer to perform. If no cases in the case base are similar enough to the input stimulus, then the observer may not select the correct action to perform. For this approach, during each case base search if no case, C_i, has a similarity to the input stimulus, I, above a threshold, T, then the input stimulus is logged so it can be solved using active learning ($\forall C_i, sim(I, C_i) < T$). This threshold value will influence the number of stimuli that are used for active learning, with higher threshold values resulting in more stimuli being logged.
- **Secondary Case Base:** When learning from 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 stimuli, so it may not be appropriate to have cases from two different experts in a single case base[2]. 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, a primary and secondary, cases from the secondary case base can be compared to those in the primary case base. Similarly to the runtime approach, any cases that have no similar cases in the primary case base can be logged for active learning. Secondly, if the secondary case base has cases with actions that are rare or non-existent in the primary case base those cases can be logged as well. While there is no guarantee that the problems in these cases will result in the expert performing those rare actions, after active learning, it does help guide the search for cases with rare solutions.

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

[2] For example, if one expert is a defender and the other is an forward on a soccer team. The observer may only want to behave in an defensive manner.

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 stimuli, 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.

5 Determining a Connecting Sequence

When a problem is identified for active learning using the techniques described in the previous section, it must be presented to the expert to be solved. The most direct approach would be to present each problem to the expert individually. However, as was described in Section 3, the expert might maintain a world model based on previously encountered problems. Only presenting the expert with a single problem may result in different behaviour than if the expert was given that problem as part of a sequence.

In order to ensure that the expert is able to solve the problem in the proper context, by building a world model before encountering the problem, it becomes necessary to determine a series of problems to present to the expert before the problem of interest. However, for a given problem, a series of preceding problems may not be known. We look to determine these unknown preceding problems using the following method:

1. For a problem, P_1, find the most similar case, C, in the case base.
2. Extract the problem, P_2, from C.

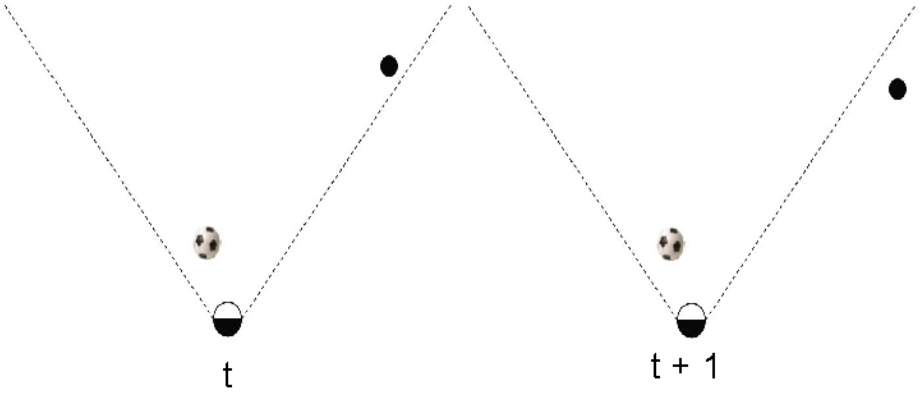

Fig. 2. Changing number of objects visible in the expert's field of vision

3. Determine a series of connecting problems, L, such that the stimuli change by no more than α between the ith and $(i+1)th$ problems in L. The α value represents the percentage of change between the stimuli. For example, the position of an object would change by at most α percent between problems.

The goal of this is to minimize the number of connecting problems that must be created, by starting with a problem that is similar to the final problem, and to gradually change the stimuli. Stimuli are changed gradually so there are no sudden large changes in what the expert senses. For example, if the visual stimulus received by the expert changed drastically it might appear like visible objects are suddenly changing locations.

We further decompose the stimulus received by the expert, S, into a collection of individual stimuli. For example, these stimuli might be visible objects, sounds, touch, or other sensory inputs. We define each stimulus, S_i, to be composed of k_i sub-stimuli ($S_i = \{s_{i1}, \ldots, s_{ik_i}\}$). Each stimulus potentially having a different number of sub-stimuli is due to the fact that the expert may not have a complete view of the environment. The expert will only sense a subset of the possible stimuli, with the remaining stimuli being unknown to the expert. For example, in Figure 2 we can see that objects can move out of (or into) the experts field of vision, thereby changing the number of stimuli the expert can sense.

In Algorithm 1, we describe how a series of connecting problems can be created that link together start and end problems. Each stimulus, s_s, from the start problem is matched[3] with a stimulus, s_e, from the end problem. The start stimulus is then modified, by a maximum of α, to be more like the end stimulus. This modified stimulus, s_c, is then added to the connecting problem that is currently being constructed, P_c. If P_c is equal to the end problem, the algorithm terminates. Otherwise, P_c is added to the series of connecting problems and is then used recursively as the next start problem.

Unlike when problems are randomly created, the connecting problems will be guaranteed to have a valid number of stimuli of each type. This is because the number of stimuli will be bound between the number in the start problem and the number in the end problem. For example, if P_s contained *5* stimuli and P_e contained *2* stimuli then all connection problems would contain between *2 and 5* stimuli. However, no testing is performed to ensure the validity of the relations between the stimuli. For example, all flags must be a fixed distance apart from each other but the algorithm never tests to ensure this it true in connecting cases. Future work will involve identifying these rules and forcing connecting cases to follow them.

6 Experimental Results

Due to the possibility that the start and end problems can have different numbers of stimuli, two situations can arise. First, if there are more start stimuli than end stimuli then not all start stimuli will have a match. The *modify* function will

[3] A detailed description of how stimuli can be matched is described in [1].

Algorithm: L = connectors(P_s, P_e, α)
Data: start problem P_s, end problem P_e, maximum change α
Result: the series of connecting problems L
$L = \{\emptyset\}$;
$P_c = \{\emptyset\}$;
$M = P_e$;
foreach $s_s \in P_s$ **do**
  $s_e = match(s_s, M)$;
  M -= s_e;
  $s_c = modify(s_s, s_e, \alpha)$;
  P_c += s_c;
end
foreach $s_e \in M$ **do**
  $s_c = modify(\emptyset, s_e, \alpha)$;
  P_c += s_c;
end
if $P_c == P_e$ **then**
  return $\emptyset$;
else
  $L = P_c + connectors(P_c, P_e, \alpha)$;
  return L;
end

Algorithm 1. Determine a series of connecting problems

attempt to remove those stimuli. An example would be to move the position of an object outside the field of vision so it can no longer be seen. Second, when there are more end stimuli than start stimuli it means stimuli needed to be added. This could involve introducing an object at the boundary of the field of vision.

The experiments we perform will attempt to answer the following questions:

- Are there certain experts that require problems to be presented in a specific sequence or can they be presented in a random order?
- Does estimating a series of preceding problems, and presenting that series along with the problem of interest to the expert, help in determining the correct solution?
- Can secondary case bases be mined in order to identify problems that may result in rare solutions?

6.1 Experimental Setup

The domain we use is simulated RoboCup soccer [17]. In the RoboCup Simulation League, the environment contains objects that belong to a fixed number of *object types*. Although each individual object on the field is unique, an agent is often unable to distinguish between objects of the same type due to noise. For example, the agent would be able to see a teammate but might not be able to tell what specific teammate it is. Additionally, the agent may not care which

specific object it is but only what type of object it is. For these reasons, objects of the same type are treated as interchangeable. In the RoboCup Simulation League we define the following object types:

$$Type = \{Ball, Goalnet, Flag, Line, Teammate, Opponent, Unknownplayer\}$$

The stimuli contained in a case are then a collection of objects that are within the expert's field of vision (as shown in Figure 3), and their location relative to the expert. The expert can then perform an action: kick, dash, or turn.

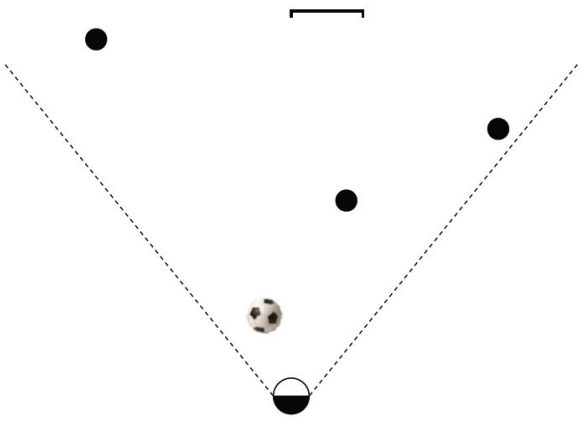

Fig. 3. Field of vision of a soccer playing agent

The expert that will be observed is a player from the CMUnited[4] soccer team [18]. CMUnited are the former champions of the RoboCup Simulation League and use a layered learning architecture and a number of strategies including formation strategies. CMUnited players can have multiple states of behaviour and maintain internal models of the world, making them a good candidate to experiment on.

Data was generated by watching the CMUnited team playing against a very simple opposing team[5], with each team composed of 11 players. Cases were generated by observing complete soccer games, with a total of 25 complete games being observed.

In the RoboCup Simulation League, the players connect to a server that maintains the model of the environment and acts as a referee for the game. The server sends the agent messages that contain information on the objects the agent can

[4] The standard CMUnited source code was modified slightly. The default version of the code would stop functioning if unexpected inputs were given, for example if stimuli were given in a random order.

[5] The team used was Krislet [19] who simply chase the ball around the field and kick it toward the opponents goal.

currently see, and the agent can then send messages to the server about the action it wishes to perform. For the active learning process, we create a fake server for the agents to connect to. The fake server can then send messages to the agent related to the problems that are to be solved, and the resulting messages from the agent can then be logged.

Since each problem used in these experiments will be extracted from a pre-existing case, there will also be a known solution to those problems in the case. The known solution will be compared to the solution generated with active learning and the f-measure will be used to measure the performance. We define the f-measure, F, for a single action type, i, as:

$$F_i = \frac{2 \times precision_i \times recall_i}{precision_i + recall_i} \tag{1}$$

with

$$precision_i = \frac{c_i}{t_i} \tag{2}$$

and

$$recall_i = \frac{c_i}{n_i} \tag{3}$$

In the above equations, c_i is the number of times the known action and generated action matched, t_i is the total number of times the action was generated and n_i is the number of times the action should have been generated. The f-measure takes into account how accurately an action is selected (the recall) as well as if when the action is selected it is selected correctly (the precision). The global f-measure, combining the f-measures for all A actions, is:

$$F_{global} = \frac{1}{A} \sum_{i=1}^{A} F_i \tag{4}$$

6.2 Importance of Problem Order

Thus far, the assumption has been made that some experts will only solve problems correctly when given a sequence of problems, not just an individual problem. In order to validate this we modify the ordering of problems presented to the expert during active learning and examine the affects. Two variations of the ordering were examined: a random ordering and the original ordering. The case bases, for each of the 25 complete games, were presented to the expert using each of the orderings with the results presented in Table 1.

We can see from the results that there is a large performance difference between using the original ordering and using a random ordering. Using the original ordering, which maintains the temporal ordering of problems, performs significantly better. This verifies our assumption that the CMUnited team relies on past stimuli to maintain an internal world model. In fact, the CMUnited agent would often output warning messages when the random ordering was used since the stimuli it was receiving was changing so drastically. One item of note is that

even when the original ordering was used there were still situations where the expert responded with a different solution than the known solution. A likely reason for this is that the expert has some degree of randomness in its action selection process or relies on other stimuli which are not encoded in the cases.

Table 1. Comparison of results

		Precision			Recall		
	f-measure	**dash**	**turn**	**kick**	**dash**	**turn**	**kick**
Original	0.82 (+/- 0.04)	0.85	0.77	0.81	0.89	0.75	0.85
Random	0.54 (+/- 0.11)	0.62	0.44	0.56	0.56	0.42	0.65
With Connecting	0.80 (+/- 0.03)	0.86	0.77	0.75	0.78	0.79	0.85
Without Connecting	0.68 (+/- 0.06)	0.77	0.72	0.58	0.68	0.68	0.67

6.3 Applying Active Learning

These experiments aim to demonstrate the benefit of estimating a series of preceding problems and presenting those problems to the expert before the problem of interest. A case is selected at random from among the CMUnited cases, and a sequence of 20 cases is extracted from the case base such that the randomly selected case is at the end of the sequence[6]. A randomly sized sequence of preceding cases, between 1 and 10 cases, were then removed from the sequence. This left the sequence containing the first 9 to 18 cases from the original sequence as well as the last case. The last case and the case that now preceded it were then used to estimate a sequence of problems that connect them. The problems portions of these two cases were used as input to Algorithm 1, with a value of $\alpha = 5\%$, in order to generate connecting problems.

Active learning was then used on the sequence, both with the connecting problems included in the sequence and without. Each run of the experiments extracted 100 sequences, and the runs were performed 25 times. The results can be found in Table 1.

With these result, we can see a distinct improvement in performance when connecting problems are created and included in the sequences used in active learning. This is likely due to the fact that the connecting problems help to gradually change the stimuli presented to the expert, rather than cause a large instantaneous change. When connecting problems are used there is only a slight decrease in performance (and not a statistically significant difference using a t-test with p=0.05) compared to presenting problems to the expert in the original order they were observed. This tells us that the connecting problems are accurate estimates of the actual problems that preceded a case. We can also see that not using connecting problems performed poorly, but not as poorly as presenting problems in a random order. This is because part of the sequence is preserved, but a gap exists in the sequence that can cause stimuli to change drastically. When the results were examined in more detail, it was found that the method

[6] If there are not that many cases before it, that case is not used.

that did not use connecting sequences performed better when fewer cases were removed from the original sequence due to the smaller gap that was created.

As a second set of experiments, we look to examine other case bases to identify problems with specific solutions. In previous work in learning by observing a RoboCup player [1,5], the solution space was found to be highly imbalanced (approximately 67.8% dash, 32.1% turn and 0.1% kick). Using passive learning, there would be significantly more *dash* actions compared to *kick* actions. This solution imbalance resulted in difficulty correctly selecting these rare actions. Thus, being able to generate more cases that have these rare solutions might be useful for increasing the performance of the CBR system. Such a solution imbalance could also occur depending on the circumstances under which the expert is observed. For example, if a soccer playing agent is observed playing a game against a far superior opponent it may never get the opportunity to perform certain behaviours like kicking the ball.

Case bases that were created by observing another soccer agent, called Krislet [19], are examined and all cases that have the *kick* action are extracted. While we cannot use these cases directly, since Krislet plays soccer differently than CMUnited and may react differently to stimuli, the problems in these cases can provide a good starting point for active learning. The method for determining connecting case, described in Section 5, was used on on the problems in these cases (using $\alpha = 5\%$) and then the resulting sequences of problems are presented to the CMUnited agent. In total, cases from Krislet playing 25 full games of soccer were examined and 883 cases were found that had the *kick* action. Of those, active learning found CMUnited produced the *kick* action in 634 of the problems. Comparatively, selecting 883 cases at random only resulted in CMUnited performing the *kick* action 67 times. We can see that using such a targeted approach helps guide the search for problems with rare solutions.

Identifying these rare actions does not guarantee an improvement in the performance but is likely to improve performance is some situations. In an extreme situation where there were no *kick* actions in the case base, the f-measure for the kick action would be *0* since no test cases would ever be properly classified as *kicks*. By actively obtaining cases with a *kick* action, it would then be possible to properly classify some of the *kick* test cases and thereby increase the f-measure results. This can be thought of as a sampling method, since it helps increase the number of cases for a specific class of action.

7 Conclusions and Future Work

Our work has attempted to address the limited control over the problems in a case base when cases are obtained through passive learning, specifically when learning is done by observation. We present an approach that incorporates active learning by identifying problems that are not represented in the case base, either during runtime or by examining other case bases, and presenting those problems to an expert to solve. This approach aims to make the active learning as nonintrusive as possible by making the expert think the problems, made up

of sensory stimuli, are coming from the environment and not from an outside source.

Our results show that problems can not be presented to an expert individually. Instead problems must be provided in the proper context, by giving the expert a sequence of problems, so that the expert can properly build a world model before attempting to solve the problem of interest. This is a result of the inherent temporal link between cases that are learnt from observation. The approach described, where two problems are linked with a series of connecting problems, was found to produce solutions which are highly similar to the expected solutions. Additionally, we show how mining a related case base can be used to identify problems that have solutions which are rare. This technique can be used to help balance the distribution of the solution space by boosting the number of occurrences of rare solutions. Our results show that even when a set of cases that represent a complete soccer game are presented to the expert, in the correct order, that the solution accuracy is not perfect. This could be due to stimuli that are not included in a case, like inter-agent communication, or an amount of randomness in the action selection process.

While this work has shown the benefit of active learning using a series of connecting problems in a simulated soccer domain, the results could be applicable in a variety of domains. Providing the proper context for a problem, in the form of a series of preceding problems, would be applicable when learning from any expert that uses previously solved problems to maintain a world model. Although the limited scope of our experiments do not allow us to draw broader conclusions about the performance benefit of our techniques, our results are promising and future work will examine other domains and non-classification tasks. Another area of interest is examining if there are certain problems that can produce multiple solutions and if the cause of this can be identified. If there exist highly similar problems that have different solutions this might indicate that other information, like previous problems, are being used during reasoning. Also, future work will look at if examining the influence of problem ordering can be used to measure the complexity of an expert's reasoning process in order to determine how long a sequence needs to be given to the expert.

Full results, data sets, sourcecode and videos related to this work are available online[7].

References

1. Floyd, M.W., Esfandiari, B., Lam, K.: A case-based reasoning approach to imitating RoboCup players. In: Twenty-First International Florida Artificial Intelligence Research Society Conference, pp. 251–256 (2008)
2. Romdhane, H., Lamontagne, L.: Reinforcement of local pattern cases for playing Tetris. In: Twenty-First International Florida Artificial Intelligence Research Society Conference, pp. 263–268 (2008)

[7] http://rcscene.sf.net

3. Ontañón, S., Mishra, K., Sugandh, N., Ram, A.: Case-based planning and execution for real-time strategy games. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS, vol. 4626, pp. 164–178. Springer, Heidelberg (2007)
4. Jære, M.D., Aamodt, A., Skalle, P.: Representing temporal knowledge for case-based prediction. In: Craw, S., Preece, A.D. (eds.) ECCBR 2002. LNCS, vol. 2416, pp. 174–188. Springer, Heidelberg (2002)
5. Floyd, M.W., Davoust, A., Esfandiari, B.: Considerations for real-time spatially-aware case-based reasoning: A case study in robotic soccer imitation. In: Althoff, K.-D., Bergmann, R., Minor, M., Hanft, A. (eds.) ECCBR 2008. LNCS, vol. 5239, pp. 195–209. Springer, Heidelberg (2008)
6. Flinter, S., Keane, M.T.: On the automatic generation of cases libraries by chunking chess games. In: 1st International Conference on Case-Based Reasoning, pp. 421–430 (1995)
7. Atkeson, C.G., Schaal, S.: Robot learning from demonstration. In: 14th International Conference on Machine Learning, pp. 12–20 (1997)
8. Coates, A., Abbeel, P., Ng, A.Y.: Learning for control from multiple demonstrations. In: 25th International Conference on Machine Learning, pp. 144–151 (2008)
9. Grollman, D.H., Jenkins, O.C.: Sparse incremental learning for interactive robot control policy estimation. In: IEEE International Conference on Robotics and Automation, pp. 3315–3320 (2008)
10. Yang, C., Farley, B., Orchard, R.: Automated case creation and management for diagnostic CBR systems. Applied Intelligence 28(1), 17–28 (2008)
11. Asiimwe, S., Craw, S., Taylor, B., Wiratunga, N.: Case authoring: From textual reports to knowledge-rich cases. In: Weber, R.O., Richter, M.M. (eds.) ICCBR 2007. LNCS, vol. 4626, pp. 179–193. Springer, Heidelberg (2007)
12. Powell, J.H., Hauff, B.M., Hastings, J.D.: Evaluating the effectiveness of exploration and accumulated experience in automatic case elicitation. In: Muñoz-Ávila, H., Ricci, F. (eds.) ICCBR 2005. LNCS, vol. 3620, pp. 397–407. Springer, Heidelberg (2005)
13. Powell, J.H., Hastings, J.D.: 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)
14. Massie, S., Craw, S., Wiratunga, N.: Complexity-guided case discovery for case based reasoning. In: The Twentieth National Conference on Artificial Intelligence, pp. 216–221 (2005)
15. McSherry, D.: Automating case selection in the construction of a case library. Knowledge-Based Systems 13(2-3), 133–140 (2000)
16. Lieber, J., d'Aquin, M., Badra, F., Napoli, A.: Modeling adaptation of breast cancer treatment decision protocols in the kasimir project. Applied Intelligence 28(3), 261–274 (2008)
17. RoboCup: Robocup official site (2009), `http://www.robocup.org`
18. Stone, P., Riley, P., Veloso, M.M.: The CMUnited-99 champion simulator team. In: Veloso, M.M., Pagello, E., Kitano, H. (eds.) RoboCup 1999. LNCS, vol. 1856, pp. 35–48. Springer, Heidelberg (2000)
19. Langner, K.: The Krislet Java Client (1999), `http://www.ida.liu.se/~frehe/RoboCup/Libs`