2021-08-06

Predicting Human Behavior in Repeated Games with Attitude Vectors

Brian L. James
Brigham Young University

Predicting Human Behavior in Repeated Games with Attitude Vectors

Brian L. James

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Jacob Crandall, Chair
Kevin Seppi
Jonathon Sillito

Department of Computer Science

Brigham Young University

ABSTRACT

Predicting Human Behavior in Repeated Games with Attitude Vectors

Brian L. James
Department of Computer Science, BYU
Master of Science

As Artificial Intelligence systems are used by human users at an increasing frequency, the need for such systems to understand and predict human behavior likewise increases. In my work, I have considered how to predict human behavior in repeated games. These repeated games can be applied as a foundation to many situations where a person may interact with an AI, In an attempt to create such a foundation, I have built a system using Attitude Vectors used in automata to predict actions based on prior actions and communications. These Attitude Vector Automata (AVA) can transform information from actions in one game with a given payoff matrix into actions in another game. Results show that prediction accuracy was ultimately below other, similar work, in general in several repeated games. There are however some aspects, such as scenarios involving lying, in which my predictor showed potential to outperform these other systems. Ultimately, there is potential in using ideas presented as AVA to build a potentially more robust system for future efforts in human behavior prediction.

Keywords: generalizing, human machine interaction, artificial intelligence, S, Attitude Vector, Attitude Vector Automata, AVA

Contents

List of Figures

1 Introduction

In an increasingly complex world, it is more important than ever to bridge the gap between the power of technology and the people who use it. Computing technology in particular adds great complexity to daily life, while simultaneously offering perhaps the greatest ability of any technology for usability. At the forefront of both of these aspects is Artificial Intelligence (AI). Through its diverse applications comes the ability to process large, complex data, from language to image processing. Such applications make performing complex tasks easier for scientists, and can similarly be leveraged to bring these and other technology to a broad spectrum of users by cleverly bridging the gap between complex mathematical foundations and human interaction. Thus the work of developing AI agents which understand and cooperate with humans is essential [2].

A helpful tool for studying and modeling interactions between two agents is game theory. In game theory, these interactions are often described as normal-form games [3] depicted as payoff matrices where the rows and columns represent the possible actions of each agent while the corresponding value pair indicates the value (or payoff) for each agent for the given action pair. This approach quantifies interactions in a manner useful for developing strategies, especially as it relates to those involving one or more AI agents.

In order to get the most use out of any strategies an AI agent may learn from humans, it is important to consider how that knowledge can be generalized. In other words, strategies observed in a given payoff matrix should help guide an agent's behavior in new situations. By implementing a means to generalize, an agent can better utilize past data in novel circumstances. This flexibility also appropriately brings the ability closer to the human ability to apply knowledge across domains.

One quite successful algorithm for working with humans has already been developed for two player matrix games and a is key inspiration in my own work: S# [1]. This algorithm uses a set of programmed experts to determine its actions and is even capable of communicating through a set of messages. Given its success, research has been done based on S# to attempt to model humans, which has led to some good results [4]. I refer to this algorithm, which models S# by predicting its most likely choice of expert to predict human plans, as MAP-S# throughout this paper. MAP-S# has been show to effectively predict human behavior in a several repeated matrix games with messages, also refered to as cheap talk, though its performance degrades considerably in scenarios in which people lie [4].

For my research, I used a data-driven approach to attempt to overcome some of the weaknesses found in MAP-S#'s predictive modeling. I set out to make an algorithm which, rather than being programmed, could be learned from training on data of humans playing matrix games. In doing this, I focused in particular on developing a system capable of better predicting dishonest players as these predictions in particular were more difficult for MAP-S# to make. Further, I wanted to make sure that strategies learned from one game could be applied to different games, even when the payoff matrix used was different.

2 Related Work

Given the importance of designing machines capable of working with others, it is unsurprising that a great deal of prior research has been done in this area. The entire field of game theory has been developed with the goal of understanding these kinds of problems. The following is but a brief overview of past and ongoing work related to human behavior modeling in repeated games. Specifically, I consider how intelligent machines have been designed to model humans and other agents.

2.1 Opponent Modeling

When working with multiple intelligent agents, the common approach is to create machines that attempt to model their associates in some way. Various approaches have been developed in an attempt to solve these opponent modeling problems. Perhaps the most direct form of this is in the area of policy reconstruction. These agents attempt to reconstruct the other agent's decision making processes through one of several means. This has been done through Fictitious Play [5], wherein average frequencies of moves are used to approximate the other agent's behaviour. This can be modified by considering the other agents' actions based on the agent's own actions as in [6]. Another approach to modeling others is through case-based modeling, wherein the agent compares its current situation with other cases and uses these to determine how to act [7]. Machine-learning techniques have also been used to model others, including deterministic finite automata [8], decision trees [9], and neural networks [10].

While making models to fit a specific opponent may lead to ultimately better results, it is not always practical to do so. In situations where a machine must quickly adapt to a new partner, one option is to fit them to previously determined types. Here again, methods including deterministic finite automata [11], decision trees [9], and neural networks [12]; [13] can be used to represent types which can be fit to the strategy used by the agent's counterpart. By including multiple strategies, AVA is in some ways similar, though the approach is not exactly as these.

Just as one would design an intelligent agent to model those it interacts with, so too would it be reasonable to believe that the agents it will interact with may also model it. The understanding that other intelligent beings have their own models to describe the world is referred to as Theory of Mind [14], which is something people develop at a young age to understand how others behave [15]. Through Theory of Mind, an agent may learn to understand the intentions of others [16]. Similar attempts to leverage this ability have included agents which recursively modeling what each being thinks the other believes until reaching some predetermined depth at which the agent assumes the other agent will act rationally [17]. As AVA is designed to predict humans, it would be a potential candidate for guiding an AI agent's own Theory of Mind.

When developing a machine meant to model people there are a few differences not typically seen in other machines that must be considered. For instance, many agents assume that others only use one model. While this works well for simple agents and may be able to approximate more complex ones, it will not necessarily cover human opponents as effectively. If a person sees that their strategy of interaction isn't working to their expectations, they will almost certainly adopt a new strategy, which is an element AVA attempts to cover by learning from multiple pflayers. This does add a layer of complexity, but a few approaches have been attempted previously to account for this. For instance, [18] created an algorithm to start by varying models until convergence is reached. [19] have constructed agents which are allowed to change between static models periodically. Meanwhile others have created algorithms to map interaction histories to models [20]; [21], while still others have adjusted models by primarily weighting recent interactions [22].

When humans interact, not only do they sometimes change strategies, they also sometimes act irrationality. Often, attempts to create intelligent systems are interested in achieving some optimal value, and behaving in any manner that does not support that goal is rendered impossible. People do not always act in their best interest, however, allowing emotions and fallacies to cause them to act in ways a designer may see as suboptimal. Still, there has been some work done to attempt to model this behavior, and for an agent to interact with people it may be useful to capture some essence of this. In their work, [23] use a satisficing algorithm for an agent to learn how to behave in the prisoner's dilemma, learning to adjust expectations based on the opponent's actions. In a variety of other works (e.g., [24]), agents are constructed to attempt to mimic human behavior

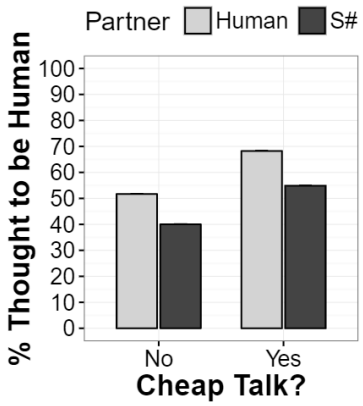


Figure 1: In previous trials, data indicated that it is difficult for people to distinguish whether they are interacting with S# or another human. This was true both when players engaged in cheap talk (through chat) and when they did not. Figure from [1].

through algorithms which are adjusted until they approximate people in making economic decisions. While not explicitly satisficing, by focusing on making correct predictions rather than optimizing payoff, it was my intention to create an algorithm which could handle similar suboptimal strategies humans sometimes use.

2.2 Modeling Humans

While much of the work on interactions may improve a machine's ability to interact successfully with humans regardless of the original intent, there are still things to be learned by focusing specifically on human behavior. The potential benefits derived from such attempts has inspired researchers in a number of studies. One popular topic of study is the prisoner's dilemma, which is one of the games considered in my work. In [25], success has been found modeling humans in this scenario, demonstrating that greater disparity between payoffs for cooperating versus defecting improves results. Other research involving the prisoner's dilemma has demonstrated that humans tend to be somewhat lenient when playing with others [26].

Still more study has been done in the attempt to model humans in other situations. Research involving more complex sequential games has been conducted in an effort to model humans using SARSA and Q-Learning [27]. Another model, SHARP, has been developed to model humans in solving security problems such as Stackelberg Security Games [28]. Further study has been done to attempt to shed light on the nature of human cooperation using a dual-process cognition, wherein an agent has both a default, intuitive process or a deliberation process which can be taken with a penalty to the payoff [29]. Through all of these, it is people's tendency to cooperate which stands out, which is seen in much of the data I considered in my work.

2.3 Using S# to Model Humans

A highly successful model for playing repeated matrix games, there has already been work done based on using S# to predict human behavior [4]. In S#, a set of designed experts are compared for anticipated performance, which is compared against the algorithm's current expected value. From these it selects a satisficing expert, which dictates the agent's behavior for a period (a small sequence of rounds). As play progresses the expectation is either raised or lowered according to performance

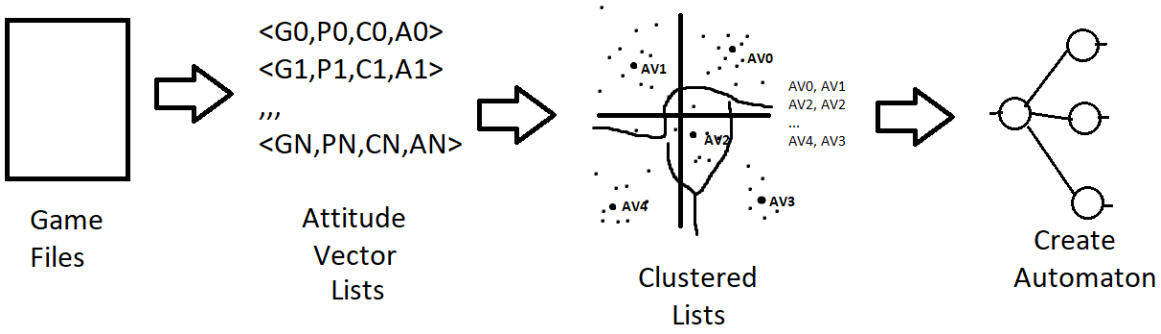


Figure 2: An overview of how AVA reads and stores data. Data is taken by AVA, converted to the general Attitude Vector space, and stored in trees to create strategy automata. The automaton is then capable of predicting human behavior in other matrix games.

and different experts are selected as needed. S# also includes the ability to communicate using a set of messages. One notable feature of this feature is that S# always follows what it claims it will do. This performs very well when compared with humans and other agents [1]. As shown in Figure 1, the behavior of S# is difficult for humans to distinguish from other humans. It is perhaps not surprising, therefore, that MAP-S#, a behavior-modeling algorithm based on S# [4], predicts human behavior in repeated matrix games with cheap talk at a high success rate (observed to be about 89%). While my model does not use S#, AVA is highly inspired by its approach. Similar to S# it uses multiple strategies, though these are learned from recorded games played by humans rather than designed experts. As previously noted AVA also doesn't use satisficing, though such strategies can be learned.

3 Overview to the Approach

In designing Attitude Vector Automata (AVA), I set out to build a system designed to predict human behavior in two player matrix games. Further, I wanted to develop AVA to be robust enough to reasonably predict behavior even when encountering a novel situation (payoff matrix). In developing AVA, I limited this further to the case of symmetric matrices, though application to asymmetric games could be done similarly.

Building a system capable of learning and predicting general strategies in matrix games required solving a number of essential problems, as detailed in Figure 2. Once training data is read, actions and messages must be encoded into a general state. Next, generalized data is combined in order to find most probable responses to any given sequence of actions. Finally, to apply the learned strategies to new games, AVA reads actions and messages in, transforming the data into the general space so it can predict the most probable general response which is applied to the current payoff matrix to determine the most probable action a human player would make. The following is a closer look at how this is accomplished.

3.1 Generalizing Actions

One of humanity's greatest strengths is their ability to transfer knowledge gained in one area to others. Ideas such as cooperation, revenge, forgiveness, and deceit are all examples of how people may act in a myriad of games, social encounters, and many other circumstances. These are typically learned as children and then applied throughout people's lives with little to no need for additional learning to understand how to exhibit these general behaviors in new situations. For a system aiming to simulate human behavior, this level of adaptation must be mimicked. It is for this exact reason that one of the first and last things AVA does involves translating specific actions to and from a general space.

For AVA to work in a general space, that space must first be detailed. To form such a space, I had to consider what information this could be extracted from. For matrix games, all decisions revolve around the payoffs. Very often there is one pair of actions in which one player gets the highest payoff and typically a different action pair corresponding to the highest payoff. Similarly, for any meaningful payoff matrix there will also be action pairs giving each player a lower payoff. In my attempt to encapsulate this information I devised a two-axis system of Attitude Vectors.

In describing Attitude Vectors I wanted to characterize both sides of each axis to help understand the significance of each. The first axis I considered in developing Attitude Vectors considers the payoff to individual players. To this end, players who only seek the highest payoff regardless of the affect on the other player can be characterised as "greedy," while players who seek (or accept) the lowest payoff typically do so to the benefit of the other player, thus this is considered a "placating" attitude. The second axis considers the combined payoff of both players for their actions. Thus, getting the highest combined score is considered a "cooperative" attitude, while getting a low combined score is classified an "absurd" attitude.

3.2 Learning Strategies

With a general space created, data needs to be processed into strategies. This is done by grouping rounds as features, identifying common attitudes, and combining them in a structure describing the strategies. In this form, attitudes can be predicted and converted to the most probable action.

Before data can be combined into automata, it must be sorted into a common format. Each match up is transformed into a list of features. These features describe the action taken by each player and the relevant messages made by each player. These four elements are represented as Attitude Vectors. For actions, the payoffs each obtains is used to calculate based on the payoff matrix used for the game, as described below. The Attitude Vector obtained for each player is then averaged with the previous round's corresponding Attitude Vector to account for strategies involving alternating between actions. Messages are handled in a similar manner if they refer to actions. If multiple messages are used to suggest actions to be taken, the representative Attitude Vector is calculated for each and the average is taken. In the event no messages are sent relating to what actions to take, the message is represented by the zero vector.

Once the actions and messages are converted to their representative Attitude Vectors, similar Attitude Vectors are grouped together. Attitude Vectors are separated by association with actions and messages and then clustered. As different games will naturally produce different Attitude Vectors, this allows for similar strategies in different games to be combined.

Once actions are converted and standardized the data is ready to be stored for use in predicting strategies. For this, I used a tree structure where nodes represent actions and messages. In this tree, groups of four layers represent the memory of rounds, with two final layers representing the action and message Attitude Vectors for the "current" round. Features extracted from games

are read in and added to the count contained in the tree's nodes. This way, when encountering new games AVA determines the most probable actions based on the frequency of each Attitude Vector with the given history. If a prediction is not found by following the given history, the query is considered with few rounds until a satisfactory solution is determined.

With the tree structure as described, multiple strategies will be describe. These can be traced by following common predicted Attitude Vectors and examining possible futures based on how the column player may act and, in turn, how the row player may respond. By identifying cases where two Attitude Vectors are likely, these can be separated into new trees until common strategies are extracted into their own predictive trees. For a given game, identifying which of such possibilities a player follows can help determine more precisely which strategy is likely being employed, switching which tree to focus on until the strategy appears to change.

4 Mathematical Framework

In order to truly learn what strategies humans use, there must be a way to convert observations into a form that can be applied to future (previously unseen) scenarios. Further, as humans are able to apply strategies from one scenario to another, so too should an algorithm aiming to predict humans be able to store and recall information in a more abstract level. To accomplish this, AVA converts actions to general features in the form of Attitude Vectors. From this converted state, data can be organized and recalled to predict actions in future games.

4.1 Attitude Vectors

The key to learning a strategy is understanding the effect a player's actions will have on themselves and other players. In matrix games, the effect of players choosing an action is the payoff each player obtains. As this is true regardless of setup, the relative payoffs frame our general space for considering game play abstractly. In AVA, I use a four-tuple referred to as an Attitude Vector to accomplish this, as described in the previous section.

The four values that make up the Attitude Vector are all within the range [0,1]. These four can be thought of as two pairs, each pair describing an axis. As such, pairs are defined such that only one attitude in each pair can be nonzero for any particular Attitude Vector. This way it is impossible for a player to be both greedy and placating or cooperative and absurd.

With general definitions of each attitude given, actions can be converted to their Attitude Vector counterparts. The player in question chooses action a_p with the expectation that the other player will choose player a_o. The payoff for the player is given by $v(a_p, a_o)$ and payoff for the other player is $v(a_o, a_p)$ due to the symmetric nature of the games considered. I also define v_{pmax} as the maximum payoff the player may attain, v_{pmin} as the minimum payoff for the player, v_{max} as the maximum combined payoff for both players, and v_{min} as the minimum combined payoff for both players. Finally, v_{avg} is the mean value between the maximum and minimum combined payoffs. Mathematically, the attitudes are defined as below:

$$Greedy = \begin{cases} \frac{v(a_p,a_o)-v_{pmin}}{v_{pmax}-v_{pmin}} & \text{if } v(a_p,a_o) > v(a_o,a_p) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$Placate = \begin{cases} \frac{v(a_o,a_p)-v_{pmin}}{v_{pmax}-v_{pmin}} & \text{if } v(a_o,a_p) > v(a_p,a_o) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

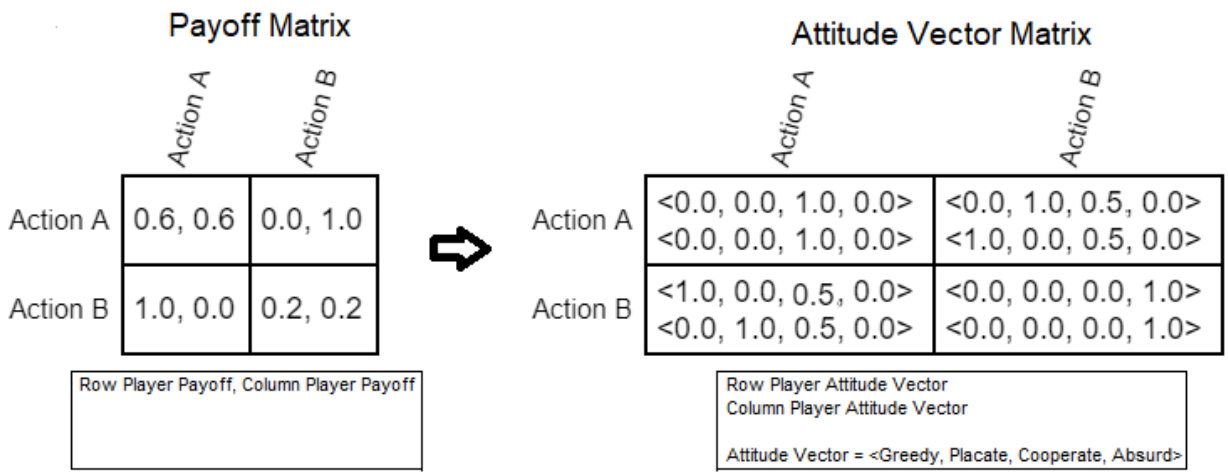


Figure 3: A payoff matrix (prisoner's dilemma) is transformed to give the Attitude Vector an action pair is represented by. This is used to convert actions and messages into a form that can be stored and reused in other games.

W

$$Cooperative = \begin{cases} \frac{v(a_p,a_o)+v(a_o,a_p)-v_{avg}}{v_{max}-v_{avg}} & \text{if } v(a_p,a_o)+v(a_o,a_p) > v_{avg} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$Absurd = \begin{cases} \frac{v_{avg}-v(a_p,a_o)-v(a_o,a_p)}{v_{max}-v_{avg}} & \text{if } v_{avg} > v(a_p,a_o)+v(a_o,a_p) \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

With a means for converting actions to Attitude Vectors defined, payoff matrices are used to make Attitude Vector matrices. With these, recorded game play can be easily converted into Attitude Vectors. To simplify predictions, AVA assumes that players were able to predict the action the other player would take. However, in order to account for matrices where a strategy may be best defined between two action pairs (such as the cooperative strategy of the Alternator Game) the presumed Attitude Vector for any round after the first is calculated as the average value of the current and previous round. Thus the strategy of alternating between two action pairs wherein said strategy is cooperative maps to the single cooperative action pair of another.

To be of use, there needs to be a means of converting Attitude Vectors into actions. To do this, the action pairs of a game matrix are transformed into Attitude Vectors. To convert Attitude Vectors, the algorithm calculates the Euclidean distance from the Attitude Vector given by each pair. The closest pair is the most similar to the intended Attitude and its associated action pair is the most probable action pair a strategy would select.

4.2 Building and Applying Strategies

With a method for converting actions into a general space as well as the reverse transformation defined, these are next organized into strategies, to be later applied to new games. I start by reading transcripts of matrix games previously played, including the actions and messages both players submitted. These are processed and categorized, then stored for later retrieval and application.

On receiving a game transcript, each round of play is converted into a feature, consisting of four Attitude Vectors representing the player's action, the player's messages, the other player's actions, and the other player's messages. Actions are converted into Attitude Vectors as demonstrated above (with an example given in Figure‘3, then the Attitude Vector for each round is stored as the average between the current and previous round to account for instances only possible when alternating. In the case of the first action, this is simply transformed as if the same action was played in the (non-existent) preceding round. Messages also use this basic approach, though must take into account the possibility of multiple suggested actions to perform or not. To do this, messages are processed into a list of action pairs suggested to perform and actions suggested not to do. Messages which contradict themselves are treated as empty messages and their Attitude Vectors are set to the zero vector. Otherwise messages suggesting a single pair of actions is processed as previously described, while multiple actions are averaged to give the resulting mean values. Finally, suggesting a player not take a specified action averages all non-prohibited Action Vectors.

Once all the games have been processed into lists of features they are clustered into representative groups. This is done using the k-means algorithms, grouping Attitude Vectors for actions and messages separately. Values of k are compared using the Silhouette method [30], with values for k ranging from two to eight in both cases. Once clusters are determined, listed features are converted to the centroid of the cluster they pertain to. These lists of converted features are then ready to be stored for future recall.

To be able to use the data in future games, feature lists are stored in a tree structure as illustrated in Figure‘4. From the root, branches represent the action used by the player, then the message sent by the player, the action used by the other player, then the message sent. This pattern repeats for each round the tree holds in memory, terminating with the player's actions and messages. The leaves of the tree contain a count of the number of instances their particular path is followed in the lists of features. Smaller trees are kept for instances when an exact match is not possible for the full history.

After being populated, the tree structure could simply select the most likely next action based on the path taken for the given history, though this risks losing some similar strategies. To avoid this, the tree can be split by identifying histories leading to multiple, frequently diverging possibilities as described in Figure 5. By repeatedly sorting through nodes looking for nodes with the greatest difference between how many sequences of actions and messages they represent, nodes can be found with two or more significant alternate actions to predict, then follow possibilities from the second most frequent option to remove from the original tree and put into its own. With a group of such trees it is then possible to track the accuracy of each and switch the active tree when the current one underperforms.

In order to split the tree as described, AVA first puts the root node of the tree into a priority queue, sorting based on the total number of sequences the node is involved in. Nodes are taken from the queue and checked whether the node's children's children are leaf nodes. If so, these are added to a list, otherwise their children are added back to the priority queue. Once the queue is empty or all remaining nodes are involved in a small number of recorded sequences (two or less) the nodes from the list have their children's counts checked, finding the child with the second highest count. The node with the highest second highest count represents the action most likely to be missed by the single tree and therefore it, with any following messages, can be removed into a new tree.

With the sequence for starting the second tree found, this sequence can be used to move subsequent possible sequences from the original tree into the new tree. To do this, all observed

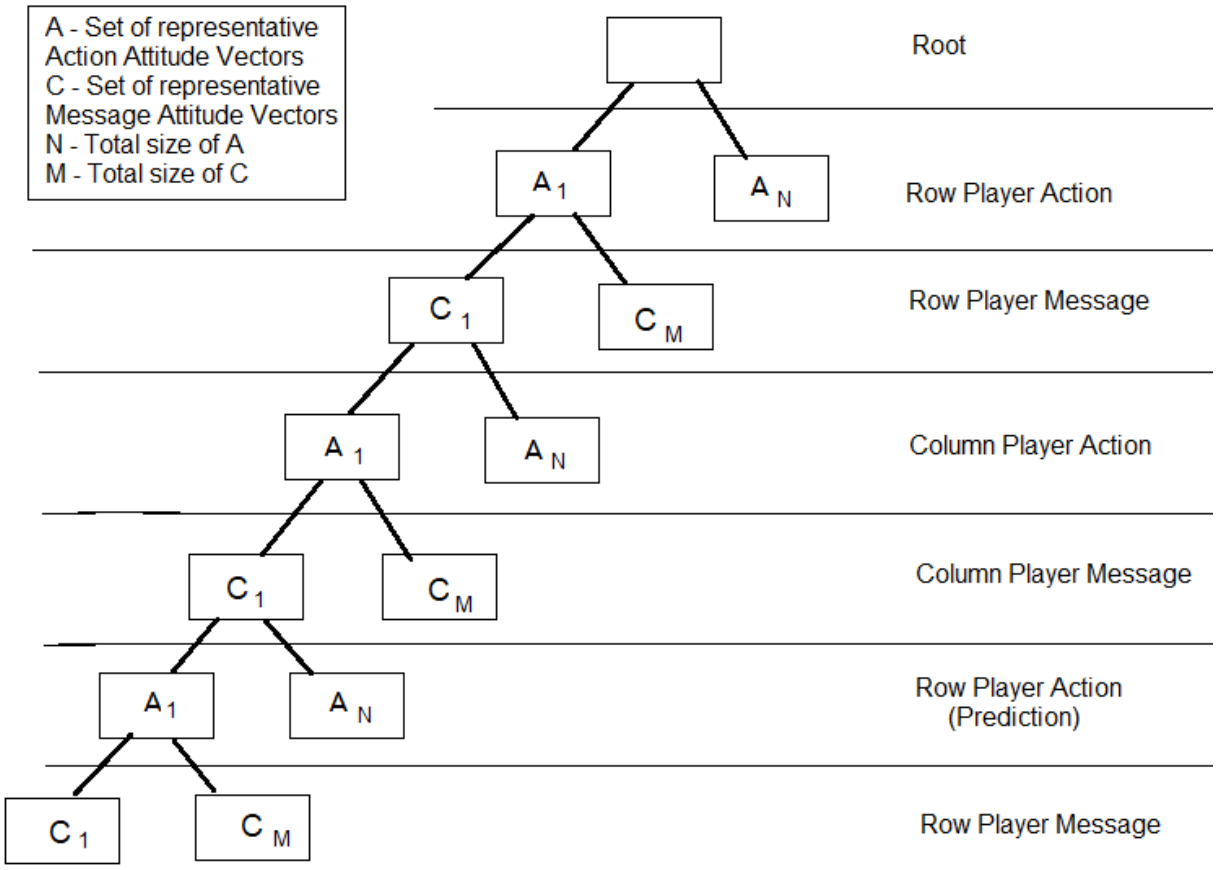


Figure 4: Once game play histories are converted into representative Attitude Vectors the information is stored in a tree structure. The final two layers are used to record how many times a player acted and sent messages given the prior rounds (represented by higher layers. These are used to make predictions on the test data. Between these and the root nodes represent previous interactions, with four layers for each prior interaction AVA can remember when making predictions.

```
Given a tree T whose nodes' count is the number of
    observations which reach the node
ExtractTree(T):
Create empty priority queue Q
Create empty collection C
From T, add root node to Q
While Q is not empty:
    Remove next node N
    If N's children's children are leaf nodes:
        If N has more than one child, add N to C
    Else:
        Add N's children to Q
Calculate child with second highest count for each
    node in C.
Create empty tree T'
TransferSequence(T, T', S) where S is the child
    with the highest count from the previous step.
```

```
Given trees T, T' whose nodes' count is the number
    of observations which reach the node, and
    sequence S defining a node with length
    one less than the depth of T
TransferSequence(T, T', S)
For each leaf L reachable from S in T:
    If the count of L is nonzero:
        Find the leaf L' in T' reached by the same
            sequence as L in T
        Set the count of L' to the count of L
        Set the count of L to zero
Create sequence S' by removing the first four
    elements of S
For each node N reachable four nodes deeper from
    S' by sequence R:
    If the count of N is nonzero:
        TransferSequence(T, T', R)
```

Figure 5: The algorithms used for dividing information into multiple trees. The ExtractTree algorithm identifies a sequence to extract, using the TransferSequence algorithm to remove sequences which may follow the initial sequence to be extracted.

sequences stemming from the given sequence are determined, moved to a list, then removed from the original tree. These are systematically added to the new tree, then used to consider possible futures by removing the oldest round (represented by the first four nodes), shifting the sequence by one round. Any sequences in the original tree reachable from the history are then entered into the same process to remove them and add them to the new tree until no such sequences can be found. Finally, this entire process can be repeated to find more sequences where splitting by predicted action yields more automata.

With games processed into trees of information, the data can be applied to new games. As with any game, the new game is converted into Attitude Vectors. These are used as the history to predict the next Attitude Vector. To convert the Attitude Vector to an action, the previous action's Attitude Vector is calculated ignoring history, a_p, and the action is predicted based on an Attitude Vector calculated by calculating the distance between the two and adding the difference to the predicted Attitude Vector, then choosing the action pair best describing the calculated Attitude Vector.

5 Data

In order to learn how human players behave in two player matrix games, data of humans playing these games was used to train and test AVA. This data was collected from two data sets generated during previous studies with S# [1, 31], in which human players were matched against either S# or another human. In these games, both players were able to send messages to their partner at the start of each round, selected from a predefined set of speech phrases. After sending these messages, each player simultaneously selected an action for that round.

In the data I examined, players were matched in the three different games depicted in Figure 6. These games are Alternator, Chicken, and a Prisoner's Dilemma. In Alternator, players select from one of three options. To cooperate players alternate between the joint actions in

Alternator

	Action A	Action B	Action C
Action A	0.0, 0.0	0.3, 0.7	1.0, 0.4
Action B	0.7, 0.3	0.1, 0.1	0.3, 0.45
Action C	0.4, 1.0	0.45, 0.3	0.4, 0.4

Chicken

	Action A	Action B
Action A	0.0, 0.0	1.0, 0.33
Action B	0.33, 1.0	0.84, 0.84

Prisoner's Dilemma

	Action A	Action B
Action A	0.6, 0.6	0.0, 1.0
Action B	1.0, 0.0	0.2, 0.2

Figure 6: Payoff matrices used for playing Alternator, Chicken, and Prisoner's Dilemma. Payoffs are listed for row player, then column player based on which action they choose.

bottom-left and top-right cells. Chicken and the Prisoner's Dilemma each provide two joint actions which can be effectively described as cooperating or defecting. In both cases, a defecting player will outperform a cooperating one if the other player cooperates. In Chicken both players get zero payoff if both defect, whereas in the Prisoner's Dilemma this simply gives a decreased payoff. Further, in the Prisoner's Dilemma, choosing the cooperative action when the other player defects results in zero payoff to the cooperating player. In every case, the payoff matrices are symmetric, so player order has no effect on how strategies are played.

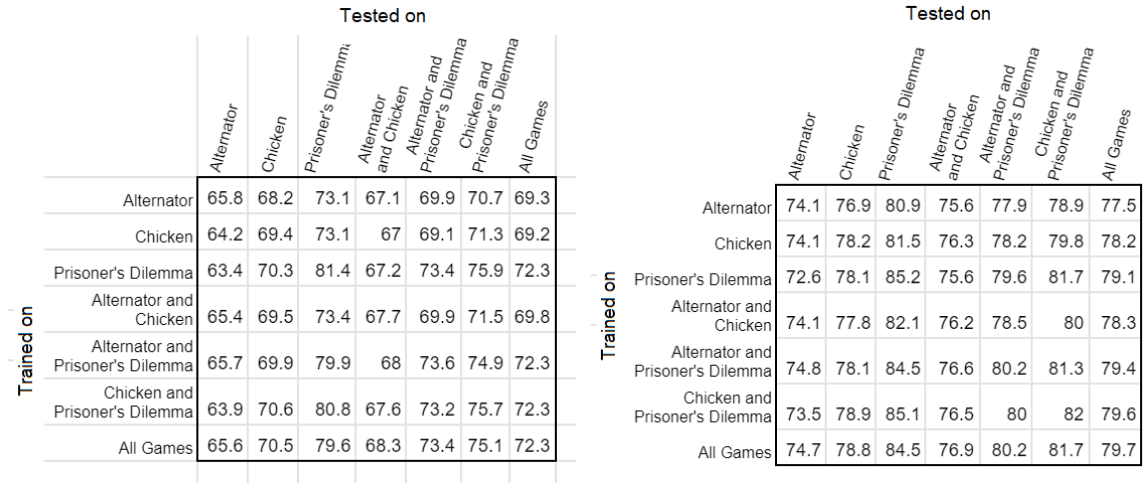
For my research, data was collected from two studies. The first study [1] includes 21 instances of Alternator, 22 instances of Chicken, and 24 instances of Prisoner's Dilemma. The second study [31] includes 144 instances of the Alternator, 144 of Chicken, and 145 of Prisoner's dilemma. The first players listed in the first set of data were used in preparing the model, and final results were taken by training on these same players, then testing on the second players. Following this, AVA was trained on both players from the first data set and tested on the human players from the second data set.

In every game used with AVA, players had the opportunity to use a variety of strategies. Some players tended to play cooperatively with honest communication, while others tried to maximize their own profit and used messaging to be deceitful. Still others allow themselves to be bullied to playing an inferior strategy, playing poorly due to frustration. The S# algorithm was designed to play honestly, and so it was my intention to design an algorithm capable of learning to handle dishonesty, as well as a broader variety of players.

6 Overview of Results

I considered two results to be most important in determining the effectiveness, strengths, and weaknesses of AVA. First, to get an overall sense of AVA's effectiveness, I trained and tested my algorithm using four different memory lengths (represented by four levels of nodes in the tree structure per round remembered), averaging overall scores over 200 training sessions per memory length. Results considered automata trained on each game type individually and combined, measuring accuracy on test data by individual and combined game types. Second, in order to better understand the ability of AVA to generalize across games, I also looked at round-by-round predictions made by AVA when trained on the same game type, a different game type, or all game types.

The first results I looked at came from predicting the actions taken by the second player in my first data set, with automata trained on the actions and messages of the first player. I first



Tested on

Trained on	Alternator	Chicken	Prisoner's Dilemma	Alternator and Chicken	Alternator and Prisoner's Dilemma	Chicken and Prisoner's Dilemma	All Games
Alternator	65.8	68.2	73.1	67.1	69.9	70.7	69.3
Chicken	64.2	69.4	73.1	67	69.1	71.3	69.2
Prisoner's Dilemma	63.4	70.3	81.4	67.2	73.4	75.9	72.3
Alternator and Chicken	65.4	69.5	73.4	67.7	69.9	71.5	69.8
Alternator and Prisoner's Dilemma	65.7	69.9	79.9	68	73.6	74.9	72.3
Chicken and Prisoner's Dilemma	63.9	70.6	80.8	67.6	73.2	75.7	72.3
All Games	65.6	70.5	79.6	68.3	73.4	75.1	72.3

Tested on

Trained on	Alternator	Chicken	Prisoner's Dilemma	Alternator and Chicken	Alternator and Prisoner's Dilemma	Chicken and Prisoner's Dilemma	All Games
Alternator	74.1	76.9	80.9	75.6	77.9	78.9	77.5
Chicken	74.1	78.2	81.5	76.3	78.2	79.8	78.2
Prisoner's Dilemma	72.6	78.1	85.2	75.6	79.6	81.7	79.1
Alternator and Chicken	74.1	77.8	82.1	76.2	78.5	80	78.3
Alternator and Prisoner's Dilemma	74.8	78.1	84.5	76.6	80.2	81.3	79.4
Chicken and Prisoner's Dilemma	73.5	78.9	85.1	76.5	80	82	79.6
All Games	74.7	78.8	84.5	76.9	80.2	81.7	79.7

Figure 7: Average accuracy of predicted actions for the first data set using (left) a single tree and (right) multiple trees. Averaged over 800 attempts at varying memory lengths. Rows indicate training data used, columns indicate test data used.

considered results obtained when using a single automaton, storing all player data in a single tree for use in predictions, shown in Figure 7(left). Results here ranged from 63.4% - 81.4% accuracy. Scores in this case were highest for automata trained on prisoner's dilemma games, while results were weakest for those trained on alternator. Both the maximum and minimum average scores reflect these, with both instances occurring when tested solely on the prisoner's dilemma data. These trends were repeated when using multiple automata as shown in Figure 7(right), though the average results improved, ranging from 72.6% to 85.2%.

After getting results from the first data set, I turned my attention to the larger second set of data, the results of which are shown in Figure 8. On this data set, the automata trained on Chicken performed the best overall when using the single automaton. However, when trained on Prisoner's Dilemma, it performed the worst. Overall these ranged from 55.7% to 70.5%. Using multiple automata again saw an improvement in overall results, increasing the average scores to 69.8% to 82.4%. Again automata trained on Chicken performed the best, though those trained on Alternator saw a smaller improvement and therefore accounted for the lowest average results.

Apart from overall results, I also looked at how AVA performed throughout individual games. This involved recording the actions, messages, Attitude Vectors, and predictions for each round. As the multiple automata approach performed consistently better, it is this approach which I considered.

Before discussing the goals of outperforming Map-S#, better prediction of liars, and the ability to generalize, it is worth briefly exploring AVA's strengths and weaknesses in handling strategies to try to better understand its potential usefulness for future endeavors. To delve further into this, I looked at the round-by-round predictions to see where AVA succeeded and where it failed when using multiple automata (the highest performing AVA automata). In doing this, I first looked at results from the first data set, using patterns found to see how well similar behavior was predicted in the second data set. As automata used to test the second data set would have learned from the same training data used in the first set plus the data tested on in the same set, this would give me some idea of what kinds of strategies further training could improve.

Trained on \ Tested on	Alternator	Chicken	Prisoner's Dilemma	Alternator and Chicken	Alternator and Prisoner's Dilemma	Chicken and Prisoner's Dilemma	All Games
Alternator	60.8	69.9	55.7	65.3	58.2	62.7	62.1
Chicken	61.1	69.9	57.1	65.5	59.1	63.5	62.7
Prisoner's Dilemma	60.4	69.3	58.7	64.8	59.5	64	62.8
Alternator and Chicken	60.4	70.5	56.8	65.5	58.6	63.6	62.6
Alternator and Prisoner's Dilemma	60.4	69.7	58.9	65	59.6	64.3	63
Chicken and Prisoner's Dilemma	60.3	69.9	59.9	65.1	60.1	64.9	63.3
All Games	60.5	70.5	59.7	65.5	60.1	65.1	63.5

Trained on \ Tested on	Alternator	Chicken	Prisoner's Dilemma	Alternator and Chicken	Alternator and Prisoner's Dilemma	Chicken and Prisoner's Dilemma	All Games
Alternator	71.4	81	75.7	76.2	73.6	78.3	76.1
Chicken	70.8	82	73.1	76.4	71.9	77.5	75.3
Prisoner's Dilemma	69.9	81.2	77.2	75.5	73.6	79.2	76.1
Alternator and Chicken	70.8	82.3	73.5	76.5	72.2	77.9	75.5
Alternator and Prisoner's Dilemma	69.9	81.9	77.2	75.9	73.6	79.5	76.3
Chicken and Prisoner's Dilemma	69.8	81.8	76.9	75.8	73.4	79.4	76.2
All Games	70.4	82.4	77	76.4	73.7	79.7	76.6

Figure 8: Average results for the second data set using (left) a single tree and (right) multiple trees. Averaged over 800 attempts at varying memory lengths. Rows indicate training data used, columns indicate test data used.

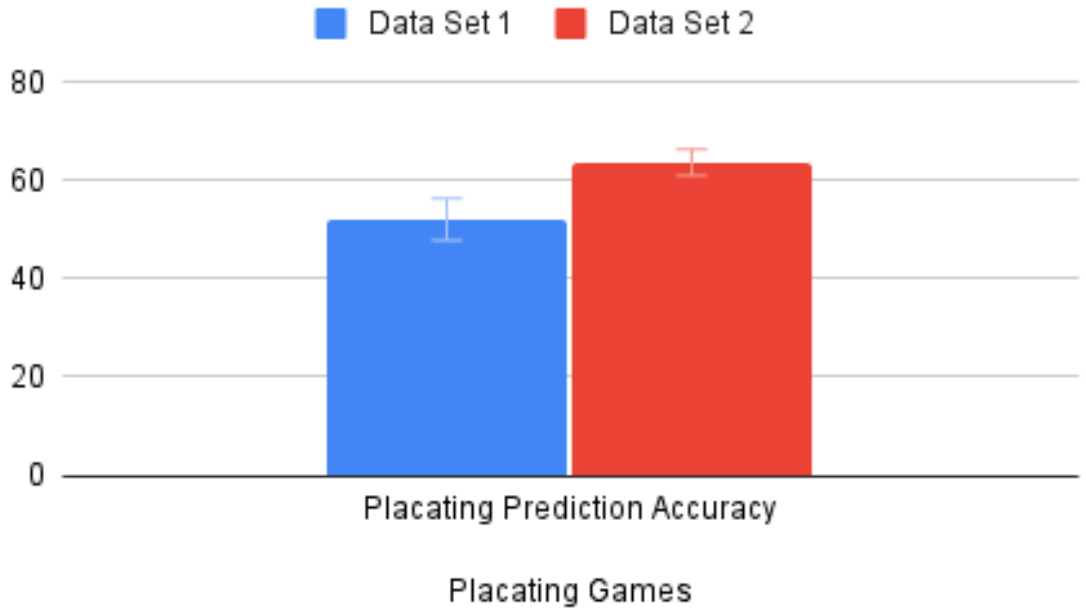


Figure 9: Prediction accuracy for players who repeatedly placate the other player. Accuracy improves with more data (strategies) to learn from. Error bars show the standard error of the mean.

From the first data set, it quickly became apparent that AVA generally performed well when the player acted consistently. As AVA considers things in terms of Attitude Vectors, consistency in this case means maintaining the same Attitude Vector round after round, not necessarily the same action. For instance, in the alternator game players only reach the highest possible cooperative solution when alternating between actions. This also occurred for players alternating between cooperative and greedy behavior, continuing to show strong predictive results. There was, however, one repeated behavior which was not well predicted in the first data set: placating. In fact, in game play where a player consistently gave in to the other player AVA consistently predicted retaliation, which didn't occur for many rounds. Despite the consistently poor prediction in the first data set with regard to players who placated, this prediction changed with the second data set (Figure 9). This suggests a great strength and perhaps the greatest reason to pursue further work: the ability to learn. Through learning from a greater variety of strategies, AVA is able to adapt and apply these in new situations, giving it a strong ability to broaden its abilities with more data to learn from.

Finally, it must be noted that where AVA typically does well with consistent behavior (except as noted), inconsistent behavior can be much harder to predict. This is perhaps to be expected, as some of the least accurate predictions stem from players who, for one reason or another, seem to behave inconsistently. It is possible that there are some underlying strategies involved in such cases, though presently one has not been consistently identified.

7 Analysis of Results

In this section, I reflect on the strengths and weaknesses of AVA given the results presented in the previous section. I first compare the general predictive accuracy of AVA with that of Map-S#. Given the reduced performance of Map-S# when people lied, I then look at AVA's ability to predict human behavior when humans lied. Finally, I discuss the ability of AVA to generalize across games.

7.1 Comparing with MAP-S#

In making AVA, one of my primary goals was to outperform MAP-S# in predicting human behavior. Unfortunately, the results show that AVA's predictive accuracy lags behind those produced by MAP-S# on the [1] data as demonstrated in Figure 10. In [4], MAP-S# was found to predict with 89.02% accuracy. By contrast, at best AVA predicts with an average accuracy of 84.5% using the first data set and 82.4% when testing on the second data set. This is using the most successful automata and considering all test data in each case. Using the average of all automata, these results decrease to 79.5% and 76.6% for the first and second data sets respectively. Ultimately, predictions are at best 5% less accurate than S# overall, and typically 10% to 13% worse based on the results.

7.2 Predicting Liars

While AVA did not predict human behavior as well as MAP-S#, it is still worth investigating performance in specific cases. Particularly, MAP-S# showed a noticeable drop in performance when attempting to predict the behavior of lying players. A lying player is any who indicates intention through messaging to choose an action different from what they ultimately select. [4] [4] determined that MAP-S# drops to an average of 68.35% given lying players. For lying players in the the first data set, the average score when testing on all lying players was 62.6%, as shown in Figure 11. However, if cases where both players lie are excluded, this increases to 77.8%. As the

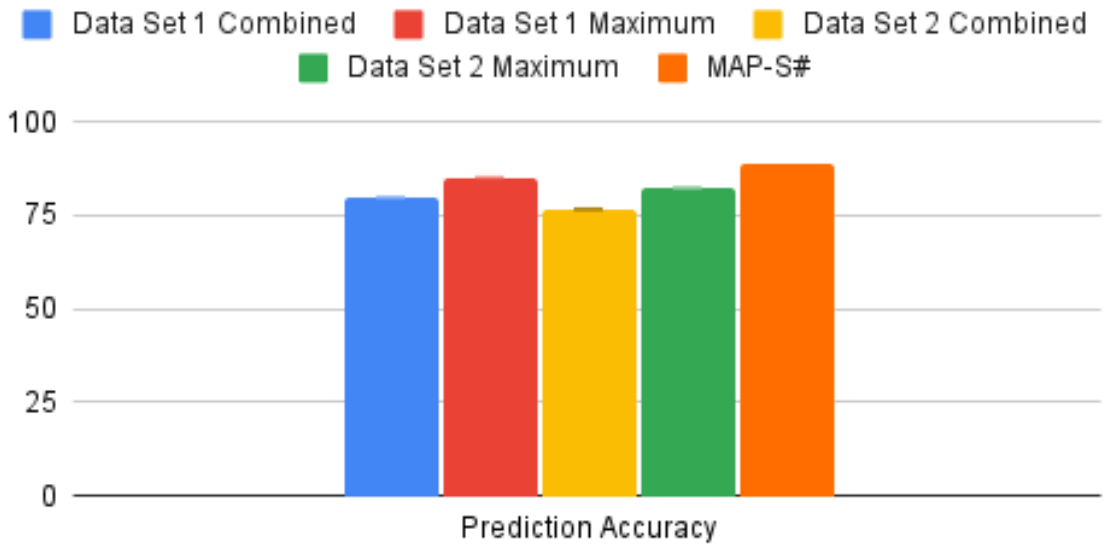


Figure 10: Comparing how well AVA and MAP-S# predicted human behavior. For AVA I include both results of multiple automata trained and tested on all games and the maximum prediction results considering all games individually and combined. MAP-S# uses data from [1]. Error bars show the standard error of the mean.

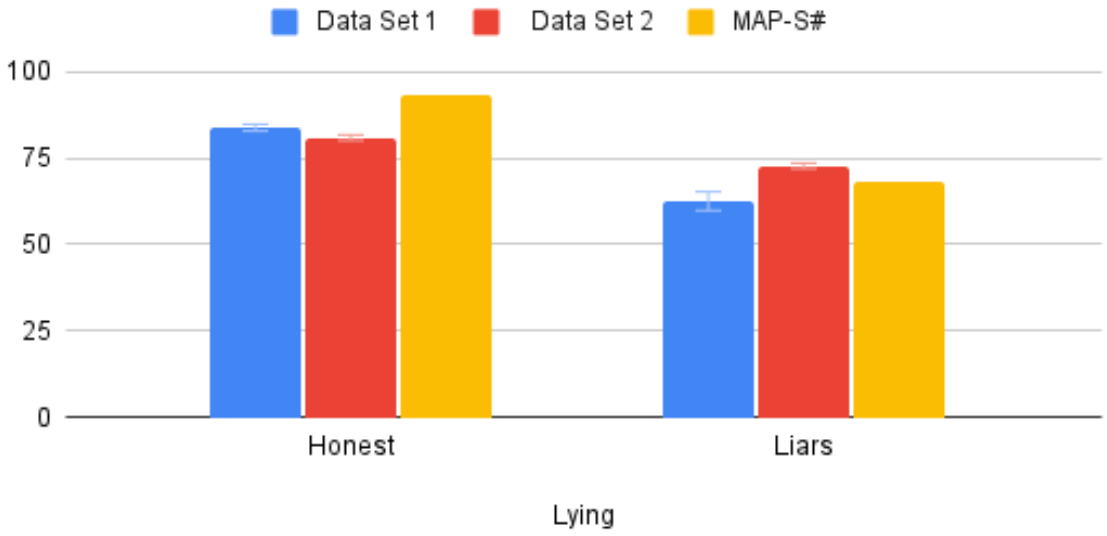


Figure 11: Lying was a significant challenge for MAP-S# to model. Here it can be seen that AVA does perform somewhat better on the second data set. Error bars show the standard error of the mean.

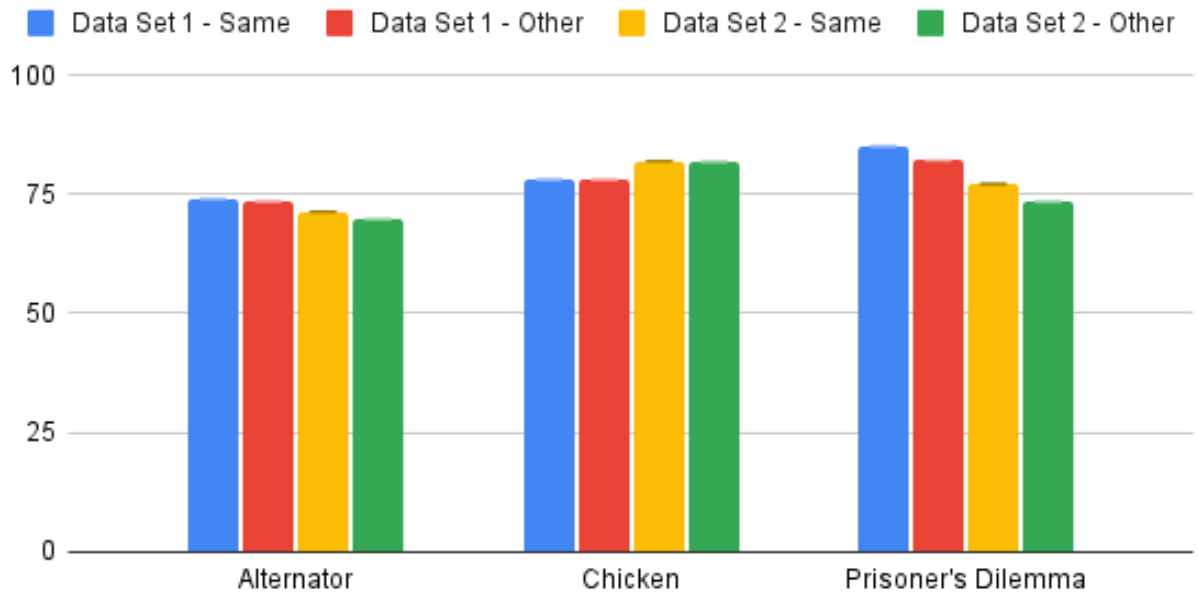


Figure 12: A comparison between accuracy on test data when test data came from the same or different game as the training data. Error bars show the standard error of the mean.

second data set consists of only human players against honest agents, I could only consider when the predicted player lies, which yields an average prediction rate of 72.8%. While still lower than the average score, in the latter data set at least, when comparing the results as in Figure 11 some improvement can be seen in predictive power. While this may hint at possible improvement from a larger training set, further studies would be required to make this determination.

7.3 Generalizing to New Games

Thus far in this discussion, I have primarily to compared results with those obtained by MAP-S#. Although overall performance is relatively lacking, the ability to generalize is noteworthy. As shown in Figure 12 in both data sets, average results on any game type are similar, with automata trained on other game types typically performing just slightly worse than when trained on the same game type. In a few instances, predictions of automata trained on other games even exceed what automata trained on the same game type achieve. This general similarity in prediction power suggests that using Attitude Vectors as a means for generalizing across games has potential. This is important, as it allows the algorithm to be trained in specific repeated games and then used in other kinds of repeated games without a substantial decrease in performance.

8 Conclusions and Future Work

As seen through the analysis of my results, AVA was unable to meet its overall goal of outperforming MAP-S# when predicting human behavior. Frequent changes in apparent strategy was a particular complication in making accurate predictions, and this lack of flexibility played a role in AVA's failure to meet its goal. Despite this, AVA succeeds to some extent in predicting the behavior of liars and even more so in generalizing to different matrices through the use of Attitude Vectors. Ultimately,

there is ample room for further study to be done in the future to improve the algorithm, either through modifying the algorithm itself or through the use of more training data.

Looking to possible ways to explore the foundation I have laid out, there are a number of changes which could prove fruitful. One limitation placed on AVA during my tests was a separation of training and test data. However, this does limit the strategies available for AVA to learn and apply, and it could be interesting to allow for learning while looking at new data. If something like my algorithm were to be used in a real-world setting, this is presumably how it would function, allowing for a continuously growing repository of strategies to learn from. Another approach to explore would be using the probability distribution of possible actions to predict, rather than my method of selecting the most likely Attitude Vector and simply returning the most probable result. With the distribution the action could be voted on based on predicted probability for different Attitude Vectors and how much each Attitude Vector correlates to each action. Finally, an expanded or otherwise modified feature set beyond the Attitude Vectors detailed here may be better able to capture some key details not considered in this research. Such features may include whether people follow what they say or defer to the other's proposal, Attitude Vectors averaged over multiple rounds, and so on.

References

[1] Jacob W. Crandall, Mayada Oudah, Tennom Chenlinangjia, Fatimah Ishowo-Oloko, Sherief Abdallah, Jean-François Bonnefon, Manuel Cebrián, Azim Shariff, Michael A. Goodrich, and Iyad Rahwan. Cooperating with machines. *Nature Communications*, 9, 2018. URL `https://doi.org/10.1038/s41467-017-02597-8DO`.

[2] Elisa Bertino, Finale Doshi-Velez, Maria L. Gini, Daniel Lopresti, and David Parkes. Artificial intelligence & cooperation. *CoRR*, abs/2012.06034, 2020. URL `https://arxiv.org/abs/2012.06034`.

[3] H. Z. Gintis. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Behavior*. Princeton University Press, 2000.

[4] N. Mathema, M. A. Goodrich, and J. W. Crandall. Predicting plans and actions in two-player repeated games. *AAAI 2020 Workshop on Plan, Activity, and Intent Recognition,*, 2020.

[5] George W. Brown. Iterative solution of games by fictitious play. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, New York, 1951.

[6] Sandip Sen and Neeraj Arora. Learning to take risks. In *AAAI Workshop on Multiagent Learning*, pages 59 – 64, 1997.

[7] Janet Kolodner. *Case-based Reasoning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-237-2.

[8] David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *AAAI Proceedings*, pages 62 – 67, 1996.

[9] Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenfeld. Teamwork with limited knowledge of teammates. In *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence*, July 2013. URL http://www.cs.utexas.edu/users/ai-lab/?barrett:aaai13.

[10] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484 – 489, January 2016.

[11] Michael Rovatsos, Gerhard Weiß, and Marco Wolf. Adaptive agents and multi-agent systems. chapter Multiagent Learning for Open Systems: A Study in Opponent Classification, pages 66–87. Springer-Verlag, Berlin, Heidelberg, 2003. ISBN 3-540-40068-0. URL http://dl.acm.org/citation.cfm?id=1805771.1805777.

[12] Alan Lockett, Charles Chen, and Risto Miikkulainen. Evolving explicit opponent models for game play. In *Genetic and Evolutionary Computation Conference (GECCO-2007)*, 2007. URL http://nn.cs.utexas.edu/?lockett:gecco07.

[13] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1804–1813, New York, New York, USA, 20–22 Jun 2016. PMLR. URL http://proceedings.mlr.press/v48/he16.html.

[14] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40:e253, 2017. doi: 10.1017/S0140525X16001837.

[15] H. Gweon and R. Saxe. Chapter 20 - developmental cognitive neuroscience of theory of mind. In John L.R. Rubenstein and Pasko Rakic, editors, *Neural Circuit Development and Function in the Brain*, pages 367 – 377. Academic Press, Oxford, 2013. ISBN 978-0-12-397267-5. doi: https://doi.org/10.1016/B978-0-12-397267-5.00057-1. URL http://www.sciencedirect.com/science/article/pii/B9780123972675000571.

[16] Chris L. Baker, Rebecca Saxe, and Joshua B. Tenenbaum. Bayesian theory of mind: Modeling joint belief-desire attribution. In *CogSci*, 2011.

[17] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 1*, AAAI'96, pages 120–125. AAAI Press, 1996. ISBN 0-262-51091-X. URL http://dl.acm.org/citation.cfm?id=1892875.1892893.

[18] Vincent Conitzer and Tuomas Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1):23–43, May 2007. ISSN 1573-0565. doi: 10.1007/s10994-006-0143-1. URL https://doi.org/10.1007/s10994-006-0143-1.

[19] Pablo Hernandez-Leal, Yusen Zhan, Matthew E. Taylor, L. Enrique Sucar, and Enrique Munoz De Cote. Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems*, 31(4):767–789, July 2017. ISSN 1387-2532. doi: 10.1007/s10458-016-9352-6. URL https://doi.org/10.1007/s10458-016-9352-6.

[20] Stefano V. Albrecht, Jacob W. Crandall, and Subramanian Ramamoorthy. Belief and truth in hypothesised behaviours. *Artif. Intell.*, 235(C):63–94, June 2016. ISSN 0004-3702. doi: 10.1016/j.artint.2016.02.004. URL https://doi.org/10.1016/j.artint.2016.02.004.

[21] Nolan Bard and Michael Bowling. Particle filtering for dynamic agent modelling in simplified poker. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI)*, pages 515–521, 2007.

[22] Stefano V. Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 1155–1156, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-1993-5. URL http://dl.acm.org/citation.cfm?id=2484920.2485118.

[23] Jeff L. Stimpson, Michael A. Goodrich, and Lawrence C. Walters. Satisficing and learning cooperation in the prisoner's dilemma. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'01, pages 535–540, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-812-5, 978-1-558-60812-2. URL http://dl.acm.org/citation.cfm?id=1642090.1642164.

[24] W Brian Arthur. Designing Economic Agents that Act Like Human Agents: A Behavioral Approach to Bounded Rationality. *American Economic Review*, 81(2):353–359, May 1991. URL https://ideas.repec.org/a/aea/aecrev/v81y1991i2p353-59.html.

[25] John Nay and Yevgeniy Vorobeychik. Predicting human cooperation. 2016. URL https://doi.org/10.1371/journal.pone.0155656.

[26] Drew Fudenberg, David G. Rand, and Anna Dreber. Slow to anger and fast to forgive: Cooperation in an uncertain world. *American Economic Review*, 102(2):720–49, April 2012. doi: 10.1257/aer.102.2.720. URL http://www.aeaweb.org/articles?id=10.1257/aer.102.2.720.

[27] R. Ceren, P. Doshi, M. Meisel, A. Goodie, and D. Hall. On modeling human learning in sequential games with delayed reinforcements. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 3108–3113, Oct 2013. doi: 10.1109/SMC.2013.530.

[28] Debarun Kar, Fei Fang, Francesco Maria Delle Fave, Nicole Sintov, and Milind Tambe. "A game of thrones": When human behavior models compete in repeated stackelberg security games. In *AAMAS*, 2015.

[29] Adam Bear and David G. Rand. Intuition, deliberation, and the evolution of cooperation. *Proceedings of the National Academy of Sciences*, 113(4):936–941, 2016. ISSN 0027-8424. doi: 10.1073/pnas.1517780113. URL `http://www.pnas.org/content/113/4/936`.

[30] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. ISSN 0377-0427. doi: https://doi.org/10.1016/0377-0427(87)90125-7. URL `https://www.sciencedirect.com/science/article/pii/0377042787901257`.

[31] Mayada Oudah, Talal Rahwan, Tawna Crandall, and Jacob W. Crandall. How AI wins friends and influences people in repeated games with cheap talk. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.