Using Selective-Sampling Simulations in Poker

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

Abstract

Until recently, AI research that used games as an experimental testbed has concentrated on perfect information games. Many of these games have been amenable to so-called brute-force search techniques. In contrast, games of imperfect information, such as bridge and poker, contain hidden knowledge making similar search techniques impractical. This paper describes work being done on developing a world-class poker-playing program. Part of the program's playing strength comes from real-time simulations. The program generates an instance of the missing data, subject to any constraints that have been learned, and then searches the game tree to determine a numerical result. By repeating this a sufficient number of times, a statistically meaningful sample can be obtained to be used in the program's decision–making process.

For constructing programs to play two-player deterministic perfect information games, there is a well-defined framework based on the alpha-beta search algorithm. For imperfect information games, no comparable framework exists. In this paper we propose *selective sampling* simulations as a general-purpose framework for building programs to achieve high performance in imperfect information games.

Introduction

The 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; games whose 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. Here the complete state is not known to any player, and a 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 instantiations of the missing information.

Two examples of imperfect information games are bridge and poker. Recently, at least two research groups have made a concerted effort to achieve high-performance bridge-playing programs [Ginsberg, 1996; Ginsberg, 1998; 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, poker has been largely ignored by the computing community. However, poker has a number of attributes that make it an interesting and challenging problem for AI research [Billings *et al*., 1998b].

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 to be 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; Papp, 1998]. 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 for a hand. The previous version of Loki used an expert-knowledge evaluation function to make betting decisions. Although this betting strategy allowed Loki to play better than average poker, it was inadequate to play world-class poker, since continually upgrading this knowledge is difficult and error-prone.

Loki now bases its betting strategy on a simulation-based approach that we call *selective sampling*. It simulates the outcome of each hand, by generating opponent hands from the sample space of all *appropriate* opponent hands and tries each betting alternative to see which one 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. With brute-force search, the search implicitly uncovers information that can improve the quality of a decision. With selective sampling, the quality of the sample selection and the simulation over this sample, improves the chances that the decision is the correct one.

In examining the literature, one finds that various forms of simulation-based approaches have been used in backgammon [Tesauro, 1995], bridge [Ginsberg, 1998], Scrabble[1] [Sheppard, 1998] and now poker. There are many similarities in the methods used in all four games.

1 ™ of Milton-Bradley.

This is a pre-print of a copyrighted paper that will appear in 1999 AAAI Syring Symposium Search Techniques for Problem Solving under Uncertainty and Incomplete Information.

For deterministic perfect information games, there is a well-known framework for constructing 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 become this framework.

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. 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 in each betting round.[1] 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.

Building a Poker Program

A minimum set of requirements for a strong poker-playing program includes hand strength, hand potential, betting strategy, bluffing, unpredictability and opponent modeling. What follows is a brief description of each; implementation details for Loki can be found in [Billings *et al*., 1998a; Billings *et al*., 1998b, Papp, 1998]. There are several other identifiable characteristics which may not be necessary to play reasonably strong poker, but may eventually be required for world-class play.

Hand strength assesses how strong your hand is in relation to the other hands. At a minimum, it is a function of your cards and the current community cards. A better hand strength computation takes into account the number of players still in the game, your position at the table, and the history of betting for the hand. 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 assesses the probability of a hand improving (or being overtaken) 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 (five cards of the same suit) as more community cards are dealt. At a minimum, hand potential is a function of your cards 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 model 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 unpredictable play.

Bluffing allows you to make a profit from weak hands, 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 betting strategy over time, opponents may be induced to make mistakes based on an incorrect model.

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

Simulation-Based Betting Strategy

The original betting strategy consisted of expert-defined rules, based on hand strength, hand potential, game conditions, and probabilities. A professional poker player (Billings) 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, computer-oriented approach was needed.

In effect, this 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, the

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

same approach is not feasible for imperfect information games because there are too many possibilities to consider.

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 (no further financial investment), calling ($10 to match the outstanding bet), or raising ($10 to call, plus an additional $10). 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 is, in general, a prohibitively expensive computation. Since we cannot consider all possible combinations of hands, future cards, and actions, we examine only an appropriate representative sample from the possibilities. The larger the sample, and the more informed the selection process, the higher the probability that we can draw meaningful conclusions.

An Expected Value Based Betting Strategy

Loki'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 performs a simulation to get an estimate of the *expected value* (EV) of each betting action. 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 case 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 is 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 select the most probable hands that our opponents may have, we use *selective sampling*.

Selective Sampling

When we do the simulation, we have specific information that can be used to bias the selection of cards (i.e. sample selectively). 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, and the random generation of each opponent's two card hand is based on those probabilities.

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 have a routine, `ProbTriple()`, which computes 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.

Probability triples

To play out a simulated game, we need to predict how an opponent will play in any particular situation. This is not necessarily deterministic, so we want to predict the probability that they will fold, check/call, or bet/raise. This mixed strategy is represented as a triple $[f,c,r]$, $f+c+r = 1.0$, and is computed by the routine `ProbTriple()`. This is, in effect, a static evaluation function, and could be used as a complete (non-deterministic) betting strategy.

For the purpose of the simulations, it is not essential to predict the exact action the opponent will take in every case. An error will not be serious provided that the selected action results in a similar computed EV. For example, in a particular situation whether an opponent calls or raises may result in a very similar EV for us. In this case, it will not adversely affect the computation to assume that the opponent will always call. However, the better we are able to predict the opponent's actual behavior, the better we can exploit strategic weaknesses.

Loki does opponent modeling, meaning that it gathers historical data on how each opponent plays [Billings *et al.*, 1998a]. This information has been used in the calculation of hand strength and potential by appropriately skewing the probability of each possible opponent hand. The `ProbTriple()` routine can also facilitate opponent modeling, but now we can distinguish not only *what* hands an opponent is likely to play, but also *how* they will play them. For example, Loki can measure the aggressiveness of each player, and use this information to make better inferences about the implications of each observed action.

The future behavior of an opponent is, strictly speaking, unknowable. Predicting how they will play their hand is a subjective assessment, and may be more successful for some players than others. We wish to separate the subjective (but necessary) elements of poker from the objective aspects. By doing so, we can make the program structure (e.g. alpha-beta framework in perfect-information games) orthogonal to the application-dependent knowledge (the evaluation function).

[1] "Best" is subjective. Here we do not consider other plays, such as deliberately misrepresenting the hand to the opponents.

Results

The number of trials per simulation is chosen to meet real-time constraints and statistical significance. In our experiments, we performed 500 trials per simulation, since the EVs obtained after 500 trials are quite stable. The average absolute difference in EV after 500 trials and after 2000 trials is small and rarely results in a significant change in an assessment. The difference between 100 trials and 500 trials was much more significant; the variance with 100 trials is far too high.

To reduce the overall number of trials per simulation, we stop the simulation early if an *obvious* action is found. We currently define an obvious action as any action where the separation between the EV of the best action and the EV of the second best action is greater than the sum of the standard deviations of the EVs. This criterion for an obvious action is extremely conservative, and results in declaring fewer than 5% of actions as obvious. More liberal criteria for distinguishing obvious moves need to be tested to produce more frequent cutoffs while retaining an acceptable margin of error.

Adding simulation to our best version of Loki improves the program's performance (as judged by computer self-play, which may not be representative of play with humans). Taking our old betting strategy and using it in the simulations results in a program that wins, on average $1,075 more per 1,000 hands of $10/$20 poker. The extra winnings of roughly $1 per hand represent a large increase, as judged by human poker player standards.

The above experiment did not use the `ProbTriple()` facility, since the old betting strategy returns a decision (fold, call/check, bet/raise) as opposed to probabilities for each. We have implemented a simple, fast routine for `ProbTriple()` (less than one page of code). Using it in the simulations causes the program to win an average of $880 per 1,000 hands, as compared to our best non-simulation program. This is encouraging, since even with a naïve betting strategy, the simulations magnify the results to produce something credible. We are working on improving this routine to do a better job generating probabilities, while maintaining its significant speed advantage over our old betting routine.

Loki plays on the Internet (on irc.poker). In the near future we will replace the current version of Loki that is playing with a new simulation-based version.

Comments

It should be obvious that the simulation approach must be better than the static approach, since it uses a selective search to augment and refine a static evaluation function. Playing out relevant scenarios can only improve the default values obtained by heuristics, resulting in a more accurate estimate.

As has been seen in other search algorithms, 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). In effect, the simulated search magnifies the quality of the results.

A simulation-based approach has the advantage of simplifying the expert knowledge required to achieve high performance. This is similar to what has been observed in two-player games, where deep search compensates for limited knowledge. It also has the advantage of isolating the expert knowledge into a single function. In effect, the probability triple routine is viewed as a black box by the EV engine; only the poker expert has to deal with its internals. Since the more objective aspects of the game can eventually be well-solved, 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.

A Framework for Non-Deterministic Game-Playing Programs

Using simulations for imperfect information games is not new. Consider the following three games:

1 In Scrabble, the opponent's tiles are unknown, but this is constrained by the tiles in the computer's hand and those that have appeared on the board. A simulation consists of repeatedly generating a plausible set of tiles for the opponent. Then each trial consists of a 2 to 4 ply search of the game tree, trying to determine which move for the computer leads to the maximum number of points [Sheppard, 1998]. A simulation-based approach has been used for a long time in Scrabble programs. Brian Sheppard, the author of the Scrabble program Maven, coined the term "simulator" for this type of game-playing program structure.

2 In backgammon, "rollouts" of certain positions are done by simulation, and are now generally regarded as the best available estimates for the equity of a given position. The unknown element is the non-deterministic dice rolls. A simulation consists of generating a series of dice rolls, playing through to the end of the game, and then recording the result [Tesauro, 1995].

3 In bridge, the hidden information is the cards that each player has. A simulation consists of dealing cards to the opponents in a manner that is consistent with the bidding. The hand is then played out and the result determined. Repeated deals are played until enough confidence has been gained to decide which card to play [Ginsberg, 1996; Ginsberg, 1998].

In the above examples, the programs are not using Monte Carlo sampling to generate hidden information: they use *selective sampling*, sampling biased towards taking advantage of all the available information. We want to distinguish selective sampling from traditional Monte Carlo techniques, in that we are using information about the game state to skew the underlying probability distribution, rather than assuming uniform or other fixed probability distributions. Monte Carlo techniques may eventually converge on the right answer, but selective sampling allows for faster convergence and less variance.

Two examples illustrate this point (besides the poker example discussed earlier). First, the Scrabble program Maven does not randomly assign 7 of the remaining unknown tiles to the opponent. Instead, it biases its choice to give the opponent a "nice" hand [Sheppard, 1998]. Strong players like to have a balanced hand with lots of potential; a random assignment of letters does not achieve that. Second, in bridge the assignment of cards to an opponent should be subject to any information obtained from the bidding. If one opponent has indicated point strength, then the assignment of cards to that opponent should reflect this information [Ginsberg, 1998].

The alpha-beta framework has proven to be an effective tool for the design of two-player, zero-sum, deterministic games with perfect information. It has been around for over 30 years, and in that time the basic structure has not changed much, although there have been numerous algorithmic enhancements to improve the search efficiency. Figure 1 illustrates this framework. It has the following properties:

1 The program usually iterates on the search depth (iterative deepening).
2 The search has full breadth, but limited depth.
3 Heuristic evaluation usually occurs at the leaf nodes of the search.
4 All interior node alternatives are usually considered, except those that can be logically eliminated (such as alpha-beta cutoffs).

```
search_depth = 0;
pos = current_state_of_the_game;
while( ( search_depth <= MAX_DEPTH ) and
             ( resources remaining ) )
{
  search_depth = search_depth + 1;
  for( each legal move m )
  {
    score[m] = AlphaBeta( pos.m, search_depth );
  }
  best = max( score[] );
}
play move best;
```

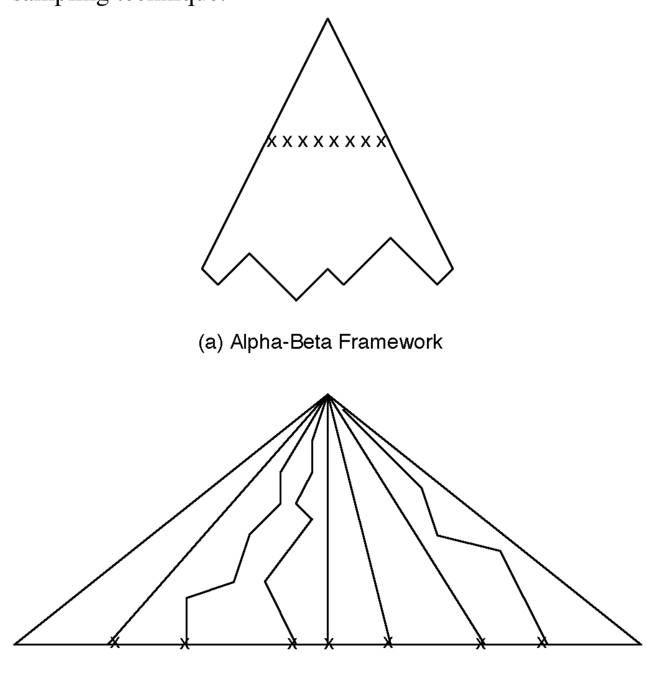
Figure 1. Framework for two-player, zero-sum, perfect information games.

The search gathers confidence in its move choice by searching deeper along each line. Figure 2a) shows where in the search the evaluations occur. The deeper the search, the greater the confidence in the move choice, although diminishing returns quickly takes over. There is usually no statistical evidence to support the choice of best move. The alpha-beta algorithm is designed to identify a "best" move, and not differentiate between any other moves. Hence, the selection of the best move may be brittle, in that a single node misevaluation can propagate to the root of the search and alter the best move choice.

In an imperfect information game, it is often impractical to build the entire game tree of all possibilities [Koller and Pfeffer, 1997]. This is especially true for poker because of multiple opponents and the number of cards in the deck. One instance of the imperfect and non-deterministic information is applied to each specific trial. Hence, a representative sample of the search space is looked at to gather statistical evidence on which move is best. Figure 3 shows the pseudo-code for this approach. Some characteristics of this approach include:

1 The program iterates on the number of samples taken.
2 The search done for each sample usually goes to the end of the game. For poker, leaf node evaluations can be the game result.
3 The search is often full depth. In poker, the search goes to the end of the game, but in backgammon this is impractical.
4 Heuristic evaluation usually occurs at the interior nodes of the search to determine the appropriate opponent actions and our action.
5 Usually a subset of interior node alternatives are considered, to reduce the cost of a sample. In poker, we consider a single action at each opponent's turn.

The simulation benefits from *selective* samples that use information from the game state (i.e. are context sensitive), rather than a uniform distribution or other fixed distribution sampling technique.



(a) Alpha-Beta Framework

(b) Sampling Framework

Figure 2. Comparing two different search frameworks.

Similar to alpha-beta, confidence in the answer increases as more nodes are evaluated. However, diminishing returns take over after a statistically significant number of trials have been performed. Selective sampling greatly reduces the number of nodes to search, just as cutoffs reduces the search tree size for alpha-beta.

For poker, each sample taken increases the program's confidence in the EV for that betting decision. The program is not only sampling the distribution of opponent hands. Since it considers only one opponent action at each decision point, it is also sampling part of the decision tree. Figure 2b) illustrates the portion of the total search space explored by Loki (or by any other game that always simulates to leaf nodes).

```
obvious_move = NO;
trials = 0;
while( ( trials <= MAX_TRIALS ) and
         ( obvious_move == NO ) )
{
  trials = trials + 1;
  Pos = current_state_of_the_game +
        ( selective_sampling to
          generate_missing_information );
  for( each legal move m )
  {
    value[m] += Search( pos.m, info );
  }
  if( ∃ i such that
       value[ i ] >> value[ j ]( ∀j, j ≠ i ) )
  {
    obvious_move = YES;
  }
}
select decision based on value[];
```

Figure 3. Framework for two-player, zero-sum, imperfect information games.

An important feature of the simulation-based framework is the notion of an obvious move. 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 choice.

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. A more sophisticated evaluation function can improve the quality of the search (simulation), at the cost of reducing the size of the tree (number of samples). Finding the right balance between the cost per trial and the number of trials is an interesting problem.

Conclusions

A simulation-based betting strategy for poker appears to be superior to the static evaluation-based alternative. The success of our approach depends on the quality of the `ProbTriple()` routine, which contains the expert's knowledge. However, even our crude initial `ProbTriple()` routine is better than our best, hand-tuned betting strategy. We are still in the early stages of our work, and the probability triple generating routine is still primitive. We believe that significant gains can be made by improving this routine, and refining the selection methods to use more game-state information.

This paper proposes that the selective sampling simulation-based framework should become a standard technique for games having elements of non-determinism and imperfect information. This powerful method gathers statistical evidence to compensate for a lack of information. Selective sampling is important for increasing the quality of the information obtained.

While the notion of simulation-based selective sampling is not new to game-playing program developers, it is a technique that is repeatedly discovered. This technique needs to be recognized as a fundamental tool for developing not only game-playing programs, but many other applications that deal with imperfect information.

Acknowledgments

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

References

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

D. Billings, D. Papp, J. Schaeffer and D. Szafron, 1998a. "Opponent Modeling in Poker", AAAI, pp. 493-499.

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

M. Ginsberg, 1996. "Partition Search", AAAI, pp. 228-233.

M. Ginsberg, 1998. "GIB: Steps Towards an Expert-Level Bridge-Playing Program", unpublished manuscript.

D. Koller and A. Pfeffer, 1997. "Representations and Solutions for Game-Theoretic Problems," *Artificial Intelligence* 94(1-2), 167-215.

D. Papp, 1998. "Dealing with Imperfect Information in Poker", M.Sc. thesis, Department of Computing Science, University of Alberta.

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

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.