Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference

A Comparison of Case Acquisition Strategies for Learning from Observations of State-Based Experts

Santiago Ontañón
Drexel University
Philadelphia, PA, USA
santi@cs.drexel.edu

Michael W. Floyd
Carleton University
Ottawa, Ontario, Canada
mfloyd@sce.carleton.ca

Abstract

This paper focuses on case acquisition strategies in the context of Case-based Learning from Observation (CBLfO). In Learning from Observation (LfO), a system learns behaviors by observing an expert rather than being explicitly programmed. Specifically, we focus on the problem of learning behaviors from experts that reason using internal state information, that is, information that can not be directly observed. The unobservability of this state information means that the behaviors can not be represented by a simple perception-to-action mapping. We propose a new case acquisition strategy called *Similarity-based Chunking*, and compare it with existing strategies to address this problem. Additionally, since standard classification accuracy in predicting the expert's actions is known to be a poor measure for evaluating LfO systems, we propose a new evaluation procedure based on two complementary metrics: behavior performance and similarity with the expert.

Introduction

Learning from Observation (LfO) (Ontañón, Montaña, and Gonzalez 2011) is a machine learning paradigm that aims at automatically learning behaviors via unobtrusive observation of an actor or expert performing those behaviors. Work on LfO in the literature is often referred to by a number of largely synonymous terms such as behavioral cloning, imitation learning, apprenticeship learning or learning from demonstration (with the small difference that in learning from demonstration it is assumed that the expert actively collaborates in the demonstration process).

In this paper we discuss Case-based Learning from Observation (CBLfO) (Floyd, Esfandiari, and Lam 2008), i.e. approaches to LfO that use case-based reasoning. CBLfO can be decomposed into two primary stages: *case acquisition* and *deployment*. In the first stage, case acquisition, cases are learned by observing an expert and in the second stage, deployment, those cases are used by a CBR system to replicate the observed behavior of the expert. In this paper we focus on the first stage and specifically on the problem of learning cases for behaviors that require the expert to reason with unobservable internal information. We would like to emphasize that our focus is restricted to case acquisition strategies that are only based on observations and not those that require the supplemental interaction with the expert during learning (Floyd and Esfandiari 2011b).

In many application domains, the behaviors that we are interested in learning require the expert to reason using internal state information. This means that the behavior of the expert cannot be represented by a regular perception to action mapping (e.g. a policy), since the internal state of the expert (which might depend, for example, on past events or actions that are not in the current perceptual state) also influences the choice of action. For example, consider the example of trying to learn car driving behavior from observation. The driving speed exhibited by the driver depends on the last speed limit sign seen. Even when there is no speed limit sign in sight, the driver remembers the last one seen and maintains such speed.

This paper presents two main contributions to case acquisition in CBLfO. First, we present a new case acquisition strategy designed to learn behaviors that require reasoning with internal state information and compare it with existing case acquisition strategies. Second, we show that classification accuracy, a common metric in the LfO literature, is not enough to assess the performance of LfO algorithms. We propose an evaluation procedure that makes explicit the strengths and weaknesses of the LfO algorithm by taking into account both the performance of the resulting agent and how similar to the expert the exhibited behavior is.

The remainder of this paper is structured as follows. First, we introduce our application domain, real-time strategy games. We then introduce the necessary background on CBLfO, followed by a description of the different case acquisition strategies used in our study. After that, we present an empirical evaluation of the different strategies. The paper closes with a discussion of the existing related work, as well as concluding remarks and future research directions.

Real-time Strategy Games

Real-time Strategy (RTS) games are complex domains, offering a significant challenge to both humans and artificial intelligence (AI). Designing AI techniques that can play RTS games is a challenging problem because RTS games have huge decision and state spaces, they are nondeterministic, partially observable and real-time (Ontañón

et al. 2010; Aha, Molineaux, and Ponsen 2005). Moreover, RTS games require spatial and temporal reasoning, resource management, adversarial planning, uncertainty management and opponent modelling (Buro 2003). The size of the state and decision space, and the fact that the games are real-time (leaving very little time for decision making), makes standard game tree search approaches inapplicable (although they have been tried (Chung, Buro, and Schaeffer 2005)).

The most common approach is to hard-code human strategies (for example McCoy and Mateas (2008)). The main difficulties here is that humans combine a collection of different strategies at many different levels of abstraction, and it is not obvious how to make all of those strategies interact in a unified integrated architecture.

Given the complexity of such hard-coded strategies, automatic learning techniques have been proposed. However, they cannot cope with the complexity of the full game and have to be applied to specific sub-tasks in the game or to an abstracted version of the game. For example, Aha, Molineaux, and Ponsen (2005) use case-based reasoning to learn to select between a repertoire of predefined strategies in different game situations. Another approach is reinforcement learning (Marthi et al. 2005; Sharma et al. 2007), although it only works for small instances of RTS games with a limited number of units to control (single digits).

Learning from observation approaches have the advantage that they do not have to explore the whole state space of the game. For example, Könik and Laird (2006) study how to learn goal-subgoal decompositions by learning from human demonstrations. Weber and Mateas (2009) use learning from demonstration to predict the opponent's strategy. The work presented in this paper is a natural continuation of the work of Floyd, Esfandiari, and Lam (2008), Ontañón et al. (2010), and Lamontagne, Rugamba, and Mineau (2012).

Case-based Learning from Observation

Case-based Learning from Observation (CBLfO) approaches the LfO problem by capturing the behavior of the expert in a series of cases, which can later be retrieved and reused for replicating the expert's behavior. Specifically, in LfO, the observed behavior is represented as a set of *demonstrations* or *learning traces*, from which to learn.

A learning trace consists of a list of triples $D = [\langle t_1, S_1, a_1\rangle, ..., \langle t_n, S_n, a_n\rangle]$, where each triple contains a time stamp t_i, game state S_i, and an action a_i. The action a_i is the action executed by the expert at time t_i in response to game state S_i.

We will consider a game state S to be represented as a feature vector, and an action a by a name and a set of parameters. For example, the action $a = Build(U_0, \text{“}Barracks\text{”}, (23, 18))$ represents an action commanding unit U_0 to build a building of type “$Barracks$” at position $(23, 18)$.

The operation of a Case-based Learning from Observation agent proceeds in two stages:

1. *Case Acquisition*: the agent analyzes a set of demonstrations $\{D_1, ..., D_m\}$ and generates a case base from them.
2. *Deployment*: the agent uses CBR in order to perform a task (in our situation, play an RTS game) using the case base previously acquired from the demonstrations.

In this paper, we will primarily focus on the case acquisition phase, although case acquisition techniques have implications on how the deployment stage is performed.

We will represent a *case* as a pair $\langle S, P\rangle$, where S is a game state and P is a plan. A case represents the fact that the plan P was demonstrated by the expert as the correct thing to do at game state S. The solution of a case (the plan) can either be a single action or a complex plan. The case acquisition strategies discussed in this paper only use plans that are, at their most complex, a series of actions but systems using more sophisticated plans also exist (Ontañón 2012).

Case Acquisition

Case acquisition is the problem of generating a set of cases $\{\langle S_1, P_1\rangle, ..., \langle S_m, P_m\rangle\}$ from a set of learning traces $D_1, ..., D_n$, and is one of the most important processes in CBLfO since the performance of the system strongly depends on the quality of the cases in the case base.

This section first presents three existing case acquisition strategies (*Reactive Learning*, *Monolithic Sequential Learning*, and *Temporal Backtracking*), and then introduces a new strategy called *Similarity-based Chunking* that is specifically designed to address the problem of learning behaviors from experts who reason with internal states. Other case acquisition strategies from the literature include hierarchical or dependency-graph algorithms (Ontañón 2012). However, those algorithms exploit additional domain knowledge, which we do not assume is available to the LfO agent.

Reactive Learning

Given an expert trace, which contains a sequence of triplets, the *reactive learning* strategy (RL) learns one case per entry in the trace. Thus, from a trace consisting of n entries:

$$[\langle t_1, S_1, a_1\rangle, ..., \langle t_n, S_n, a_n\rangle]$$

it will learn the following n cases:

$$\{\langle S_1, a_1\rangle, ..., \langle S_n, a_n\rangle\}$$

In this strategy, the solution portion of the case is a plan that only contains a single action. This case acquisition strategy was introduced by Floyd, Esfandiari, and Lam (2008). The top-left part of Figure 1 shows an example of the cases created from an expert trace consisting of 11 actions.

Monolithic Sequential Learning

As we will show in the empirical evaluation, one of the issues of the previous strategy is that it is purely reactive and the resulting CBR system has problems properly sequencing actions. Notice that this is expected, since the one piece of information that is lost in the reactive learning strategy is the precise order of the actions. The *monolithic sequential learning* strategy (MSL) takes the completely opposite approach, given an expert demonstration with n actions, it learns a single case:

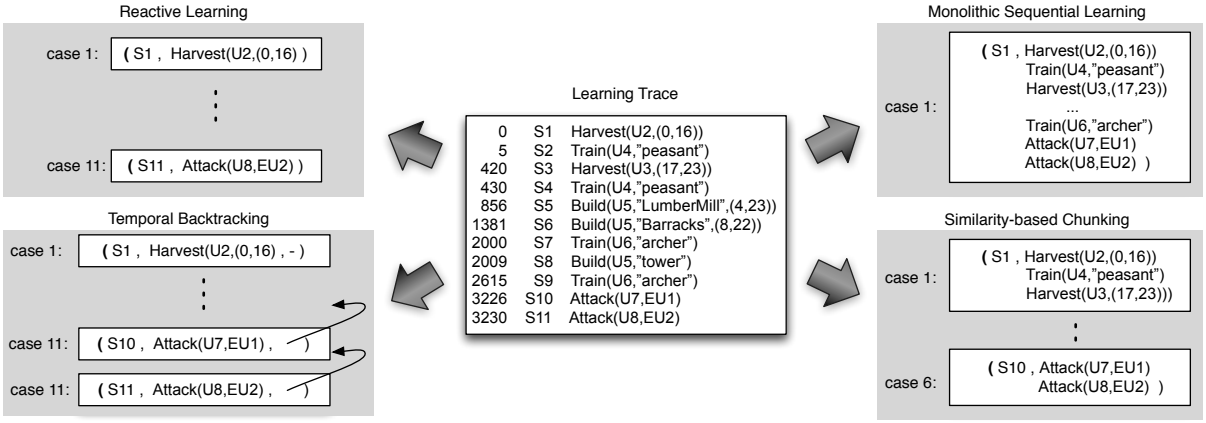

Figure 1: A visual comparison of the four case acquisition strategies used in our experiments

$$\langle S_1, sequence(a_1, ..., a_n) \rangle$$

where $sequence(a_1, ..., a_n)$ represents a sequential plan where all the actions are executed in exactly the same order as the expert executed them. Thus, this strategy learns a single case per demonstration. Figure 1 shows an example of this learning strategy (top-right). As we will show below, the problem with this strategy is that once a case has been retrieved, actions will be executed in a fixed sequence, without any reactivity to the dynamics of the domain.

Temporal Backtracking Learning

Temporal Backtracking Learning (TBL) is an improvement over the basic Reactive Learning strategy by extending the case definition to preserve the order in which the actions occurred while maintaining reactivity. Specifically, given a sequence of cases learned using the Reactive Learning strategy from a single demonstration ($C_1 = \langle S_1, a_1 \rangle, ..., C_n = \langle S_n, a_n \rangle$), Temporal-Backtracking annotates each case with a link to the case that preceded it, resulting in the following set of cases (as illustrated in the bottom-left of Figure 1):

$$\begin{aligned} C_1 &= \langle S_1, P_1, - \rangle, \\ C_2 &= \langle S_2, P_2, C_1 \rangle, \\ &... \\ C_n &= \langle S_n, P_n, C_{n-1} \rangle \end{aligned}$$

When using cases defined in this manner, case retrieval is modified in the following way. When we need to retrieve a case given a query, we first retrieve all the cases more similar than a certain threshold k_1 to the query. Then, we evaluate whether the actions in the set of retrieved cases are different from one another. If all the retrieved cases predict the same action, then that action is used, otherwise, *temporal backtracking* is performed. Temporal backtracking uses the temporal links between cases (the third element in the case triple) to get more information, in the form of previously encountered game states and performed actions, to discriminate between retrieved cases. It starts by going one time step back and comparing the similarity between the action in the previous case with the past action of the query. Only those retrieved cases with a similarity above another threshold k_2 are kept. If there is still disagreement between the actions predicted, then we compare the game states in the previous cases (keeping only cases with similarity above k_3). If there was still a disagreement, then we would move one more step back in time (using thresholds k_2 and k_3 for past action similarity and past state similarity respectively), and so on, until we cannot go back in time anymore, or until the actions in the remaining set of cases are all equivalent (or more similar than a given threshold). See Floyd and Esfandiari (2011a) for exact details of the retrieval algorithm.

Similarity-based Chunking Learning

As we will show in the experimental evaluation section, while Temporal Backtracking Learning is a significant improvement over Reactive Learning, it still sometimes fails to account for some sequencing of actions which causes its performance during game play to be drastically reduced. *Similarity-based Chunking Learning* (SBCL) is a middle ground between Reactive Learning and Monolithic Sequential Learning, which divides the trace into contiguous *chunks*, trying to capture the key aspects of action sequencing, while maintaining some of the reactivity.

It often happens that two consecutive states in a learning trace are highly similar. This is a problem, since it means that when using a Reactive Learning strategy, the two cases generated from those two states will be hard to distinguish and action sequencing might suffer. The SBCL strategy chunks together those highly similar consecutive entries, as follows:

1. Start at the first entry of the learning trace (the triple $\langle t_1, S_1, a_1 \rangle$).
2. Mark the current entry ($\langle t_i, S_i, a_i \rangle$) as the beginning of a chunk. Move to the next entry ($\langle t_{i+1}, S_{i+1}, a_{i+1} \rangle$).
3. If the similarity of the current entry, the jth entry, to the entry at the start of the chunk is higher than a threshold k ($similarity(S_j, S_i) > k$), add the current entry to the current chunk, move to the next entry ($\langle t_{j+1}, S_{j+1}, a_{j+1} \rangle$), and repeat Step 3. Otherwise, the current chunk has ended so a new case is created. If the chunk starts at the ith entry and contains m entries, the

case will have the game state from the ith entry and a sequence of m actions ($\langle S_i, sequence(a_i, ..., a_{i+(m-1)})\rangle$). The process then returns to Step 2.

4. Repeat until the end of the trace is reached.

An illustration of this strategy can be seen at the bottom-right corner of Figure 1. Similarity-based chunking is related to the idea of sequential case acquisition via conditional entropy by Lamontagne, Rugamba, and Mineau (2012), but where actions are grouped together by the similarity of the game state, rather than by the frequency in which they appear in a sequence in demonstrations. However, they have different goals: our strategy aims at solving problems related to state-based experts, whereas the conditional entropy strategy is designed to reduce the size of the case-base while still performing similarly to the reactive learning strategy.

Experimental Evaluation

In order to compare the different case acquisition strategies, we implemented them in the *Darmok 2* (D2) system and used a real-time strategy game called S3 as our domain. We evaluated the performance of each of the case acquisition strategies along two different dimensions. First, performance was measured by counting how many games the system was able to win after learning. Second, similarity to the expert's behavior was measured by evaluating the learning system's accuracy in predicting the actions that the expert would have executed in given situations.

Darmok 2

We used the *Darmok 2* (D2) case-based reasoning system to compare the different case-acquisition strategies. D2 implements the *on-line case-based planning* cycle, a high-level framework to develop case-based planning systems that operate in real-time environments (Ontañón et al. 2010), and was designed to play real-time strategy games.

The D2 system maintains the current plan to be executed (which is initially empty). If the plan is empty, D2 retrieves a case and the plan in the retrieved case is used as the current plan. The retrieved plans might have to be adapted using transformational adaptation techniques. D2 tracks the execution status of the plan in execution, and if any part of the plan fails and cannot be adapted by the plan adaptation algorithm of D2, the plan is discarded. Each time the current plan finishes, or is discarded, D2 restarts the cycle by retrieving another case. For a more in-depth description of D2, the reader is referred to a formal definition of the system (Ontañón et al. 2009; Ontañón et al. 2010).

S3

The Darmok 2 system was used in a strategy game called *S3*, a simplified version of Warcraft. In S3, players need to collect wood (by chopping trees) and gold (by mining gold mines) in order to construct buildings and train units to defeat their opponents. To achieve those goals, there are 8 different action operators they can use (each of them with 2 to 5 parameters). We used a collection of 5 different maps with different characteristics: two maps contained several islands connected by small bridges, two maps contained walls of trees that players had to cut through in order to reach the enemy, and one map consisted of a labyrinth of trees with each player on one side. Maps in S3 are represented by a two-dimensional grid, where in each cell we can have grass, water or trees. The maps used in our evaluation had a size of 64×32 cells. S3 contains 4 built-in AIs, implementing 4 different strategies: *footman rush*, *archers rush*, *catapults rush* and *knights rush*.

	Wins	Ties	Score	Accuracy
Expert	12.0	6.0	42.0	100.0%
RL	0.0	2.4	2.4	42.3%
SML	2.6	2.8	10.6	36.4%
TBL	0.8	2.4	4.8	58.7%
SBCL	2.4	0.4	7.6	45.5%

Table 1: Game results and similarity with the expert's behavior for each case acquisition strategy

Experimental Settings

We created a fifth AI, which we call the *expert*, implementing a *defensive knights rush* strategy where a formation of defensive towers are created and then knights are sent to attack the enemy. The expert's behavior has internal state: it keeps track of how many units it has sent to attack so far, and changes it's behavior according to that. Thus, it is not possible to represent the behavior of this expert as a perception-to-action mapping.

Expert traces were generated by making this strategy play one game against each of the other AIs (including itself) in each of the maps. The expert executed an average of 134.7 actions (minimum 47 and maximum 242) per trace. Then, we selected 5 of these traces, one per map, where the expert wins for the training set. When running experiments, when a game reached 100000 cycles (over half an hour of running time), it was stopped and considered a tie.

Performance in Playing the Game Results

Table 1 shows the average number of wins, ties, and overall score (3 points for a win and 1 point for a tie) that our system obtained using the different case acquisition strategies. Each version of our system played against the 5 AIs (the 4 built-in AIs and the expert) in 5 different maps (25 games per experiment), and each experiment was repeated 5 times (total of 125 games per strategy). For the Temporal Backtracking strategy, we used used the thresholds $k_1 = 0.9$, $k_2 = 0.5$ and $k_3 = 0.7$. For Similarity-based Chunking, we used the threshold $k = 0.85$. These parameters were set via experimentation to obtain the best results.

These results clearly indicate that strategies that learn sequential actions (Sequential Monolithic and Similarity-Based Chunking) obtain a much larger number of wins than the other strategies. We believe this is due to the fact that sequencing actions in an RTS game is key to winning the game, and also because the behavior of the expert can only be learned by learning action sequencing beyond a situation-to-action mapping. Also, notice that the Sequential Monolithic Learning strategy only manages to play the game at

all thanks to the adaptation component of Darmok 2, which can adapt the actions in the retrieved case to the situation at hand. Otherwise, blindly replicating the actions of the expert in a different game would not result in any meaningful play (this is true for all strategies, but particularly for this one).

However, these results do not indicate whether the system behaves similar to the expert or not. They don't indicate whether the system was successful in learning from observation or not, only if the agent was able to achieve the primary goal of the expert (winning the game). Additionally, it may not be possible for a learning agent to have a clearly defined goal to measure its success unless that goal is explicitly given by a domain expert.

Similarity with Expert's Behavior Results

Table 1 also shows the accuracy with which our system could predict the actions executed by the expert using each case acquisition strategy, measured using a leave-one-out strategy. The system learned from all but one trace and tried to predict the actions for each game state in the remaining trace. Moreover, given that the actions in RTS games are complex, we only measured the accuracy with which the system predicted the type of the action (whether it is an *attack* action, a *harvest* action, etc.) regardless of the parameters (there are 8 different types of actions).

Table 1 shows that even though the Sequential Monolithic Learning strategy performed better in the game, it actually does a poor job in predicting the actions executed by the expert. This was expected since this strategy blindly repeats the sequence of actions in the retrieved case, which might not correspond to what the expert would do in the situation at hand. The strategy that better replicated the behavior of the expert is the Temporal Backtracking Learning strategy. Similarity-based Chunking outperforms Sequential Monolithic Learning and Reactive Learning, but does not reach the level of Temporal Backtracking Learning. Figure 2 shows a visualization of how the classification accuracy of the system varies as the game progresses. The horizontal axis represents time, measured as number of actions performed. Reactive keeps the same average accuracy throughout, while we can see that SML performs very well at the beginning of the game, but drops very quickly. TBL keeps a better accuracy throughout the game. SBCL shows a similar trend to SML, but maintaining a higher accuracy for longer.

Analyzing the results, we see that the metrics used in our evaluation (performance and similarity with expert) rank the strategies differently, since they analyze different features of the behavior of the system. According to performance (using the score metric) we obtained: SML $>$ SBCL $>$ TBL $>$ RL. According to similarity (using the average accuracy), we obtained: TBL $>$ SBCL $>$ RL $>$ SML. All the differences were found to be statistically significant using a paired t-test with $p < 0.05$.

Both metrics have been extensively used in the literature, and we argue that it is very important to use both of them in LfO work in order to get a complete picture of the performance of the LfO algorithms. Depending on the application task, one metric might be more important than the other. Thus, the only conclusive results in our evaluation is that our new technique, SBCL, performs better than RL in both metrics. However, if we use their average rank, with a rank of 1 being the best and 4 the worst, the best case acquisition strategies are SBCL (average rank of 2) and TBL (average rank of 2), followed by SML (average rank of 2.5) and RL (average rank of 3.5).

Related Work

Previous Case-based Learning from Observation systems, as we have discussed in earlier sections, are ill-suited for learning from experts with internal states because they learn policies to map the expert's current perception to an action. Learning from observation systems that use other learning approaches (Grollman and Jenkins 2007; Thurau and Bauckhage 2003; Coates, Abbeel, and Ng 2008) suffer the same limitations because they attempt to learn similar mapping policies.

The need for using sequences of problems and solutions during reasoning has been explored in other CBR research such as case-based plan recognition (Kerkez and Cox 2003). Martín and Plaza (2004) note that reasoning often can not be performed at a single point in time but requires reasoning over a period of time. It was found that reasoning with a sequence of past inputs allows inputs that are otherwise similar to be differentiated (Shih 2001).

Approaches that do take into account sequences of past actions have been called trace-based reasoning (Mille 2006) and episode-based reasoning (Sànchez-Marrè et al. 2005). 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 (Doumat, Egyed-Zsigmond, and Pinon 2010) or a fixed-length sequence of both inputs and actions (Champin, Prié, and Mille 2003; Sànchez-Marrè et al. 2005). While these approaches could be used to learn from experts who have internal states, the fact that they make use of fixed-length sequences during retrieval limits their applicability to learning from observation.

Conclusions and Future Work

This paper has focused on case acquisition strategies for Case-based LfO systems that are designed to learn state-based behaviors. We have introduced a new case acquisition strategy, Similarity-based Chunking, and compared it against other case acquisition strategies from the literature.

Additionally, we have paid special attention to the methodology to evaluate the performance of our system, by using two complementary metrics: performance in game play and similarity with the expert's behavior. We have shown that those two metrics analyze completely different aspects of the performance of the system, and thus, argue that both should be used when comparing LfO algorithms.

As part of our future work, we would like to develop a better understanding of the different trade-offs between the existing case acquisition strategies in domains other than RTS games. Also, we would like to explore the role that different case adaptation strategies have in the performance of Case-based LfO systems and how adaptation depends on the type

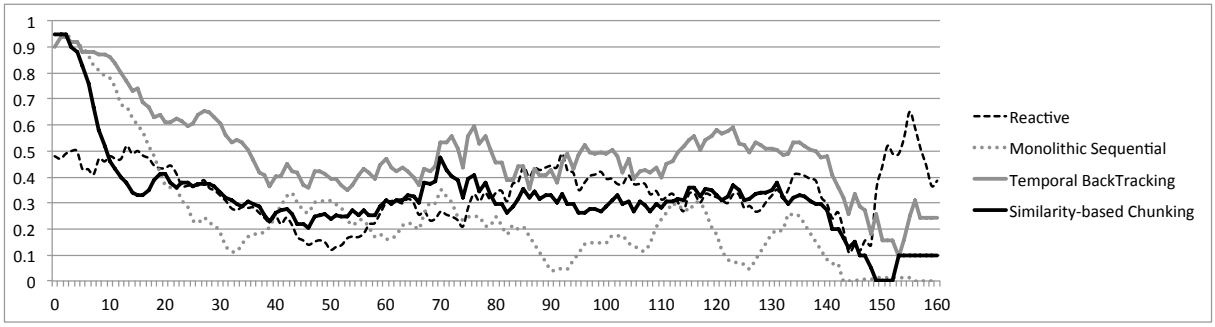

Figure 2: A visualization of the classification accuracy of the system in predicting the expert's actions over time using each of the different case acquisition techniques. The horizontal axis represents time, measured in number of actions issued.

of case acquisition technique being employed. Finally, the obtained results show that there is a significant amount of room for improvement, indicating we can still improve our case retrieval and adaptation techniques.

References

Aha, D.; Molineaux, M.; and Ponsen, M. 2005. Learning to win: Case-based plan selection in a real-time strategy game. In *6th International Conference on Case-Based Reasoning*, 5–20.

Buro, M. 2003. Real-time strategy games: A new AI research challenge. In *18th International Joint Conference on Artificial Intelligence*, 1534–1535.

Champin, P.-A.; Prié, Y.; and Mille, A. 2003. MUSETTE: Modeling USEs and Tasks for Tracing Experience. In *ICCBR Workshop From Structured Cases to Unstructured Problem Solving Episodes For Experience-Based Assistance*, 279–286.

Chung, M.; Buro, M.; and Schaeffer, J. 2005. Monte Carlo planning in RTS games. In *IEEE Symposium on Computational Intelligence and Games*.

Coates, A.; Abbeel, P.; and Ng, A. Y. 2008. Learning for control from multiple demonstrations. In *25th International Conference on Machine Learning*, 144–151.

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

Floyd, M. W., and Esfandiari, B. 2011a. Learning state-based behaviour using temporally related cases. In *16th United Kingdom Workshop on Case-Based Reasoning*, 34–45.

Floyd, M. W., and Esfandiari, B. 2011b. Supplemental case acquisition using mixed-initiative control. In *24th International Florida Artificial Intelligence Research Society Conference*, 395–400.

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

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

Kerkez, B., and Cox, M. T. 2003. Incremental case-based plan recognition with local predictions. *International Journal on Artificial Intelligence Tools* 12(04):413–463.

Könik, T., and Laird, J. E. 2006. Learning goal hierarchies from structured observations and expert annotations. *Mach. Learn.* 64(1-3):263–287.

Lamontagne, L.; Rugamba, F.; and Mineau, G. 2012. Acquisition of cases in sequential games using conditional entropy. In *ICCBR Workshop on TRUE: Traces for Reusing Users' Experience - Cases, Episodes and Stories*, 203–212.

Marthi, B.; Russell, S.; Latham, D.; and Guestrin, C. 2005. Concurrent hierarchical reinforcement learning. In *19th International Joint Conference on Artificial Intelligence*, 779–785.

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

McCoy, J., and Mateas, M. 2008. An integrated agent for playing real-time strategy games. In *23rd Conference on Artificial Intelligence*, 1313–1318.

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

Ontañón, S.; Bonnette, K.; Mahindrakar, P.; Gómez-Martín, M. A.; Long, K.; Radhakrishnan, J.; Shah, R.; and Ram, A. 2009. Learning from human demonstrations for real-time case-based planning. In *IJCAI Workshop on Learning Structural Knowledge From Observations*.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2010. On-line case-based planning. *Computational Intelligence* 26(1):84–119.

Ontañón, S.; Montaña, J. L.; and Gonzalez, A. J. 2011. Towards a unified framework for learning from observation. In *IJCAI Workshop on Agents Learning Interactively from Human Teachers*.

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

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

Sharma, M.; Homes, M.; Santamaria, J.; Irani, A.; Isbell, C.; and Ram, A. 2007. Transfer learning in real time strategy games using hybrid CBR/RL. In *20th International Joint Conference on Artificial Intelligence*, 1041–1046.

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

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

Weber, B. G., and Mateas, M. 2009. A data mining approach to strategy prediction. In *IEEE Symposium on Computational Intelligence and Games*.