International Journal of Fuzzy Logic and Intelligent Systems, vol. 10, no. 3, September 2010, pp.170-177
DOI : 10.5391/IJFIS.2010.10.3.170

# A Learning AI Algorithm for Poker with Embedded Opponent Modeling

**Seong-gon Kim* and Yong-Gi Kim****

***CISE department, University of Florida, USA**
****Dept of Computer Science, Gyeongsang National University, Korea**
**Corresponding author: ygkim@gnu.ac.kr**

**Abstract**

Poker is a game of imperfect information where competing players must deal with multiple risk factors stemming from unknown information while making the best decision to win, and this makes it an interesting test-bed for artificial intelligence research. This paper introduces a new learning AI algorithm with embedded opponent modeling that can be used for these types of situations and we use this AI and apply it to a poker program. The new AI will be based on several graphs with each of its nodes representing inputs, and the algorithm will learn the optimal decision to make by updating the weight of the edges connecting these nodes and returning a probability for each action the graphs represent.

**Key Word** : Learning AI Algorithm, Poker, Opponent Modeling

## 1. Introduction

Artificial intelligence has long been applied to games such as Chess[1], Checkers[2][3], Go[4], Othello[5][6] and other various board games with positive results in beating human players and has gained much public attention in the process. But because players have entire knowledge of the game states, computers have always had the advantage over human players when calculating strategies and moves and making future predictions in these types of games. On the other hand, games such as scrabble or poker are different in that these are games of imperfect information; deriving each play style of an opponent becomes essential, and human "gut instincts" and reasoning have had more success than algorithmic computer computation in the past. This is where machine learning is of most interest.

What this project will implement is poker, Texas Hold'em style in particular. A new graph based machine learning algorithm that focuses on the strength of relationships between the various observable variables to produce an output. It will learn to play poker from a blank state with an implicit opponent model to decide on a play action without having to predict the opponent's action explicitly. The goal will be to determine whether machine learning can benefit the poker playing performance of AI poker bots as opposed to strategies based on pure mathematical and statistical calculations, and if so, how much of a benefit it would be. The project will also explore the advantages, if any, of the new machine learning algorithm by comparing and playing rule based and random chance based poker opponents against it.

Manuscript received Mar. 15. 2010; revised . Aug. 27, 2010; accepted Sep. 7, 2010

## 2. Texas Hold'em

There are several important aspects to consider when playing Texas Hold'em. The first is the *Pre-Flop Evaluation*. Pre-flop play in Hold'em has been extensively studied in poker literature[7] and there exists a general strategy to use for each given pre-flop hand since there are only 1326 possible combinations and 169 distinct hand types for the initial two cards. Generally, higher ranked cards have a higher probability to win over lower ranked cards and suited hands are considered better than cards of different suits, although this approach largely depends on other several factors such as a players position from the dealer button, whether the game is limit or no-limit and how many players are at the table, and the chances of winning largely depends on the players ability to decide which action by considering these factors.

Another critical factor when playing Hold'em is the *Hand Strength Assessment* during each stage of the game[8]. For example, suppose we have a hand of A-K and the flop comes A-J-3. We would have a higher chance of winning over a J-K hand and our hand strength will be stronger, but a lower chance to win against a J-3 and thus, a weaker hand. Using a computer, we are able to simulate all possible hands an opponent might have and calculate where our hand ranks among these hands.

A similar but equally important factor to consider is the *Hand Potential*[8]. Consider the previous example where our opponent has J-3. Although our hand strength is weaker, if our hand was suited and two of the cards on the flop were of the same suit, we would have a better potential which would most likely win after the Turn and River. Much like finding our hand strength, we are able to simulate all possible future Turns and Rivers and compare our hand against the possible hands an opponent might have.

Knowing how strong our hand is how do we make an action decision? The general practice is to use *Pot odds*[9]. Pot odds percentage is given by the following equation:

*Pot Odds* = Bet to call / (Total pot + Bet to call)

In other words, pot odds are the ratio of the current size of the pot to the cost of the contemplated call. For example, if the pot contains $30, and a player must call $10 to stay in the hand, then the player has 30-to-10, or 3-to-1 pot odds and a 10 / (30 + 10) = 0.25 or 25% pot odds percentage. This pot odds percentage is compared to the probability of winning a hand with a future card in order to estimate the call's expected value. If the pot odds are 25% and the calculated chances of your hand winning is 30%, then you would call the bet because the pot odds are in your favor (i.e. a larger expected return value).

## 3. New Graph Based Learning AI

Currently there exist a lot of poker programs, such as *Poki*[10][11], *Loki*[11] or *Hyperborean*[12], that uses various AI and machine learning techniques. These programs use an independent opponent modeling structures which do not directly produce an action and relies on estimated value calculations[13]. This results in the performance being too dependent upon the opponent model. This can be a problem when the opponent model is outdated, which is usually the case when not learning in real time. Our goal is to create a flexible and dynamic strategy learning algorithm that incorporates opponent modeling so that it can adapt to different opponents while directly producing an output action. Thus, a need for a new AI algorithm arises that can memorize states of a poker game and learn from the mistakes it makes.

### 3.1. General structure

We need a structure that can hold various states of an environment and our approach is to use undirected graphs. Figure 1 shows the general structure of the graphs, here with two decision choices.

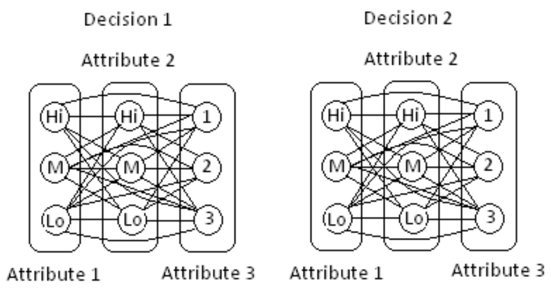

Fig. 1 Graph based learning AI with two decisions

Each graph will represent a single decision. Vertices are grouped together to represent an attribute where each vertex in the group is a possible input for that attribute. The weighted edges will represent the “strength” between the inputs.

Given a set of inputs, we can calculate the summation of the weighted edges that connect these input vertices and get an output value for each graph. We can then compare the graph values to get the strength of each decision when given these inputs.

After the decision is made, we can apply a positive or negative reinforcement to the decision by updating the weights used. For example, when the decision turns out to be wrong, we subtract a reinforcement value from each of the weights used and this will be taken into account the next time any of the same vertices from this set are given as input. Likewise, for a correct decision, we add a positive reinforcement value which will affect the strength of this particular decision the next time a similar input is given. An example with a simplified decision graphs using XOR is shown in Fig 2.

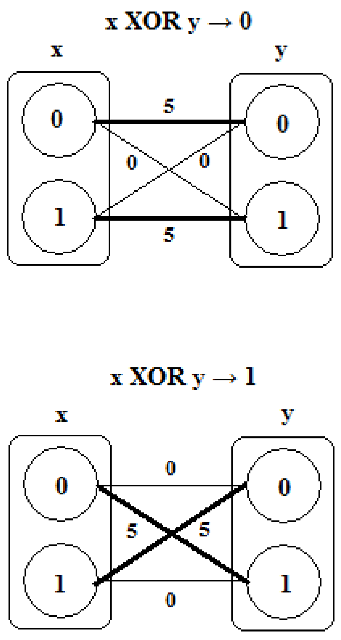

Fig. 2 An XOR example

When inputs x and y are different, the weights of the second graph (5 + 5 = 10) will return a higher value than the first graph (0 + 0 = 0) indicating that x XOR y = 1. On the other hand, if x and y have the same bits, the weights of the first graph will have a larger value indicating that x XOR y = 0.

The intuition behind the structure is that given a set of inputs, the graph with the most successful, or strongly connected, state and most vertices and edges similar to the input relations will return the highest value. The advantage of this graph is that it can memorize a vast number of states and also generate multiple values for each of the contemplated decisions. This is crucial to Hold’em, where each learned opponent model of any given state still holds valuable information that can be used in later games. A disadvantage of this structure is that it requires a long computation time since each decision requires a graph. But since poker generally has 3 decisions: fold, check/call and bet/raise, this should not be much of a problem.

Fig 3 shows this decision graph structure applied to a Call decision for poker.

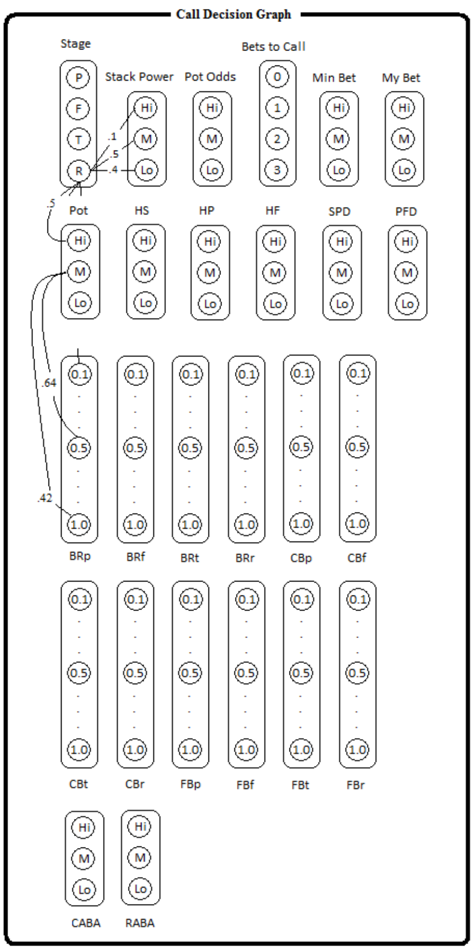

Fig. 3 Decision graph for calls

HS, HP, and HF denote attributes Hand Strength, Hand Potential, and Hand Future, while SPD and PFD denote the Hand Strength / Hand Potential Difference and Hand Potential / Hand Future Difference, respectively. The bottom attributes BR, CB, and FB denote Rate of Bets, Calls to Bets, and Folds to Bets, while the trailing lowercase characters p, f, t, and r stand for each of the stage pre-flop, flop, turn, and river. Finally CABA and RABA denote the Call Amount to Bet Amount and Raise Amount to Bet Amount attributes, all of which will be explained shortly. Also note that although omitted in Figure 3, every vertex in an attribute group has an edge with all the other vertices in different attribute groups.

Because there are a fairly large amount of edges to traverse and read, the graphs were implemented by mapping the edges and weights on to matrices instead of an actual graph structure shown in Fig 3, since the trade off of space for faster computation time is enormous when it comes to a game like poker where a decision is also constrained by time.

The attributes that were used for the graphs can largely be classified into two categories: Hand/State Assessment Inputs and Opponent Modeling Inputs. These two groups of inputs, 26 attributes in total, goes into the graph and finds the strength of the relations between attribute values to determine which decision would be the best to make for any given situation.

## 3.2. Hand and state assessment inputs

There are a total of 12 attributes that represent the state of the game. Table 1 shows these attributes. The first is *stage*, indicating which stage of the game the play is in and has the values of Pre- Flop, Flop, Turn and River.

Table 1. Hand and state assessment inputs and their description

| Inputs | Description |
|---|---|
| Stage | Pre-Flop, Flop, Turn, River |
| Stack Power | L_Stack/(L_Stack + Op_Stack) |
| Pot Odds | Bet / (Pot + Bet) |
| Bets to Call | # of bets that can be made |
| Min Bet | Min. bet to play |
| My Bet | Opponents bet to call |
| Pot | Pot size |
| Hand Strength | Rank of current hand |
| Hand Potential | Rank of possible hand |
| Hand Future | Rank of hand on River |
| SP Difference | Hand Strength–Hand Potential |
| PF Difference | Hand Potential–Hand Future |

Stack Power is the ratio of the learning agents stack amount over the entire amount of money in the game, give by:

$$\textit{Stack Power} = \text{L\_STACK} / (\text{L\_STACK} + \text{OP\_STACK})$$

where L_STACK is the learning agents stack and OP_STACK is the opponents stack. This will gauge how well the learning agent is doing in the game and allow the agent to change its play style accordingly, since the agent must play tight when short stacked and bet aggressively when it is winning.

*Pot odds*, as explained before, is given as an input so the agent can decide whether calling a bet or raise is worth the risk.

*Bets to call* is the number of bets that can be made. This is a good indicator to how much commitment the opponent is making to the pot or to a particular hand. When there are no bets to call, the opponent has made a full commitment to the hand and will most likely have a good had. When there are two or more bets to call, the opponent either has a weak hand or is slow playing a strong hand.

*Min Bet* is the minimum bet that has to be called for the learning agent to stay in play and *My Bet* is the amount that the opponent has to call to stay in. *Pot* is how much money is currently in the pot. These attributes were used to differentiate situations and learn how the opponent makes a decision or how the learner should respond in these situations.

*Hand Strength, Hand Potential* and *Hand Future* are the current hand strength, the hand strength given all possible cards that can turn up the next stage, and the calculated hand strength for the stage after, and are by far the most important factors in the game. Note that the Hand Future value is not used during a Turn or Flop and Hand Potential is not used during the River. These values are calculated by using a Hand Evaluator and enumerating every possible hand the opponent might have and all possible hands that can come in the next stages of the game, then comparing it against the learner's hand. Because of the tedious calculations and vast number of possible hands, this required a fast computational algorithm. This was implemented using Paul D. Senzee's pre-computed perfect hash function on top of Cactus Kev's Poker Hand Evaluation technique. Because there are 2.6 million unique hands, learning algorithms cannot be computed efficiently without some type of hand evaluator. The basic idea is that of these 2.6 million unique playable hands, there are only 7462 distinct poker hand values when grouped by their ranking and this abstraction from unique to distinct allows faster calculation and less training time for the learner. The trade off is that the abstraction lessens the accuracy of the learner since there is a certain level of information loss when converting to distinct hands, but the disadvantages are minimal compared to the benefits.

Finally, the *SP Difference* and *PF Difference* are the difference between the hand strength and hand potential, and the hand potential and hand future, respectively. This is a good indicator for the possibility of straights or flushes that can be overlooked by the hand strength, potential and future.

### 3.3. Opponent modeling inputs

It has been shown that using opponent models produce significant improvement in performance[17], not just in the game of poker, but in other games as well[18][19]. The graph algorithm also takes into account the play style of an opponent, but "embeds" this into the algorithm as a whole through inputs, rather than having a separate model. There are a total of 14 attributes used that distinguishes an opponent. Table 3 shows these attributes.

Table 2. Opponent Modeling Inputs (Attributes) and their description

| Inputs | Descriptions |
|---|---|
| Rate of Bets on Pre-Flop | # of Bets / # of Pre-Flops |
| Rate of Bets on Flop | # of Bets / # of Flops |
| Rate of Bets on Turn | # of Bets / # of Turns |
| Rate of Bets on River | # of Bets / # of Rivers |
| Calls to Bets Pre-Flop | # of Calls / # of Pre-Flop Bets |
| Calls to Bets on Flop | # of Calls / # of Flop Bets |
| Calls to Bets Turn | # of Calls / # of Turn Bets |
| Calls to Bets River | # of Calls / # of River Bets |
| Folds To Bets on Pre-Flop | # of Folds / # of Pre-Flop Bets |
| Folds To Bets on Flop | # of Folds / # of Flop Bets |
| Folds To Bets on Turn | # of Folds / # of Turn Bets |
| Folds To Bets on River | # of Folds / # of River Bets |
| Call Amount to Bet Amount | Call Amount / Bet Amount |
| Raise Amount to Bet Amount | Raise Amount / Bet Amount |

The *Rate of Bets* is the ratio of bets made by an opponent in a particular stage and the number of those stages played. The *Calls to Bets* value is the ratio of Calls made by an opponent to the Bets made by the learner, for each stage. The *Folds to Bets* value is the ratio of opponent folds to the learner's bets. There are values for each of the four stages and this will help the learner determine its opponents play style. For example, a low Rate of Bets value in the Flop stage and a high value in the River stage could be a good indicator that the opponent likes to slow play strong hands. A high Folds to Bets value in the Flop stage could indicate that the opponent does not chase cards and is thus a conservative type player.

There are two more attributes, the *Call Amount to Bet Amount* and the *Raise Amount to Bet Amount*, which is the ratio of the opponents call and raise amounts to the learners bet amount, respectively. This shows on average how much an opponent is willing to call or how much an opponent will raise given a bet. It also provides insight into the opponent for the learner.

### 3.4. Graph based AI applied to poker

By applying our graph based AI to the poker program we can assign a graph to each action. 3 types of poker playing programs were implemented using this AI. Table 4 gives us a list of the implemented programs.

Table 3. Implemented AIs and Opponents

| Learning AI | Opponent |
|---|---|
| 6-Value Based AI | Aggressive |
| 3-Value Based AI | Neutral |
| 3-Value Based AI with Base Strategy | Conservative |
|  | Bluff / Base Strategy |

The 6-Value Based AI uses 6 graphs, each corresponding to fold, check/call, bet/raise 0.1, bet/raise 0.2, bet/raise 0.3 and bet/raise 0.4. The bet/raise actions will match the bet so that the pot odds for the opponent will match its following decimal number. For example, if bet/raise 0.2 was chosen with 30$ in the pot, then a bet will be made such that the pot odds for the opponent will be 0.2, so the learner will make a bet/raise of 10$:

$$\textit{Opponent Pod Odds} = \text{Bet} / (30 + 2 * \text{Bet}) = 0.2$$

$$B = 30 / 3 = 10\$$$

The 3-Value Based AI is similar to the 6-Value Based AI but it uses 3 graphs, each for fold, check/call, bet/raise instead and is used for limit Hold'em.

The 3-Value Based AI with Base Strategy uses the 3-Value Based AI values and a base strategy system with hard coded values, and the two values are combined at to produce a decision. Figure 4 shows the architecture of this program.

The attributes are mapped to 5 values: High, High-Medium, Medium, Medium-Low and Low, then used as input for the graphs in our AI structure.

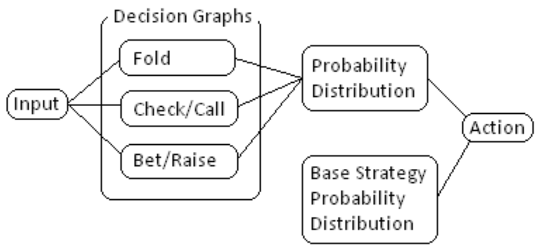

Fig. 4 3-value based AI with base strategy

Each graph will return a calculated value based on the weights between the inputs and we can obtain a probability distribution from these values[11]. Then, by using a random number generator, we select an action according to its probability distribution. So if a value of 40 was returned for fold, 60 was returned for check/call and 100 was returned for bet/raise, then we fold 20% of the time ( 40 / (40 + 60 + 100) = 0.2 ), check or call 30% of the time and bet or raise 50% of the time. The advantages of using a probability distribution is that this will keep the actions random enough so that an opponent cannot easily find out the strategy used and this will also prevent getting stuck at a local maximum when searching for an optimal action. We continue this for every betting round of each stage of the game until there is a winner. If round ends with a winner, then we train the decision graphs with a reinforcement value according to the pot amount that was won or lost. A large winning will give a strong positive reinforcement for the actions taken throughout the round and the same value will be distributed among actions not taken as negative reinforcement. This is to maintain a balance of values between the decision graphs and make sure that there is a lesser probability that other actions are taken. Likewise, a loss, with the exception of folding, will feed back a negative reinforcement so that there will be a lesser chance that the learner makes the same decisions in similar situations, while the same value will be distributed among other actions as a positive reinforcement. When the learner folds, there is no reinforcement, since in poker, there is no way of knowing if the decision to fold was right or wrong without looking at the opponents hand.

This AI will require a long time to train compared to opponent modeling poker programs since rather than learning an opponent model, it will start by learning how to play poker and will also learn what strategy to use against specific opponents and which attributes to look for. Then it uses this knowledge against new opponents by matching the opponent type and game state to similar situations encountered during training. Unlike other poker bots that use machine learning instances only when in play, this poker program opts for more of a long term solution by training its self from a blank state and retaining those experiences for future use. Every game it plays will also be used as training data at the end of the game, so it can dynamically adapt to each individual game too.

A problem this algorithm has is determining the accuracy and errors. Because of the probabilistic nature of Hold'em, even without the imperfect information aspects of the game, it is still impossible to find out whether an action was an optimal decision or not[14]. For example, winning just the blinds with a strong hand such as a four of a kind is still a win, but it would not be an optimal solution since decisions to increase the winning pot size were not taken.

## 4. Experiments and Analysis

The experiments were conducted with a 1 on 1 heads up game against each of the opponents to eliminate the decision factors arising when playing multiple opponents. Also, Pre-Flop decisions were eliminated and all players were made to check or call all Pre-flop hands in order to prevent folding the majority of Pre-Flop hands, which is often the case during heads up games, and also to maximize the number of actual plays in different situations.

To obtain meaningful empirical results for the poker programs, it is necessary to run a series of experiments against different opponents. Four opponents each with different play styles were used for the experiments: Aggressive Opponent, Neutral Opponent and Conservative Opponent, each of their names being self explanatory. Table 3 gives a list of the Opponents implemented. Aggressive Opponent bets/raises aggressively, even with a considerably weak hand, while Conservative Opponent only bets/raises when it has a very strong hand and will fold most bets, and Neutral Opponent falls somewhere between the two. The percentile distribution for each of the opponents actions were determined by empirical data based on what is normally seen in common poker games. No formulae were used to determine these distributions and in real games, they are often different depending on the level of poker skill the poker table shows. Base Opponent was implemented as a neutral type player with a tendency to bluff with weak hands, and its play strategy was also used as the base for the 3-Value Based AI with Base Strategy program.

Many experiments were performed to establish reliable results and, and only a cross-section of those tests are presented here.

### 4.1. Trained 6-value AI vs. opponents

The 6-Value AI was trained against each of its opponents for exactly 200 rounds before facing its opponents one by one for another 200 rounds. The results are shown in Fig 5 and Fig 6.

The AI played worse against the Neutral and Conservative opponents but compensated by showing a stronger performance against the Aggressive opponent. This was because the AI combined its experiences against each of the opponents from training in an attempt to converge to, or more likely average out to, an optimal strategy that plays well against all the opponents.

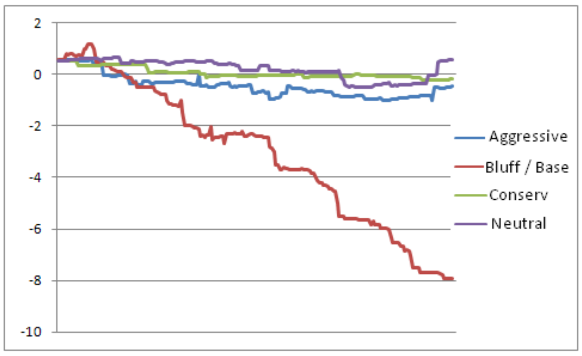

Fig. 5 Winning pot size ratio of 6-Value Poker AI vs. Opponents after training 200 rounds with each opponent

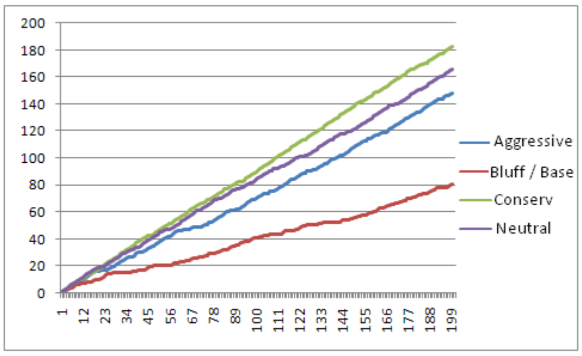

Fig. 6 The number of rounds the trained 6-Value Poker AI won

By training against multiple opponents, the graph structures should have updated the weights of the common vertices that affect all types of play, such as hand strength and potential, while keeping the opponent modeling vertices stale, and thus increase play performance. But this was not the case as the winning pot size ratio hovered around 0.0 with a slight increase at the end. This seemed to be an indicator that there might have been too many decision graphs compared to the number of attribute values supplied, and this aspect might have not been able to show its full potentials within the 200 rounds.

It is also important to note that the 6-Value Poker AI still played poorly, if not worse, against the Base opponent's random bluffing style strategy, and multiple training sessions did not fix the problem.

### 4.2. Trained 3-value poker AI vs. opponents

The 3-Value Poker AI structure was trained 200 rounds against each opponent then played against each opponent for an additional 200 rounds separately.

Similar to the first experiment, training once again had the averaging out effect on the 3-Value Poker AI. Notice from Fig 7 that the performance against the Neutral and Conservative opponent slightly decreased, while there was a much larger performance boost against both the Base and Aggressive opponent.

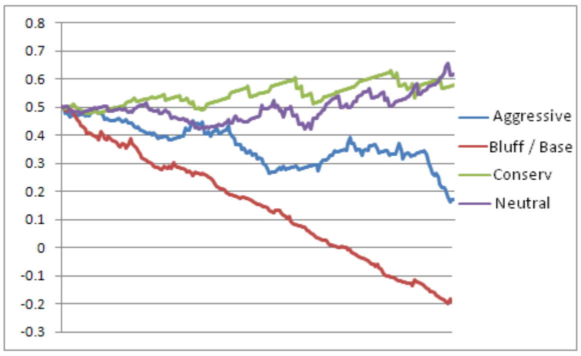

Fig. 7 Winning pot size ratio of 3-Value Poker AI vs. Opponents after training 200 rounds with each opponent

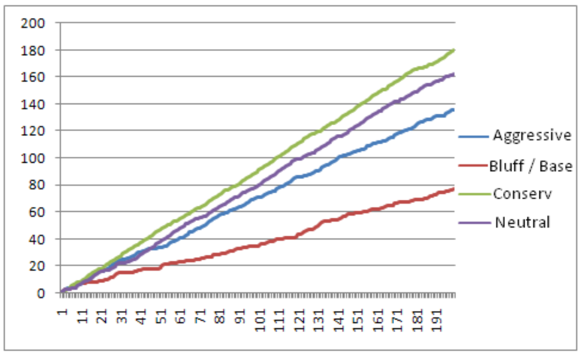

Fig. 8 The number of rounds the trained 3-Value Poker AI won

While still not generating a profit, the play strategy becomes more stable and can hold out for a longer time.

Since the graph based structure requires a lot of training example while having a wide mixture of various different opponents to distinguish which attribute vertices are important in which situation, we shift our focus to creating program so that it could use the 3-Value Poker AI as a booster to its own strategy so that the AI will not need to satisfy the requirement and still be useful. Because the Bases opponents bluffing strategy was the hard opponent to win against, we will choose the Base opponent as the base strategy to add on to our AI (hence the name Base).

### 4.3. Trained 3-valued poker AI with base strategy vs. opponents

In our final experiment, we trained the AI along with the Base Strategy for 200 rounds with each opponent then had it face each of the opponents for another 200 rounds separately. Figure 9 and Figure 10 shows the surprising results.

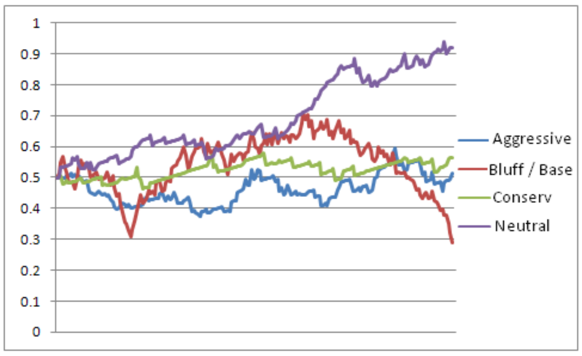

Fig. 9 Winning pot size ratio of 3-Valued Ai with Base Strategy vs. Opponents after training 200 rounds with each opponend

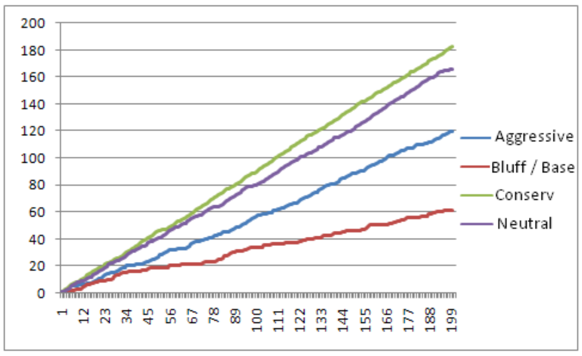

Fig. 10 The number of hands won by 3-Value AI with Base Strategy

The AI begins to return profit and does not even get close to losing. The AI also makes correct decision and converges to a near optimal value. For example, when playing against the Conservative Opponent, the AI continues to bet and raise hard because it knows that the Conservative Opponent will fold near 90% of the time. Once the Conservative Opponent bets or raises, the AI knows it has a very strong had and proceeds to fold.

Although it will require more training to create a stronger AI, through these experiments shows that the graphed based learning structure has great potentials to improve.

## 5. Conclusion and Future Work

These experiments show that the presented graph based learning algorithm can be used for poker. Nevertheless, this algorithm is the first time being used and is in its infant stage. It has a lot of areas that needs to be worked on. One such are would be the mapping of continuous values to a discrete number of vertices. One idea would be to use fuzzy logic and use functions instead of vertices. Also some aspects of poker still not researched to the full extent in this paper were pre-flop strategies and playing against multiple opponents. The goal of the research was to find out how the new graph based learning AI can be used and applied to poker, and in a more general sense, how well machine learning performs in situations of imperfect information. For those reasons, a lot of detailed aspects of poker have been omitted and there is still plenty of room for improvement in this new learning structure.

The new graph based learning AI successfully found winning strategies to play against different opponents and showed that it can improve over time. However, this does not necessarily mean that it will be effective against human opponent. Humans naturally have the most sophisticated opponent modeling abilities and can easily change their play style. This creates a reason to investigate the usage performance of machines on humans. As seen in the experiments, random play style was not well classified in general and this can relate to real life poker situations where human factors set in. Such would be fatigue, short play time or dynamically changing opponents at a given table that result in changing styles for the learner. Also, different strategies which were not included in the experiment, such as slow plays, strutting fishing etc, could affect the performance of the learner.

Poker has a limitless amount of research topics and as long as poker is around, the research will continue on.

## References

[1] Fürnkranz, J., "Machine Learning in Computer Chess: The Next Generation"*, Austrian Research Institute for Artificial Intelligence*, Vienna, TR-96-11, 1996.

[2] Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P., Szafron, D., "A World Championship Caliber Checkers Program"*, Artificial Intelligence*, Vol. 53, No. 2-3, pp.273-290, 1992.

[3] Schaeffer, J., Burch, N., Björnsson, Y., Kishimoto, A., Müller, M., Lake, R., Lu, P., Sutphen, S., "Checkers is Solved", *Science*, Vol. 317, No. 5844, pp.1518 – 1522, September 2007.

[4] Wu, L., Baldi, P., "A scalable machine learning approach to Go", *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt, and T. Hoffman (ed.), Cambridge, MA: MIT Press, pp. 1521-1528, 2007.

[5] Buro, M., "The Evolution of Strong Othello Programs", *Entertainment Computing - Technology and Applications*, R. Nakatsu and J. Hoshino (ed.), Kluwer, pp.81-88, 2003.

[6] Buro, M., "How Machines have Learned to Play Othello", *IEEE Intelligent Systems*, Vol. 14(6), pp.12-14, 1999.

[7] Slansky, D., Malmuth, M., *Hold'em Poker for Advanced Players,*Two Plus Two Publishing, 1994.

[8] Billings, D., Papp, D., Schaeffer, J. Szafron, D., "Opponent Modeling in Poker", *AAAI'98* , Madison, Wisconsin, pp.493-499, 1998.

[9] Slansky, D., *The Theory of Poker,* Two Plus Two Publishing, 1999.

[10] Chopra, A., "Knowledge and Strategy-based Computer Player for Texas Hold'em Poker", *MS Thesis*, University of Edinburgh, 2006.

[11] Billings, D., Davidson, A., Schaeffer, J. Szafron, D., "The Challenge of Poker", *Artificial Intelligence*, Vol. 134(1-2), pp.201-240, 2002.

[12] Johanson, M., "Robust Strategies and Counter-Strategies:

Building a Champion Level Computer Poker Player", *MS Thesis*, University of Alberta, 2007.

[13] Chen, B., Ankenman, J., *The Mathematics of* Poker, ConJelCo Publishing, 2006.

[14] Darse Billings, *"Algorithms and assessment in Computer Poker"*, *Ph.D. dissertation*, University of Alberta, 2006.

[15] Davidson, A., Billings, D., Shaeffer, J., Szafron, D., "Improved Opponent Modeling in Poker", *ICAI'2000* , Las Vegas, Nevada, pp.1467-1473, 2000.

[16] Billings, D., Papp, D., Schaeffer, J., Szafron, D., "Poker as a Testbed for AI Research", *AI'98* ,Vancouver, BC, Canada, Vol. 1418, pp.228 – 238, 1998.

[17] Lazaric, A., Quaresimale, M., Restelli, M., "On the usefulness of opponent modeling: the Kuhn Poker case study", *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems*, Estoril, Portugal, Vol. 3, Pages: 1345-1348, 2008.

[18]Richards, M., Amir, E., "Opponent modeling in scrabble", *IJCAI*, pp. 1482–1487, 2007.

[19] Sheppard, B., "World-championship-caliber scrabble", *Artificial Intelligence*, 134(1-2):241–275, 2002.

**Seong-Gon Kim** received the B.S. degree in computer science from University of Illinois at Urbana-Champaign, Illinois, U.S.A in 2008, and the M.S. degree in computer science and engineering from University of Florida, Florida, U.S.A. in 2010. He is currently a Researcher/Engineer in LG Electronics User Platform Lab. His current research interests include machine learning, pattern recognition, and intelligent robotics.

**Yong-Gi Kim** received the B.S. degree in electrical engineering from Seoul National University, Seoul, Korea in 1978, the M.S. degree in computer science from University of Montana, U.S.A. in 1987, and the Ph.D. degrees in computer and information sciences from Florida State University, U.S.A. in 1991. He was a Visiting Scholar in the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Illinois, U.S.A., from 2008 to 2009. He is currently a Professor in the Department of Computer Science, Gyeongsang National University, Korea. His current research interests include soft computing, intelligent systems and autonomous underwater vehicles.