The Challenge of Poker

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

December 6, 2001

Abstract

Poker is an interesting test-bed for artificial intelligence research. It is a game of imperfect information, where multiple competing agents must deal with probabilistic knowledge, risk assessment, and possible deception, not unlike decisions made in the real world. Opponent modeling is another difficult problem in decision-making applications, and it is essential to achieving high performance in poker.

This paper describes the design considerations and architecture of the poker program *Poki*. In addition to methods for hand evaluation and betting strategy, *Poki* uses learning techniques to construct statistical models of each opponent, and dynamically adapts to exploit observed patterns and tendencies. The result is a program capable of playing reasonably strong poker, but there remains considerable research to be done to play at a world-class level.

1 Introduction

The artificial intelligence community has recently benefited from the positive publicity generated by chess, checkers, backgammon, and Othello programs that are capable of defeating the best human players. However, there is an important difference between these board games and popular card games like bridge and poker. In the board games, players have complete knowledge of the entire game state, since everything is visible to both participants. In contrast, bridge and poker involve *imperfect information,* since the other players' cards are not known. Traditional methods like deep search have not been sufficient to play these games well, and dealing with imperfect information is the main reason that progress on strong bridge and poker programs has lagged behind the advances in other games. However, it is also the reason these games promise greater potential research benefits.

Poker has a rich history of study in other academic fields. Economists and mathematicians have applied a variety of analytical techniques to poker-related problems. For example, the earliest investigations in *game theory*, by luminaries such as John von Neumann and John Nash, used simplified poker to illustrate the fundamental principles [22, 23, 38].

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

We are attempting to build a program that is capable of playing poker at a world-class level. We have chosen to study the game of Texas Hold'em, which is one of the most strategically complex and popular variants of poker. Our experiences with our first program, called *Loki*, were positive [6, 7]. In 1999, we rewrote the program, christening the new system *Poki*.

These programs have been playing on Internet poker servers since 1997, and have accrued an impressive winning record, albeit against weak opponents. Early versions of the program were only able to break even against better opposition, but recent improvements have made the program substantially stronger, and it is now winning comfortably in those more difficult games. Although most of these Internet games simulate real game conditions quite well, it would be premature to extrapolate that degree of success to games where real money is at stake. Regardless, analysis of *Poki*'s play indicates that it is not yet ready to challenge the best human players. Ongoing research is attempting to bridge that gap.

Section 2 of this article reviews previous work and related research on poker. Section 3 provides an overview of Texas Hold'em, including an illustrative example of strategic concepts, and a minimal set of requirements necessary to achieve world-class play. An overview of *Poki*'s architecture is described in Section 4. Section 5 discusses the program's betting strategy, detailing some of the components of the system. The special problem of opponent modeling is addressed in Section 6. Experimental methods and the performance of the program are assessed in Section 7. Section 8 provides a generalized framework for non-deterministic games, based on *Poki*'s simulation search strategy. Section 9 discusses the challenges that remain for building a world-class poker-playing program.

2 Other Research

There are several ways that poker can be used for artificial intelligence research. One approach is to study simplified variants that are easier to analyze. We have already mentioned some of the the founding work in game theory, which could only handle extremely simple poker games. An example is Kuhn's game for two players, using a three-card deck, one-card hands, and one betting round, with at most two betting decisions [21]. While this was sufficient to demonstrate certain fundamental principles of game theory, it bears little resemblance to normal competitive poker variations.

Mathematicians have also explored many interesting problems related to poker, and highly simplified variations are again sufficient to provide complex problems ([26] for example).

Another way to reduce the complexity of the problem is to look at a subset of the game, and try to address each sub-problem in isolation. Several attempts have been made to apply machine learning techniques to a particular aspect of poker (some examples include [9, 20, 34, 39]). Similarly, many studies only look at two-player poker games. Multi-player games are vastly more complicated, even with the usual assumption of no co-operative behavior between players. The danger with any type of simplification is that it can destroy the most challenging and interesting aspects of the problem.

An alternate approach, which we advocate, is to tackle the entire problem: choose a real variant of poker and address all of the considerations necessary to build a program that performs at a level comparable to or beyond the best human players. Clearly this is a most ambitious undertaking, but also the one that promises the most exciting research contributions if successful.

Nicholas Findler worked on and off for 20 years on a poker-playing program for 5-card Draw poker [12]. His primary objective was to model human cognitive processes, and he developed a program that could learn. While successful to a degree, the program itself was not reported to be a strong player. Furthermore, the game of 5-card Draw, although quite popular at that time, is not as strategically complex as other poker games, such as 7-card Stud and Texas Hold'em.

Some success in analyzing larger scale poker variants was achieved by Norman Zadeh in the 1970s, and much of this work is still of value today [40, 41]. Other individuals, including expert players with a background in mathematics, have gained considerable insight into "real" poker by using partial mathematical analyses, simulation, and *ad hoc* expert experience ([33] is a popular example).

There is a viable middle-ground between the theoretical and empirical approaches. Recently, Daphne Koller and Avi Pfeffer have revived the possibility of investigating poker from a game-theoretic point of view [19]. They presented a new algorithm for finding optimal randomized strategies in two-player imperfect information games, which avoids the usual exponential blow-up of the problem size when converting it to *normal form*. This algorithm is used in their *Gala* system, a tool for specifying and solving a greatly extended range of such problems. However, the size of the translated problems is still proportional to the size of the game tree, which is prohibitively large for most common variations of poker. For this reason, the authors concluded "...we are nowhere close to being able to solve huge games such as full-scale poker, and it is unlikely that we will ever be able to do so."

Nevertheless, this does raise the interesting possibility of computing *near-optimal* solutions for real poker variants, which might require far less computation to obtain a satisfactory answer. This is analogous to efficient approximation algorithms for certain combinatorial optimization problems that are known to be intractable (NP-hard).

One obvious technique for simplifying the problem is to use abstraction, collecting many instances of similar sub-problems into a single class. There are many states in the poker game tree that are isomorphic to each other (for example, a hand where all relevant cards are hearts and diamonds is isomorphic to two corresponding hands with all spades and clubs). Beyond this, strictly distinct cases might be so similar that the appropriate strategy is essentially identical. For example, the smallest card of a hand being a deuce instead of a trey may have no bearing on the outcome. This is analogous to the approach used by Matt Ginsberg in *partition search*, where he defined equivalence classes for the smallest cards of each suit in a bridge hand [14]. Jiefu Shi and Michael Littman have made some preliminary attempts along these lines to produce near-optimal solutions for a scaled-down version of Texas Hold'em [31].

A second method is aimed at constructing a shallower game tree, using expected value estimates to effectively truncate subtrees. This is similar to the method used so successfully in most perfect information games, where an evaluation function is applied to the leaves of a depth-limited search. However, it is not as easy to accomplish because, unlike perfect information games, the states of a poker game tree are not independent of each other (specifically, we cannot distinguish states where the opponent has different possible hidden cards). Ken Takusagawa, a former student of Koller and Pfeffer, has extended their work by combining this method with abstraction, to produce some near-optimal solutions for particular scenarios of Texas Hold'em [35]. Alex Selby has applied the Simplex algorithm directly to two-player pre-flop Hold'em, and has computed optimal solutions for that re-defined game, using expected values in place of the post-flop phase [28].

Our own empirical studies over the past few years have used similar methods of abstraction and expected value estimation to reduce the computational complexity of the problem, so the approaches are not as different as they may at first appear. It will be interesting to see if these theoretical "hybrid techniques" can be applied directly to a competitive poker program in the future.

3 Texas Hold'em

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 generally considered to be the most strategically complex poker variant that is widely played in casinos and card clubs. It is also convenient because it has particularly simple rules and logistics.

We assume the reader is familiar with the ranking of poker hands (if not, many good introductions to poker can be found on the Internet). In the following description, and throughout the paper, italics are used for common poker terms, which are defined in the glossary (Appendix A).

3.1 Rules of Play

A *hand*[1] of Texas Hold'em begins with the *pre-flop*. Each player is dealt two *hole cards* face down, followed by the first round of betting, which is started with two forced bets called the *small blind* and the *big blind*. Three *community cards*, collectively called the *flop*, are then dealt face up on the table, 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. The players still active in the game at that time reveal their two hole cards for the *showdown*. The best five-card poker hand formed from each player's two private hole cards and the five public 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 amount of each bet is strictly controlled in each betting round.[2] There are two denominations of bets, called a *small bet* and a *big bet*, which will be \$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 always \$20. In general, when it is a player's turn to act, one of three betting options is available: *fold*, *check/call*, or *bet/raise*.[3] 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.

3.2 Poker Strategy

To illustrate some of the decisions one must face in Hold'em, we will present a sample hand, with some typical reasoning a good player might go through. This hand is relatively basic, in order to make the example easier to follow. Many complex interactions can contribute to much more difficult situations, but it is hoped that this example will suffice to demonstrate some of the strategic richness of the game.

The game is \$10-\$20 Limit Hold'em with ten players. We "have the button", meaning that we will be the last to act in each betting round, which is an advantage. The two players to the left of us post the *small blind* (\$5) and the *big blind* (\$10), and the cards are dealt. The action begins with the player to

[1] The term "hand" is used in two ways: to denote a player's private cards, and to refer to one complete deal, or game. We have not tried to avoid the possible ambiguity, preferring to use the same terminology used by most serious poker players whenever possible. In each instance, the intended meaning of "hand" should be clear from the context.

[2] In No-limit Texas Hold'em, there are no restrictions on the size of bets; a player may wager any amount, up to their entire stack, at any time.

[3] A *check* and a *call* are logically equivalent, in that the betting level is not increased. The term *check* is used when the current betting level is zero, and *call* when there has been a wager in the current betting round. Similarly, a *bet* and a *raise* are logically equivalent, but the term *bet* is used for the first wager of a betting round.

the left of the big blind, who calls \$10 (we will refer to this player as "EP", for "early position"). The next three players fold (throwing their cards into the discard pile), a middle position player (MP) calls \$10, and the next two players fold.

We are next to act and have 7♢-6♢. A strong poker player would know that this is a reasonably good drawing hand, which should be profitable to play for one bet from late position against several players. This would not be a good hand to call a raise with, or to play against only one or two opponents. From previous hands played, we know that EP is a *tight* (conservative) player. We expect that EP probably has two big cards, since he called in early position (but didn't raise, making large pairs highly unlikely for this particular player). Our opponent modeling has concluded that MP is a *loose* player, who sees the flop about 70% of the time, so he could have almost anything (*eg.* any pair, any two cards of the same suit, or even a hand like 6-4 of different suits). The small blind is an extremely tight player who will probably fold most hands rather than calling another \$5. The big blind almost always defends her blind (*i.e.* she will call a raise).

A raise in this situation, for deceptive purposes, is not completely out of the question. However, it would be inappropriate against this particular set of opponents (it might be more suitable in a game with higher limits). We call the \$10, the small blind calls, and the big blind checks.

The flop is Q♠-7♡-4♢. We have *second pair* (connecting with the second largest card on the board) for a hand of moderate strength and moderate potential for improvement. If we do not currently have the best hand, there are five direct *outs* (outcomes) that can immediately improve our hand (7♠, 7♣, 6♠, 6♣, 6♡). We also have some indirect flush and straight potential, which will come in about 7% of the time,[4] and can be treated as roughly three direct outs. The board texture is fairly *dry*, with only a few possible straight draws, and no direct flush draws. Therefore, any bets by the opponents are likely to indicate a *made hand* (*eg.* a pair) rather than a *draw* (a hand where additional cards are needed), unless they are a chronic bluffer. An expert player wouldn't actually need to go through this thought process—it would simply be known the moment the flop hits the table, through experience and pattern recognition.

Both blinds check, EP bets, and MP folds (see Figure 1). There is \$60 in the pot, and it will cost us \$10 to call. We believe the bettor seldom bluffs, and almost certainly has a Queen, given his early position pre-flop call.[5] The small blind is known to *check-raise* on occasion, and might also have a Queen, but is more likely to have a poor match with the board cards, because he is highly selective before the flop. We have never observed the big blind check-raising in the past, so the danger of being trapped for an extra bet is not too high.

If we play, we must decide whether to raise, trying to drive the other players out of the hand, or call, inviting others to call also. If there was a good chance of currently having the best hand, we would be much more inclined to raise.

[4] 73 out of 990 outcomes (43 flushes and 30 straights).

[5] Ironically, reasonably good players are often the most predictable, whereas *very good* players are not.

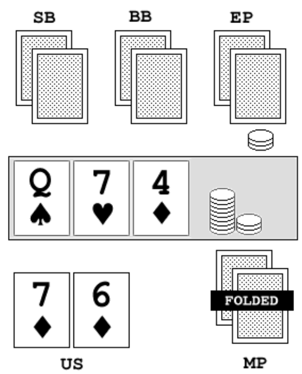

Figure 1: Sample hand after the flop.

However, we feel that chance is relatively small in the current situation. We might also want to drive out other players who are looking to hit the same cards we want, such as 5-3, which needs a 6 to make a straight against our two pair. However, the added equity from having an extra bet in the pot is normally greater than the risk of shared outs, so we are happy to let the blinds draw with us against the bettor.

From our previous study and experience, we know that calling in this situation is a small positive expectation play, but we still cannot rule out the possibility of *raising for a free-card.* If we raise now, we may induce the bettor to call and then check to us next round, when we can also check and get a second card for "free" (actually for half-price). We need to assess the likelihood that EP will re-raise immediately (costing us two extra bets, which is a very bad result), or will call but then bet into us again on the turn anyway (costing us one extra bet). Since we do not feel we have much control over this particular player, we reject the fancy raise maneuver, and just call the \$10. Both of the blinds fold, so we are now one-on-one with the bettor. Despite the many factors to consider, our decision is made quickly (normally within one second when it is our turn).

The turn card is the 5♡ and EP bets. The 5♡ gives us an open-ended draw to a straight, in addition to our other outs. In terms of expected value, this is essentially a "free pass" to the river, as we now have a clearly correct call of \$20 to win \$90. However, we again need to consider raising. This opponent will probably give us credit for having a very strong hand, since the 5♡ connects for several plausible two pair hands or straights. We could also be *slow-playing* a very strong hand, like a *set* (three of a kind using a *pocket pair*, such as 4♠-4♣). Since we're quite certain he has only one pair, this particular opponent might even fold the best hand, especially if his *kicker* (side-card) is weak. At the very least, he will probably check to us on the river, when we can also check, unless we improve our hand. Thus we would be investing the same amount of money as calling twice to reach the showdown, and we would be earning an extra big bet whenever we make our draw. On the other hand, we don't necessarily have to call that last bet on the river (although if we fold too often, we will become vulnerable to bluffing). We decide to make the expert play in this situation, confidently raising immediately after his bet. He thinks about his decision for a long time, and reluctantly calls.

The river card is the 5♠, and our opponent immediately checks. We know that he is not comfortable with his hand, so we can consider bluffing with what we believe is the second-best hand. From our past sessions we know that once this player goes to the river, he will usually see the hand through to the end. In effect, his decision after our raise was whether to fold, or to call two more bets. Since a bluff in this situation is unlikely to be profitable, we stick to our plan and check. He shows Q♣-J♣, we say "good hand", and throw our cards into the discard pile.

Now we consider what effect this hand has had on our *table image*, in anticipation of how the players at the table will react to our future actions. The better players might have a pretty good idea of what we had (a small pair that picked up a good draw on the turn), and won't make any major adjustments to their perception of our play. Our opponent, EP, is more likely to call us down if a similar situation arises, so we might earn an extra bet on a strong hand later. Weaker players may think we are a somewhat wild gambler, so we expect them to call even more liberally against us. This reinforces our plan of seldom bluffing against them, but betting for value with more marginal hands.

3.3 Requirements for a World-Class Poker Player

We have identified several necessary attributes for an algorithm to play poker at a world-class level. A system may handle some of these requirements indirectly, rather than by explicit design, but all of them must be solved at least satisfactorily if a program is to compete with the best human players. We present one or more ways of solving each requirement, but there are many different approaches that could be just as viable, or possibly much better. Furthermore, these components are not independent of each other. They must be continually refined and integrated as new capabilities are added to the system.

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

Hand potential computes the probability that a hand will improve to win, or that a leading hand will lose, after future community cards appear. For example, a hand that contains four cards in the same suit may have a low hand strength, but has good potential to win with a flush as additional community cards are dealt. Conversely, a hand with a high pair might be expected to decrease in strength if many draws are available for opposing hands. At a minimum, hand potential is a function of the cards in the hand and the current community cards. However, a better calculation would use all of the additional factors described in the hand strength computation.

Bluffing makes it possible to win with a weak hand,[6] and creates doubt on the part of the opponent, thereby increasing the amount won on subsequent strong 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 would bluff this percentage of hands, indiscriminately. In practice, other factors (such as hand potential) should also considered. A better system would identify profitable bluffing opportunities by deducing the opponent's approximate hand strength and predicting their probability of folding.

Unpredictability makes it difficult for opponents to form an accurate model of our strategy. Mixing strategies (occasionally handling a given situation in different ways) hides information about the nature of our current hand. By varying our playing style over time, opponents may be induced to make mistakes based on incorrect beliefs.

Opponent modeling determines a likely probability distribution of the opponent's hand. Minimal opponent modeling might use a single generic model for all opponents. This can be improved by modifying those probabilities based on the personal betting history and collected statistics of each opponent.

Certain fundamental principles of poker, such as *pot odds*, are taken as a given. There are several other identifiable characteristics that might not be necessary to play reasonably strong poker, but may eventually be required for world-class play. Collectively, these concepts are part of an overall *betting strategy*, which determines whether we fold, call, or raise in any particular situation. The most important of these attributes for poker-playing programs are discussed in greater detail in the following sections.

4 *Poki*'s Architecture

A poker game consists of a *dealer* together with multiple *players* that represent either human players or computer players. In our Java implementation, these players are defined as objects. The dealer handles the addition and removal of players from the game, deals the cards to each player at the start of a new hand, prompts each player for an appropriate action when it is their turn, broadcasts player actions to other players, and updates a public game context as the game progresses. The game context contains all of the public information about the game, including the names and relative locations of the players, and the board cards.

We have implemented several different dealer interfaces: an IRC-Dealer for playing against other players on the Internet Relay Chat poker server, a

[6] Other forms of deception such as *slow-playing* (calling with a strong hand) are not considered here.

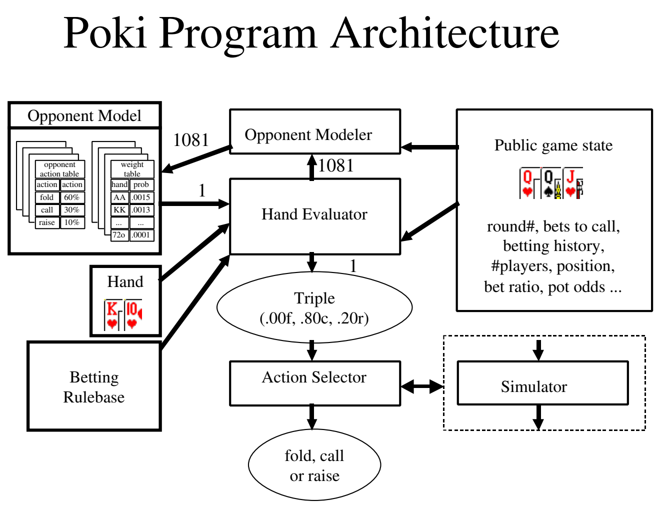

Figure 2: The architecture of *Poki*.

Tournament-Dealer for self-play experiments, and a TCP/IP-Dealer that allows *Poki* to play against humans using a web browser, and against other programs using a published protocol (see `http://www.cs.ualberta.ca/~games/poker/`).

An overview of *Poki*'s architecture is shown in Figure 2. Although each version of the program represents and uses the available information in a different way, all versions share a common high-level architecture.

In addition to the public game context, *Poki* stores private information: its current hand, and a collection of statistical opponent models. The assessment of the initial two-card hand is explained in Section 5.1, and the first-round betting decisions are made with a simple rule-based system. The opponent model (essentially a probability distribution over all possible hands) is maintained for each player participating in the game, including *Poki* itself, as detailed in Section 6. The Opponent Modeler uses the Hand Evaluator, a simplified rule-based Betting Strategy, and learned parameters about each player to update the current model after each opponent action, as described in Section 5.2.4. After the flop, the Hand Evaluator in turn uses the opponent model and the game state information to assess the value of *Poki's* hand in the current context, as explained in Section 5.2.1 and 5.2.2. Thus, there is a certain amount of cyclic feedback among the core components of the system. The evaluation is used by a more sophisticated rule-based Betting Strategy to determine a plan (how often to fold, call, or raise in the current situation), and a specific action is chosen, as discussed in Section 5.2.5 and throughout Section 5. The entire process is repeated each time it is our turn to act. For a more advanced decision procedure, the Simulator iterates this process using different instantiations of opponent hands, as discussed in Section 5.3.

5 Betting Strategy

Betting strategies before the flop and after the flop are significantly different. Before the flop there is little information available to influence the betting decision (just two hole cards and the previous player actions), and a relatively simple expert system is sufficient for competent play. After the flop the program can analyze how all possible opponent holdings combine with the given public cards, and many other factors are relevant to each decision. A post-flop betting strategy uses the full game context, the private hand, and the applicable opponent models to generate an action. Three betting strategies will be described in this paper, one for the pre-flop and two for the post-flop.

5.1 Pre-flop Betting Strategy

There are $\{52 \text{ choose } 2\} = 1326$ possible hands prior to the flop. The value of one of these hands is called an *income rate*, and is based on a simple technique that we will call a *roll-out simulation*. This is an off-line computation that consists of playing several million hands (trials) where all players call the first bet (*i.e.* the big blind), and then all the remaining cards are dealt out without any further betting. This highly unrealistic *always call assumption* does not necessarily reflect an accurate estimate for the expected value of the hand. However, it does provide a first-order approximation, and the *relative* values of the hands are reasonably accurate for the given situation.

More generally, this method is referred to as the *all-in equity*. It is a calculation of the percentage expectation for the current hand assuming the player is *all-in*,[7] and all active hands proceed to the showdown. It can be applied at any phase of the game, and serves as a baseline estimate of the expected value of a hand in any given situation.

5.1.1 Comparing Pre-Flop Strategies

The best known and most widely respected expert opinion on pre-flop play is that of David Sklansky, a professional poker player and author of the most important books on the game [32, 33]. In *Texas Hold'em for the Advanced Player* [33] he prescribes a hand classification scheme to be used in typical middle limit games (*eg.* \$20-\$40 limit Hold'em). There is a strong correlation between his rankings and the results of the roll-out simulations.

Before proceeding to a closer comparison of the two ranking systems, a few caveats should be mentioned. First, there is no single ranking of starting hands

[7] Under normal *table stakes* rules, a player who does not have enough money on the table to meet the outstanding bet can go *all-in*, and remains eligible to win the portion of the pot contributed to. The betting continues (toward a *side-pot*) for the remaining active hands.

that applies to all situations. An expert player will make adjustments based on the prevailing conditions (for example, a *loose game* (many players seeing the flop), a *wild game* (lots of gambling), etc.). Furthermore, the true expectation of each hand will depend on the precise context at the time of each betting decision. For example, a hand is assessed very differently after all previous players have folded than it would be after one or more players have called. The general guidelines must cover a wide variety of situations, so naturally there will be exceptions. Sklansky's recommendations are also intended for a full game of ten players. A completely different set of hand rankings are necessary for short-handed games, and this is reflected in the different income rates computed by roll-out simulations with fewer players in the hand.

Table 1 shows how the roll-out simulations compare to Sklansky's rankings. In the tables, "s" refers to a *suited* hand (two cards of the same suit), "o" refers to an *offsuit* hand (two cards of different suits), and "*" indicates a *pocket pair* (two cards of the same rank). Table 1 is divided into eight groups, corresponding to Sklansky's rating system, with Group 1 being the best hands, and Group 8 being weak hands that should only be played under special circumstances (*eg.* for one bet after many players have called). In general, there is a strong correlation between Sklansky's rankings and the income rates obtained from roll-out simulations.

The simulation values demonstrate a bias in favor of certain hands that play well against many players, known as "good multi-way hands". These are cards that can easily draw to a very strong hand, such as a flush (*eg. suited* hands like A♡-2♡), a straight (*eg. connectors* like 8♡-7♣), or three of a kind (*eg.* a *pocket pair* like 2♡-2♣). Since all ten players proceed to the showdown in a roll-out simulation, the average winning hand needs to be considerably stronger than in a real ten player game (where typically half of the players will fold before the flop, and many hands are won uncontested before the showdown). By the same reasoning, large pairs may be undervalued, because of the unaccounted potential of winning without improvement against a smaller number of opponents.

Conversely, Sklansky's rankings show evidence of a bias in favor of unsuited connectors, where suited hands should be preferred.[8] Certain small-card combinations, such as 7♠-6♠, may have been given a higher ranking by Sklansky because they add a good balance of deception to the overall play list (for example, one does not want the opposition to conclude that we cannot have a 7 when the flop is 7♡-7♢-3♣). However, the hands intended for information hiding purposes should not extend to the unsuited connectors like 7♣-6♡, which have a much lower overall expectation.

There are also a few instances of small logical errors in Sklansky's rankings. For example, 43s is ranked in Group 7, ahead of 53s in Group 8, but it can be shown that 53s logically dominates 43s, because it has the same straight and flush potential, with better high-card strength. Similarly, 52s dominates 42s and 32s, but 52s is not ranked in any of the eight groups, whereas the latter are

[8] The highest valued hands not in Sklansky's rankings are T7s (+231) and Q7s (+209).

Group 1		Group 2		Group 3		Group 4	
+2112	AA*	+714	TT*	+553	99*	+481	T9s [1]
+1615	KK*	+915	AQs	+657	JTs	+515	KQo
+1224	QQ*	+813	AJs	+720	QJs	+450	88*
+935	JJ*	+858	KQs	+767	KJs	+655	QTs
+1071	AKs	+718	AKo	+736	ATs	+338	98s [1]
				+555	AQo	+449	J9s
						+430	AJo
						+694	KTs

Group 5		Group 6		Group 7		Group 8	
+364	77*	+304	66*	+214	44*	-75	87o [2]
+270	87s [1]	+335	ATo	+92	J9o [2]	+87	53s [3] (> 43s)
+452	Q9s	+238	55*	+41	43s [3]	+119	A9o
+353	T8s [1]	+185	86s	+141	75s	+65	Q9o
+391	KJo	+306	KTo	+127	T9o	-129	76o [2]
+359	QJo	+287	QTo	+199	33*	-42	42s [3] (< 52s)
+305	JTo	+167	54s	-15	98o [2]	-83	32s [3] (< 52s)
+222	76s [1]	+485	K9s	+106	64s	+144	96s
+245	97s [1]	+327	J8s	+196	22*	+85	85s
+538	A9s			+356	K8s	-51	J8o [2]
+469	A8s			+309	K7s	+206	J7s
+427	A7s			+278	K6s	-158	65o [2]
+386	A6s			+245	K5s	-181	54o [2]
+448	A5s			+227	K4s	+41	74s
+422	A4s			+211	K3s	+85	K9o
+392	A3s			+192	K2s	-10	T8o
+356	A2s			+317	Q8s		
+191	65s [1]						

Three possible explanations for the differences: [1] small card balancing, [2] bias for unsuited connectors, and [3] logical error (inconsistent).

Table 1: Income rate values versus Sklansky groupings.

members of Group 8.

Since the differences are not large, it is clear that roll-out simulations provide an acceptable means of quantifying the pre-flop value of each hand. This information is currently used as part of a formula-based expert system for playing before the flop, which is not unlike the guidelines given by Sklansky in the aforementioned text. We prefer to use the computed results, rather than transcribing the Sklansky rules, because (a) we wish to eliminate the use of human knowledge whenever possible, (b) the roll-out simulation information is quantitative rather than qualitative, and (c) the algorithmic approach can be applied to many different specific situations (such as having exactly six players in the game), whereas Sklansky gives only a few recommendations for atypical circumstances.

Future versions of the program should be even more autonomous, adapting to the observed game conditions and making context-sensitive decisions on its own.

5.1.2 Iterated Roll-out Simulations

An interesting refinement to roll-out simulation is to use repeated iterations of the technique, where the previous results govern the betting decision for each player. In the ten player case, a negative value in the previous simulation would dictate that the hand be folded, rather than calling the big blind. This drastically reduces the number of active players in each hand, producing a more realistic distribution of opponents and probable hands. The result is a reduction in the bias toward multi-way hands, and a much better estimation of the hands that can be played profitably when ten players are originally dealt in.

After each round of simulations has reached a reasonable degree of stability, another iteration is performed. This process eventually reaches an equilibrium, defining a set of hands that can be played profitably against the blinds and the other unknown hands. The results are most applicable to the "play or don't play" decision for each player. Although much better than a simple roll-out simulation, this technique is still far from perfect, because other important considerations such as betting position and known opponent actions have not been accounted for.

In our experiments, each iteration lasted for 50,000 trials. A diminishing noise factor was added to each income rate, analogous to the cooling factor used in simulated annealing. This gives negative expectation hands a chance to recover as the prevailing context changes. After ten generations, the remaining positive expectation hands were played for another 500,000 trials, to ensure stability. The resulting set of profitable hands, shown in Table 2, is in strong agreement with expert opinion on this matter. The table shows a comparison of the income rates for 10-player roll-out simulations (IR-10) and the results refined by iterating (Iterated). The values shown are in *milli-bets* (*eg.* a hand with an income rate of +1000 should win an average of one small bet each time it is played). The iterated values are reasonable estimates of actual income rates, unlike the simple roll-out values, which are only used as relative measures.

Hand	IR-10	Iterated	Hand	IR-10	Iterated	Hand	IR-10	Iterated
AA*	+2112	+2920	ATs	+736	+640	KQo	+515	+310
KK*	+1615	+2180	99*	+553	+630	QTs	+655	+280
QQ*	+1224	+1700	KQs	+858	+620	QJs	+720	+270
JJ*	+935	+1270	AQo	+555	+560	A9s	+538	+220
TT*	+714	+920	KJs	+767	+480	ATo	+335	+200
AKs	+1071	+860	88*	+450	+450	KTs	+694	+190
AKo	+718	+850	77*	+364	+390	KJo	+391	+160
AQs	+915	+780	AJo	+430	+380	A8s	+469	+110
AJs	+813	+680	JTs	+657	+360	66*	+304	+40

Table 2: Iterated income rate (profitable hands).

One of the factors used by Sklansky and other experts is the possibility of a hand being *dominated*. For example, AQ is said to dominate AJ, because the AQ has a tremendous advantage if they are in a hand against each other (an Ace on board does not help the AJ). In contrast, AQ does not dominate the inferior holding of KJ, because they are striving to hit different cards. The role of domination is clearly demonstrated in the results of the iterated roll-out simulations. Examples include the increased value of large pairs and AK unsuited, and the diminished value of KQ (which is dominated by AA, KK, QQ, AK, and AQ).

Iterated roll-out simulations have also been used to compute accurate expected values for two-player pre-flop Hold'em. The resulting betting decisions are in very good agreement with Alex Selby's computation of the optimal game-theoretic strategy, in which he used an adaptation of the Simplex algorithm for solving this game directly [28].[9] The small number of cases where the strategies differ are all near the boundary conditions between raise and call, or call and fold. Furthermore, the expected values are always close to the threshold for making the alternate choice, with a difference usually less than 0.1 small bets.

5.2 Basic Betting Strategy

The basic betting strategy after the flop chooses an action using three steps:

1. Compute the *effective hand strength*, EHS, of *Poki*'s hand relative to the board.

2. Use the game context, a set of betting rules, and formulas to translate the EHS into a *probability triple*: $\{Pr(fold), Pr(call), Pr(raise)\}$.

3. Generate a random number in the range zero to one, and use it to choose

[9] We are assuming that an optimal solution to the re-defined game of pre-flop Hold'em will serve as a near-optimal solution to the pre-flop phase of real Hold'em (*i.e.* that a "perfect" solution to a simpler game will be a "good" solution to the full-scale version).

```
HandStrength(ourcards,boardcards)
{
   ahead = tied = behind = 0
   ourrank = Rank(ourcards,boardcards)
   /* Consider all two card combinations of the remaining cards.*/
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank)          ahead += 1
      else if(ourrank==opprank)     tied += 1
      else /* < */               behind += 1
   }
   handstrength = (ahead+tied/2) / (ahead+tied+behind)
   return(handstrength)
}
```

Figure 3: Hand strength calculation.

an action from the probability distribution. This contributes to the unpredictability of the program.

EHS is a measure of how well the program's hand stands in relationship to the remaining active opponents in the game. It is a combination of the current hand strength (HS) and positive potential (PPot) for the hand to improve. These are discussed in the following sections.

5.2.1 Hand Strength

The *hand strength*, HS, is the probability that a given hand is better than that of an active opponent. Suppose an opponent is equally likely to have any possible two hole card combination.[10] All of these opponent hands can be enumerated, identifying when *Poki*'s hand is better (+1), tied ($+\frac{1}{2}$), or worse (0). Taking the summation and dividing by the total number of possible opponent hands gives the (unweighted) hand strength. Figure 3 gives the algorithm for a simple hand strength calculation.

Suppose our hand is A♢-Q♣ and the flop is J♡-4♣-3♡. There are 47 remaining unknown cards and therefore {47 choose 2} = 1,081 possible hands an opponent might hold. In this example, any three of a kind, two pair, one pair, or AK is better (444 cases), the remaining AQ combinations are equal (9 cases), and the rest of the hands are worse (628 cases). Counting ties as one half, this corresponds to a percentile ranking, or hand strength, of 0.585. In other words, there is a 58.5% chance that A♢-Q♣ is better than a random hand.

The hand strength calculation is with respect to one opponent, but can be extrapolated to multiple opponents by raising it to the power of the number of

[10] This is not true, in general, but simplifies the presentation of the algorithm. We eliminate this assumption and generalize the algorithm in the next section.

A♢-Q♣ hole cards		J♡-4♣-3♡ board cards			
5 Cards	7 Cards				
	Ahead	Tied	Behind	Sum	
Ahead	449,005	3,211	169,504	628x990 =	621,720
Tied	0	8,370	540	9x990 =	8,910
Behind	91,981	1,036	346,543	444x990 =	439,560
Sum	540,986	12,617	516,587	1,081x990 =	1,070,190

Table 3: Hand potential example.

active opponents.[11] Against five opponents with random hands, the adjusted hand strength, HS_5, is $0.585^5 = 0.069$. Hence, the presence of the additional opponents has reduced the likelihood of A♢-Q♣ being the best hand to only 6.9%.

5.2.2 Hand Potential

After the flop, there are still two more board cards to be revealed. On the turn, there is one more card to be dealt. We want to determine the potential impact of these cards. The *positive potential*, PPot, is the chance that a hand which is not currently the best improves to win at the showdown. The *negative potential*, NPot, is the chance that a currently leading hand ends up losing.

PPot and NPot are calculated by enumerating over all possible hole cards for the opponent, like the hand strength calculation, and also over all possible board cards. For all combinations of opponent hands and future cards, we count the number of times *Poki's* hand is behind, but ends up ahead (PPot), and the number of times *Poki*'s hand is ahead but ends up behind (NPot). The algorithm is given in Figure 4, and the results for the preceding example are shown in Table 3. In this example, if the hand A♢-Q♣ is ahead against one opponent after five cards, then after 7 cards there is a 449,005/621,720 = 72% chance of still being ahead.

Computing the potential on the flop can be expensive, given the real-time constraints of the game (about one second per decision). There are $\{45 \text{ choose } 2\} = 990$ possible turn and river cards to consider for each possible two-card holding by the opponent. In practice, a fast approximation of the PPot calculation may be used, such as considering only the next one card to come. Previous implementations have used a fast function to produce a crude estimate of PPot, which was within 5% of the actual value about 95% of the time.

[11] This assumes that all of the opponent hands are independent of each other. Strictly speaking, this is not true. To be a useful estimate for the multi-player case, the error from this assumption must be less than the error introduced from other approximations made by the system. More accurate means are available, but we defer that discussion in the interest of clarity.

```
HandPotential(ourcards,boardcards)
{
   /* Hand potential array, each index represents ahead, tied, and behind.  */
   integer array HP[3][3]  /* initialize to 0 */
   integer array HPTotal[3] /* initialize to 0 */

   ourrank = Rank(ourcards,boardcards)
   /* Consider all two card combinations of the remaining cards for the opponent.*/
   for each case(oppcards)
   {
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank)      index = ahead
      else if(ourrank=opprank) index = tied
      else /* < */             index = behind
      HPTotal[index] += 1

      /* All possible board cards to come.  */
      for each case(turn)
      {
         for each case(river)
         {  /* Final 5-card board */
            board = [boardcards,turn,river]
            ourbest = Rank(ourcards,board)
            oppbest = Rank(oppcards,board)
            if(ourbest>oppbest)      HP[index][ahead ]+=1
            else if(ourbest==oppbest)HP[index][tied  ]+=1
            else /* < */             HP[index][behind]+=1
         }
      }
   }

   /* PPot:  were behind but moved ahead.  */
   PPot = (HP[behind][ahead] + HP[behind][tied]/2 + HP[tied][ahead]/2)
        / (HPTotal[behind]+HPTotal[tied]/2)
   /* NPot:  were ahead but fell behind.  */
   NPot = (HP[ahead][behind] + HP[tied][behind]/2 + HP[ahead][tied]/2)
        / (HPTotal[ahead]+HPTotal[tied]/2)
   return(PPot,NPot)
}
```

Figure 4: Hand potential calculation.

5.2.3 Effective Hand Strength

The *effective hand strength*, EHS, combines hand strength and potential to give a single measure of the relative strength of *Poki*'s hand against an active opponent. One simple formula for computing the probability of winning at the showdown [12] is:

$$
\begin{aligned}
Pr(win) &= Pr(ahead) \times Pr(opponent\ does\ not\ improve) \\
&\quad + Pr(behind) \times Pr(we\ improve) \\
&= HS \times (1 - NPot) \quad + \quad (1 - HS) \times PPot
\end{aligned}
$$

In practice, we generally want to bet when we currently have the best hand, regardless of negative potential, so that an opponent with a marginal hand must either fold, or pay to draw. Hence, NPot is not as important as PPot for betting purposes. Since we are interested in the probability that our hand is either currently the best, or will improve to become the best, one possible formula for EHS sets NPot $=$ 0, giving:

$$EHS = HS + (1 - HS) \times PPot \tag{1}$$

This has the effect of betting a hand aggressively despite good draws being possible for opponent hands, which is a desirable behavior.

For n active opponents, this can be generalized to:

$$EHS = HS^n + (1 - HS^n) \times PPot \tag{2}$$

assuming that the same EHS calculation suffices for all opponents. This is not a good assumption, since each opponent has a different style. A better generalization is to have a different HS and PPot for each opponent i. EHS with respect to each opponent can then be defined as:

$$EHS_i = HS_i + (1 - HS_i) \times PPot_i \tag{3}$$

Modifying these calculations based on individual opponents is the subject of Section 6.

5.2.4 Weighting the Enumerations

The calculations of hand strength and hand potential in Figures 3 and 4 assume that all two card combinations are equally likely. However, the probability of each hand being played to a particular point in the game will vary. For example, the probability that the opponent holds Ace-King is much higher than 7-2 after the flop, because most players will fold 7-2 before the flop.

To account for this, *Poki* maintains a *weight table* for each opponent. The table has an entry for every possible two card hand, where each value is the

[12] The formula can be made more precise by accounting for ties, but becomes less readable.

```
UpdateWeightTable(Action A, WeightTable WT, GameContext GC, OpponentModel OM)
{
   foreach (entry E in WT)
   {
      ProbabilityDistribution PT[FOLD,CALL,RAISE]

      PT = PredictOpponentAction(OM, E, GC)
      WT[E] = WT[E] * PT[A]
   }
}
```

Figure 5: Updating the Weight Table.

conditional probability of the opponent having played those cards to the current point in the game. To get a better estimate of hand strength, each hand in the enumeration is multiplied by its corresponding probability in the weight table.

In practice, the weights have a value in the range zero to one, rather than absolute probabilities (summing to one), because only the relative sizes of the weights affect the later calculations. When a new hand begins, all entries are initialized to a weight of one. As cards become known (*Poki*'s private cards or the public board cards), many hands become impossible, and the weight is set to zero.

After each betting action, the weight table for that opponent is updated in a process called *re-weighting*. For example, suppose an opponent calls before the flop. The updated weight for the hand 7-2 might be 0.01, since it should normally be folded. The probability of Ace-King might be 0.40, since it would seldom be folded before the flop, but is often raised. The relative value for each hand is increased or decreased to be consistent with every opponent action.

The strength of each possible hand is assessed, and a *mixed strategy* (probable distribution of actions) is determined by a formula-based betting strategy. These values are then used to update the weight table after each opponent action. The algorithm is shown in Figure 5.

For example, assume that the observed player action is a bet or raise, and that the weight table currently has entries:

[A♠-K♣, 0.40], ..., [Q♢-2♢, 0.20], ...

Further assume that in the given situation, the PredictOpponentAction procedure (see Figure 5) generates probability distributions $\{Pr(fold), Pr(check/call), Pr(bet/raise)\}$ of $\{0.0,\ 0.7,\ 0.3\}$ for the hand A♠-K♣, and $\{0.0,\ 0.1,\ 0.9\}$ for the hand Q♢-2♢. After re-weighting, the new weight table entry for A♠-K♣ will be $0.4 \times 0.3 = 0.12$, and $0.2 \times 0.9 = 0.18$ for Q♢-2♢. Had the opponent checked or called in this situation, the weights would be 0.28 and 0.02, respectively.[13]

[13] In the parlance of Bayesian (conditional) probabilities, the old weight table represents the *prior distribution* of the opponent's cards, and the new weight table is the *posterior distribution*.

	SB	BB	EP	MP	Poki
Pre-flop					
	small blind	big blind	call	call	call
	call	check			
Flop Q♠ 7♡ 4♢					
	check	check	bet	fold	call
	fold	fold			
Turn 5♡					
			bet		raise
			call		
River 5♠					
			check		check

Table 4: Betting scenario for hand described in Section 3.2.

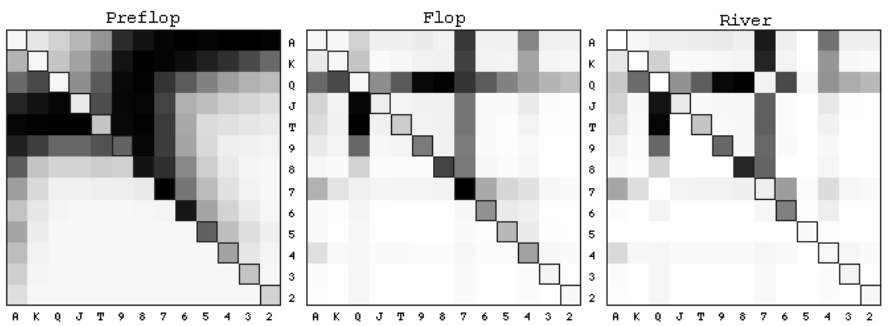

Figure 6: Progressive weight tables for one opponent in the example hand.

Table 4 shows a possible game scenario based on the example given in Section 3.2 (with the five players that immediately folded in the pre-flop removed). In this hand, player EP is assumed to be a default player rather than the well-modeled tight opponent described previously. Figure 6 shows *Poki*'s weight table for EP at three stages of the hand (pre-flop, flop, and river). In each figure, darker cells correspond to higher relative weights. Suited hands are shown in the upper right portion of the grid, and unsuited hands are on the lower left. The program gathers more information as the hand is played, refining the distribution of hands that are consistent with the betting actions of EP.

5.2.5 Probability Triples and Evaluation Functions

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. This representation of future actions (analogous to a *mixed strategy* in game theory) is used in three places in *Poki*:

1. The basic betting strategy uses a probability triple to decide on a course of action (fold, call, or raise).
2. The opponent modeling component (Section 6) uses an array of probability triples to update the opponent weight tables.
3. In a simulation-based betting strategy (Section 5.3) probability triples are used to choose actions for simulated opponent hands.

Hand strength (HS), potential (PPot, NPot), and effective hand strength (EHS), are simple algorithms for capturing some of the probabilistic information needed to make a good decision. However, there are many other factors that influence the betting decision. These include things like *pot odds*, *implied odds*, relative betting position, betting history of the current hand, etc. Hence the probability triple generation routine consists of *ad hoc* rules and formulas that use EHS, the opponent model, game conditions, and probability estimates to assess the likelihood of each possible betting action. A professional poker player (Billings) defined this system based on crude estimates of the return on investment for each betting decision. We refer to this as either a rule-based or formula-based betting strategy. The precise details of this procedure will not be discussed, as they are of limited scientific interest.

An important advantage of the probability triple abstraction is that most of the expert-defined knowledge in *Poki* has been gathered together into the triple-generation routines. This is similar to the way that external knowledge is restricted to the evaluation function in alpha-beta search. 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 Hold'em-specific knowledge, complex expert-defined rule systems, and knowledge of human behavior are all separated from the engine that uses this input for its calculations. The essential algorithms should be applicable to other poker variants with little or no modification, and perhaps to substantially different domains.

5.3 Selective Sampling and Simulation-based Betting Strategy

Having an expert identify all the betting rules necessary to play poker is time consuming and difficult. The game is strategically complex, and decisions must be based on the exact context of the current game, and historical information of past sessions. A system based on expert rules is unlikely to produce a world-class level of play, because covering every relevant situation in sufficient detail is not feasible. We believe that dynamic, adaptive, computer-oriented techniques will be essential to compete with the best human players.

As mentioned above, a knowledge-based betting strategy is analogous to a static evaluation function in deterministic perfect information games. Given the current state of the game, it attempts to determine the action that yields the best result. The corresponding analogue would be to add search to the evaluation function. While this is easy to achieve in a game such as chess (consider all possible moves as deeply as resources permit), the same approach is not directly applicable to poker. There are fundamental differences in the structure of imperfect information game trees, and the total number of possibilities to consider is prohibitive.

Toward this end, *Poki* supports a simulation-based betting strategy. It consists of playing out many likely scenarios, keeping track of how much money each decision will win or lose. Every time it faces a decision, *Poki* 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 single trial consists of playing out the hand from the current state of the game through to the end. Many trials produce a *full-information simulation* (which is not to be confused with the simpler roll-out simulations mentioned in Section 5.1).

Each trial is played out twice—once to consider the consequences of a check or call, and once to consider a bet or raise. In each trial, a hand is assigned to each opponent, based on the probabilities maintained in their weight table. The resulting instance is simulated to the end, and the amount of money won or lost is determined. Probability triples are used to determine the future actions of *Poki* and the opponents, based on the two cards they are assigned for that trial and threshold values determined by the specific opponent model. The average over all trials in which we check or call is the *call EV*, and the average for the matching trials where we bet or raise is the *raise EV*. The *fold EV* can be calculated without simulation, since there is no future profit or loss.

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 (prefer a raise, then a call, then a fold). To increase the program's unpredictability, we can randomize the selection between betting actions whose EVs are close in value, but the level of noise in the simulation already provides some natural variation for close decisions.[14]

Enumerating all possible opponent hands and future community cards would be analogous to exhaustive game tree search, and is impractical for poker. Simulation is analogous to a selective expansion of some branches of a game tree. To get a good approximation of the expected value of each betting action, one must have a preference for expanding and evaluating the nodes that are most likely to occur. To obtain a correctly weighted average, all of the possibilities must be considered in proportion to the underlying non-uniform probability distribution of the opponent hands and future community cards. We use the term *selective sampling* to indicate that the assignment of probable hands to each opponent is consistent with this distribution.

[14]Unfortunately, this simple approach does convey some useful information to observant opponents, in that the strength of our hand and the betting level are too closely correlated. Moving toward a near-optimal mixed strategy would provide better information-hiding, and may be necessary to reach the world-class level.

At each betting decision, a player must choose a single action. The choice is strongly correlated to the quality of the cards that they have, and we can use the opponent model and formula-based betting strategy to compute the likelihood that the player will fold, call, or raise in each instance. The player's action is then randomly selected based on this probability distribution, and the simulation proceeds. As shown in Figure 2, the Simulator calls the opponent model to obtain each of our opponent's betting actions and our own actions. Where two or three alternatives are equally viable, the resulting EVs should be nearly equal, so there is little consequence if the "wrong" action is chosen.

It is reasonable to expect that the simulation approach will be better than the static approach, because it essentially uses a selective search to augment and refine a static evaluation function. Barring serious misconceptions, or bad luck on a limited sample size, playing out many relevant scenarios will improve the estimates obtained by heuristics alone, resulting in a more accurate assessment overall.

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

- hand strength (fraction of trials where our hand is better than the one assigned to the opponent),
- hand potential (fraction of trials where our hand improves to the best, or is overtaken), and
- subtle considerations not addressed in the simplistic betting strategy (*eg. 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 produce advanced betting tactics like *check-raising* as an emergent property, even if the basic strategy used within each trial is incapable of this play.

At the heart of the simulation is the evaluation function, discussed in Section 5.2.5. The better the quality of the evaluation function, the better the simulation results will be. Furthermore, the evaluation system must be compatible and harmonious with the nature of the simulations. Since the formula-based betting strategy was developed and tuned for the original system, it may not be entirely consistent or appropriate for use in the simulation-based version. It is possible that built-in biases which were useful (or compensated for) in the original version are sources of serious systemic error when used as the evaluation function for simulations. It may be the case that a simpler function would be more balanced, producing better results.

One of the interesting results of work on alpha-beta search is 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 itself improves the basic evaluation, refining the quality of the approximation. As with alpha-beta, there are important tradeoffs to consider. 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 reducing the total number of trials performed in real-time.

Variations of selective sampling have been used in other games, including Scrabble [30], backgammon [36], and bridge [15]. *Likelihood weighting* is another method of biasing stochastic simulations [13, 29]. In our case, the goal is different because we need to differentiate between EVs (for call/check, bet/raise) instead of counting events. Poker also imposes tight real-time constraints (typically a maximum of a few seconds per decision). This forces us to maximize the information gained from a limited number of samples. The problem of handling unlikely events (which is a concern for any sampling-based result) is smoothly handled by the re-weighting system (Section 5.2.4), allowing *Poki* to dynamically adjust the likelihood of an event based on observed actions. An unlikely event with a large payoff figures naturally into the EV calculations.

6 Opponent Modeling

No poker strategy is complete without a good opponent modeling system. A strong poker player must develop a dynamically changing (adaptive) model of each opponent, to identify potential weaknesses.

In traditional games, such as chess, this aspect of strategy is not required to achieve a world-class level of play. In perfect information games, it has been sufficient to play an objectively best move, without special regard for the opponent. If the opponent plays sub-optimally, then continuing to play good objective moves will naturally exploit those errors. Opponent modeling has been studied in the context of two-player games, but the research has not translated into significant performance benefits [8, 17, 18].

In poker, the situation is different. Two opponents can make opposite kinds of errors—both can be exploited, but it requires a different response for each. For example, one opponent may bluff too much, the other too little. We adjust by calling more frequently against the former, and less frequently against the latter. To simply call with the optimal frequency would decline an opportunity for increased profit, which is how the game is scored. Even very strong players can employ radically different styles, so it is essential to try to deduce each opponent's basic approach to the game, regardless of how well they play.

6.1 RoShamBo

The necessity of modeling the opponent is nicely illustrated in the game of RoShamBo (also known as Rock-Paper-Scissors). This is a well-known "kid's game", where each player chooses an action simultaneously, and there is a cyclic set of outcomes: scissors beats paper, paper beats rock, and rock beats scissors (choosing the same action results in a tie). The game-theoretic optimal strategy for this zero sum game is also well-known: one chooses any of the three actions uniformly at random. However, the optimal strategy is oblivious to opponent actions, and is not exploitive. The best one can do using the optimal strategy is to break even in the long run (an expected value of zero, even if the opponent *always* goes rock). Contrary to popular belief, the game is actually very complex when trying to out-guess an intelligent opponent.

The International RoShamBo Programming Competition[15] is an annual contest for programs that play Rock-Paper-Scissors [3]. More than 50 entries were submitted from all over the world for each competition. Every program plays every other program in a round-robin tournament, with each match consisting of 1,000 games. Scores are based on total games won, and on the match results (with the match declared a draw if the scores are not different by a statistically significant margin). Since the optimal strategy can only draw each match, it consistently finishes in the middle of the pack, and has no chance of winning the tournament.

The authors of the top entries, including some well-known AI researchers, have commented that writing a strong RoShamBo program was much more challenging than they initially expected [4, 11]. The best programs do sophisticated analysis of the full history of the current match in order to predict the opponent's next action, while avoiding being predictable themselves. Programs that used a simple rule-base for making their decisions consistently finished near the bottom of the standings. All of the top programs define completely general methods for pattern detection, some of which are remarkably elegant. Given the simple nature of RoShamBo, some of these nice ideas may be applicable to the much more complex problems faced by a poker playing system.

6.2 Statistics-based Opponent Modeling

In poker, opponent modeling is used in at least two different ways. We want a general method of deducing the strength of the opponent's hand, based on their actions. We also want to predict their specific action in a given situation.

At the heart of an opponent modeling system is a *predictor*. The predictor's job is to map any given game context into a probability distribution over the opponent's potential actions. In limit poker, this distribution can be represented by a *probability triple* $\{Pr(fold),\ Pr(call),\ Pr(raise)\}$.

One way to predict an opponent action would be to use our own betting strategy, or some other set of rules, to make a rational choice on behalf of the opponent. When we use this type of fixed strategy as a predictor, we are assuming the player will play in one particular "reasonable" manner, and we refer to it as *generic opponent modeling* (GOM).

Another obvious method for predicting opponent actions is to expect them to continue to behave as they have done in the past. For example, if an opponent is observed to bet 40% of the time immediately after the flop, we can infer that they will normally bet with the top 40% of their hands in that situation (including a certain percentage of weak hands that have a good draw). When

[15] See `http://www.cs.ualberta.ca/~games`.

we use an opponent's personal history of actions to make predictions, we call it *specific opponent modeling* (SOM).

Our first opponent modeling effort was based on the collection of simple statistical information, primarily on the betting frequencies in a variety of contexts. For example, a basic system distinguishes twelve contexts, based on the betting round (pre-flop, flop, turn, or river), and the betting level (zero, one, or two or more bets). For any particular situation, we use the historical frequencies to determine the opponent's normal requirements (*i.e.* the average effective hand strength) for the observed action. This threshold is used as input into a formula-based betting strategy that generates a *mixed strategy* of rational actions for the given game context (see Section 5.2.5).

However, this is a limited definition of distinct contexts, since it does not account for many relevant properties, such as the number of active opponents, the relative betting position, or the texture of the board cards (*eg.* whether many draws are possible). Establishing a suitable set of conditions for defining the various situations is not an easy task. There are important trade-offs that determine how quickly the algorithm can learn and apply its empirically discovered knowledge. If a context is defined too broadly, it will fail to capture relevant information from very different circumstances. If it is too narrow, it will take too long to experience enough examples of each scenario, and spotting general trends becomes increasingly difficult. Equally important to deciding how many equivalence classes to use is knowing what kinds of contextual information are most relevant in practice.

Furthermore, there are many considerations that are specific to each player. For example, some players will have a strong affinity for flush draws, and will raise or re-raise on the flop with only a draw. Knowing these kinds of personality-specific characteristics can certainly improve the program's performance against typical human players, but this type of modeling has not yet been fully explored.

Opponent modeling in poker appears to have many of the characteristics of the most difficult problems in machine learning—noise, uncertainty, an unbounded number of dimensions to explore, and a need to quickly learn and generalize from relatively small number of heterogeneous training examples.[16] As well, the real-time nature of poker (a few seconds per betting decision) limits the effectiveness of most popular learning algorithms.

6.3 Neural Networks-based Opponent Modeling

To create a more general system for opponent modeling, we implemented a neural network for predicting the opponent's next action in any given context. Guessing the next action is useful for planning advanced betting strategies, such as a *check-raise*, and is also used in each trial of a full-information simulation (see Section 5.3).

[16] By "heterogeneous" we mean that not all games and actions reveal the same type or amount of information. For example, if a player folds a hand, we do not get to see their cards.

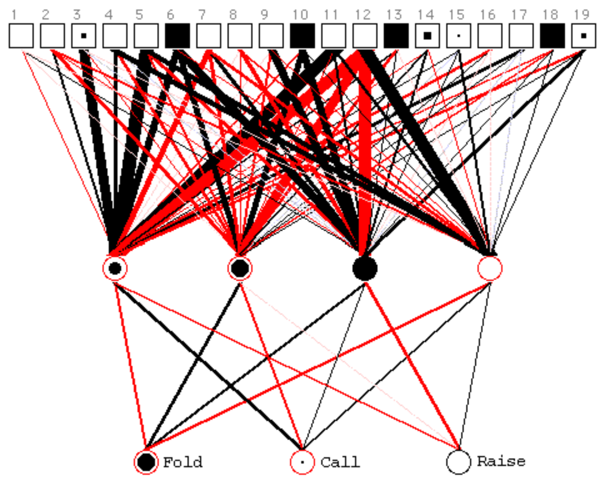

Figure 7: A neural network predicting an opponent's future action.

A standard feed-forward neural net was trained on contextual data collected from online games against real human opponents. The networks contain a set of nineteen inputs corresponding to properties of the game context, such as the number of active players, *texture* of the board, opponent's position, and so on. These are easily identified factors that may either influence, or are correlated with a player's next action.

The output layer consists of three nodes corresponding to the fold, call, and raise probabilities. Given a set of inputs, the network will produce a probability distribution of the opponent's next action in that context (by normalizing the values of the three output nodes).

By graphically displaying the relative connection strengths, we are able to determine which input parameters have the largest effects on the output. After observing networks trained on many different opponents, it is clear that certain factors are dominant in predicting the actions of most opponents, while other variables are almost completely irrelevant. The accuracy of these networks (and other prediction methods) is measured by cross-validating with the real data collected from past games with each opponent. Details are available in a previous paper [10].

Figure 7 shows a typical neural network after being trained on a few hundred hands played by a particular opponent. The inputs are the on the top row, with the activation level ranging from zero (fully white) to one (fully black). The thickness of the lines represent the magnitude of the weights (black being positive, grey being negative). In this example, the connections from input node number twelve (true if the opponent's last action was a raise) are very strong, indicating that it is highly correlated with what the opponent will do next. The bottom row shows the network predicting that the opponent will probably fold, with a small chance of calling.

The first result of this study was the identification of new features to focus on when modeling common opponents. This produced a relatively small set of context equivalence classes which significantly improved the statistical opponent modeling reported previously [10]. We are currently experimenting with using a real-time neural network system to replace the frequency table method entirely. Preliminary results from games with both human and computer opponents suggest that this may lead to a dramatic improvement.

7 Performance Evaluation

Measuring the performance of a poker-playing program is difficult. *Poki* is a complex system of interacting components, and changing a single component often has cascading effects, leading to unpredictable and unforeseen behavior. We have employed a variety of methods for assessing the program, but none of them is completely adequate.

7.1 Experimental Methodology

Poker is a game of high variance, and the element of luck dominates the outcome of any one hand. Among evenly matched players, the effects of good or bad fortune are still significant even after several thousand hands. Measurements are always susceptible to high levels of noise and anomalous games. Furthermore, players are constantly adapting during this time, improving their understanding of each opponent, or changing styles to make it more difficult for others to form an accurate model of them.

Self-play experiments are a simple way to test new features, by playing older versions of the program against newer versions. This provides an easily controlled closed environment, where many thousands of hands can be played quickly.

To reduce variance we use a duplicate tournament system similar to that used in duplicate bridge. Since each hand can be played with no memory of preceding hands, in a ten-player game each deal of the cards can be replayed ten times, shuffling the seating arrangement each time so that every player holds each hand once. This reduces the amount of noise considerably, and also reduces the effects of relative seating position (for example, it would be advantageous to always act immediately after a particularly aggressive or unpredictable player). However, this method still admits a lot of variance. For example, one player might choose to fold a marginal hand whereas another might play in that same situation, possibly winning or losing many bets.

Another assessment method attempts to compute an objective measurement of the expected value for each decision, using the perfect information of the actual situation. For example, a weak looking hand might actually win 20% of the time against the current field, and the EV for making a "loose call" in that situation might be +0.6 bets, compared to -0.6 bets for a more conservative fold, or -0.2 bets for a raise. When comparing two or more players, this kind of specific evaluation of an action can be applied to the first differing decision of each deal, since the subsequent betting actions are not comparable (being different contexts). While this method is attractive in principle, it is somewhat difficult to define a reliable EV measure for all situations, and consequently it has not been used extensively to date.

The major drawback of self-play experiments is that they lack the wide variety of styles and game conditions exhibited by real players. Other researchers have previously commented on the "myopia" of self-play games in chess [2]. The problem is much more acute and limiting for the development of a poker-playing system, because the style of the opponent is of paramount importance to correct play. A program that does very well against normal opponents may be vulnerable to a particular type of erratic or irrational player, even if their play is objectively worse. Although we try to create a variety of computer opponents by varying parameter settings of the players (*eg.* percentage of hands played, aggressiveness, advanced betting strategies, etc.), the range of styles is still much more restricted than that of human opponents.

Even with a carefully selected, well-balanced field of artificial opponents, it is important to not over-interpret the results of any one experiment. Often all that can be concluded is the relative ranking of the algorithms amongst themselves. One particular strategy may dominate in a self-play experiment, even though another approach is more robust in real games against human opponents.

A good demonstration of this limitation was seen in the testing of early simulation-based betting strategies. The results of self-play experiments were very encouraging, and occasionally spectacular. However, this was largely due to the pure aggressiveness of the new strategy, which was particularly effective at exploiting the overly conservative nature of its computer opponents at that time. When testing the new betting strategy in online games, it was much less successful against reasonably strong human opposition, who were able to adapt quickly.

For this reason, playing games against real human opponents is still indispensable for proper evaluation. Unfortunately, this entails other sources of inaccuracy.

A poker program can participate in a real game with willing participants, using a laptop computer on the table. This turns out to be surprisingly difficult, due to the fast pace of a real game and the amount of information to be entered. Even with numerous single-character accelerators, text entry is a bottleneck to the process. A well-designed graphical interface might help considerably, and an automatic card-reader (*eg.* a bar-code scanner) could prevent the operator from giving away useful information, since only the program would know its hand. However, it may always be more practical to have human players participate in a virtual game, rather than having programs compete in the physical world.

For more than three years, our programs have regularly participated in on-line poker games against human opposition on the Internet Relay Chat (IRC). Players connect to the IRC poker server and participate in numerous games that are conducted by dedicated software. No real money is at stake, but the accumulated bank-roll for each player is preserved between sessions, and a variety of statistics are maintained. There is a hierarchy of games for limit Hold'em, and a player must win a specified amount in the introductory level games to qualify for the higher tiered games.

These lowest level games (open to everyone) vary from wild to fairly normal, offering a wide variety of game conditions to test the program. The second and third tier games resemble typical games in a casino or card room. Most of these players take the game seriously, and some are very strong (including some professionals). Since *Poki* has been a consistent winner in these higher tiered games (and is in the top 10% of all players on the server), we believe the program plays better than the average player in a low-limit casino game.

Recently, several online poker servers have begun offering real-money games played over the Internet. The response has been very favorable, and it is normal to have more than 1,000 players logged into a virtual card room at any given time. With the agreement of the entrepreneurs, this might provide a future venue for testing programs in a completely realistic setting.

Another form of online poker is a free Java web applet, where users can play at a table with poker programs and other people. *Poki* currently hosts such a facility, which provides an interesting hybrid between self-play experiments and games against humans.[17]

While online poker is useful for measuring the progress of a program, it is not a controlled environment. The game is constantly changing, and positive results over a given time-frame can easily be due to playing against a weaker set of opponents, rather than actual improvements to the algorithm. Considering that it may take thousands of hands to measure small improvements, it is difficult to obtain precise quantified results. There is also no guarantee that an objectively stronger program will be more successful in this particular style of game. Certain plays that might be good against master players could be inappropriate for the more common opponents in these games. Moreover, regular players may have acquired a lot of experience against previous versions of *Poki*, making it difficult to achieve the same level of performance.

As a result, it is still beneficial to have a master poker player review hundreds of hands played by the program, looking for errors or dubious decisions. Needless to say, this is a slow and laborious method of assessment. A human master can also play against one or more versions of the program, probing for weaknesses or unbalanced strategy. Based on these direct encounters, we believe *Poki* is an intermediate level player, but has not yet reached the master level.

7.2 Experimental Results

The unit of measurement for program performance is the average number of small bets won per hand (sb/hand). For example, in a game of \$10/\$20 Hold'em with 40 hands per hour, an income rate of +0.05 sb/hand translates into \$20 per hour. Human players sometimes use this metric in preference to dollars per

[17] See `http://www.cs.ualberta.ca/~games`.

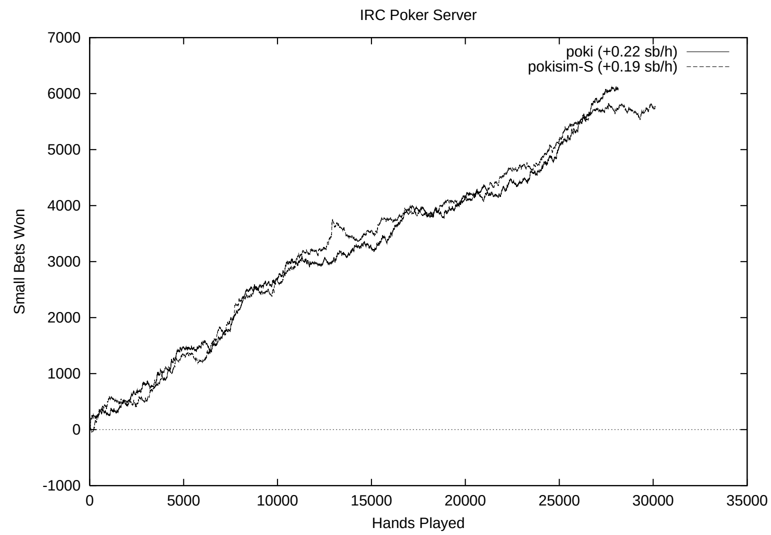

Figure 8: *Poki*'s performance on the IRC poker server (introductory level games).

hour, since it is not dependent on the speed of play, which can vary from 20 to 60 hands per hour.

Since no variance reduction methods are available for online games, we generally test new algorithms for a minimum of 20,000 hands before interpreting the results. On this scale, the trends are usually clear and stable amid the noise. Unfortunately, it can take several weeks to accumulate this data, depending on the popularity of the online game in question.

Any embellishment resulting in an improvement of +0.05 sb/hand in self-play experiments against previous versions is considered to be significant. However, this does not always translate into comparable gains in actual games, as many factors affect the ultimate win rate. Nevertheless, the program has made steady progress over the course of the project. In recent play on the IRC poker server, *Poki* has consistently performed between +0.10 and +0.20 sb/hand in the lowest level games, and between +0.07 and +0.10 sb/hand in the higher tiered games against stronger opposition.

The results of simulation-based betting strategies have so far been inconsistent. Despite some programming errors that were discovered later, the earliest (1998) versions of simulation-based *Loki* outperformed the regular formula-based version in both self-play experiments ($+0.10 \pm 0.04$ sb/hand), and in the introductory level games of IRC (+0.13 sb/hand *vs* +0.08 sb/hand). However, it lost slowly in the more advanced IRC games, whereas the regular version would at least break even.

The more recent versions are substantially stronger, but a similar pattern is

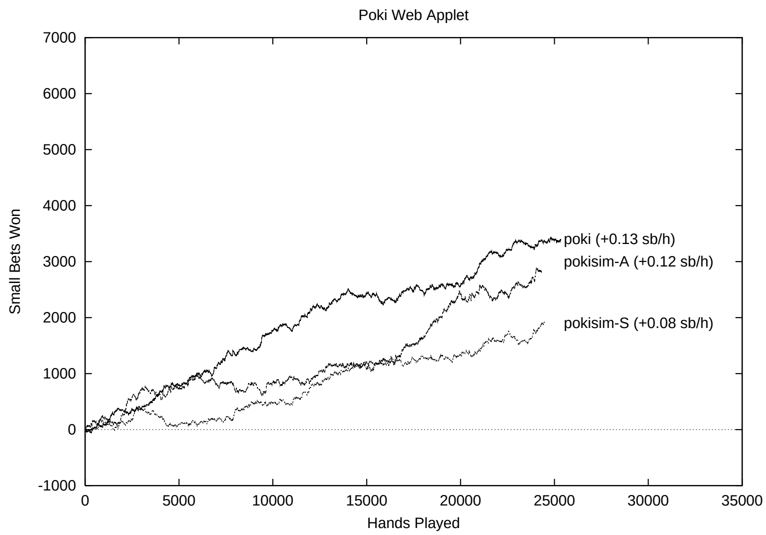

Figure 9: *Poki*'s performance on the web applet.

apparent. Figure 8 shows that both the regular betting strategy (labeled "poki") and the simulation-based betting strategy (labeled "pokisim-S") win at about +0.20 sb/hand in the introductory level games on the IRC poker server. It is quite likely that differences in playing strength cannot be demonstrated against this particular level of opposition, since both may be close to their maximum income rate for this game. In other words, there are diminishing returns after achieving a very high win rate, and further improvement becomes increasingly difficult. However, there is a clear difference in the more advanced games, where the regular betting strategy routinely wins at about +0.09 sb/hand, but the simulation-based version could only break even (peaking at +0.01 sb/hand after 5,000 hands, but returning to zero after 10,000 hands).

When the simulation-based versions were introduced, some of the credit for their success was probably due to the solid reputation that the more conservative versions of *Poki* had previously established. Many opponents required several hundred hands to adjust to the more aggressive style resulting from the simulations. However, the stronger opposition was able to adapt much quicker, and learned to exploit certain weaknesses that had not been detrimental against weaker players.

Figure 9 shows some recent results using the online web applet. This game consists of several computer players (some of which are intentionally weaker than the most recent versions of *Poki*), and at least one human opponent at all times. Since the artificial players are quite conservative, this game is quite a bit *tighter* than most IRC games, and the win rate for the regular formula-based betting strategy is +0.13 sb/hand. The simulation-based betting strategy performs at

+0.08 sb/hand, indicating that this particular set of opponents are much less vulnerable to its strategy differences than the players in the introductory IRC games.

A new simulation-based player (labeled "pokisim-A") maintains three different methods for opponent modeling (statistical frequencies, the rule-based method used by *Poki*, and a real-time neural network predictor), and uses whichever one has been most successful for each opponent in the past. Not surprisingly, it outperforms the single approach, earning +0.12 sb/hand, for a 50% overall improvement in this particular game. This is roughly the same degree of success as the formula-based strategy ("poki"), despite the fact that the original system has benefited from much more tuning, and that the underlying evaluation function was not designed for this fundamentally different approach.

We note that the variance is quite a bit higher in this experiment, which is the more common situation.[18] The results could be quite misleading if interpreted after only 5,000, or even after 15,000 hands. The two bottom lines cross over at 15,000 hands, but "pokisim-S" is lower before and after that point.

There have been hundreds of self-play experiments over the last few years, testing individual enhancements, and the effects of different game conditions. We refer the reader to our previous publications for further details [5, 6, 7, 10, 24, 25, 27].

8 A Framework for Non-Deterministic Game-Playing Programs

Using simulations for non-deterministic games is not new. Consider the following three games:

1. In Scrabble, the opponent's tiles are unknown, so the outcome of future turns must be determined probabilistically. A simulation consists of repeatedly generating a plausible set of tiles for the opponent. Each trial might involve a two ply or four ply search of the game tree, to determine which move leads to the maximum gain in points for the program. 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 [30].

2. In backgammon, simulation is used for "rollouts" of the remainder of a game, and are now generally regarded to be the best available estimates for the equity of a given position. A simulation consists of generating a series of dice rolls, and playing through to the end of the game with a strong program choosing moves for both sides. Gerry Tesauro has shown

[18] The relatively low variance in the previous figure may again be a result of both programs being close to maximal gains against that particular level of opposition.

that relatively simple rollouts can achieve a level of play comparable to the original neural network evaluation function of *TD-Gammon* [36, 37].

3. In bridge, the cards of other players are hidden information. A simulation consists of assigning 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 out to decide which play produces the highest probability of success. Matt Ginsberg has used this technique in *GIB* to achieve a world-class level for play of the hand [15].

In the above examples, the programs are not using traditional Monte Carlo simulation to generate the unknown information. They use selective sampling, biased to take advantage of all the available information. In each case, and in poker, we are using information about the game state to skew the underlying probability distribution, rather than assuming a uniform or other fixed probability distribution. Monte Carlo techniques might eventually converge on the right answer, but selective sampling allows reduced variance and faster convergence.

In the Scrabble example, *Maven* does not assign tiles for the opponent by choosing from the remaining unknown tiles uniformly at random. It biases its choice to give the opponent a "nice" hand, because strong players usually make plays that leave them with good tiles for future turns (such as letters that may score the 50 point bonus for using all tiles). It also samples without replacement, to ensure that every remaining tile is selected equally often, thereby reducing the natural variance [30]. In backgammon, future dice rolls are generated randomly, but the choice of moves is made by an external player agent. In bridge, the assignment of cards to an opponent is subject to the information obtained from the bidding. If one opponent has indicated high point strength, then the assignment of cards to that opponent reflects this information [15].

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 more than 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). The search technique usually has the following properties:

1. The search is full breadth, but limited depth. That is, all move alternatives are considered, except those that can be logically eliminated (such as alpha-beta cutoffs).

2. Heuristic evaluation occurs at the leaf nodes of the search tree, which are interior nodes of the game tree.

3. The search gets progressively deeper (iterative deepening), until real-time constraints dictate that a choice be made.

The alpha-beta algorithm typically uses integer values for positions and is designed to identify a single "best" move, not differentiating between other

```
SimulationFramework()
{
   obvious_move = NO
   trials = 0
   while( ( trials <= MAX_TRIALS ) and ( obvious_move == NO ) )
   {
      trials = trials + 1
      position = current_state_of_the_game +
          ( selective_sampling to generate_missing_information )
      for( each legal move m )
      {
         value[m] += PlayOut( position.m, info )
      }
      if( exists i such that value[i] >> value[j]( forall j ≠ i ) )
      {
         obvious_move = YES
      }
   }
   select move based on value[]
}
```

Figure 10: Framework for two-player, zero-sum, imperfect information games.

moves. The selection of the best move may be brittle, in that a single node mis-evaluation can propagate to the root of the search and alter the move choice. As the search progresses, the bounds on the value of each move are narrowed, and the certainty of the best move choice increases. The deeper the search, the greater the confidence in the selected move, and after a certain point there are diminishing returns for further search.

In an imperfect information game of respectable size, it is impossible to examine the entire game tree of possibilities [19]. This is especially true for poker because of the many opponents, each making their own decisions. The pseudo-code for the proposed method of selective sampling is shown in Figure 10 [5]. This approach has the following properties:

1. The search is full depth, but limited breadth. That is, each line is played out to the end of the game (in poker, to the showdown or until one player wins uncontested).

2. Heuristic evaluation occurs at the interior nodes of the search tree to decide on future moves by the players. Outcomes are determined at the leaf nodes of the game tree, and are 100% accurate.

3. The search gets progressively wider, performing trials consistent with the probability distribution of hidden information, until real-time constraints dictate that a choice be made.

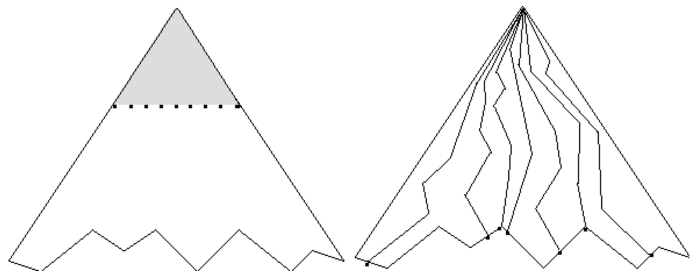

Figure 11: Comparing two search frameworks.

The expected values of all move alternatives are computed, and the resulting choice may be a randomized *mixed strategy.* As the search progresses, the values for each move become more precise, and the certainty of the best move choice increases.[19] The more trials performed, the greater the confidence in the selected move, and after a certain point there are diminishing returns for performing additional trials.

Although the move sequences examined during an alpha-beta search are systematic and non-random, it can be viewed as a sampling of related positions, used as evidence to support the choice of best move. In the case of selective sampling, the evidence is statistical, and the confidence can be measured precisely. The two contrasting methods are depicted in Figure 11, with alpha-beta search on the left and simulation-based search on the right.

As noted previously, it is not essential to continue each trial to the end of the game. In non-deterministic games, the expected value of internal game tree nodes can also be heuristically estimated with a score (as in Scrabble), an evaluation function (as in backgammon), or other methods (such as the roll-out simulations described in Section 5.1).

An important feature of the simulation-based framework is the notion of an obvious move. Although some alpha-beta programs try to incorporate an obvious move feature, the technique is usually *ad hoc* and based on programmer experience, rather than a sound analytic technique (an exception is the B* proof procedure [1]). In the simulation-based framework, an obvious move is well-defined. If one choice exceeds the alternatives by a statistically significant margin, we can stop the simulation early and take that action, with precise knowledge of the mathematical validity of the decision. Like alpha-beta pruning, this early cut-off may prove to be an effective means for reducing the required amount of search effort, especially if it is applied at all levels of the imperfect information game tree.

The proposed framework is not a complete ready-made solution for non-deterministic games, any more than alpha-beta search is the only thing required for high-performance in a particular deterministic game. As discussed in Sec-

[19]The "best" move is somewhat subjective. Here we do not consider certain plays, such as deliberately misrepresenting the hand to the opponents.

tion 5.3, there are many trade-offs to be considered and explored. One must find a good balance between the amount of effort spent on each trial, and the total number of trials completed in the allotted time. There are many different ways to create an evaluation function, and as with other strong game programs, speed and consistency may be more important than explicit knowledge and complexity.

9 Conclusions and Future Work

Poker is a complex game, with many different aspects, from mathematics and hidden information to human psychology and motivation. To master the game, a player must handle all of them at least adequately, and excel in most. Strong play also requires a player to be adaptive and unpredictable – any form of fixed recipe can and will be exploited by a good opponent. Good players must dynamically alter their style, based on the current game conditions and on historical knowledge (including past sessions). In contrast, traditional games like chess are somewhat homogeneous in nature, where one can focus very deeply on one particular type of strategy.

Like other computer game-playing research, poker has a well-defined goal, and the relative degree of success is measurable – whether the program plays the game well, or doesn't. We have resisted the temptation of focusing only on the clearly tractable problems, in favor of grounding the research on those topics that actually affect the bottom line the most. As a result, developing *Poki* has been a cyclic process. We improve one ability of the program until it becomes apparent that another property is the performance bottleneck. Some of the components in the current system are extremely simplistic (such as a constant where a formula or an adaptive method would be better), but do not yet appear to limit overall performance. Others have received much more attention, but are still woefully inadequate.

Human poker players are very good at understanding their opponent, often forming an accurate model based on a single data point (and occasionally before the first hand is dealt!). Programs may never be able to match the best players in this area, but they must at least try to reduce the gap, since they can clearly be superior in other aspects of the game. Although *Poki* has successfully used opponent modeling to improve its level of play, it is abundantly clear that these are only the first steps, and there are numerous opportunities for improvement.

For example, the current system becomes slower to adjust as more information is collected on a particular opponent. This "build-up of inertia" after thousands of data points have been observed can be detrimental if the player happens to be in an uncommon mood that day. Moreover, past success may have largely been due to opponents staying with a fixed style that does not vary over time (most computer opponents certainly have this property). It is much more difficult to track good players who constantly "change gears" for a relatively brief time. Although recent actions are mixed with the long-term record, a superior historical decay function could allow the system to keep up with current events better.

It is easy to gather lots of data on each opponent, but it is difficult to discern the most useful features. It is possible that simpler metrics may be better predictors of an opponent's future behavior. There are also several techniques in the literature for learning in noisy domains where one must make inferences based on limited data, which have not yet been explored.

For the simulations, the major problem is the high variance in the results. Even with noise reduction techniques, the standard deviation can still be high. Faster machines and parallel computations might help to base decisions on a larger sample size. This eventually has diminishing returns, and our empirical results suggest that the benefits may be small beyond a necessary minimum number of data points (roughly 500). Once the critical minimum can be attained in real-time, the more important issue is whether the trials are fair and representative of the situation being modeled.

For the game of bridge, simulations have successfully allowed computer programs to play hands at a world-class level [15]. Nevertheless, limitations in the simulation-based approach and the high variance have prompted Matt Ginsberg, the author of *GIB*, to look at other solutions, including building the entire search tree [16]. We too may have to look for new approaches to overcome the limitations of simulations.

The poker project is rich in research opportunities, and there is no shortage of new ideas to investigate. Having explored some fairly straight-forward techniques to accomplish a reasonable level of play, we are now contemplating re-formulations that might produce a breakthrough to a world-class level of play. Toward this end, some of our current research has moved toward empirical techniques for deriving game-theoretic near-optimal solutions for betting strategies. We have also given more attention to two-player Hold'em, in which many of the flaws of the current system are emphasized.

However, it is not clear if a single unifying framework is possible for poker programs. Certain abilities, such as the accurate estimation of expected values in real time, will eventually be well-solved. However other aspects, like opponent modeling, are impossible to solve perfectly, since even the opponents may not understand what drives their actions!

A Glossary of Poker Terms

This appendix contains definitions of common poker terms used in this paper. More extensive poker glossaries are available on the world wide web, such as `http://www.kimberg.com/poker/dictionary.html`, or `http://conjelco.com/pokglossary.html`.

- **All-in** To have one's entire stake committed to the current pot. Action continues toward a **side pot**, with the all-in player being eligible to win only the main pot.

- **All-in Equity** The expected income if the current hand was permitted to go to the **showdown** with no further betting.
- **Bet** To make the first wager of a betting round (compare **raise**).
- **Big Bet** The largest bet size in limit poker. $20 in $10-$20 Hold'em.
- **Big Blind** A forced **bet** made before the deal of the cards. $10 in $10-$20 Hold'em, posted by the second player to the left of the **button**.
- **Blind** A forced **bet** made before the deal of the cards (see **small blind** and **big blind**).
- **Bluff** To **bet** with the expectation of losing if called.
- **Board** The **community cards** shared by all players.
- **Button** The last player to act in each betting round in Texas Hold'em.
- **Call** To match the current level of betting. If the current level of betting is zero, the term **check** is preferred.
- **Check** To decline to make the first wager of a betting round (compare **call**).
- **Check-Raise** To **check** on the first action and then **raise** in the same betting round after someone else has **bet**.
- **Community Cards** The public cards shared by all players.
- **Connectors** Two cards differing by one in rank, such as 7-6. More likely to make a straight than other combinations.
- **Draw** A hand with good potential to make a strong hand, such as a *straight draw* or a *flush draw* (compare **made hand**).
- **Dry** Lacking possible draws or betting action, as in a *dry board* or a *dry game*.
- **Flop** The first three **community cards** dealt in Hold'em, followed by the second betting round (compare **board**).
- **Fold** To discard a hand instead of matching the outstanding **bet**, thereby losing any chance of winning the pot.
- **Free-Card Raise** To **raise** on the **flop** intending to **check** on the **turn**.
- **Hand** (a) A player's private cards (two **hole cards** in Hold'em). (b) One complete game, from the dealing of the cards to the **showdown** (or until one player wins uncontested).
- **Hole Card** A private card in Hold'em.

- **Implied Odds** The **pot odds** based on the probable future size of the pot instead of the current size of the pot.
- **Income rate** The expected amount a hand will win.
- **Kicker** A side card, often deciding the winner when two hands are otherwise tied (*eg.* a player holding Q-J when the **board** is Q-7-4 has **top pair** with a Jack kicker).
- **Loose Game** A game having several **loose players**.
- **Loose Player** A player who does not fold often (*eg.* one who plays most hands at least to the **flop** in Hold'em).
- **Made hand** A hand with a good chance of currently being the best, such as **top pair** on the **flop** in Hold'em (compare **draw**).
- **Mixed strategy** Handling a particular situation in more than one way, such as to sometimes **call**, and sometimes **raise**.
- **Near-optimal** A good approximation of a game-theoretic optimal solution.
- **Offsuit** Two cards of different suits (compare **suited**).
- **Open-Ended Draw** A **draw** to a straight with eight cards to make the straight, such as 6-5 with a **board** of Q-7-4 in Hold'em.
- **Outs** Cards that will improve a hand to a probable winner (compare **draw**).
- **Pocket Pair** Two cards of the same rank, such as 6-6. More likely to make three of a kind than other combinations (see **set**).
- **Pot Odds** The ratio of the size of the pot to the size of the outstanding bet, used to determine if a **draw** will have a positive expected value.
- **Pre-flop** In Hold'em, the first betting round after the deal of the cards and before the **flop**.
- **Raise** To increase the current level of betting. If the current level of betting is zero, the term **bet** is preferred.
- **Raising for a Free-card** To **raise** on the **flop** intending to **check** on the **turn**.
- **River** The fifth **community card** dealt in Hold'em, followed by the fourth (and final) betting round.
- **Second pair** Matching the second highest **community card** in Hold'em, such as having 7-6 with a **board** of Q-7-4.

- **Set** Three of a kind, formed with a **pocket pair** and one card of matching rank on the **board**. A powerful well-disguised hand (compare **trips**).
- **Showdown** The revealing of cards at the end of a **hand** to determine the winner and award the pot.
- **Side-pot** A second pot for the bets made by active players after another player is **all-in**.
- **Slow-play** To **call** with a strong hand, and then **raise** in a later betting round, for purposes of deception.
- **Small Bet** The smallest bet size in limit poker. $10 in $10-$20 Hold'em.
- **Small Blind** A forced **bet** made before the deal of the cards. $5 in $10-$20 Hold'em, posted by the first player to the left of the **button**.
- **Suited** Two cards of the same suit, such as both Hearts. More likely to make a flush than other combinations (compare **offsuit**).
- **Table Image** The general perception other players have of one's play.
- **Table Stakes** A poker rule allowing a player who cannot match the outstanding bet to go **all-in** with his remaining money, and proceed to the **showdown** (also see **side pot**).
- **Texture of the Board** Classification of the type of **board**, such as having lots of high cards, or not having many draws (see **dry**).
- **Tight Player** A player who usually folds unless the situation is clearly profitable (*eg.* one who folds most hands before the **flop** in Hold'em).
- **Top pair** Matching the highest **community card** in Hold'em, such as having Q-J with a **board** of Q-7-4.
- **Trips** Three of a kind, formed with one **hole card** and two cards of matching rank on the **board**. A strong hand, but not well-disguised (compare **set**).
- **Turn** The fourth **community card** dealt in Hold'em, followed by the third betting round.
- **Wild Game** A game with a lot of raising and re-raising. Also called an **action game**.

Acknowledgments

This research was supported by the Natural Sciences and Engineering Council of Canada and the Alberta Informatics Circle of Research Excellence.

We gratefully acknowledge the many comments and suggestions made by the anonymous referees.

References

[1] H. Berliner. The B* tree search algorithm: A best-first proof procedure. *Artificial Intelligence*, 12:23–40, 1979.

[2] H. Berliner, G. Goetsch, and M. Campbell. Measuring the performance potential of chess programs. *Artificial Intelligence*, 43:7–20, 1990.

[3] D. Billings. The first international Roshambo programming competition. *International Computer Games Association Journal*, 23(1):42–50, 2000.

[4] D. Billings. Thoughts on Roshambo. *International Computer Games Association Journal*, 23(1):3–8, 2000.

[5] D. Billings, D. Papp, L. Pena, J̃. Schaeffer, and D. Szafron. Using selective-sampling simulations in poker. In *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, pages 13–18, 1999.

[6] D. Billings, D. Papp, J. Schaeffer, and D. Szafron. Opponent modeling in poker. In *AAAI National Conference*, pages 493–499, 1998.

[7] D. Billings, L. Pena, J. Schaeffer, and D. Szafron. Using probabilistic knowledge and simulation to play poker. In *AAAI National Conference*, pages 697–703, 1999.

[8] D. Carmel and S. Markovitch. Incorporating opponent models into adversary search. In *AAAI National Conference*, pages 120–125, 1995.

[9] C. Cheng. Recognizing poker hands with genetic programming and restricted iteration. In J. Koza, editor, *Genetic Algorithms and Genetic Programming at Stanford*, 1997.

[10] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence (IC-AI'2000)*, pages 1467–1473, 2000.

[11] D. Egnor. Iocaine powder. *International Computer Games Association Journal*, 23(1):33–35, 2000.

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

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

[14] M. Ginsberg. Partition search. In *AAAI National Conference*, pages 228–233, 1996.

[15] M. Ginsberg. GIB: Steps toward an expert-level bridge-playing program. In *International Joint Conference on Artificial Intelligence*, pages 584–589, 1999.

[16] M. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, 2001. To appear.

[17] H. Iida, J. Uiterwijk, J. van den Herik, and Ĩ. Herschberg. *Thoughts on the Application of Opponent-Model Search*, pages 61–78. University of Maastricht, 1995.

[18] P. Jansen. *Using Knowledge About the Opponent in Game-Tree Search*. PhD thesis, Carnegie-Mellon University, 1992.

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

[20] K. Korb and A. Nicholson. Bayesian poker. In *Uncertainty in Artificial Intelligence*, pages 343–350, 1999.

[21] H. W. Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

[22] J. F. Nash. Non-cooperative games. *Annals of Mathematics*, 54:286–295, 1951.

[23] J. F. Nash and L. S. Shapley. A simple three-person poker game. *Contributions to the Theory of Games*, 1:105–116, 1950.

[24] D. Papp. Dealing with imperfect information in poker. Master's thesis, University of Alberta, 1998.

[25] L. Pena. Probabilities and simulations in poker. Master's thesis, University of Alberta, 1999.

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

[27] J. Schaeffer, D. Billings, L. Pena, and D. Szafron. Learning to play strong poker. In M. Kubat and J. Fuernkranz, editors, *Learning in Games*. Nova Science Publishers, 2001. To appear.

[28] A. Selby. Optimal heads-up preflop poker. 1999. http://www.archduke.demon.co.uk/simplex/.

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

[30] B. Sheppard. Computer scrabble. *Artificial Intelligence*, 2001. Elsewhere in this issue.

[31] J. Shi and M. Littman. Abstraction models for game theoretic poker. In *Computer Games'00*. Springer-Verlag, 2001. To appear.

[32] D. Sklansky. *The Theory of Poker*. Two Plus Two Publishing, 1992.

[33] D. Sklansky and M. Malmuth. *Hold'em Poker for Advanced Players*. Two Plus Two Publishing, 2nd edition, 1994.

[34] S. Smith. Flexible learning of problem solving heuristics through adaptive search. In *International Joint Conference on Artificial Intelligence*, pages 422–425, 1983.

[35] K. Takusagawa. *Nash Equilibrium of Texas Hold'em Poker*. Undergraduate thesis, Stanford University, 2000.

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

[37] G. Tesauro. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 2001. Elsewhere in this issue.

[38] J. von Neumann and O. Morgenstern. *The Theory of Games and Economic Behavior*. Princeton University Press, 2nd edition, 1947.

[39] D. Waterman. A generalization learning technique for automating the learning of heuristics. *Artificial Intelligence*, 1:121–170, 1970.

[40] N. Zadeh. *Winning Poker Systems*. Prentice Hall, 1974.

[41] N. Zadeh. Computation of optimal poker strategies. *Operations Research*, 25(4):541–562, 1977.