Learning Opponent Strategies through First Order Induction

Katie Genter
Dept. of Computer Science
The Univ. of Texas at Austin
Austin, TX 78712 USA
katie@cs.utexas.edu

Santiago Ontañón
IIIA - CSIC
Bellaterra, Spain
santi@iiia.csic.es

Ashwin Ram
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332 USA
ashwin@cc.gatech.edu

Abstract

In a competitive game it is important to identify the opponent's strategy as quickly and accurately as possible so that an effective response can be staged. In this vain, this paper summarizes our work in exploring the use of the first order inductive learning (FOIL) algorithm for learning rules which can be used to represent opponent strategies. Specifically, we use these learned rules to perform plan recognition and classify an opponent strategy as one of multiple learned strategies. Our experiments validate this novel approach in a simple real-time strategy game.

Introduction

Consider any competitive game — professional soccer, StarCraft, poker — they all require participants to have a solid strategy to be successful. But often the best way to defeat an opponent is to identify their strategy, as then their actions can be predicted and an effective response can be staged.

One way to identity an opponent's strategy is through plan recognition (Schmidt, Sridharan, and Goodson 1978). Plan recognition is the process of inferring the plan of an intelligent agent from observations of the agent's actions or the effects of those actions. Plans can be represented in various forms. In this paper we focus on learning opponent strategies that are composed of collections of rules. Many different algorithms could be used to learn these rules (**?**), but we chose the first order inductive learning (FOIL) algorithm (Quinlan 1990) because of its ability to take in predicates and produce a rule list in which the individual rules are ranked by their Laplace accuracy.

There is previous work in both rule learning and plan recognition in the context of games (e.g. (Kabanza et al. 2010; **?**)) — but to the best of our knowledge, ours is the first to present an inductive logic programming approach to plan recognition. In particular, we input gameplay traces in which a particular strategy is exhibited into the FOIL algorithm, and we let the resulting rules represent that strategy. After learning multiple strategies, we can predict which learned strategy an opponent is following by comparing the rules for the opponent's trace against the rules for the learned strategies.

Learning Strategies through FOIL

Every time a game is played, a trace created with information from that game. At each time step in a game, we record all of the relevant features for each entity entry in the game state. Specifically, for our BattleCity example domain, we record information concerning the time, identification code, type, x position, y position, heading, and owner for each entity entry. Actions are represented by an action name and the set of parameters required by the action. In our experimental domain, there are only two possible actions: Move(player,direction) and Fire(player).

Traces are difficult to input and use by FOIL in their original form, so we translate the relevant information from each trace into a more usable form. In particular, we define a set of attributes $F = \{f_1, ..., f_n\}$ to be extracted from each entry as either true or false. In this way, each entry S_i is translated to a set of identifiers $F(S_i)$ that correspond to all of the attributes in S_i. We can then easily input these sets of identifiers for a trace into the FOIL algorithm as a modified trace $T' = [\langle t_1, F(S_1)\rangle, ..., \langle t_n, F(S_n)\rangle]$.

Using Rules to Represent Strategies

After gathering traces that demonstrate specific opponent strategies, we use FOIL to learn rules to represent each specific strategy. For this, we consider each attribute f_i separately. Given an attribute f_i and a modified trace T', a training example can be generated for each instant t_j (except for t_1). In particular, if f_j is true in $F(S_j)$, then we will create a *positive* example $e_j = \langle F(S_{j-1}), +\rangle$. Likewise, if f_j is false, then we create a *negative* example.

Using the set of training examples collected for an attribute f_i, FOIL produces a set of rules which predict whether attribute f_i will be true in the next cycle given the current game state. We consider each attribute f_i similarly until FOIL has generated a rule set for each defined attribute. Together, these rule sets for each defined attribute represent a given opponent strategy.

Identifying Opponent Strategies

We use learned strategies to predict what strategy an unknown opponent is utilizing given a trace of its actions in the world. Given a new trace for which we want to identify its strategy, we gather the positively correlated rules that characterize the new agent's movements and actions in the world

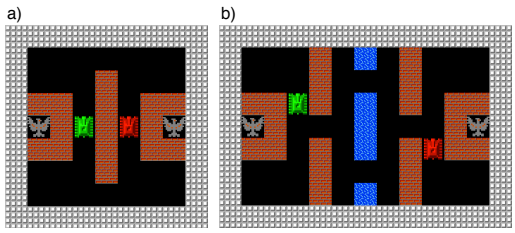

Figure 1: The two BattleCity maps used in our evaluation.

as described in the previous section. We then compare these rules to the positively correlated rules of each learned strategy and determine which learned strategy is closest to the opponent's behavior. We assign a score to represent the similarity between rules representing a known strategy and rules characterizing an unknown opponent's behavior. The strategy that earns the highest score represents the most likely strategy of the unknown opponent.

Experimental Evaluation

To evaluate our technique, we use a two-player real-time strategy game called BattleCity (see Figure 1). In this game, each player navigates their tank around a maze while trying to simultaneously destroy the opponent's tank or base and defend their own tank and base. Tanks can move in four directions and fire bullets in whichever direction the tank last moved, while bases are static. The environment is fully observable and the only sources of non-determinism in the game are the actions of the opponent.

We use two different maps in our evaluation: a simple map (Figure 1.a) and a more complex map (Figure 1.b). For each map, we recorded traces demonstrating three distinct strategies. We recorded 14 winning traces for each strategy, making a total of 42 traces per map. In each trace, a human tester controlled the tank protecting the left base, while the other player was controlled by the built-in artificial intelligence of the game. The human tester attempted to exhibit a particular playing strategy in each trace. The three strategies considered in the simple map (Figure 1.a) were:

'Right' : turn right from the start and fire through the walls towards the opponent base until it is captured.

'DownRightUp' : travel down and around the center wall, approach the opponent base from the bottom, and then fire through the wall to capture the base.

'UpRightDown' : is similar to strategy two. However, in this case travel up and around the wall, approaching the base from the top.

The three strategies considered in the complex map (Figure 1.b) were:

'Across' : position the tank across from the opponent base and fire across the water and through the wall until the base is captured.

'Down' : travel down until aligned with the lower bridge. Then fire through walls and travel across the bridge before approaching the base from the bottom. Then fire through the wall and capture the base.

'Up' : is similar to our second strategy, except now take the upper path and approach the opponent base from the top.

Map	Strategy Exhibited	Percent Correct
Simple	Right	10/10
Simple	DownRightUp	10/10
Simple	UpRightDown	9/10
Complex	Across	10/10
Complex	Down	10/10
Complex	Up	8/10

Table 1: The number of times each strategy was correctly identified on the simple map (Figure 1.a) and the complex map (Figure 1.b).

For each map, we divided the 42 traces into two sets: 12 traces (4 of each strategy) constituted the training set and 30 traces (10 of each strategy) constituted the test set. We learned rule sets for each strategy using the traces in the training set. We then used these rules to classify each of the 30 unseen traces as one of the three learned strategies. Results are found in Table 1.

It is important to note that although the human tester tried to demonstrate the same strategy in each trace recorded for a particular strategy, traces exhibiting the same strategy are often rather different due to uncertainty introduced by the opponent. For example, the enemy might block the path the tester is trying to move through, causing a delay or slight deviation from the prescribed strategy. Lastly, it is important to note that the attributes used in our evaluation were not fine tuned — in fact, the same set of attributes were used in a previous paper (XXX and YYY 2009).

Future Work

One future step is to evaluate our approach in domains that are more complicated than BattleCity. Additionally, other approaches for identifying opponent strategies should be considered. It could also be interesting to use learned strategies to predict the next move that will be made by the adversary. For this, it might be best to consider partial strategies learned from partial traces, as a partial strategy may more easily match a trace if they cover similar time steps.

References

Kabanza, F.; Bellefeuille, P.; Bisson, F.; Benaskeur, A. R.; and Irandoust, H. 2010. Opponent behavior recognition for real-time strategy games. In *Proceedings of the AAAI Workshop on Plan, Activity, and Intent Recognition (PAIR-10)*.

Quinlan, J. R. 1990. Learning logical definitions from relations. *Machine Learning* 5:239–266.

Schmidt, C.; Sridharan, N.; and Goodson, J. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1-2):45–83.

XXX, and YYY. 2009. Learning from human demonstrations for real-time case-based planning. In *IJCAI-09 Workshop on Learning Structural Knowledge From Observations*.