Learning to Play Strong Poker

Jonathan Schaeffer, Darse Billings, Lourdes Peña, Duane Szafron
Department of Computing Science
University of Alberta
Edmonton, Alberta Canada T6G 2H1
{jonathan, darse, pena, duane}@cs.ualberta.ca

Abstract

Poker is an interesting test-bed for artificial intelligence research. It is a game of imperfect knowledge, where multiple competing agents must deal with risk management, opponent modeling, unreliable information, and deception, much like decision-making applications in the real world. Opponent modeling is one of the most difficult problems in decision-making applications and in poker it is essential to achieving high performance. This paper describes and evaluates the implicit and explicit learning in the poker program *Loki*. *Loki* implicitly "learns" sophisticated strategies by selectively sampling likely cards for the opponents and then simulating the remainder of the game. The program has explicit learning for observing its opponents, constructing opponent models and dynamically adapting its play to exploit patterns in the opponents' play. The result is a program capable of playing reasonably strong poker, but there remains considerable research to be done to play at a world-class level.

1. INTRODUCTION

The artificial intelligence community has recently benefited from the tremendous publicity generated by the development of chess, checkers and Othello programs that are capable of defeating the best human players. However, there is an important difference between these board games and popular card games like bridge and poker. In the board games, players always have complete knowledge of the entire game state since it is visible to both participants. This property allows high performance to be achieved by a brute-force search of the game tree. In contrast, bridge and poker involve imperfect information since the other players' cards are not known, and search alone is insufficient to play these games well. Dealing with imperfect information is the main reason why progress on developing strong bridge and poker programs has lagged behind the advances in other games. However, it is also the reason why these games promise higher potential research benefits.

Poker has a rich history of scientific investigation. Economists and mathematicians have applied a variety of analytical techniques to certain poker-related problems. However, since "real" poker is too complex for this approach, they have studied simplified variants ([15] for example). Other individuals, including expert players with a penchant for mathematics, have gained considerable insight about "real" poker by using partial mathematical analyses, simulation, and *ad-hoc* expert experience ([18] is a popular example).

Until recently, the computing science community has largely ignored poker. However, poker has a number of attributes that make it an interesting domain for artificial intelligence (AI) research. These attributes include imperfect knowledge, multiple competing agents, risk management, opponent modeling, deception, and dealing with unreliable information. All of these are challenging dimensions to a difficult problem.

There are two ways that poker can be used as an interesting testbed for artificial intelligence research. One approach is to use simplified variants that are easier to analyze. For example, Findler worked on and off for 20 years on a poker-playing program for simplified 5-card draw poker [7]. He modeled human cognitive processes and built a program that could learn. The danger with this approach is that simplification can remove the challenging components of the problem that are of interest to AI researchers. A variant of this approach is to look at a subset of the game, and try to address each component in isolation. Several attempts have been made to apply machine learning techniques to individual aspects of poker (some examples include [19,21,6]).

The second approach, and the one that we advocate, is to tackle the entire problem: choose a real variant of poker and address all the considerations necessary to build a program that performs at a level comparable to that of the best human players. Clearly this is the most ambitious approach, but also the one that promises the most exciting research opportunities.

Recently, Koller and Pfeffer have been investigating poker from a theoretical point of view [13]. They present a new algorithm for finding optimal randomized strategies in two-player imperfect information competitive games. This is done in their *Gala* system, a tool for specifying and solving problems of imperfect information. Their system builds decision trees to find the optimal game-theoretic strategy. However the tree sizes prompted the authors to state that "...we are nowhere close to being able to solve huge games such as full-scale poker, and it is unlikely that we will ever be able to do so." In theory, their approach could be used to build an optimal poker player for a real variant of poker. In practice, it will require too many computational resources unless further improvements are discovered.

We are attempting to build a program that is capable of playing world-class poker. We have chosen to study the game of Texas Hold'em, the poker variation used to determine the world champion in the annual World Series of Poker. Hold'em is considered to be the most strategically complex poker variant that is widely played.

Our experiences with our first poker program, called *Loki,* were positive [1,14]. However, we quickly discovered two limitations to further performance gains:

1 The betting strategy—whether to fold, call, or raise in a given situation—was defined using expert knowledge. This became cumbersome, since it was awkward to define rules to cover all the possible scenarios. Furthermore, any static strategy is suspect. A successful strategy must depend on changing game conditions.

2 Initially, in games played over the Internet, *Loki* performed quite well. However, some opponents detected patterns and weaknesses in *Loki*'s play, and they altered their strategy to exploit them. An opponent can exploit any predictable strategy, both in theory and in practice. To be a strong poker player, one must model the opponent's play and adjust to it.

This paper describes and evaluates two types of learning in *Loki.* First, its knowledge-based betting strategy can be viewed as a static evaluation function. In two-player games, such as chess, the quality of the evaluation can be improved through search. In poker, imperfect information makes a search of the full game tree impractical. Instead, a simulation that samples from the set of likely scenarios can be used to enhance an evaluation. We found that a simple evaluation function augmented by search can uncover sophisticated strategies, as has been observed in perfect-information games. In other words, search compensates for a lack of knowledge. In effect, *Loki* uses simulations to implicitly learn advanced strategies.

Second, *Loki* observes and records the actions of each opponent and uses this information to build a simple model of their play. This model is used to predict each opponent's hidden cards. The program adapts to the style of each opponent and exploits any predictable actions.

We have experimentally assessed each of these styles of learning, both in the laboratory and in play with human opponents. To the best of our knowledge, *Loki* is the first successful demonstration of using real-time learning to improve performance in a high-performance game-playing program.

This paper describes our previous work on *Loki* [1,2,3,4,14] and outlines some of the future directions we are pursuing. Section 2 provides an overview of Texas Hold'em. Section 3 identifies the minimal set of requirements necessary to achieve world-class play. *Loki*'s architecture is described in Section 4. Section 5 discusses the implicit learning used in the betting strategy, while Section 6 addresses the explicit opponent modeling. The performance of the program is assessed in Section 7. Section 8 identifies future work, and Section 9 provides some conclusions.

2. TEXAS HOLD'EM

A hand of Texas Hold'em begins with the *pre-flop*, where each player is dealt two *hole cards* face down, followed by the first round of betting. Three community cards are then dealt face up on the table, called the *flop*, and the second round of betting occurs. On the *turn*, a fourth community card is dealt face up and another round of betting ensues. Finally, on the *river*, a fifth community card is dealt face up and the final round of betting occurs. All players still in the game reveal their two hole cards for the *showdown*. The best five-card poker hand formed from the two hole cards and the five community cards wins the pot. If a tie occurs, the pot is split. Texas Hold'em is typically played with 8 to 10 players.

Limit Texas Hold'em uses a structured betting system, where the order and amount of betting is strictly controlled on each betting round.[1] There are two denominations of bets, called the small bet and the big bet ($2 and $4 in this paper). In the first two betting rounds, all bets and raises are $2, while in the last two rounds, they are $4. In general, when it is a player's turn to act, one of five betting options is available: fold, call/check, or raise/bet. There is normally a maximum of three raises allowed per betting round. The betting option rotates clockwise until each player has matched the current bet or folded. If there is only one player remaining (all others having folded) that player is the winner and is awarded the pot without having to reveal their cards.

1 In No-limit Texas Hold'em, there are no restrictions on bet sizes.

3. REQUIREMENTS FOR A WORLD-CLASS POKER PLAYER

We have identified several key components that address some of the required activities of a strong poker player. However, these components are not independent. They must be continually refined as new capabilities are added to the program.

Hand strength assesses the strength of a hand in relation to the other hands. The simplest hand strength computation is a function of the cards in the hand and the current community cards. A better hand strength computation takes into account the number of players still in the game, the position of the player at the table, and the history of betting for the current game. An even more accurate calculation considers the probabilities for each possible opponent hand, based on the likelihood of each hand being played to the current point in the game.

Hand potential computes the probability that a hand will improve to win, or that a leading hand will lose, as additional community cards appear. For example, a hand that contains four cards in the same suit may have a low hand strength, but has good potential to win with a flush as more community cards are dealt. Conversely, a hand with a high pair could decrease in strength and lose to a flush as many cards of a common suit appear on the board. At a minimum, hand potential is a function of the cards in the hand and the current community cards. However, a better calculation could use all of the additional factors described in the hand strength computation.

Betting strategy determines whether to fold, call/check, or bet/raise in any given situation. A minimum strategy is based on hand strength. Refinements consider hand potential, pot odds (your winning chances compared to the expected return from the pot), bluffing, opponent modeling and trying to play unpredictably.

Bluffing allows you to make a profit from weak hands,[2] and can be used to create a false impression about your play to improve the profitability of subsequent hands. Bluffing is essential for successful play. Game theory can be used to compute a theoretically optimal bluffing frequency in certain situations. A minimal bluffing system merely bluffs this percentage of hands indiscriminately. In practice, you should also consider other factors (such as hand potential) and be able to predict the probability that your opponent will fold in order to identify profitable bluffing opportunities.

Unpredictability makes it difficult for opponents to form an accurate model of your strategy. By varying your playing strategy over time, opponents may be induced to make mistakes based on an incorrect model.

Opponent modeling allows you to determine a likely probability distribution for your opponent's hidden cards. A minimal opponent model might use a single model for all opponents in a given hand. Opponent modeling may be improved by modifying those probabilities based on the collected statistics and betting history of each opponent.

There are several other identifiable characteristics that may not be necessary to play reasonably strong poker, but may eventually be required for world-class play.

Opponent modeling is integral to successful poker play. Koller and Pfeffer have proposed a system for constructing a game-theoretic optimal player [13]. However, it is important to differentiate an *optimal* strategy from a *maximizing* strategy. The optimal player makes its decisions based on game-theoretic probabilities, without regard to specific context. The maximizing player takes into account the opponent's sub-optimal tendencies and adjusts its play to exploit these weaknesses.

In poker, a player that detects and adjusts to opponent weaknesses will win more than a player who does not. For example, against a strong conservative player, it would be correct to fold the probable second-best hand. However, against a weaker player who bluffs too much, it would be an error to fold that same hand. In real poker it is very common for opponents to play sub-optimally. A player who fails to detect and exploit these weaknesses will not win as much as a better player who does. Thus, a maximizing program will out-perform an optimal program against sub-optimal players.

Although a game-theoretic optimal solution for Hold'em would be interesting and provide a good baseline for comparing program (and human) performance, it would in no way "solve the game." To produce a world-class poker program, strong opponent modeling is essential.

4. *LOKI*'S ARCHITECTURE

This section gives a brief overview of the important components of *Loki*'s architecture [4]. Figure 1 illustrates how these components interact. In the diagram, rectangles are major components, rounded rectangles are major data structures, and ovals are actions. The data follows the arrows between components. An annotated arrow indicates how many times data moves between the components for each of our betting actions.

The architecture revolves around generating and using *probability triples*. It is an ordered triple of values, PT = $[f,c,r]$, such that $f + c + r = 1.0$, representing the probability distribution that the next betting action in a given context should be a fold, call, or raise, respectively. The Triple Generator contains our poker knowledge, and is analogous to an evaluation function in two-player games. The Triple Generator calls the Hand Evaluator to evaluate any two-card hand in the current context. It uses the resulting hand value, the current game state, and expert-defined betting rules to compute the triple. To evaluate a hand, the Hand Evaluator enumerates over all possible opponent hands and counts how many of them would win, lose or tie the given hand.

[2] Other forms of deception (such as calling with a strong hand) are not considered here.

Each time it is *Loki*'s turn to bet, the Action Selector uses a single probability triple to decide what action to take. For example, if the triple [0.0,0.8,0.2] were generated, then the Action Selector would never fold, call 80% of the time and raise 20% of the time. A random number is generated to select one of these actions so that the program varies its play, even in identical situations. Although this is analogous to a *mixed strategy* in game theory, the probability triple implicitly contains contextual information.

After the flop, the probability for each possible opponent hand is different. For example, the probability that Ace-Ace hole cards are held is much higher than the cards 7-2, since most players will fold 7-2 before the flop. There is a weight table for each opponent. Each weight table contains one value for each possible two-card hand that the opponent could hold. The value is the probability that the hand would be played exactly as that opponent has played so far. For example, assume that an opponent called before the flop. The updated probability value for the hand 7-2 might be 2% since it normally should be folded. Similarly the probability of Ace-King might be 60% since it would seldom be folded before the flop, but is often raised. After an opponent action, the Opponent Modeler updates the Weight Table for that opponent in a process called *re-weighting*. The value for each hand is increased or decreased to be consistent with the opponent's action. The Hand Evaluator uses the Weight Table in assessing the strength of each possible hand, and these values are in turn used to update the Weight Table after each opponent action. The absolute values of these probabilities are of little consequence, since only the relative weights affect the later calculations. The details are discussed in Section 6.

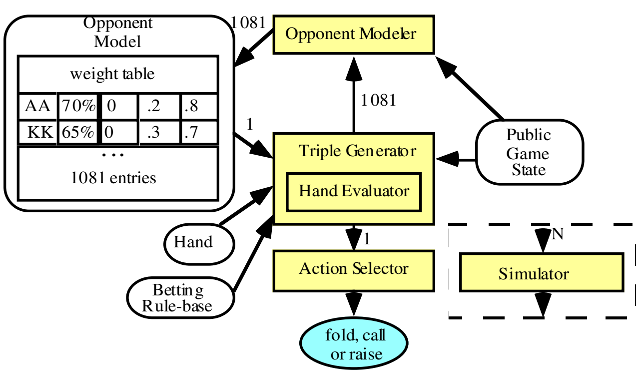

Figure 1. The architecture of *Loki*.

Probability triples are used in three places in *Loki.* The Action Selector uses a probability triple to decide on a course of action (fold, call, raise) as previously described. The Simulator uses probability triples to choose actions for simulated opponent hands (see Section 5). The Opponent Modeler uses an array of probability triples to update the model of each opponent (see Section 6).

An important advantage of the probability triple representation is that imperfect information is restricted to the Triple Generator and does not affect the rest of the program. This is similar to the way that alpha-beta search restricts knowledge to the evaluation function. The probability triple framework allows the "messy" elements of the program to be amalgamated into one component, which can then be treated as a "black box" by the rest of the system. Thus, aspects like game-specific information, complex expert-defined rule systems, and knowledge of human behavior are all isolated from the engine that uses this input.

5. IMPLICIT LEARNING

Loki's original Action Selector component consisted of expert-defined rules that used hand strength, hand potential, game conditions, and probabilities to decide on an action. A professional poker player defined the system as a first approximation of the return on investment for each betting decision. As other aspects of *Loki* improved, this simplistic betting strategy became the limiting factor to the playing strength of the program. Unfortunately, any rule-based system is inherently rigid, and even simple changes were difficult to implement and verify for correctness. A more flexible, computation-based approach was needed.

In effect, a knowledge-based betting strategy is equivalent to a static evaluation function. Given the current state of the game and the hole cards, it attempts to determine the action that yields the best result. If we use deterministic perfect information games as a model, the obvious extension is to add search to the evaluation function. While this is easy to achieve in a perfect-information game such as chess (consider all possible moves as deeply as resources permit), the same approach is not feasible for real imperfect information games because there are too many possibilities to consider [13].

Having an expert identify all the betting rules necessary to play world-class poker is time consuming and difficult. Decisions must be based on context, within a game and between games. Covering all eventualities is not practical. In such a system, the expert does the learning, transferring his knowledge into new or modified rules. We prefer a dynamic computation-based approach, where the program does the learning as it plays.

Loki's improved betting strategy consists of playing out many likely scenarios to determine how much money each decision will win or lose. Every time it faces a decision, *Loki* invokes the Simulator to get an estimate of the *expected value* (EV) of each betting action (see the dashed box in Figure 1 with the Simulator replacing the Action Selector). A simulation consists of playing out the hand a specified number of times, from the current state of the game through to the end. Folding is considered to have a zero EV, because we do not make any future profit or loss. Each trial is played out twice—once to consider the consequences of a check/call and once to consider a bet/raise. In each trial, cards are dealt to each opponent (based on the probabilities maintained in the Weight Table), the resulting game is simulated to the end, and the amount of money won or lost is determined. Probability triples are used to simulate the actions of the opponents based on the two cards they are assigned for that trial. The average over all of the trials is taken as the EV of each action. In the current implementation we simply choose the action with the greatest expectation, folding if both expectations are negative. If two actions have the same expectation, we opt for the most aggressive one (call over fold and raise over call). To increase the programs unpredictability, we can randomize the selection of betting actions whose EVs are close in value.

Enumerating all possible opponent hands and future community cards is analogous to exhaustive game tree search and is impractical for poker. However, simulation is analogous to a selective expansion of some branches of a game tree. To get a good approximation of the expected value of each betting action, one must have a preference for expanding and evaluating the nodes that are most likely to occur. To obtain a correctly weighted average, all of the possibilities must be considered in proportion to the underlying probability distribution of the opponent hands and future community cards. The distribution of future community cards is uniform across unseen cards, but the probable opponent hands are not! We use *selective sampling* to select the most probable hands for each opponent.

When simulating a hand, we have specific information that can be used to bias the selection of cards. For example, a player who has been raising is more likely to have a strong hand than a player who has just called every bet. For each opponent, *Loki* maintains a probability distribution over the entire set of possible hands (the Weight Table), and the random generation of each opponent's hole cards is based on those probabilities. Thus, we are biasing our selection of hole cards for the opponent to the ones that are most likely to occur.

At each node in the decision tree, a player must choose between one of three alternatives. Since the choice is strongly correlated to the quality of the cards that they have, we can use the Triple Generator to compute the likelihood that the player will fold, check/call, or bet/raise in each instance. The player's action is then randomly selected based on the probability distribution, and the simulation proceeds. As shown in Figure 1, the Simulator calls the TripleGenerator to obtain each of our betting actions and each of our opponent actions. Where two actions are equally viable, the resulting EVs should be nearly equal, so there is little consequence if the "wrong" action is chosen.

It should be obvious that the simulation approach must be better than the static approach, since it essentially uses a selective search to augment and refine a static evaluation function. Barring a serious misconception (or bad luck on a limited sample size), playing out relevant scenarios will improve the default values obtained by heuristics, resulting in a more accurate estimate.

As seen in other search domains, the search itself contains implicit knowledge. A simulation contains inherent information that improves the basic evaluation. For example, a simulation contains implicit knowledge such as:

- hand strength (fraction of trials where our hand is better than the one assigned to the opponent),
- hand potential (fraction of trials where our hand improves to the best, or is overtaken), and
- subtle implications not addressed in the simplistic betting strategy (e.g. "implied odds"—extra bets won after a successful draw).

It also allows complex strategies to be *uncovered without providing additional expert knowledge*. For example, simulations can result in the emergence of advanced betting tactics like a check-raise, even if the basic strategy without simulation is incapable of this play.

At the heart of the simulation is an evaluation function. The better the quality of the evaluation function, the better the simulation results will be. One of the interesting results of work on alpha-beta has been that even a simple evaluation function can result in a powerful program. We see a similar situation in poker. The implicit knowledge contained in the search improves the basic evaluation, magnifying the quality of the search. As with alpha-beta, there are tradeoffs to be made. A more sophisticated evaluation function can reduce the size of the tree, at the cost of more time spent on each node. In simulation analysis, we can improve the accuracy of each trial, but at the expense of the total number of trials performed in real-time.

Variations of selective sampling have been used in other games, including Scrabble [17], backgammon [20], and bridge [9]. Selective sampling is similar to the idea of likelihood weighting in stochastic simulation [8,16]. In our case, the goal is different because we need to differentiate between EVs (for call/check, bet/raise) instead of counting events. Also, poker complicates matters by imposing tight real-time constraints (typically a maximum of two seconds). This forces us to maximize the information gained from a limited number of samples. Further, the problem of handling unlikely events (which is a concern for any sampling-based result) is smoothly handled by our re-weighting system (Section 6), allowing *Loki* to dynamically adjust the likelihood of an event based on observed actions. An unlikely event with a big payoff figures naturally into the EV calculations.

6. EXPLICIT LEARNING

In strategic games like chess, the performance loss by ignoring opponent modeling is small, and hence it is usually ignored (although it has been studied [5,11,12]). In contrast, not only does opponent modeling have tremendous value in poker, it can be the distinguishing feature between players at different skill levels. If a set of players all have a comparable knowledge of poker fundamentals, the ability to alter decisions based on an accurate model of the opponent may have a greater impact on success than any other strategic principle.

To assess a hand, the Hand Evaluator compares those cards against all possible opponent holdings. Naively, one could treat all opponent hands as equally likely, however this skews the hand evaluations compared to more realistic assumptions. Many weak hands are likely to have been folded before the flop, making them less likely to be held later in the hand. Similarly, a hand made strong by the turn and river cards may have been folded on the flop. Therefore, for each starting hand, we need to define a probability that our opponent would have played that hand in the observed manner. We call the probabilities for each of these (52 choose 2) = 1,326 subcases *weights* since they act as multipliers in the enumeration computations.[3]

The use of these weights is the first step toward opponent modeling since we are changing our computations based on the relative probabilities our opponent's possible hole cards. The simplest approach to determining these weights is to treat all opponents the same, calculating a single set of weights to reflect "reasonable" behavior, and use them for all opponents. An initial set of weights was determined by ranking the starting hands (as determined by off-line simulations) and assigning a probability commensurate with the average return on investment of each hand. These results closely approximate the ranking of hands by strong players [18].

In *Loki*, the Opponent Modeler uses probability triples to update the Weight Table after each opponent action. To accomplish this, the Triple Generator is called for each possible two-card hand. It then multiplies each weight in the Weight Table by the entry in the probability triple that corresponds to the opponent's action. For example, suppose the previous weight for Ace-Ace is 0.7 (meaning that *if* it has been dealt, there is a 70% chance the opponent would have played it in exactly the manner observed so far), and the opponent now calls. If the probability triple for the current context is [0.0, 0.2, 0.8], then the updated weight for this case would be 0.7 x 0.2 = 0.14. The relative likelihood of the opponent holding Ace-Ace has *decreased* to 14% because they did not raise.

The call value of 0.2 reflects the possibility that this particular opponent might deliberately try to mislead us by calling instead of raising. Using a probability distribution allows us to account for uncertainty in our beliefs. This process of updating the weight table is repeated for each entry.

The above corresponds to what we call *Generic Opponent Modeling (GOM).* Each hand is viewed in isolation and all opponents are treated as the same player. Each player's Weight Table is initially identical, and gets modified based on their betting action. Although rather simplistic, this model is quite powerful in that it does a good job of skewing the hand evaluations to take into account the most likely opponent holdings.

Obviously, treating all opponents the same is clearly wrong. Each player has a different style, ranging from *loose* (plays most hands beyond the flop) to *tight* (usually plays the few hands that have a very high probability of winning), and from aggressive to conservative. Knowing the style of the opponents allows a player to adjust their betting decisions. For example, if a perceived tight player is betting aggressively, there is a good chance that they have a strong hand. A loose player will play many marginal hands or may bluff a lot. This is useful information and may allow you to fold a strong hand or call with a weak one when it is correct to do so. In general, a bet made by a loose or aggressive player should not be taken as seriously as one made by a tight or conservative player.

Specific Opponent Modeling (SOM) customizes the probability triple function to represent the playing style of each opponent. For a given game, the reweighting factor applied to the entries of the Weight table is adjusted by betting frequency statistics gathered on that opponent from previous hands. This results in a shift of the assumed call and raise thresholds for each player. In the case of a tight player, the call and raise thresholds will increase, indicating fewer hands that are likely to be played. Conversely, a loose player's thresholds will be lowered. During each round of a game, the history of previous actions by the opponent is used to influence the probability triple generated for that opponent.

In competitive poker, opponent modeling is much more complex than portrayed here. For example, players can act to mislead their opponents into constructing an erroneous model. Early in a session a strong poker player may try to create the impression of being very conservative, only to exploit that image later in that session when the opponents are using an incorrect opponent model. A strong player has to adapt their model to the opponents varying their playing style.

7. EXPERIMENTS

Self-play experiments offer a convenient method for the comparison of two or more versions of the program. Our experiments use a duplicate tournament system, based on the same principle as duplicate bridge. Since each hand can be played with no memory of preceding hands, it is possible to replay the same deal, but with the participants holding a different set of hole cards each time. Our tournament system simulates a ten-player game, where each deal is replayed ten times, shuffling the seating arrangement so that every participant has the opportunity to play each set of hole cards once. This arrangement greatly reduces the "luck element" of the game, since each player will have the same number of good and bad hands. The differences in the performance of players will therefore be based more strongly on the quality of the

[3] The probability that an opponent holds a particular hand is the weight of that subcase divided by the sum of the weights for all the subcases.

decisions made in each situation. This reduction in natural variance allows meaningful results to be obtained with a smaller number of trials than in a typical game setting. Nevertheless, it is important to not over-interpret the results of one experiment.

Experiments have been performed with *Loki* to measure the performance of generic opponent modeling (GOM), simulation (S), and both combined (GOM+S). The results were obtained by playing a self-play tournament containing two enhanced versions of *Loki* against 8 unenhanced versions. A tournament consisted of 2,500 different deals (*i.e.* 25,000 games). Each simulation consisted of 500 trials, since the results obtained after 500 trials were reasonably stable.[4]

The metric used to measure program performance is the average number of small bets won per hand (sb/hand), a metric that is sometimes used by human players. For example, in a game of \$10/\$20 Hold'em, an improvement of +0.10 sb/hand translates into an extra \$30 per hour (based on 30 hands per hour). Anything above +0.05 small bets per hand is considered a large improvement. In play on an Internet poker server against human opponents, *Loki* has consistently performed at or above the +0.05 sb/hand level.

The experiments showed that GOM improved performance by 0.031 $\pm$0.019 sb/hand, simulations improved by 0.093 $\pm$0.04 sb/hand, and the combination was worth 0.095 $\pm$0.045 sb/hand (note that these are newer numbers than those appearing in [2,3,4]). The results reported here may be slightly misleading since each experiment used two similar programs. As has been shown in chess, one has to be careful about interpreting the results of these types of experiments.

GOM is a significant gain as expected. Given that all players in the tournaments were variants of *Loki*, the wide variety of play that is seen in human play is missing. Hence, GOM may be of greater benefit against typical human opponents. Simulations, on the other hand, are a huge win in self-play experiments against non-simulation opponents. As expected, they have a naturally occurring higher variance. The use of simulations represents a large improvement in the quality and variety of the betting strategies employed by *Loki* (or, possibly, overcome a serious weakness in the older version of the program). Whereas our initial knowledge-based betting strategy routine [1,14] was limited by the amount of knowledge we could code and tune, the simulation-based approach has no such restrictions. The simulations implicitly enable advanced betting strategies, with a degree of unpredictability that makes it harder for the opponents to model *Loki.*

Note that although each feature is a win by itself, the combination is not necessarily additive because there may be some interdependence between GOM and simulations (i.e. both ideas may exploit the same weaknesses). As well, the magnitude of the simulation improvement is such that it hides the effects of combining it with GOM. The larger the winning margin, the smaller the opportunity there is for demonstrating further improvement against the same opposition.

Each set of improvements reported over the past two years were measured against the previous strongest versions of *Loki*. As a result, the magnitude of the change may be dampened over time, simply because it is being tested against generally stronger opposition. For example, if you have three generations of poker-playing programs (A, B, and C) with B defeating A by 0.1 sb/hand and C is better than B by 0.1 sb/hand, it does not follow that C will be .2 sb/hand better than A.

Specific opponent modeling (SOM) is harder to measure, due in part to the nature of our self-play experiments. In previous work we demonstrated improvements for both GOM and SOM against a static default model [2]. However, since that time *Loki* has improved significantly (for example, with improved reweighting and simulations). A consequence is that our simplistic SOM model has not yet added significantly to the performance of the stronger version of *Loki*. Improving SOM is our current focus, and some of the ideas we are pursuing are discussed in the next section.

Loki has been tested in more realistic games against human opposition. For this purpose, the program participates in an on-line poker game, running on the Internet Relay Chat (IRC). Human players connect to IRC and participate in games conducted by dedicated server programs. No real money is at stake, but bankroll statistics on each player are maintained. The new versions of *Loki* using GOM and simulations win consistently when playing on the IRC server. Although there is a high level of variance in this environment, there is strong evidence that GOM is a major advance in the program's playing strength against human opposition (as opposed to the self-play experiments where the advantage was not as significant). The performance of the program depends strongly on which players happen to be playing, and on IRC it ranges from novices to professional players. Consequently, it is dangerous to quantify the results of our recent improvements to *Loki*.

8. ONGOING RESEARCH

The work reported here is our first experience with a betting strategy based on simulations and opponent modeling. Each area has numerous opportunities for improvement, some of which are currently being addressed. Indeed, the poker project is rich in research opportunities. There is no shortage of good ideas to investigate; only a shortage of time and resources.

For the simulations, the major problem is variance (standard deviation) in the results. We have identified several ways in which the experiments could be conducted

[4] The average absolute difference in expected value in increasing from 500 to 2,000 trials was small and seldom resulted in a significant change to an assessment. The difference between 100 trials and 500 trials was much more significant; the variance with 100 trials was too high.

with less noise. Nevertheless, even with these enhancements, we expect the variance to still be high. Faster machines and parallel computations might be helpful since this will result in a larger sample of simulation data points. However, this has diminishing returns and our experiments suggest that beyond a critical minimum number of simulation data points (in the 100-500 range) the benefits may be small.

There are tradeoffs in doing the simulations. Currently, each data point contains a small amount of information about the expected value. Given the simplicity of the calculation, one can acquire numerous data points. Alternatively, one could do fewer simulations, but have each return a more accurate value. The quantity versus quality trade-off needs to be explored in more detail.

For the game of bridge, simulations have successfully allowed computers to play hands at a world-class level (GIB [9]). Nevertheless, limitations in the simulation-based approach and the high variance have prompted the author of GIB, Matt Ginsberg, to look at other solutions (including building the entire search tree) [10]. We too may have to look for new approaches to overcome the limits of simulations.

In the area of opponent modeling, there are numerous avenues that can be explored. One serious limitation of our current work that needs to be address is the resistance to change that is built into our system. Our opponent modeling works well in some cases because most of the opponents have a fixed style that does not vary over time (certainly the computer opponents that we use in our self-play experiments have this property). However, it does not necessarily follow that opponent modeling will be as successful in games against human players as it is in the closed experiments. Humans are also very good at opponent modeling, and can be much less predictable than the players in our experiments. We have not yet investigated making our opponent models quickly responsive to perceived changes in an opponent's style. For a strong human player, a single data point is often sufficient for them to set or alter their model of an opponent. Our models are far too slow to adapt. This must change!

A sampling of some of the ideas being investigated include:

- Use the simulations to refine the opponent modeling. Having done a simulation, record the expected reaction for each opponent. If their actions frequently differ from what is predicted, then *Loki* can adjust its opponent model.
- With opponent modeling, it is easy to gather lots of data. The problem is filtering it and attaching the appropriate importance to it. Without this, our modeling will be too slow to react, or base its decisions on irrelevant information. We are investigating condensing the data into simpler metrics that may be better predictors of an opponent's style and future behavior. For example, measuring the amount of money that a player invests per game may be a good predictor of loose/tight play.
- Previous specific opponent modeling was hampered by the crude method used for collecting and applying observed statistics. Much of the relevant context was ignored for simplicity, such as combinations of actions within the same betting round. A more sophisticated method for observing and utilizing opponent behavior would allow for a more flexible and accurate opponent model. For example, we are currently experimenting with modifying our model based on sequences of opponent's actions. A check followed by a raise (typically a show of strength) has more meaning than looking at these two actions in isolation.
- *Loki* does not currently use showdown information. The cards seen at the showdown reveal clues about how that opponent perceived each decision during the hand. These hindsight observations can be used to adaptively measure important characteristics like aggressiveness, bluffing frequency, predictability, affinity for draws, and so forth.
- We have yet to fully explore the variety of techniques available in the literature for learning in a noisy domain where one must make inferences based on limited data.

9. CONCLUSIONS

To master the game of poker, one must be adaptive. Any form of deterministic play can and will be exploited by a good opponent. A player must change their style based on the dynamic game conditions observed over a series of hands (looking at each hand in isolation is an artificial limitation). Our work has made some progress towards achieving a poker-playing program that can learn and adapt. *Loki* successfully uses opponent modeling to improve its play. However, it is abundantly clear that these are only the first steps, and there is considerable room for improvement.

Poker is a complex game. Strong play requires the player to excel in many different aspects of the game. Developing *Loki* has been a cyclic process. We improve one aspect of the program until it becomes apparent that another aspect is the performance bottleneck. That problem is then tackled until it is no longer the limiting factor, and new weaknesses in the program's play are revealed. We made our initial foray into opponent modeling and were pleased with the results. With the success of the new simulation-based betting strategy, opponent modeling is now back on the critical path since it will offer the biggest performance gains. We will now refocus our efforts on that topic, until it too moves off the critical path.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Council of Canada.

References

[1] Billings, D., Papp, D., Schaeffer, J. and Szafron, D. (1997). Poker as a testbed for machine intelligence research. *Advances in Artificial Intelligence*, R. Mercer and E. Neufeld (editors), Springer Verlag, pp. 1-15.

[2] Billings, D., Papp, L., Schaeffer, J. and Szafron, D. (1998). Opponent modeling in poker, AAAI, pp. 493-999.

[3] Billings, D. Papp, D., Peña, L., Schaeffer, J. and Szafron, D. (1999). Using selective-sampling simulations in poker, AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information, pp. 13-18.

[4] Billings, D., Peña, L., Schaeffer, J. and Szafron, D. (1999). Using probabilistic knowledge and simulation to play poker, AAAI, to appear.

[5] Carmel, D. and Markovitch, S. (1995). Incorporating opponent models into adversary search. AAAI, pp. 120-125.

[6] Cheng, C. (1997). Recognizing poker hands with genetic programming and restricted iteration. *Genetic Algorithms and Genetic programming at Stanford*, J. Koza (editor), Stanford, California.

[7] Findler, N. (1977). Studies in machine cognition using the game of poker. *Communications of the ACM* 20(4):230-245.

[8] Fung, R. and Chang, K. (1989). Weighting and integrating evidence for stochastic simulation in Bayesian networks, *Uncertainty in Artificial Intelligence*, Morgan Kaufmann.

[9] Ginsberg, M. (1999). GIB: Steps towards an expert-level bridge-playing program, IJCAI, to appear.

[10] Ginsberg, M. (1999) Personal communication, April 13.

[11] Iida, H., Uiterwijk, J., van den Herik, J. and Herschberg, I. (1995). Thoughts on the application of opponent-model search. In *Advances in Computer Chess 7*, University of Maastricht, pp. 61-78.

[12] Jansen, P. (1992). Using knowledge about the opponent in game-tree search. Ph.D. thesis, Department of Computer Science, Carnegie-Mellon University.

[13] Koller, D. and Pfeffer, A. (1997). Representations and solutions for game-theoretic problems. *Artificial Intelligence* 94(1-2), 167-215.

[14] Papp, D. (1998). Dealing with Imperfect Information in Poker, M.Sc. thesis, Department of Computing Science, University of Alberta.

[15] Sakaguchi, M. and Sakai, S. (1992). Solutions of some three-person stud and draw poker. *Mathematics Japonica*, 6(37):1147-1160.

[16] Shacter, R. and Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks, *Uncertainty in Artificial Intelligence*, Morgan Kaufmann.

[17] Sheppard, B. (1998). Email communication, October 23.

[18] Sklansky, D. and Malmuth, M. (1994). *Hold'em Poker for Advanced Players*. Two Plus Two Publishing.

[19] Smith, S. (1983). Flexible learning of problem solving heuristics through adaptive search. IJCAI, pp. 422-425.

[20] Tesauro, G. (1995). Temporal difference learning and TD-Gammon, *Communications of the ACM* 38(3):58-68.

[21] Waterman, D. (1970). A generalization learning technique for automating the learning of heuristics. *Artificial Intelligence*, vol. 1, pp. 121-170.