Planning and Learning in an Adversarial Robotic Game

Gilbert Peterson and Diane Cook

University of Texas at Arlington
Box 19015
Arlington, TX 76019-0015
{gpeterso, cook}@cse.uta.edu

Abstract[1]

This paper demonstrates the tandem use of a finite automata learning algorithm and a utility planner for an adversarial robotic domain. For many applications, robot agents need to predict the movement of objects in the environment and plan to avoid them. When the robot has no reasoning model of the object, machine learning techniques can be used to generate one. In our project, we learn a DFA model of an adversarial robot and use the automaton to predict the next move of the adversary. The robot agent plans a path to avoid the adversary at the predicted location while fulfilling the goal requirements.

Introduction

The objective of this project is to research effective methods for learning the strategies of an adversary by observation. There are two popular approaches to outperforming an opponent in an adversarial game. One is to attempt to outperform the adversary by exploring a greater number of possible moves farther into the future. The second option is to create a model of the adversary's thought processes, and then by predicting their move, to create a counter-strategy. The creation of a simple model of the adversary's strategies is the method explored in this paper. With this model, the agent then creates a plan to outperform the adversary.

The testing environment for this work is a modification of a game titled "Hunt the Wumpus." The active elements in the world are the wumpus and the agent. The agent's goal is to exit the Wumpus World having collected as many gold bars as possible, while avoiding the wumpus and bottomless pits. The adversaries in the Wumpus World are the agent and the wumpus. One robot performs the role of the agent and the second robot is the wumpus. The wumpus has a simple set of strategies to follow, randomly selected at the beginning of each game.

The agent predicts the move of the wumpus by learning a model of all the possible strategies of the wumpus. The reasoning model that the wumpus uses is a computationally bounded model. The agent can learn a finite automata representing the move selections for the strategies of the wumpus.

In addition to learning the finite automaton structure, the agent learns a probability distribution from each state to each following state based on previously seen behaviors of the wumpus. Using the probability distribution of a state, the agent predicts the most probable move the wumpus will execute next, or assigns a probability of occupation to all of the states the wumpus may move to.

The agent creates a plan of steps to reach the desired goals through the use of a utility planner. The utility planner places a utility value and reward on each state of the world. The utility value calculation is a function of the known positions of the pits, gold bars, and the wumpus. The value placed on each state of the world represents the desire of the agent to be in that state. For instance, if there is a pit in a location, it will have a negative value, because death is not something the agent wants. All of the world states have a utility value. The agent selects a move that places it in the state with the maximum utility value.

The next section of the paper discuses related work on the use of a finite automaton to learn adversarial behavior in non-zero sum games, DFA learning, and utility planning. We then introduce the Wumpus World and changes made for using the robots, followed by an explanation of the finite automaton learning process and explain the learning of an automaton as the selection for learning an adversarial strategy. The approach used for utility planning is then described, and finally, we discuss the findings.

Related Work

Learning the minimal DFA for a given set of positive and negative examples is a known NP-Hard problem (Gold, 1967). There are a number of algorithms and approaches that in polynomial time learn an approximate minimal DFA.

One approach that stems from the later work of Gold is the US-L* algorithm proposed by Carmel and Markovitch, 1995. The US-L* algorithm uses an observation table represented by (S, E, T) to represent the DFA. The observation table consists of the closed set of strings S, the tests E that are the suffix-closed set of strings, and a two-dimensional table T that maintains the output of the elements in the sets of S and E.

The RPNI algorithm (Oncina and Garcia, 1992) and the RPNI2 algorithm (Dupont, 1992) break the example set

into sets of positive and negative examples. The positive example set becomes the initial FA tree structure. The negative example set becomes a test set to reduce the positive tree to a minimal DFA. The RPNI2 algorithm differs from the RPNI algorithm in that it performs the reduction incrementally. Incremental reduction of DFA can be implemented using homing sequences (Rivest and Shapire,1989). The proof that the RPNI algorithm for simple problems is PAC (Probably Approximately Correct) learnable is in Parekh and Hanovar, 1995.

The RPNI algorithm is the algorithm used to learn the DFA representing the opponent strategies. The reason for this selection is the implementation to fit the algorithm to the domain is straightforward.

To outsmart an adversary, there are two different approaches that may be taken. The first approach is to explore more of the game space. Variations on the minimax algorithm perform this task. The second option is to learn a model of the adversaries' behavior and use that to predict their behavior.

In learning a model of the opponent, the approach assumes the adversary uses a computationally bounded model to make move decisions instead of playing and searching for the minimax optimal move. The computational bounded model being used to make decisions is most often a finite automata (Freud, et al., 1995). By learning the opponent model, the agent can gain a higher optimal performance.

Learning the opponent model is being studied by Carmel and Markovitch(1994). They use a DFA learning algorithm based on Angluins L* (Angluin, 1987). Another approach for learning the model of the DFA is by Mor, Goldman, and Rosneschein (1996), in that the agent knows how many states exist in the DFA and uses a payoff matrix to decide how states should interconnect.

The wumpus in the Wumpus World makes a move selection based on a finite automaton. This is different from many two player games, where both players are striving for the same goal. The desired outcome of the agent is to correctly model and predict the wumpus' next move and plan a solution. With a correct prediction, the agent can then create a plan to outsmart the wumpus.

The planning agent used is a utility based planner. An excellent source of discussion of reinforcement learning and the utility planner can be found in Reinforcement Learning by Richard S. Sutton and Andrew G. Barto. The the utility based planner was chosen because the Wumpus World is compartmentalized into a grid of deterministic size and elements. The only unknowns occur in dealing with the wumpus.

We chose to use robots as the wumpus and the agent for a more realistic environment than a simulator. The use of the robots also serves to increase the amount of uncertainty associated with the wumpus position.

Wumpus World

The Wumpus World domain is based on an early computer game. The basis for the game is an agent who explores an N by N grid world while avoiding a creature named the wumpus. Additional elements of the world consist of bottomless pits (which don't affect the wumpus), and bars of gold. The objective of the game is to collect as many of the gold bars as possible, return to the initial grid location [1,1] and exit the cave. The information the agent senses each turn to aid in locating the gold and avoiding the wumpus and pits, is a five element percept. If the agent is in a square containing the wumpus or directly adjacent squares the agent perceives a stench. If the agent is in a square directly adjacent to a pit it will perceive a breeze. If there is a gold bar in the same location as the agent it will perceive glitter. If the agent runs into a wall it will perceive a bump. If the agent shoots its arrow and kills the wumpus it will hear a scream. The actions allowed the agent are to move forward, turn left, turn right, grab gold, shoot the arrow, and climb out of the cave. The actions allowed the wumpus are to move forward, turn left, turn right, and do nothing.

Two robots perform in the roles of the agent and the wumpus. Because of this changes are made to the Wumpus World definition to use the robot's sonar ring instead of the five element percept. The sonar ring senses the walls of the world, the other robot, and obstacles. Obstacles inserted into the world make the task of sensing the other robot more complex. The agent and wumpus possess the locations of the pits and gold bars, because the robots' only input is the sonar ring.

The robots playing the roles of the agent and wumpus are Trilobots. The Trilobot robots have an eight element sonar ring. The sonar ring reads distance at 22.5 degree intervals. A picture of the robots in a portion of wumpus world is shown in Figure 1.

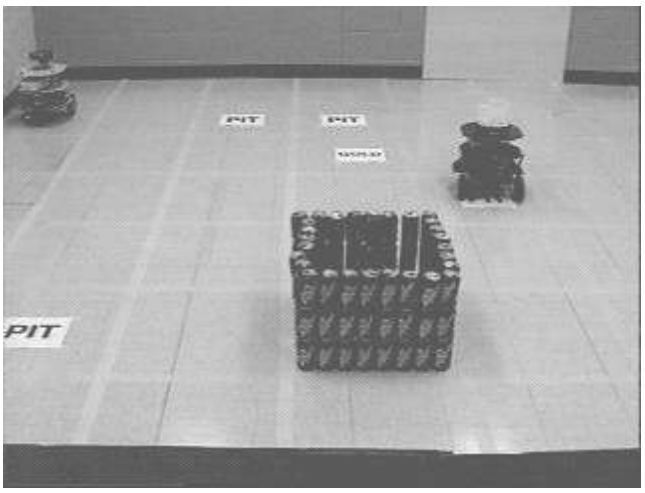

Figure 1

Learning the Wumpus Strategies

The strategies that the wumpus can employ are to 1) move to a gold bar and circle it clockwise, 2) move to a gold bar

obstacle until the Agent is close and then attack. The wumpus will only follow one of these strategies through a given game. A modified RPNI algorithm infers a DFA structure representative of the wumpus strategies given a series of sub-objectives. Each sub-objective is the fulfillment of part of the wumpus strategy. An example is that one of the wumpus strategies is to move to a gold bar and circle it clockwise. The sub-objectives that make up the strategy are the action of moving to the bar followed by circling the bar clockwise. The sub-objective also describes elements of the task such as which of the up to three gold bars the wumpus should approach.

The use of sub-objectives seems counter-intuitive. It would be logical to learn the wumpus strategies by representing the Wumpus World states as states in the automata, and the wumpus actions as the transitions. For an 8x8 world there are 64 states to store, representing all locations where the wumpus can reside. Information on the agent position would also be needed (in case the wumpus is attacking the agent), but this squares the number of possible states. All possible configurations of gold bars and obstacles need representing because they too relate to the strategies. With all this information included in the state, an immense amount of information would be represented.

Instead of representing the world state information, the sub-objectives perform this task. This creates an automaton of the repetitive elements of the wumpus strategies, and abstracts away the specifics of the Wumpus World domain. This representation reduces the amount of data to store and allows the method to scale to larger problems. The transitions from state to state instead of being a specific move, since the sub-objectives are states, are transitions of the wumpus from fulfilling one sub-objective to the state of fulfilling another.

A modified RPNI algorithm learns a DFA structure representative of the five behavior patterns given a series of sub-objectives. The sub-objectives are derived from the moves made by the wumpus as observed by the agent. The sub-objective classifications are 1)move closer to gold bar 1,2 or 3, 2) move closer to obstacle 1,2,3, 3) attack the agent, 4) hide from the agent, 5) sit or do nothing, and 6) move closer to the north wall, south wall, east wall, or west wall. Each of the wumpus strategies is a combination of several of these sub-objectives. The agent determines the move made by the wumpus by comparing the previous and current believed positions of the wumpus. The sub-objective determination begins by first, calculating relative distances between the wumpus and all the elements in the world. The relative distances compared with a rule base results in the most probable sub-objective. The agent retains the sub-objectives made from the offset of the trial in a transition list.

The algorithm chosen to learn the DFA is a modification of the RPNI algorithm (Oncia and Garcia 1992). The RPNI algorithm performs an ordered search of an accepting DFA by attempting to merge consecutive elements of the prefix set. The prefix set is a set of all of the prefixes of the acceptable strings. The first step is to create a PTA(instance+). The PTA(instance+) is a finite automata tree created from only the positive examples. Each state included in the PTA(instance+) has a corresponding element in the set of prefixes of the positive instance set. Each step of the algorithm attempts to merge a prefix of the positive instance set with possible suffixes of the PTA(instance+), while remaining consistent with the set of negative instances.

The RPNI algorithm uses a set of positive and a set of negative samples to derive a DFA based on the acceptance of states. For the wumpus world all states are acceptor states since what is being looked for is the next move, not string acceptance or rejection. Because there are no negative samples, we modify the RPNI algorithm to retract based on matching states and pattern repetition rather than on the accepting string.

The modified RPNI algorithm begins by creating a PTA(instance^{+}) just as in the original, with the instance+ set coming from the transition list. The prefix set that is incrementally searched and merged in RPNI does not carry over to the modified version. Instead of using the prefix set, the modified RPNI algorithm performs the search space from the initial node. Each path is then searched for possible loops.

For the Wumpus World the initial node is always sit. The nodes used in the DFA structure of the Wumpus World upon creation will point to themselves. This saves a step during retraction, since for the domain each sub-objective can transition to itself. The initial structure is a sit node which upon receiving a sit returns to itself. Each edge also has a probability attached. The probability represents the number of times the agent observed the wumpus make the transition.

The agent updates a probability attached to each transition as it continues to observe the behaviors of the wumpus. The agent uses these probabilities to predict the next sub-objective the wumpus will meet. An example learned automaton is in Figure 2.

Utility Driven Planning

Once the agent has learned a DFA representing the wumpus behavior patterns, the DFA predicts the next move of the wumpus and a utility update function determines a path for the agent. The agent uses a value iteration algorithm to calculate the utilities of each of the states in the wumpus domain. The value iteration algorithm prescribes a path to gather the gold while avoiding the pits and wumpus. Since the wumpus and agent move simultaneously, the value iteration algorithm runs each turn before the agent selects a move.

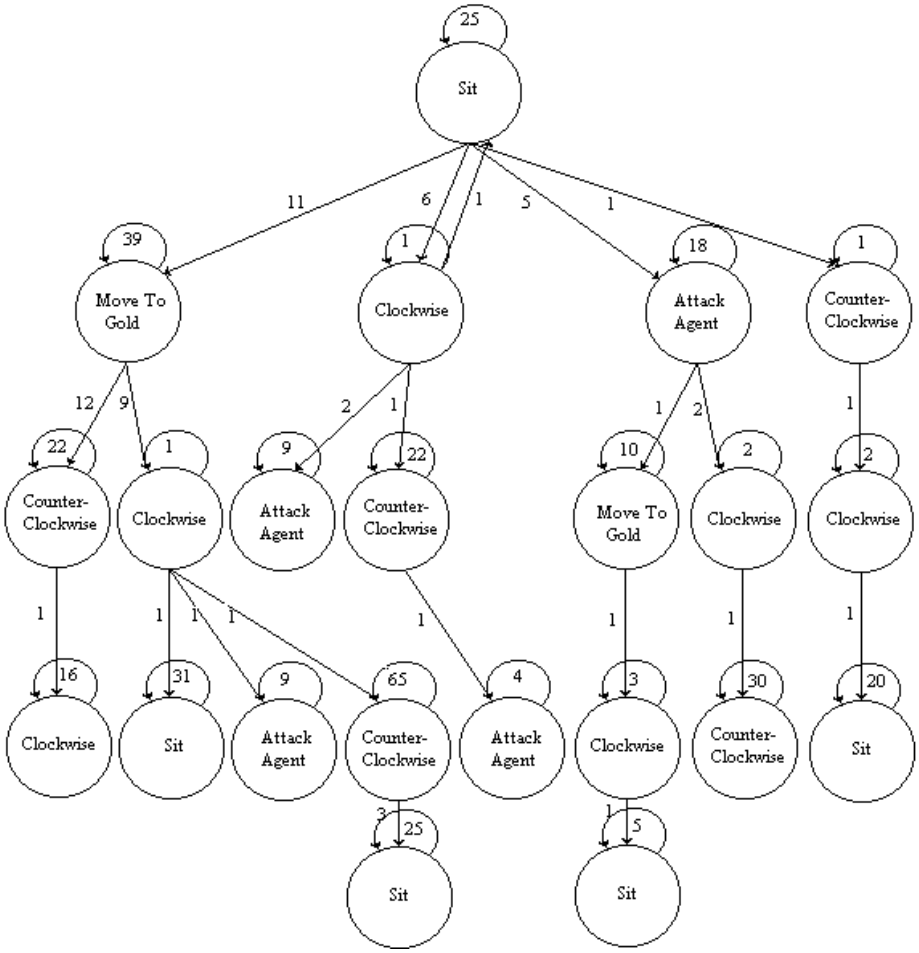

Figure 2

The value iteration algorithm formula is

$$\pi'[i] \leftarrow R[i] + \max_a \Sigma_j M^a_{ij} \pi[j].$$

The value iteration serves to update the utility values in the policy (π), and calculate a new utility for each location The new utility value for a location is the sum of the reward for the state and the maximum utility value of all the states reachable from that location in a single move. After recalculating all of the utility values, the temporary policy becomes the action policy.

The value iteration algorithm uses a reward array (R[]) <x, y, numgold>, a utility array (π[]) <x, y, transitions, numgold>, and a model (M^a_{ij}) describing the agent's moves to find a path through the wumpus domain. The reward array contains all of the rewards for all of the possible states the agent can reach. Each state in the reward array consists of an x and y location and the number of gold bars the agent has in possession at that time. Each state the agent can be in and each move the agent can make will have a separate utility value. The state information for the agent in this table as in the reward array is the x and y location, and the number of gold bars the agent possesses at the time. The model describing the agent's moves is a set of conditionals that emulate the agent in the environment. An example would be if the agent is next to and facing a wall he can not move into it, or if the agent is facing up and chooses to go forward his y location will increase by one.

Because a close-to-optimal path is desired, each location in the reward array is initialized to -0.05. If the agent knows or believes that there is a pit or wumpus in a given grid location, the reward for this location is -10.0. This keeps the value iteration algorithm from prescribing a path through a lethal location. Locations with gold receive a reward assignment of 1.0, 2.0, 3.0, and so on for as many gold bars that exist in the world. The incremental values are assigned randomly and not in a specific order to create a shortest path length. Location [1,1] is assigned a value of one more than any gold pieces, so that after the gold has been collected the agent will leave. The utility array at the beginning of each turn is reset to all zeros.

The agent will alternate between updating the utility array and testing the utilities returned to determine if a path to collect all of the gold bars and exit exists. Once a path to reach all goals exists, the utility planner will exit, and the first move of the path is executed. The first move of the path is the move that will have the highest utility for that location and possible wumpus location/s.

The agent has two modes. In the first mode, the agent can output a single sub-objective that has the maximum probability of being the sub-objective the wumpus is going to try to complete. For this case, the utility planner assigns the wumpus reward of negative twenty to the one location predicted by the automaton. Alternatively, the agent can be set to output all of the possible sub-objectives possible from the current sub-objective. The negative twenty reward of the wumpus is then distributed on all the possible world locations the wumpus could be in. The reward is distributed by normalizing the probabilities of each sub-objective and multiplying the value by negative twenty. If more than one sub-objective would have the wumpus move to the same location, the two rewards are summed.

The utility planning algorithm performs well in the small Wumpus World domain. The algorithm will scale badly to larger domains as the policy increases in size the time to a solution does also. In order to maintain the utility solution methods for a plan that we were looking for the next choice may be to implement a Bayes network or neural network using the relational distances between objects with a vector for direction to perform the planning. The other possibilities are to implement a similar utility planner with later research into controlling the size and time issues (Moore and Atkinson, 1993).

Results

The first set of experiments compares the two predictive outputs of the automaton and a case with no predictive information. The tests were run with a finite automaton trained on 25 and 50 trials. A trial consists of a randomly created world and selected wumpus strategy, where each trial consists of 25 alternating turns in which the wumpus and agent both move. If the agent dies before reaching the twenty-fifth step, the trial ends. During learning the automaton the pits can not harm the agent, and he hugs the wall of the world looking for the wumpus, observing only. The wumpus world is an 8x8 world in which object positions are randomly determined.

In testing, each trial is set to last fifty steps allowing the agent plenty of time to collect all the gold bars and return to the exit. There were five test situations. The first test set uses no automaton, and thus no predictive ability. The second uses an automaton trained for twenty-five trials with the normalized return, “Uncertain” in Table 1. The third also uses an automaton trained for twenty-five trials but with the maximum prediction return, labeled “Max” in Table 1. The fourth and fifth trials use an automaton trained for fifty trials with both of the return types, uncertain and max.

The agent earns a specific number of points for each trial. The points are -1000 for dying, -1 per step (to discourage lengthy plans), +1000 per gold bar collected, and +1000 for exiting. The points totaled for the fifty trials appear in the ”Total Points Earned” column in Table 1. These points are tallied strictly for comparison between the approaches and do not feed back into the algorithm.

The results show that the use of the finite automaton to predict the wumpus move does somewhat protect the agent from the wumpus. As a result the agent is able to stay alive longer, for more steps, and collect more gold.

The agent performs better overall in using the maximum prediction from the automaton. This is in part because the normative values which although represent the probabilities of where the wumpus could be more properly, spreads the negative reward of the wumpus over more than one state. The utility iteration algorithm is a greedy planner and will plan a path through states with a negative reward to get to states with positive rewards. This is also visible in the low average number of steps, the agent frequently would attempt to plot a path through things like pits if there was no quick way around them, in the search for a short path.

The positive reward assignment for each of the gold bars should ultimately be separated into different policies. Each policy then creates a path to one of the gold bars or the exit, and using a nearest neighbor or search to choose that policy to follow. By separating the reward, each policy would have fewer peaks and the valleys might be avoided better (Whitehead, Karlsson, and Tenenberg, 1993).

The second set of tests compares the normalized output of the learned finite automaton with the actual behavior of the wumpus. For these tests, the learning of the finite automaton occurred over fifty trials. Each individual wumpus strategy was tested fifty times. The wumpus would output the sub-objective that he follows at

Table 1: Comparison of the Predicting Methods.

Test	Average Number of Steps	Total Gold Collected	Times Killed by Wumpus	Total Points Earned
No FA	6.24	18	8	-18312
FA-25 Uncertain	8.56	23	6	-6248
FA-25 Max	8.63	25	3	1569
FA-50 Uncertain	10.28	26	3	5486
FA-50 Max	10.12	31	1	14494

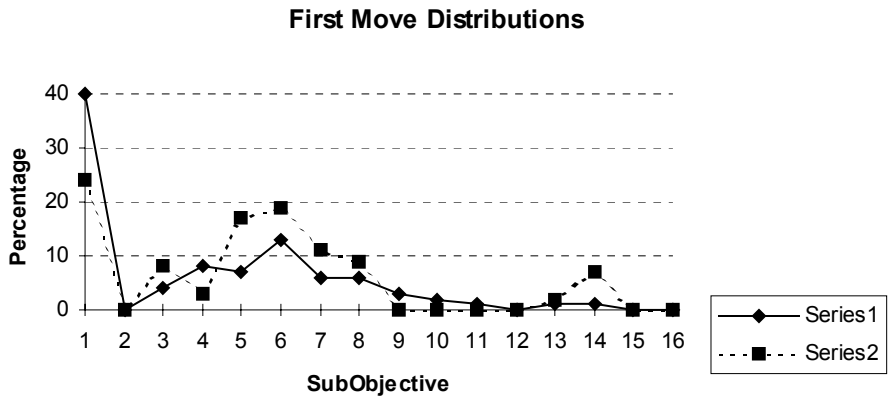

Figure 3

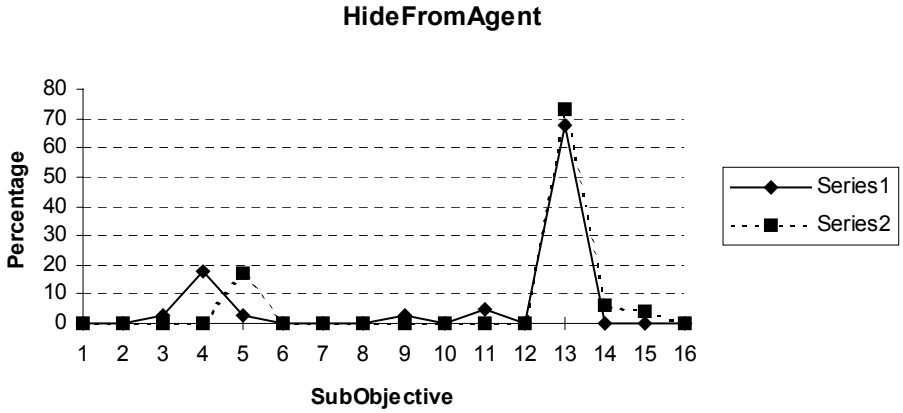

Figure 4

each step. To calculate each distribution, the output of the wumpus' transition from sub-objective to sub-objective is tallied. This is similar to the method by which the agent learned his probability distribution. The difference is the information comes from the wumpus, not the agent. For example, if the wumpus strategy was to Move to the Gold and Circle Clockwise, he might output 'move to gold 2'. For turn two, he might output 'circle the gold bar clockwise'. This would add one to the number of time the wumpus chose the sub-objective circle the gold bar clockwise from the sub-objective of 'moving to gold 2'.

The probabilities for each of the sub-objective nodes is also totaled. For each node, the sub-objective weights are normalized to probabilities so a direct comparison to the wumpus output is possible. The comparisons of the sub-objectives for the first move and the Hide From Agent strategy are in Figure 3 and Figure 4. The y-axis is the percentage of times the sub-objective on the x-axis followed the sub-objective in the title of the graph. The sub-objectives are represented as numbers 1) Sit, 2) Random, 3) Circle Gold Clockwise, 4) Circle Gold Counter Clockwise, 5) Attack Agent, 6) Move to Gold 1, 7) Move to Gold 2, 8) Move to Gold 3, 9) Move North, 10) Move South, 11) Move East, 12) Move West, 13) Hide from Agent, 14) Move to Obstacle 1, 15) Move to Obstacle 2, and 16) Move to Obstacle 3.

The legend lists the lines as Series 1, the actual distribution and Series 2, the finite automaton distribution after training for fifty trials.

The comparison graphs show that the peaks of the finite automaton sub-objectives are on the same sub-objectives as in the actual distributions. One of the reasons for differences is that the agent's viewpoint is skewed, and when the agent can not locate the wumpus, can not learn or predict his behavior. The hide sub-objective shows this best. The agent saw few transitions to the 'attacking the agent' sub-objective. This is because the agent will seek out the wumpus, and in doing so come too close and be attacked by the wumpus who is already hiding. If the agent had not seen the wumpus hiding, 'attacking the agent' represents all the information the agent perceived. Without the previous hiding information, a transition from 'hiding' to 'attacking the agent' is not incorporated into the automaton.

Another area of difference between the graphs is when the agent misclassifies the wumpus sub-objective. The method of comparing the relative distances from the wumpus with a set of rules and selecting the first match

does not always lead to a correct match. This mismatching occurs when the wumpus moves toward an area near a number of different items. Without prior knowledge of the wumpus behavior strategy, the exact sub-objective can not be determined. One extension to the rule matching method would be to assign probabilities to each of the rules fired, and select the rule that is most likely.

Conclusion

The testing done shows that by learning a model of the adversaries behavior in a finite automaton, the adversaries' behavior can be adequately predicted. By predicting the next sub-objective of the wumpus the agent is able gather more gold, stay alive longer, and get a higher score. Also by comparing the probabilities learned by the finite automaton and the wumpus behavior strategies, the distributions maintain the same probabilistic peaks. As the finite automaton is trained more, the agent performs better. Further testing is needed to train the finite automaton more in the hope that the probability distribution become closer.

The utility planner generated successful plans from the given data. However, for better performance the planner should separate the different goals into different policies and merge the utilities or resulting chosen action instead of using a single global policy. Scalability is also an issue; the utility planner fits well to the small scale of the Wumpus World may not work so well for a larger domain.

References

Angluin, D. 1987. Learning Regular Sets from Queries and Counterexamples. *Information and Computation 75*: 87-106.

Carmel, D. and Markovitch, S. 1994. Unsupervised Learning of Finite Automata: A Practical Approach. Technical Report CIS report 9504.

Carmel, D. and Markovitch, S. 1994. Learning Models of Opponent's Strategies in Game Playing. Technical Report CIS report 9318, and CIS report 9305.

Dupont, P. 1994. Incremental Regular Inference. In Miclet, L., and Higuera, C., eds*., Proceedings of the Third ICGI-96, Lecture Notes in Artificial Intelligence 1147*: 222-237. Montpellier, France: Springer.

Freud, Y., Kearns, M., Mansour, Y. Ron, D., Rubinfeld, R., and Shapire, R. E. 1995. Efficient Algorithms for Learning to Play Repeated Games Against Computationally Bounded Adversaries. *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*.

Gold, E. M. 1978. Complexity of automaton identification from given data. *Information and Control 37*(3): 302-320.

Oncina, J. and Garcia, P. 1992. Inferring Regular Languages in Polynomial Updated Time. *Pattern Recognition and Image Analysis: Selected Papers from the IVth Spanish Symposium*: 49-61.

Mor, Y., Goldman, C., and Rosenchein, J. S. 1996. Learn Your Opponent's Strategy (in Polynomial Time)!.

Moore, Andrew W., and Atkeson, Christopher G. 1993. Memory-based Reinforcement Learning: Converging with Less Data and Less Real Time. *Robot Learning*. 79-104.

Olivera, A. L. and Edwards, S. 1996. Limits of Exact Algorithms For Inference of Minimum Size Finite State Machines*. Proceedings of the Seventh International Workshop on Algorithmic Learning Theory(ALT'96)*.

Parekh, R. G. and Honavar, V. G.. 1997. Learning DFA from Simple Examples*. Proceedings of the Eighth International Workshop on Algorithmic Learning Theory (ALT'97)*, Sendai, Japan. Oct 6-8, '97 (To appear)

Rivest, R. L. and Schapire, R. E. 1989. Inference of finite automata using homing sequences. *Proceedings of the 21st ACM Symposium on Theory and Computing*: 411-420.

Russel, S. J. and Norvig, P. 1995. *Artificial Intelligence A Modern Approach*. New Jersey: Prentice Hall.

Sahota, M. K. 1994. Reactive deliberation: An architecture for real-time intelligent control in dynamic environments. *Proceedings of the 12th National Conference on Artificial Intelligence, 1303*.

Sutton, R. S. 1995. TD Models: Modeling the World at a Mixture of Time Scales. *Proceeding of the 12th International Conference on Machine Learning*: 531-539.

Sutton, R. S. and Barto, A. G. 1998. Reinforcment Learning: an introduction. MIT Press, Cambridge, Massachusetts.

Whitehead, S., Karlsson, J., and Tenenberg, J.1993. Learning Multiple Goal Behavior via Task Decomposition and Dynamic Policy Merging. *Robot Learning*. 45-78.