Better Automated Abstraction Techniques for Imperfect Information Games, with Application to Texas Hold'em Poker*

Andrew Gilpin
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
gilpin@cs.cmu.edu

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
sandholm@cs.cmu.edu

ABSTRACT

We present new approximation methods for computing game-theoretic strategies for sequential games of imperfect information. At a high level, we contribute two new ideas. First, we introduce a new state-space abstraction algorithm. In each round of the game, there is a limit to the number of strategically different situations that an equilibrium-finding algorithm can handle. Given this constraint, we use clustering to discover similar positions, and we compute the abstraction via an integer program that minimizes the expected error at each stage of the game. Second, we present a method for computing the leaf payoffs for a truncated version of the game by simulating the actions in the remaining portion of the game. This allows the equilibrium-finding algorithm to take into account the entire game tree while having to explicitly solve only a truncated version. Experiments show that each of our two new techniques improves performance dramatically in Texas Hold'em poker. The techniques lead to a drastic improvement over prior approaches for automatically generating agents, and our agent plays competitively even against the best agents overall.

Categories and Subject Descriptors

I.2 [**Computing Methodologies**]: Artificial Intelligence

General Terms

Algorithms, Economics

Keywords

Computational game theory, equilibrium computation, abstraction, automated abstraction, poker

*This material is based upon work supported by the National Science Foundation under ITR grant IIS-0427858.

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

1. INTRODUCTION

Poker is an enormously popular card game played around the world. The 2006 World Series of Poker had over $82 million in total prize money for the main event. Increasingly, poker players compete in online casinos, and television stations regularly broadcast poker tournaments. Unlike games of perfect information (such as chess and Go) in which players are perfectly informed about the state of the world, in poker, players face uncertainty stemming from the opponents' cards, and future actions by nature (*i.e.*, future cards from the deck). For these and other reasons, poker has been identified as an important research area in AI [7].

In environments with more than one agent, the outcome of one agent may depend on the actions of the other agents. Consequently, in determining what action to take, an agent must consider the possible actions of the other agents. Game theory provides the mathematical definitions (aka. solution concepts) for how rational agents should behave in such settings. However, the computational problem of actually determining such strategies remains difficult in many settings. In this paper, we present new, better computational methods for applying game theory-based approaches to large real-world games of imperfect information.

Almost since the field's founding, game theory has been used to analyze different aspects of poker [9, 49, 3, 31, 35, 2, 24, 25, 37, 22, 18, 12, 41, 42]. That early work was limited to tiny games that could be solved by hand. In fact, most of that work was focused on computing *analytical solutions* for various stylized versions of poker. For example, one typical assumption in that line of work is for the cards to be drawn uniformly at random from the unit interval, followed by a simple betting protocol. Although this approach seems likely to be of use only for extremely small games, recently there has been renewed interest in extending some of these models and determing analytical solutions from them with the goal of applying them to certain situations in real poker games [16, 15]. Simplified versions of poker have also been developed for illustrative examples in education [39]. From a cognitive modelling and analysis perspective, poker has proved to be a fertile environment for research [17, 10, 11].

For sequential games of imperfect information, one could try to find an equilibrium using the normal (matrix) form, where every contingency plan of the agent is a pure strategy for the agent. Unfortunately (even if equivalent strategies are replaced by a single strategy [30]) this representation

is generally exponential in the size of the game tree [50]. The *sequence form* representation [40, 26, 50], which considers sequences of play rather than contingency plans, is often more compact. For two-player zero-sum games, there is a polynomial-sized (in the size of the game tree) linear programming (LP) formulation based on the sequence form such that strategies for players 1 and 2 correspond to primal and dual variables. Thus, a minimax solution[1] for reasonably-sized two-player zero-sum games can be computed using this method [50, 26, 27].[2]

Using that approach, Koller and Pfeffer (1997) determined solutions to poker games with up to 140,000 nodes. That approach scales to games with about a million nodes [20]. For a medium-sized (3.1 billion nodes) variant of poker called Rhode Island Hold'em, game theory-based solutions have been developed using a lossy abstraction followed by linear programming [44], and recently optimal strategies for this game were determined using lossless automated abstraction followed by linear programming [20].

The problem of developing strong players for Texas Hold-'em remains challenging. Recently there has been a surge of research into new techniques for approaching this problem [28, 48, 7, 6, 5, 20, 19, 34, 1]. One approach has been *opponent modeling*, in which a poker-playing program attempts to identify and exploit weaknesses in the opponents [8, 13, 5, 23, 46, 47, 43]. The most successful Texas Hold'em program from that line of research is *Vexbot* [5], which combines opponent modeling with miximax search (a variant of minimax search which allows the players to move probabilistically according to some model to account for the presence of imperfect information), and is available in the commercial product *Poker Academy Pro*.

The first notable game theory-based player for Texas Hold-'em used expert-designed manual abstractions and is competitive with advanced human players [6]. A player based on the techniques developed in that paper is available in *Poker Academy Pro* as *Sparbot*. Recently, the game theory-based player *GS1* was presented, which featured automated abstraction and real-time equilibrium approximation [19, 21]. It was shown to be competitive with *Sparbot* and *Vexbot*.

We develop methods for automatically generating players for sequential games of imperfect information. Our goal is robust players that play as strongly as possible against strong opponents (rather than, say, maximally exploiting weak opponents). Thus we adopt the game theory-based approach. We present new, better techniques that allow us to approach the problem from a game-theoretic point of view, while mitigating the computational problems. We apply the methods to Texas Hold'em. Section 2 reviews the rules of that game. Section 3 overviews our approach. The main new ideas in this paper are

1. improved automated abstraction using clustering and integer programming, and
2. computing the leaf payoffs for a truncated version of the game by simulating the actions in the remaining portion of the game. This allows the equilibrium-finding algorithm to take into account the entire game tree while having to explicitly solve only a truncated version.

These are presented in detail in Sections 4 and 5, respectively. Section 6 presents experiments.

2. RULES OF TEXAS HOLD'EM POKER

There are many variations of Texas Hold'em. As most prior work on poker, we focus on the setting with two players, called *heads-up*. Another difference between variations is the betting structure. Again, as most prior research, we focus on *limit* poker, in which the betting amounts adhere to a restricted format (see next paragraph). The AAAI Computer Poker Competition held in July 2006 also used that betting structure.[3,4]

The basic rules of two-player limit Texas Hold'em are as follows.

- Before any cards are dealt, the first player, called the *small blind*, contributes one chip to the pot; the second player (*big blind*) contributes two chips.[5]
- Each player is dealt two *hole cards* from a randomly shuffled standard deck of 52 cards.
- Next, the players participate in the first of four betting rounds, called the *pre-flop*. The small blind acts first; she may either call the big blind (contribute one chip), raise (three chips), or fold (zero chips). The players then alternate either calling the current bet (contributing two chips), raising the bet (four chips), or folding (zero chips). In the event of a fold, the folding player forfeits the game and the other player wins all of the chips in the pot. Once a player calls a bet, the betting round finishes. The number of raises allowed is limited to four in each round.
- The second round is called the *flop*. Three *community cards* are dealt face-up, and a betting round takes place with bets equal to two chips. The big blind player is the first to act, and there are no blind bets placed in this round.
- The third and fourth rounds are called the *turn* and the *river*. In each round, a single card is dealt face-up, and a betting round similar to the flop betting round takes place, but with bets equal to four chips.
- The *showdown* occurs when the last betting round ends with neither player folding. Each player uses the seven cards available (their two hole cards along with the five community cards) to form the best five-card poker hand, where the hands are ranked in the usual order. The player with the best hand wins the pot; in the event of a tie, the players split the pot.

[1]Minimax solutions are robust in that there is no equilibrium selection problem: an agent's minimax strategy guarantees at least the agent's minimax value even if the opponent fails to play his minimax strategy. Throughout this paper, we are referring to a minimax solution when we use the term equilibrium.

[2]Recently this approach was extended to handle computing *sequential equilibria* [29] as well [33].

[3]Other popular variants include *no-limit*, in which players may bet any amount up to their current bankroll, and *pot-limit*, in which players may bet any amount up to the current size of the pot.

[4]Researchers have recently begun to develop game theory-based poker-playing programs for no-limit Texas Hold'em as well [34, 1].

[5]The exact monetary value of a chip is irrelevant. Thus we refer only to the quantity of chips.

3. OVERVIEW OF OUR PLAYER

As discussed in the previous section, Texas Hold'em consists of four betting rounds. We separate the game into two phases, which we describe in turn.

3.1 Phase 1

In the first phase, we solve an LP that models the strategic interaction in the first three rounds of the game, where the payoffs at the end of the third round are estimated based on an expectation of the actions in the fourth round. Thus, this model is based on actions in the entire game. (We describe the details of this payoff computation in Section 5.) We perform this first phase computation offline.

Because the LP corresponding to this three-round version of Texas Hold'em is too large to solve, we must employ abstraction techniques to reduce the size of the LP. Based on the available computational resources for solving the LP, there is a limit to the number of strategically different hands that can be considered in each round. Based on the computational resources available to us, we determined we could handle up to 15 strategically different hands in the first round, 225 in the second round, and 900 in the third round. Solving the LP took 6 days and 70 gigabytes of RAM using the barrier method in CPLEX 10.0 (using one 1.65 GHz CPU on an IBM p570 computer). Of course, as computing speed increases over time and enables larger LPs to be solved, we can apply our algorithm to compute finer abstractions. (The details of our abstraction algorithm are given in Section 4.)

3.2 Phase 2

Although the first phase computation outputs strategies for the first three rounds, we only actually use the strategies it computes for the first two rounds. We don't use the third-round strategy for actual play because 1) it may be inaccurate because the fourth round was modeled based on expectations rather than game theory, as discussed above, and 2) we can use a finer abstraction if we compute the third-round strategy in real-time.

The strategies we use for the third and fourth rounds are computed in real-time while the game is being played. We do this so that the reasoning can be specific to the situation at hand (*i.e.*, cards on the table and betting history); the mere number of such situations precludes precomputing the answer to each one of them. Once the turn card appears at the beginning of the third betting round, we compute an equilibrium approximation for the remaining game. We do this as follows. First, we update the players' hand probabilities using Bayes' rule on the Phase 1 strategies and the observed betting history. Second, we use our automated abstraction algorithm (Section 4) to construct and solve a smaller LP. Here we are considering 10 strategically different hands in the third round, and 100 in the fourth round. This is based on computational experiments we performed to find the finest abstraction for which we could usually solve (at least near-optimally) the corresponding LP in under one minute.

3.3 Comparison to other game theory-based Texas Hold'em players

Our player is closely related to two previous game theory-based Texas Hold'em players.

The first, *Sparbot* [6], also uses two phases. Like our player, *Sparbot* considers three rounds in the first phase. *Sparbot* assumes no betting in the fourth round while our player simulates fourth-round play as we will discuss in Section 5. Another difference is that *Sparbot* considers three rounds in the second phase, while we consider two. This enables us to use a much finer abstraction when computing strategies for the third and fourth rounds. This is especially important since the size of the bets in the third and fourth rounds is twice the bet level in the first two rounds.

Like our player, *Sparbot* also uses abstraction. However, the abstraction used by *Sparbot* is expert-defined, whereas ours is computed automatically by an algorithm. As computing speed increases over time, our player will be able to take advantage of that and use finer abstractions (by simply adjusting, for each betting round, the parameter that defines the number of classes that the abstraction algorithm should form). In contrast, an expert-defined abstraction would have to be manually recreated when computation speed enables finer abstraction.

The second Texas Hold'em player that our player is closely related to is *GS1* [19]. The main difference is that *GS1* is based on the vanilla *GameShrink* algorithm [20], which has drawbacks when used to create lossy abstractions, as we will discuss in Section 4. Our player uses a new automated abstraction algorithm which is better suited for computing abstractions for equilibrium approximation. Also, *GS1* only considers the first two betting rounds (and assumes no betting in the third and fourth round) when computing strategies for the first phase. In poker, this makes it particularly vulnerable to slow-playing[6] as well as making it unlikely to employ such a sophisticated strategy itself.

4. AUTOMATED ABSTRACTION USING CLUSTERING AND INTEGER PROGRAMMING

Recently, the *GameShrink* algorithm for automatically computing abstractions in sequential games of imperfect information was presented [20], and a Texas Hold'em player, *GS1*, based on *GameShrink* was shown to be competitive with *Sparbot* and *Vexbot* [19]. However, we observe that *GameShrink* suffers from three major drawbacks.[7]

- The first, and most serious, is that the abstraction that *GameShrink* computes can be highly inaccurate because the grouping of states is in a sense greedy. For example, if *GameShrink* determines that hand A is similar to hand B, and then determines that hand B is similar to hand C, it will group A and C together, despite the fact that A and C may not be very similar. The quality of the abstraction can be even worse when a longer sequence of such comparisons leads to grouping together extremely different hands. Stated differently, the greedy aspect of the algorithm leads to

[6]Slow-playing is a technique in poker in which a player with a strong hand acts as if they have a weak hand early in the game in order to extract more bets from the other player later in the game [45].

[7]When *GameShrink* is used in the lossless (exact rather than approximation) mode, these criticisms do not apply: it finds an equilibrium-preserving abstraction. However, if one were to apply the lossless mode to Texas Hold'em, the LP corresponding to each phase would be way too large to solve.

lopsided classes where large classes are likely to attract even more states into the class.

- The second drawback to *GameShrink* is that there is no way to directly specify how many classes the abstraction algorithm should yield (overall or at any specific betting round). Rather, there is a parameter (for each round) that specifies a threshold of how different states can be and still be considered the same. If one knows how large an LP can be solved, one cannot create an LP of that size by specifying the number of classes in the abstraction directly; rather one must use trial-and-error (or some variant of binary search applied to the setting of multiple parameters) to pick the similarity thresholds (one for each betting round) in a way that yields an LP of the desired size.
- The third drawback to *GameShrink* is its scalability. In particular, the time needed to compute an abstraction for a three-round truncated version of Texas Hold'em was over a month. Furthermore, it would have to be executed in the inner loop of the parameter guessing algorithm of the previous paragraph (*i.e.*, once for each setting of the parameters).

In this section we describe a new abstraction algorithm that eliminates these problems.

GameShrink operates on a data structure called the *filtered signal tree* [20]. This structure captures all of the information that the players receive from moves of nature, and is also used to represent the actual abstraction. We introduce a similar structure for our algorithm, which we will call the *abstraction tree*. For Texas Hold'em, the basic abstraction tree is initialized as follows. The root node contains $\binom{52}{2} = 1326$ children, one for each possible pair of hole cards that a player may be dealt. Each of these children has $\binom{50}{3}$ children, each corresponding to the possible flops that can appear after the two hole cards in the parent node have already been dealt. Similarly, the nodes at the next two levels have 47 and 46 children corresponding to the possible turn and river cards, respectively. Figure 1 provides an illustration. This structure is by no means limited to poker, but here for simplicity we only describe it in terms of Texas Hold'em poker since that is the primary application of this paper.

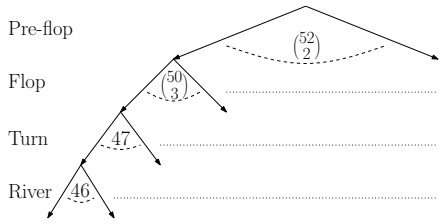

Figure 1: The initial abstraction tree of Texas Hold'em.

As described in the previous section, we limit the number of strategically different hands we can consider in the first round to 15. Thus, we need to group each of the $\binom{52}{2} = 1326$ different hands into 15 classes. We treat this as a clustering problem. To perform the clustering, we must first define a metric to determine the similarity of two hands. Letting (w, l, d) be the number of possible wins, losses, and draws (based on the roll-out of the remaining cards), we compute the hand's value as $w - l + d/2$, and we take the distance between two hands to be the absolute difference between their values. This gives us the necessary ingredients to apply the k-means clustering algorithm [32], which we specialize here to our problem:

ALGORITHM 1. *k-means clustering for poker hands*

1. *Create k centroid points in the interval between the minimum and maximum hand values.*
2. *Assign each hand to the nearest centroid.*
3. *Adjust each centroid to be the mean of their assigned hand values.*
4. *Repeat steps 2 and 3 until convergence.*

This algorithm is guaranteed to converge, but it may find a local optimum. Therefore, in our implementation we run it several times with different starting points to try to find a global optimum. For a given clustering, we can compute the error (according to the value measure) that we would expect to have when using the abstraction.

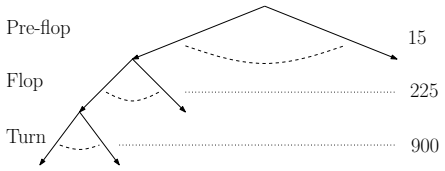

Figure 2: Abstraction tree for our player's Phase 1 equilibrium approximation.

For the later stages of the game, we again want to determine what the best abstraction classes are. Here we face the additional problem of determining how many children each parent in the abstraction tree can have. As Figure 2 illustrates, we use a limit of 225 total child edges that we can use at this level.[8] How should the right to have 225 children (abstraction classes that have not yet been generated at this stage) be divided among the 15 parents? We model and solve this problem as a 0-1 integer program [36] as follows. Our objective is to minimize the expected error in the abstraction. Thus, for each of the 15 parent nodes, we run the k-means algorithm presented above for values of k between 1 and 30.[9] We denote the expected error when node i has k children by $c_{i,k}$. We denote by p_i the probability of getting dealt a hand that is in abstraction class i (*i.e.*, in parent i); this is simply the number of hands in i divided by $\binom{52}{2}$. Based on these computations, the following 0-1 integer program finds the abstraction that minimizes the overall expected error for the second level:

$$\begin{aligned} \min \quad & \sum_{i=1}^{15} p_i \sum_{k=1}^{30} c_{i,k} x_{i,k} \\ \text{s.t.} \quad & \sum_{i=1}^{15} \sum_{k=1}^{30} k x_{i,k} \leq 225 \\ & \sum_{k=1}^{30} x_{i,k} = 1 \quad \forall i \\ & x_{i,k} \in \{0,1\} \end{aligned}$$

[8] This limit was again determined based on the size of the LP that was solvable.

[9] The maximum number of children of a particular node in an optimal abstraction will depend on several factors. For this problem, we observed that 30 was an upper bound on this number.

The decision variable $x_{i,k}$ is set to 1 if and only if node i has k children. The first constraint ensures that the limit on the overall number of children is not exceeded. The second constraint ensures that a decision is made for each node. This problem is a generalized knapsack problem, and although NP-complete, can be solved efficiently using off-the-shelf integer programming solvers (*e.g.*, CPLEX solves this problem in less than one second at the root node of the branch-and-bound search tree).

We repeat this procedure for the third betting round (with the second-round abstraction classes as the parents, and a limit of 900 on the maximum number of children). This completes the abstraction that is used in Phase 1.[10]

For Phase 2, we compute a third and fourth-round abstraction using the same approach. We do this separately for each of the $\binom{52}{4}$ possible flop and turn combinations.[11]

5. ESTIMATING PAYOFFS OF A TRUNCATED GAME

The second main new idea of this paper is estimating the leaf payoffs for a truncated version of a game by simulating the actions in the remaining portion of the game. This allows the equilibrium-finding algorithm to take into account the entire game tree while having to explicitly solve only a truncated version. This section covers the idea in the context of Texas Hold'em.

In both *Sparbot* and *GS1*, the payoffs that are computed for the leaf nodes at the end of the truncated game (Phase 1) are based on the betting history leading to that node and the expected value of winning that hand assuming that no more betting takes place in later rounds (*i.e.* the payoffs are averaged over the possible fourth-round cards drawn uniformly at random, and assuming that neither player bets in the final round). This completely ignores the fact that later betting actions affect the expected payoff of a node in the game tree.

Instead of ignoring the fourth-round betting, we in effect incorporate it into the truncated game tree by simulating the betting actions for the fourth round (using reasonable fixed randomized strategies for the fourth round), and then using these payoffs as the payoffs in the truncated game. This is intended to mitigate the negative effects of performing an equilibrium analysis on a truncated game.

At the beginning of the fourth round, each player has two hole cards and there are five community cards on the table. Letting (w, l, d) be the number of possible wins, losses, and draws for a player in that situation, we compute the hand's value using the formula $w - l + d/2$ (this is the same formula used by our clustering algorithm). For hands in the fourth round, this gives a value in the interval $[-990, 990]$. Using the history from 343,513 games of *Sparbot* in self-play

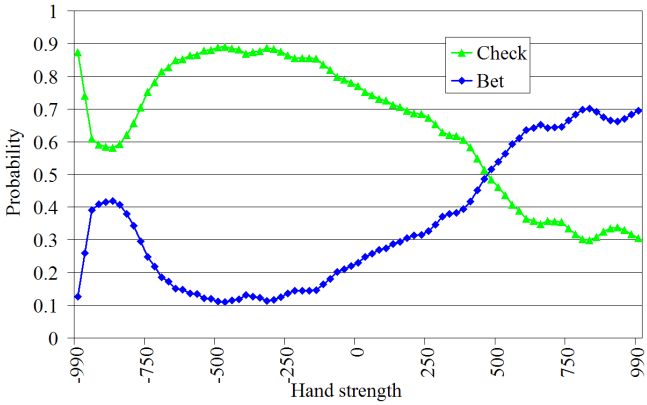

Figure 3: First player's empirical action probabilities as a function of hand strength at the beginning of the fourth betting round.

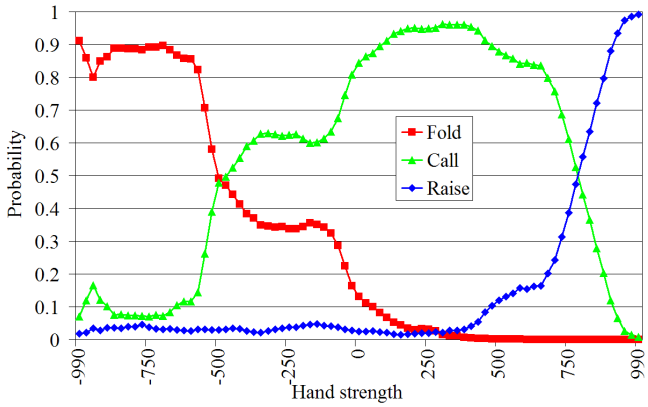

Figure 4: Second player's empirical action probabilities as a function of hand strength for the fourth betting round after the first player has bet.

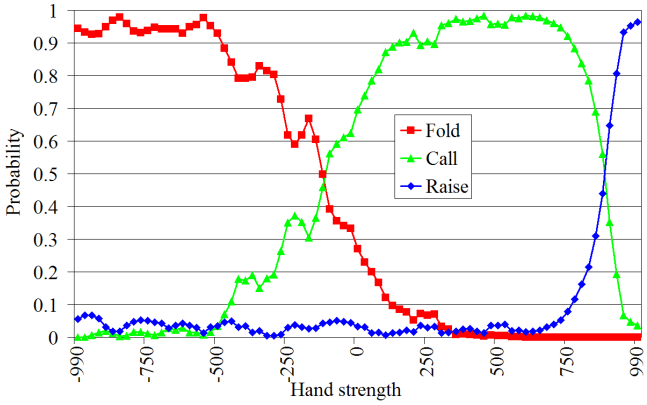

Figure 5: First player's empirical action probabilities as a function of hand strength for the fourth betting round after the first player has bet and the second player has raised.

[10]As discussed, our technique optimizes the abstraction round by round, *i.e.*, level by level in the abstraction tree. A better abstraction (even for the same similarity metric) could conceivably be obtained by optimizing all rounds in one holistic optimization. However, that seems infeasible. First, the optimization problem would be nonlinear because the probabilities at a given level depend on the abstraction at previous levels of the tree. Second, the number of decision variables in the problem would be exponential in the size of the initial abstraction tree (which itself is large), even if the number of abstraction classes for each level is fixed.

[11]Most of the computation time of the abstraction algorithm is spent running the k-means clustering algorithm. Our straightforward implementation of the latter could be improved by using sophisticated data structures such as a kd-tree [38] or performing bootstrap averaging to find the initial clusters [14]. This would also allow one to run the k-means clustering more times and thus have an even better chance of finding the global optimum of any individual k-means clustering problem.

(of which 187,850 went to the fourth round), we determined the probability of performing each possible action at each player's turn to act as a function of the hand value.[12,13] To illustrate this, Figures 3–5 show these smoothed probabilities for three particular points to act.

Of course, since these probabilities are only conditioned on the hand value (and ignore the betting history in the first three rounds), they do not exactly capture the strategy used by *Sparbot* in the fourth round. However, conditioning the probabilities on the betting history as well would have required a vast number of additional trials, as well as much more space to store the result. Conditioning the actions on hand value alone is a practical way of striking that trade-off.

6. EXPERIMENTAL RESULTS

We conducted a host of experiments against the leading prior Texas Hold'em programs, *GS1*, *SparBot*, and *VexBot*. We used the *series competition* format of the first AAAI Computer Poker Competition (held in July 2006).[14] In our experiments, our player competed with each of the other players in 50 1,000-hand series, where the players' memories are reset before each series (this resetting only affects *Vexbot* since the other players do not perform any opponent modeling). The purpose of resetting the memories after each series is to give a learning bot a fair amount of time to develop an exploitative strategy, but not a completely unrealistic amount. 1,000 hands per series is more than reasonable since most games between humans last for at most a few hundred hands [23]. The AAAI competition also used 1,000 hands per series.

The results are presented in Table 1. Based on the winning records, our agent, generated using the techniques described above, outperforms the leading prior poker-playing programs. For *GS1* (the leading prior automatically generated agent for the game), this result is statistically significant according to the sign test, with p-value 3.06×10^{-4}. In terms of overall chips won, our program beats the prior game theory-based players, *GS1* and *Sparbot*, but loses to *Vexbot* by a small margin. The margin of victory over *GS1* is statistically significant as the estimated variance of small bets won or lost per hand is ± 0.0268 small bets (*i.e.*, two chips) per hand for 50,000 hands [4]. However, the comparisons with *Sparbot* and *Vexbot* are in effect statistical ties.[15]

Interestingly, our player has a better winning percentage against *Vexbot* than it does against *Sparbot*; yet based on the win rate of average chips won and lost per hand, it performs better against *Sparbot*. This is possibly due to the fact that *Vexbot* is able to exploit a weakness in our player that enables it to win a large amount of chips on a few hands, but those particular hands do not come up often enough to affect the match winning percentages. Another possibility is that our player is playing much more conservatively than *Vexbot*, which enables it to obtain a better winning percentage, but it is not playing aggressively enough on certain hands to capture a higher chip win rate. Exploring these possibilities could potentially lead to further improvements in our player.

In addition to evaluating the performance of our player against existing players, out of scientific interest we also wanted to measure the individual effect of the two main techniques we contributed in this paper, namely our improved automated abstraction algorithm and our technique of estimating payoffs in a truncated game. The fourth row in Table 1 reports results from the comparison between our player, and our player using the old version of the *GameShrink* algorithm (as used in *GS1*) and without estimating the payoffs of the truncated game (but instead using a uniform roll-out as in *Sparbot* and *GS1*). The introduction of these two new techniques is a clear winner, with a p-value for the winning percentage of 2.27×10^{-12} and even a statistically significant win rate in terms of the number of chips won. The last two rows of Table 1 report the performance boost that each of the two new techniques yields individually. The improved automated abstraction algorithm produces a greater performance gain than the technique of estimating payoffs in the truncated game, but each of the two techniques yields a dramatic statistically significant improvement in performance.

7. CONCLUSIONS AND FUTURE RESEARCH

We presented new approximation methods for computing game-theoretic strategies for sequential games of imperfect information. We introduced two main new ideas.

- First, we introduced a new state-space abstraction algorithm. In each round of the game, there is a limit to the number of strategically different situations that an equilibrium-finding (*e.g.*, LP) algorithm can handle. Given this constraint, we use clustering to discover similar positions, and we compute the abstraction via an integer program that minimizes the expected error at each stage of the game.
- Second, we presented a method for computing the leaf payoffs for a truncated version of the game by simulating the actions in the remaining portion of the game (*e.g.*, based on statistics about actual play of fixed strong player(s) that may use mixed strategies). This allows the equilibrium-finding algorithm to take

[12]While *GS1* has been shown to be slightly stronger than *Sparbot* [19], we used *Sparbot's* fourth-round strategy because *Sparbot* plays instantaneously rather than doing real-time deliberation, so we were able to collect statistics faster.

[13]To mitigate the fact that 187,850 training examples do not densely cover all possible hand values, we bucketed hand values by rounding them to the nearest multiple of 25, and then smoothed by replacing each bucket's value by the average of it and both of its neighbors.

[14]One difference between our experiments and the format of the AAAI poker competition is that the AAAI competition performed *duplicate matches*, in which the deck shuffles are stored for each match, and replayed with the players' roles reversed. Using this approach, if one player receives particularly lucky cards during a match, this will be offset by the duplicate match in which the other player receives the lucky cards. Unfortunately, we have not yet been able to run such experiments due to the fact that the other players are only available in the *Poker Academy* software package, which does not support duplicate matches.

[15]Evaluating an agent's performance in a partially observable and stochastic environment is difficult due to the element of randomness. This difficulty is severe when evaluating poker-playing programs due to the tight relationship between luck and performance. Recent research has led to improved statistical techniques for comparing the relative performance of agents in uncertain domains [51]. We would like to incorporate these techniques in our future research (once the baseline player needed in using those techniques becomes publicly available and perhaps standardized).

Opponent	Series won by our player	Sign test p-value	Win rate of our player (small bets per hand)
GS1	38 of 50	3.06×10^{-4}	+0.0312
Sparbot	28 of 50	4.80×10^{-1}	+0.0043
Vexbot	32 of 50	6.49×10^{-2}	−0.0062
Our player without improved abstraction and without estimated payoffs	48 of 50	2.27×10^{-12}	+0.0287
Our player without improved abstraction	35 of 50	6.60×10^{-3}	+0.0273
Our player without estimated payoffs	44 of 50	3.24×10^{-8}	+0.0072

Table 1: Experimental results of our program against the leading prior programs, as well as in self-play with various features removed.

into account the entire game tree while having to explicitly solve only a truncated version.

Experiments showed that our Texas Hold'em poker-playing agent—that was generated using these techniques—drastically outperforms the best prior automatically generated agent, and is competitive with the leading prior agents overall. Based on the performance criterion of series wins and losses, our player is the strongest. We also showed that each of the two new ideas improves the performance of our player dramatically.

It is tempting to explore a tighter integration of the main two ideas of this paper. In particular, an interesting question to explore is whether it would be possible to use estimated leaf values in a truncated game (based on a statistical model) as the similarity metric for the abstraction. This would likely yield a better abstraction for the earlier rounds, but it would greatly increase the dependence of the player on the statistical model (specifically, on a particular opponent if the statistical model is built using a particular opponent). One intriguing idea is to perform the abstraction and statistical modeling in an iterative manner where the abstraction is refined based on a statistical model of the player in self-play, and where the statistical model is updated using observations of the player that uses the new abstraction. We leave this avenue as future research.

8. REFERENCES

[1] R. Andersson. Pseudo-optimal strategies in no-limit poker. Master's thesis, Umeå University, May 2006.

[2] R. Bellman. On games involving bluffing. *Rendiconti del Circolo Matematico di Palermo*, 1(2):139–156, 1952.

[3] R. Bellman and D. Blackwell. Some two-person games involving bluffing. *Proceedings of the National Academy of Sciences*, 35:600–605, 1949.

[4] D. Billings. Web posting at Poker Academy Forums, Meerkat API and AI Discussion, December 2005. `http://www.poker-academy.com/forums/viewtopic.php?t=1872`.

[5] D. Billings, M. Bowling, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Game tree search with adaptation in stochastic imperfect information games. In *Proceedings of the 4th International Conference on Computers and Games (CG)*, pages 21–34, Ramat-Gan, Israel, July 2004. Springer-Verlag.

[6] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, Acapulco, Mexico, 2003.

[7] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[8] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 493–499, Madison, WI, 1998.

[9] É. Borel. *Traité du calcul des probabilités et ses applications*, volume IV of *Applications aux jeux des hazard*. Gauthier-Villars, Paris, 1938.

[10] K. Burns. Heads-up face-off: On style and skill in the game of poker. In *Style and Meaning in Language, Art, Music, and Design: Papers from the 2004 Fall Symposium*, pages 15–22, Menlo Park, California, 2004. AAAI Press.

[11] K. Burns. Pared-down poker: Cutting to the core of command and control. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, pages 234–241, Colchester, UK, 2005.

[12] W. H. Cutler. An optimal strategy for pot-limit poker. *American Mathematical Monthly*, 82:368–376, 1975.

[13] A. Davidson. Opponent modeling in poker: Learning and acting in a hostile environment. Master's thesis, University of Alberta, 2002.

[14] I. Davidson and A. Satyanarayana. Speeding up k-means clustering by bootstrap averaging. In *IEEE Data Mining Workshop on Clustering Large Data Sets*, 2003.

[15] C. Ferguson, T. Ferguson, and C. Gawargy. Uniform(0,1) two-person poker models, 2004. Available at `http://www.math.ucla.edu/~tom/papers/poker2.pdf`.

[16] C. Ferguson and T. S. Ferguson. On the Borel and von Neumann poker models. *Game Theory and Applications*, 9:17–32, 2003.

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

[18] L. Friedman. Optimal bluffing strategies in poker. *Management Science*, 17(12):764–771, 1971.

[19] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and

real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Boston, MA, 2006.

[20] A. Gilpin and T. Sandholm. Finding equilibria in large sequential games of imperfect information. In *Proceedings of the ACM Conference on Electronic Commerce (ACM-EC)*, pages 160–169, Ann Arbor, MI, 2006.

[21] A. Gilpin and T. Sandholm. A Texas Hold'em poker player based on automated abstraction and real-time equilibrium computation. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 1453–1454, Hakodate, Japan, 2006. Demonstration Track.

[22] A. J. Goldman and J. J. Stone. A symmetric continuous poker model. *Journal of Research of the National Institute of Standards and Technology*, 64(B):35–40, 1960.

[23] B. Hoehn, F. Southey, R. C. Holte, and V. Bulitko. Effective short-term opponent exploitation in simplified poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 783–788, July 2005.

[24] R. Isaacs. A card game with bluffing. *American Mathematical Monthly*, 62:99–108, 1955.

[25] S. Karlin and R. Restrepo. Multi-stage poker models. In *Contributions to the Theory of Games*, volume 3 of *Annals of Mathematics Studies, Number 39*, pages 337–363. Princeton University Press, Princeton, New Jersey, 1957.

[26] D. Koller, N. Megiddo, and B. von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

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

[28] K. Korb, A. Nicholson, and N. Jitnah. Bayesian poker. In *Proceedings of the 15th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 343–350, Stockholm, Sweden, 1999.

[29] D. M. Kreps and R. Wilson. Sequential equilibria. *Econometrica*, 50(4):863–894, 1982.

[30] H. W. Kuhn. Extensive games. *Proc. of the National Academy of Sciences*, 36:570–576, 1950.

[31] H. W. Kuhn. A simplified two-person poker. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1 of *Annals of Mathematics Studies, 24*, pages 97–103. Princeton University Press, Princeton, New Jersey, 1950.

[32] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Berkeley, California, 1967. University of California Press.

[33] P. B. Miltersen and T. B. Sørensen. Computing sequential equilibria for two-player games. In *Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 107–116, 2006.

[34] P. B. Miltersen and T. B. Sørensen. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Honolulu, HI, USA, 2007. To appear.

[35] J. F. Nash and L. S. Shapley. A simple three-person poker game. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games*, volume 1, pages 105–116. Princeton University Press, 1950.

[36] G. Nemhauser and L. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, 1999.

[37] D. J. Newman. A model for "real" poker. *Operations Research*, 7:557–560, 1959.

[38] D. Pelleg and A. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Knowledge Discovery and Data Mining*, pages 277–281, 1999.

[39] D. H. Reiley, M. B. Urbancic, and M. Walker. Stripped-down poker: A classroom game with signaling and bluffing, February 2005. Working paper. Available at `http://economics.eller.arizona.edu/downloads/working_papers/Econ-WP-05-%11.pdf`.

[40] I. Romanovskii. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics*, 3:678–681, 1962.

[41] M. Sakaguchi. A note on the disadvantage for the sente in poker. *Mathematica Japonica*, 29:483–489, 1984.

[42] M. Sakaguchi and S. Sakai. Partial information in a simplified two person poker. *Mathematica Japonica*, 26:695–705, 1981.

[43] T. C. Schauenberg. Opponent modelling and search in poker. Master's thesis, University of Alberta, 2006.

[44] J. Shi and M. Littman. Abstraction methods for game theoretic poker. In *Computers and Games*, pages 333–345. Springer-Verlag, 2001.

[45] D. Sklansky. *The Theory of Poker*. Two Plus Two Publishing, fourth edition, 1999.

[46] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 550–558, July 2005.

[47] N. Sturtevant, M. Zinkevich, and M. Bowling. Prob-max^n: Opponent modeling in n-player games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1057–1063, Boston, MA, 2006.

[48] K. Takusagawa. Nash equilibrium of Texas Hold'em poker, 2000. Undergraduate thesis, Stanford University.

[49] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[50] B. von Stengel. Efficient computation of behavior strategies. *Games and Economic Behavior*, 14(2):220–246, 1996.

[51] M. Zinkevich, M. Bowling, N. Bard, M. Kan, and D. Billings. Optimal unbiased estimators for evaluating agent performance. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 573–578, Boston, MA, 2006.