Using Probabilistic Knowledge and Simulation to Play Poker

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

Abstract

Until recently, artificial intelligence researchers who use games as their experimental testbed have concentrated on games of perfect information. Many of these games have been amenable to brute-force search techniques. In contrast, games of imperfect information, such as bridge and poker, contain hidden information making similar search techniques impractical. This paper describes recent progress in developing a high-performance poker-playing program. The advances come in two forms. First, we introduce a new betting strategy that returns a probabilistic betting decision, a probability triple, that gives the likelihood of a fold, call or raise occurring in a given situation. This component unifies all the expert knowledge used in the program, does a better job of representing the type of decision making needed to play strong poker, and improves the way information is propagated throughout the program. Second, real-time simulations are used to compute the expected values of betting decisions. The program generates an instance of the missing data, subject to any constraints that have been learned, and then simulates the rest of the game to determine a numerical result. By repeating this a sufficient number of times, a statistically meaningful sample is used in the program's decision–making process. Experimental results show that these enhancements each represent major advances in the strength of computer poker programs.

1. Introduction

Past research efforts in computer game-playing have concentrated on building high-performance chess programs. With the Deep Blue victory over World Chess Champion Garry Kasparov, a milestone has been achieved but, more importantly, the artificial intelligence community has been liberated from the chess "problem". The consequence is that in recent years a number of interesting games have attracted the attention of AI researchers, where the research results promise a wider range of applicability than has been seen for chess.

Computer success has been achieved in deterministic perfect information games, like chess, checkers and Othello, largely due to so-called brute-force search. The correlation of search speed to program performance gave an easy recipe to program success: build a faster search engine. The Deep Blue team took this to an extreme, analyzing roughly 250 million chess positions per second.

In contrast, until recently imperfect information games have attracted little attention in the literature. In these games, no player knows the complete state and each player has to infer the missing information to maximize the chances of success. For these games, brute-force search is not successful since it is often impractical to search the game trees that result from all possible instances of the missing information.

Two examples of imperfect information games are bridge and poker. Recently, at least two research groups have made an effort to achieve high-performance bridge-playing programs [Ginsberg, 1999; Smith *et al*., 1998]. The progress has been impressive, and we may not have to wait long for a world-championship caliber program.

Until now, the computing community has largely ignored poker (a recent exception being [Koller and Pfeffer, 1997]). However, poker has several attributes that make it an interesting and challenging domain for mainstream AI research [Billings *et. al*., 1998a].

We are attempting to build a program that is capable of beating the best human poker players. 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 the most strategically complex poker variant that is widely played.

Our program, Loki, is a reasonably strong player (as judged by its success playing on the Internet) [Billings *et. al*., 1998a; 1998b]. The current limitation in the program's play is its betting strategy - deciding when to fold, call/check, or raise/bet. A betting strategy attempts to determine which betting action will maximize the expected winnings (or minimize the losses) for a hand. The previous version of Loki used several expert-knowledge evaluation functions to make betting decisions. These routines were rigid in the sense that they always returned a single value: the "best" betting decision. Although these evaluation functions allowed Loki to play better than average poker, it was inadequate to play at a world-class level, since continually upgrading this knowledge is difficult and error-prone.

This paper introduces two major advances in the capabilities of computer-poker-playing programs. Each is shown experimentally to result in substantial improvements in Loki's play.

First, this paper introduces a new betting strategy that returns a *probability triple* as the knowledge representation of the evaluation function. The routine returns three probabilities (one each for fold, call/check, and raise/bet). The program can then randomly select the betting decision in accordance with the probability triples. Representing decisions as a probability distribution better captures the type of information needed to perform well in a noisy environment, where randomized strategies and misinformation are important aspects of strong play. This component also allows us to unify the expert knowledge in a poker program, since the same component can be used for betting decisions, opponent modeling, and interpreting opponent actions.

Second, Loki now bases its betting strategy on a simulation-based approach; we call it *selective sampling*. It simulates the outcome of each hand, by generating opponent hands from the sample space of all *appropriate* possibilities, trying each betting alternative (call/check, bet/raise) to find the one that produces the highest expected winnings. A good definition of appropriate hands is one of the key concepts in defining selective sampling and it is one of the main topics of this paper. As with brute-force search in chess, the simulation (search) implicitly uncovers information that improves the quality of a decision. With selective sampling, the knowledge applied to a simulation *quantifies* the value of each choice, improving the chance of making a good decision.

Simulation-based approaches have been used in other games, such as backgammon [Tesauro, 1995], bridge [Ginsberg, 1999], and Scrabble[1] [Sheppard, 1998]. The simulation methods presented in this paper are quite similar to those used by Ginsberg in Gib, although there are several distinctions in the details, due to differences in the games.

For deterministic perfect information games, there is a well-known framework for constructing these applications (based on the alpha-beta algorithm). For games with imperfect information, no such framework exists. For handling this broader scope of games we propose that selective sampling simulation be such a framework.

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 turn over their two hidden 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. Typically, Texas Hold'em is played with 8 to 10 players.

Limit Texas Hold'em has a structured betting system, where the order and amount of betting is strictly controlled in each betting round.[2] There are two denominations of bets, called the small bet and the big bet ($10 and $20 in this paper). In the first two betting rounds, all bets and raises are $10, while in the last two rounds they are $20. In general, when it is a player's turn to act, one of three 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.

3. Building a Poker Program

A minimal set of requirements for a strong poker-playing program includes assessing hand strength and potential, betting strategy, bluffing, unpredictability and opponent modeling. Descriptions of these as they are implemented in our program, Loki, can be found in [Billings *et. al.*, 1998a; 1998b]. 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.

The architecture of the previous version of Loki, which we now call Loki-1, is shown in Figure 1. 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.

To make a betting decision, the Bettor calls the Hand Evaluator to obtain an assessment of the strength of the current cards. The Bettor uses this hand strength, the public game state data and expert-defined betting knowledge to generate an action (bet, call or raise). 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. After the flop, the probability for each possible opponent hand is different. For example, the probability that hole cards of Ace-Ace are held after the flop is much higher than 7-2, since most players will fold 7-2. Each possible hand has a weight in the Weight Table for each opponent, and these weights are modified after each opponent action. Updating the probabilities for all hands is a process called *re-weighting*. After each opponent action, the Opponent Modeler calls the Hand Evaluator once for each possible hand and increases or decreases the weight for that case to be consistent with the new information. The Hand Evaluator uses the Weight Table values to bias the calculation, giving greater weight to the more likely hands. The absolute values of the probabilities are of little consequence, since only the relative weights affect the later calculations.

Loki-1 uses expert knowledge in four places:

1. Pre-computed tables of expected income rates are used to evaluate its hand before the pre-flop, and to assign initial weight probabilities for the various possible opponent hands.
2. The Opponent Modeler applies re-weighting rules to modify the opponent hand weights based on the previous weights, new cards on the board, opponent betting actions, and other contextual information.

[1] ™ Milton-Bradley company.

[2] In No-limit Texas Hold'em, there are no restrictions on the size of bets.

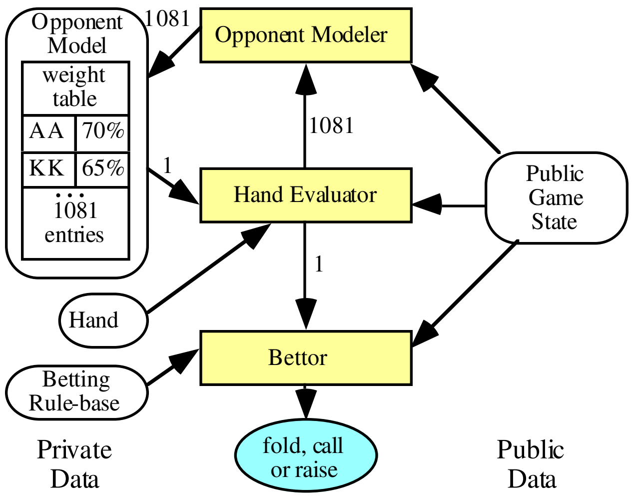

Figure 1. The architecture of Loki-1.

3. The Hand Evaluator uses enumeration techniques to compute hand strength and hand potential for any hand based on the game state and the opponent model.
4. The Bettor uses a set of expert-defined rules and a hand assessment provided by the Hand Evaluator for each betting decision: fold, call/check or raise/bet.

This design has several limitations. First, expert knowledge appears in various places in the program, making Loki difficult to maintain and improve. Second, the Bettor returns a single value (fold, call, raise), which does not reflect the probabilistic nature of betting decisions. Finally, the opponent modeler does not distinguish between the different actions that an opponent might take. A call/check versus a bet/raise gives valuable information about the opponent's cards. These issues led to a redesign of how knowledge is used in Loki.

The new version of Loki, called Loki-2 makes two fundamental changes to the architecture. First, it introduces a useful, new data object called a probability triple that is used throughout the program (Section 4). Second, simulation with selective sampling is used to refine the betting strategy (Section 5). Loki-2 can be used with or without simulation, as shown in Figure 2. With simulation, the Simulator component replaces the simpler Action Selector.

4. Probability Triples

A probability triple 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 is a fold, call, or raise, respectively. Probability triples are used in three places in Loki-2. The Action Selector uses a probability triple to decide on a course of action (fold, call, raise). The Opponent Modeler uses an array of probability triples to update the opponent weight tables. The Simulator (see Section 5) uses probability triples to choose actions for simulated opponent hands.

Each time it is Loki-2's turn to bet, the Action Selector uses a single probability triple to decide what action to take (note that the Bettor is gone). For example, if the triple [0.0,0.8,0.2] is given, then the Action Selector would call 80% of the time, raise 20% of the time, and never fold. The choice can be made by generating a random number, allowing the program to vary its play, even in identical situations. This is analagous to a *mixed strategy* in game theory, but the probablility triple implicitly contains contextual information resulting in better informed decisions which, on average, can outperform a game theoretic approach.

The Triple Generator is responsible for generating probability triples. As shown in Figure 2, this routine is now at the heart of Loki-2. The Triple Generator takes a two-card hand and calls the Hand Evaluator to evaluate the cards in the current context. It uses the resulting hand value, the current game state, and expert-defined betting rules to compute the triple. Note that in addition to using the Triple Generator to produce a triple for our known hand, it can also be used to assess the likely behavior of the opponent holding any possible hand.

For the Hand Evaluator to assess a hand, it compares that hand against all possible opponent holdings. To do this, it uses the opponent Weight Table. In Loki-2, the Opponent Modeler now uses probability triples to update this 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 there was no 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, which was not handled by the previous architecture. This process of updating the weight table is repeated for each entry.

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 algorithm. 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.

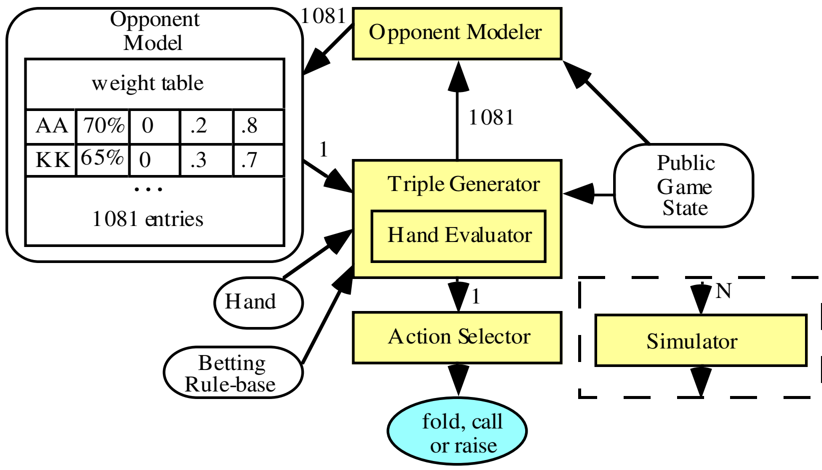

Figure 2. Using the Triple Generator in Loki-2.

The current architecture also suggests future enhancements, such as better methods for opponent modeling. For example, 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, predictability, affinity for draws, and so forth. The Opponent Modeler can maintain each of these properties for use by the Triple Generator, which combines the information in proper balance with all the other factors. The knowledge is implicitly encoded in the probability distribution, and is thereby passed on to all components of the system.

Since the more objective aspects of the game could eventually be well defined, the ultimate strength of the program may depend on the success in handling imperfect information, and the more nebulous aspects of the game, such as opponent modeling.

5. Simulation-Based Betting Strategy

The original Bettor component consisted of expert-defined rules, based on hand strength, hand potential, game conditions, and probabilities. 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, 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 [Koller and Pfeffer, 1997].

Consider a 10-player game of Texas Hold'em. By the time the flop cards are seen, some players may have folded. Let's assume one player bets, and it is Loki's turn to act. The program must choose between folding, calling or raising. Which one is the best decision?[1]

After the program's decision, every other active player will be faced with a similar choice. In effect, there is a branching factor of 3 possible actions for each player, and there may be several such decisions in each betting round. Further, there are still two betting rounds to come, each of which may involve several players, and one of many (45 or 44) unknown cards. Computing the complete poker decision tree in real time is in general, prohibitively expensive. Since we cannot consider all possible combinations of hands, future cards, and actions, we examine only a representative sample from the possibilities. A larger sample and more informed selection process will increase the probability that we can draw meaningful conclusions.

5.1 An Expected Value Based Betting Strategy

Loki-2's new 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-2 invokes the Simulator to get an estimate of the *expected value* (EV) of each betting action (see the dashed box in Figure 2 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

[1] "Best" is subjective. Here we do not consider other plays, such as deliberately misrepresenting the hand to the opponents. The expected value for a whole session is more important than the expected value for a single hand.

bet/raise. In each trial the hand is simulated to the end, and the amount of money won or lost is determined. 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. If two actions have the same expectation, we opt for the most aggressive one (call over fold and raise over call). Against human opponents, a better strategy might be to randomize the selection of betting actions whose EVs are close in value.

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 which 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. To select the candidate hands that our opponent may have, we use *selective sampling*.

5.2 Selective Sampling

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 the stakes 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 based on the hand that was generated for that player. The player's action is then randomly selected, based on the probability distribution defined by this triple, and the simulation proceeds. As shown in Figure 2, 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.

5.3 Comments

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 will improve the default values obtained by heuristics, resulting in a more accurate estimate.

As has been seen in other domains, the search itself contains implicit knowledge. A simulation contains inherent information that improves the basic evaluation: 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

An important feature of the simulation-based framework is the notion of an obvious move cut-off. Although many alpha-beta-based programs incorporate an obvious move feature, the technique is usually *ad hoc* and the heuristic is the result of programmer experience rather than a sound analytic technique (an exception is the B* proof procedure [Berliner, 1979]). In the simulation-based framework, an obvious move is statistically well-defined. As more samples are taken, if one decision point exceeds the alternatives by a statistically significant margin, one can stop the simulation early and make an action, with full knowledge of the statistical validity of the decision.

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 seen 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 number of trials performed in real-time.

Selective sampling combined with reweighting is similar to the idea of likelihood weighting in stochastic simulation [Fung and Chang, 1989; Shacter and Peot, 1989]. 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 real-time constraints. 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, allowing Loki-2 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. Experiments

To obtain meaningful empirical results, it is necessary to conduct a series of experiments under different playing conditions. Each enhancement is tested against a variety of opponents having different styles (e.g. liberal or conservative, aggressive or passive, etc.). Control experiments are run at the same time to isolate the dependent variable. In some cases, experiments are designed with built-in standards for comparison, such as playing one particular version against the identical program with an enhancement.

For each test, the parameters of the experiment (number of deals, length of simulations, etc.) are assigned to produce statistically significant results. For example, 5,000 trials might be used to compare an experimental version against a homogenous field. To test the same feature against a mixed field of opponents might require a parallel control experiment and 25,000 trials to produce stable results, due to the inherently higher variance (noise) of that environment. Many experiments were performed to establish reliable results, and only a cross-section of those tests are presented here. For instance, over 30 experiments were conducted to measure the performance of the new re-weighting system.

In this paper, we study the effects of three enhancements, two of which represent improvements to a component of the previous system, and one that is a fundamental change in the way Loki makes its decisions. The features we look at are:

R: changing the re-weighting system to use probability triples (Section 4).

B: changing from a rule-based Bettor to an Action Selector that uses probability triples and incorporates a randomized action (Section 4).

S: incorporating a Simulator to compute an EV estimate, which is used to determine an action (Section 5).

It is important to note that the enhancements were not maximized for performance. The probability-triple-based betting strategy and re-weighting were implemented in only a few hours each, owing to the improved architecture.

The changes to Loki were first assessed with self-play tournaments. A tournament consisted of playing two versions of Loki against each other: a control version (8 copies) and an enhanced version (2 copies). By restricting the tournament to two different player types, we reduced the statistical variance and achieved meaningful results with fewer hands played. To further reduce variance, tournaments followed the pattern of duplicate bridge tournaments. Each hand was played ten times. Each time the seating arrangement of the players was changed so that 1) every player held every set of hidden cards once, and 2) every player was seated in a different position relative to all the opponents. A tournament consisted of 2,500 different deals (*i.e.* 25,000 games).

The number of trials per simulation was chosen to meet real-time constraints and statistical significance. In our experiments, we performed 500 trials per simulation, since the results obtained after 500 trials were quite stable. The average absolute difference in expected value after 2000 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.

The metric used to measure program performance is the average number of small bets won per hand (sb/hand). This is a measure 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, Loki has consistently performed at or above +0.05 sb/hand.

Figure 3 shows the results of playing Loki against itself with the B and R enhancements individually and combined (B+R). Against the Loki-1 standard, B won +0.025 ± 0.007 sb/hand, R won +0.023 ± 0.0125 sb/hand and the combined B+R won +0.044 ± 0.024 sb/hand, showing that these two improvements are nearly independent of each other. Figure 3 also shows enhancement S by itself and S combined with B and R (B+R+S). Note that each feature is a win by itself and in combination with others. In general, the features are not strictly additive since there is some interdependence.

The simulation experiments generally had higher variance than those without simulation. However, all statistically significant results showed an improvement for any version of Loki augmented with selected sampling. These results are harder to accurately quantify, but an increase on the order of at least +0.05 sb/hand is evident. These results 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 type of experiments [Berliner *et al.*, 1990]. A second set of experiments was conducted to see how well the new features perform against a mixture of opponents with differing styles (as is typically seen in human play).

To vary the field of opponents, we defined several different playing styles to categorize players. Players vary from tight (T) to loose (L), depending on what fraction of hands they play to the flop. A style may range from aggressive (A) to conservative (C), depending on how frequently they bet and raise after the flop.

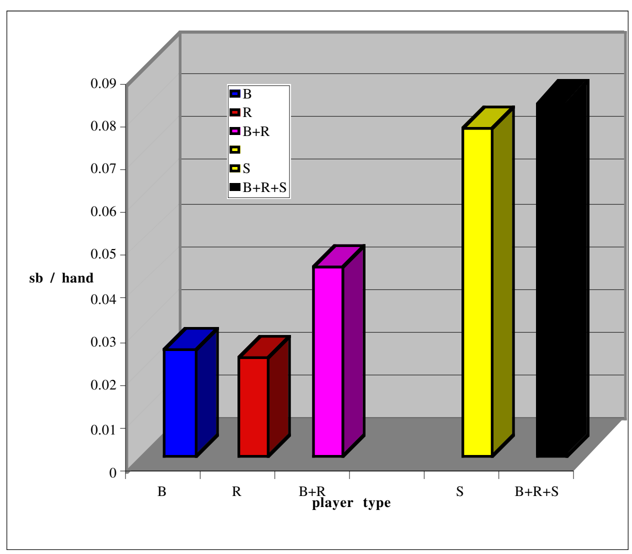

Figure 3. Experimental results of the basic Loki player versus the Loki player with enhancements.

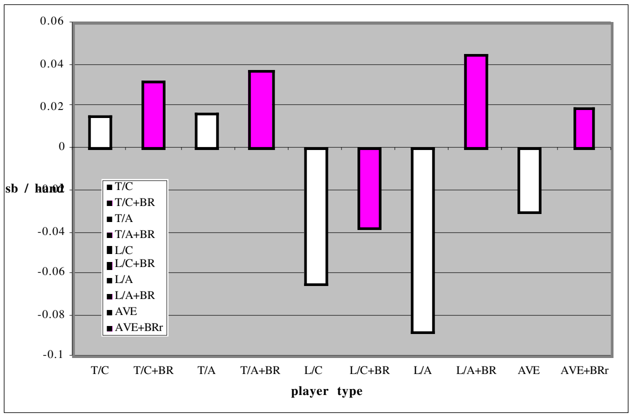

Figure 4. Experimental results in a mixed environment.

We conducted an experiment in which there was a pair of players from each of the four categories: tight/conservative, tight/aggressive, loose/conservative and loose/aggressive. In each pair, one of the players was a basic Loki-1 player and the other was a Loki-2 player with new betting strategy (B) and new re-weighting strategy (R). To fill out the field to ten players, we actually used two pairs of tight/conservative players and averaged their results. The results are shown in Figure 4. In each case, the enhanced player with BR outplayed the corresponding un-enhanced player. For example, the weakest player in the field (L/A) went from -0.088 sb/hand to +0.045 sb/hand with the B+R enhancements. There is also a data point for the average of all players. On average, an enhanced player earned +0.050 ± 0.049 sb / hand more than the corresponding un-enhanced player.

Finally, the ultimate test for Loki-2 is how it plays against human opposition. Loki-2 currently plays on an Internet Relay Chat (IRC) poker server. Interpreting the results from these games is dangerous since we have no control over the type and quality of the opponents. Nevertheless, the program is a consistent winner and appears to be better than Loki-1 in this respect. When the new features are better tuned, we expect greater success.

7. Conclusions

This paper provides two contributions to dealing with imperfect information in a poker-playing program.

First, using probability triples allows us to unify several knowledge-based components in Loki. By representing betting decisions as a probability distribution, this evaluation is better suited to representing the non-deterministic, imperfect information nature of poker. In effect, a static evaluation function now becomes a dynamic one. The added flexibility of making a probabilistic decision yields a simple Triple Generator routine that out-performs our previous best rule-based betting strategy.

Second, a simulation-based betting strategy for poker is superior to the static-evaluation-based alternative. As seen with brute-force search in games like chess, the effect of the simulation (search) magnifies the quality of the evaluation function, achieving high performance with minimal expert knowledge. Critical to this success is the notion of selective sampling; ensuring that each simulation uses data that maximizes the information gained. Selective sampling simulations are shown experimentally to significantly improve the quality of betting decisions.

We propose that the selective sampling simulation-based framework become a standard technique for games having elements of non-determinism and imperfect information. While this framework is not new to game-playing program developers, it is a technique that is repeatedly discovered and re-discovered.

Acknowledgments

This research was supported, in part, by research grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada. Computation resources were provided by MACI.

References

H. Berliner, 1979. “The B* Tree Search Algorithm: A Best First proof Procedure”, *Artificial Intelligence*, vol. 12, no. 1, pp. 23-40.

H. Berliner, G. Goetsch, M. Campbell and C. Ebeling, 1990. “Measuring the Performance Potential of Chess Programs”, *Artificial Intelligence*, vol. 43. no. 1, pp. 7-20.

D. Billings, D. Papp, J. Schaeffer and D. Szafron, 1998a. “Poker as a Testbed for Machine Intelligence Research”, in *AI’98 Advances in Artificial Intelligence* (R. Mercer and E. Neufeld, eds.), Springer Verlag, pp. 1-15.

D. Billings, D. Papp, J. Schaeffer and D. Szafron, 1998b. “Opponent Modeling in Poker”, AAAI, pp. 493-499.

D. Billings, D. Papp, L. Peña, J. Schaeffer and D. Szafron, 1999. “Using Selective-Sampling Simulations in Poker”, AAAI Spring Symposium.

R. Fung and K. Chang, 1989. “Weighting and Integrating Evidence for Stochastic Simulation in Bayesian Networks”, *Uncertainty in Artificial Intelligence*, Morgan Kaufmann.

M. Ginsberg, 1999. “GIB: Steps Towards an Expert-Level Bridge-Playing Program”, IJCAI, to appear.

D. Koller and A. Pfeffer, 1997. “Representations and Solutions for Game-Theoretic Problems,” *Artificial Intelligence,* vol. 94, no. 1-2, pp. 167-215.

R. Shacter and M. Peot, 1989. “Simulation Approaches to General probabilistic Inference on Belief Networks”, *Uncertainty in Artificial Intelligence*, Morgan Kaufmann.

B. Sheppard, 1998. Email communication, October 23.

S. Smith, D. Nau, and T. Throop, 1998. “Computer Bridge: A Big Win for AI Planning”, *AI Magazine*, vol. 19, no. 2, pp. 93-106.

G. Tesauro, 1995. “Temporal Difference Learning and TD-Gammon”, *CACM*, vol. 38, no.3, pp. 58-68.