A New Algorithm for Generating Strong Strategies in Massive Zero-Sum Games

Martin Zinkevich Michael Bowling Neil Burch
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada T6G 2E8

Abstract

In normal scenarios, computer scientists often consider the number of states in a game to capture the difficulty of learning an equilibrium. However, players do not see games in the same light: most consider Go or Chess to be more complex than Monopoly. In this paper, we discuss a new measure of game complexity that links existing state-of-the-art algorithms for computing approximate equilibria to a more human measure. In particular, we consider the range of skill in a game, i.e.how many different skill levels exist. We then modify existing techniques to design a new algorithm to compute approximate equilibria whose performance can be captured by this new measure. We use it to develop the first near Nash equilibrium for a four round abstraction of poker, and show that it would have been able to win handily the bankroll competition from last year's AAAI poker competition.

Introduction

Which is considered by humans to be the more challenging game: Go, Backgammon, Monopoly, or Poker? One might argue that Go is the most complex due to the large number of possible moves early in the game. Backgammon, although having fewer moves, actually has more possible outcomes on the first turn. Or one could argue that Monopoly is the most complex since it has the largest state space (due to the many possible levels of player wealth). Or maybe Poker's imperfect information outweighs other considerations.

All of these rationalizations actually have some merit. World champion programs in Chess and Checkers leverage the small number of possible moves under consideration to search deep into the future (Samuel 1959; Hsu 2002). Moreover, the (relatively) small number of game states in Checkers is being leveraged in an effort to outright solve the game, i.e., determine if the game is a win, loss, or draw for the first player (Schaeffer *et al.* 2003). Meanwhile, imperfect information games have been far less assailable than those involving perfect information. This is manifested in the computational needs for solving such games. A perfect information game can be solved by iterating over every state, thus requiring a linear amount of time. Imperfect information games, however, require solving a linear program (Romanovskii 1962; Koller & Megiddo 1992; Koller, Megiddo, & von Stengel 1994), which can require time cubic in the number of states. In summary, although the size of the state space, branching factor, and information properties are suggestive of difficulty they alone do not make a game challenging. However they are the sole factors in how solution techniques scale with the problem.

In this paper we suggest a new measure called the *range of skill*, which may more accurately capture the human perception of game difficulty. Imagine forming a line of Go players where the player at the head of the line can beat everyone else in the line at least 75% of the time. The second player can beat everyone *behind* her at least 75% of the time, and so on. Clearly, there is a limit to how long this line can be. Monopoly would likely have a much shorter line. And Tic-Tac-Toe's line would be even shorter. The length of a game's line of players may be a suitable measure of its difficulty. Arguments along these lines have been used to assert that Go is a more complex game than Chess since its line would be longer (Mérő & Mészáros 1990). Finding and lining up human players, however, is not exactly suitable for a computational measure. A related measure is the length of a similar line constructed using *arbitrary* strategies. We will call this the *range of skill*.[1]

The range of skill measure is more than just an academic exercise. After formalizing the measure, we then describe an algorithm for computing an approximate Nash equilibrium that constructs a list of strategies strongly resembling the lines above. We then exploit the range of skill concept to bound the computational complexity of finding an approximate Nash equilibrium with this algorithm.

As a demonstration, we use this algorithm to develop Smallbot2298, an approximate Nash equilibrium in an abstraction of limit Texas Hold'em, the poker variant used in the AAAI Computer Poker Competition. The solution is the first near-equilibrium solution to a complete four-round abstraction of the game, which is currently intractable for linear programming solvers. We also show its lack of exploitability against richer abstractions, suggesting that diminishing returns have been hit for the particular style of

[1] Arbitrary strategies do create some issues. Consider the game where both players choose a number between 1 and n, and the largest number wins. The range of skill of this game is linear in n, and yet from a human perspective it remains an extremely simple game.

abstraction. Finally, we play the strategy against the "bots" from the 2006 AAAI Computer Poker Bankroll Competition, showing that it would have won the competition had it been entered.

Foundations

We begin with some basic definitions in order to arrive at a formalization of the range of skill measure. For any set S, define $\Delta(S)$ to be the set of all distributions over S. In general, a two player zero-sum game G can be represented by a pair of strategy sets S_1 and S_2 and utility function $u : S_1 \times S_2 \rightarrow \mathbf{R}$, which we extend in the natural way by the linearity of utilities to $S_1 \times \Delta(S_2)$, $\Delta(S_1) \times S_2$, and $\Delta(S_1) \times \Delta(S_2)$. For every outcome $(s_1, s_2) \in (S_1 \times S_2)$, the utility for the first player is $u(s_1, s_2)$ and the utility for the second player is $-u(s_1, s_2)$. In addition, we say that the game is **symmetric** if $S_1 = S_2$ and $u(s_1, s_2) = -u(s_2, s_1)$.

Given a game $G = (S_1, S_2, u)$, $G' = (S'_1, S'_2, u')$ is a **restricted game** if and only if $S'_1 \subseteq S_1$, $S'_2 \subseteq S_2$, and $s_1 \in S'_1$, $s_2 \in S'_2$, $u'(s_1, s_2) = u(s_1, s_2)$.

A **Nash equilibrium** is a pair of strategies,[2] $(\sigma_1, \sigma_2) \in \Delta(S_1) \times \Delta(S_2)$ such that:

$$u(\sigma_1, \sigma_2) \geq \max_{\sigma'_1 \in \Delta(S)} u(\sigma'_1, \sigma_2) \tag{1}$$

$$u(\sigma_1, \sigma_2) \leq \min_{\sigma'_2 \in \Delta(S)} u(\sigma_1, \sigma'_2). \tag{2}$$

In other words, neither player can increase their utility be changing their strategy. For any other Nash equilibrium (σ'_1, σ'_2), $u(\sigma'_1, \sigma'_2) = u(\sigma_1, \sigma_2)$, and this is called the value of the game $v(G)$. This value can be thought of as the amount of utility the first player should expect to get by playing the game. Note that playing σ_1 guarantees the first player at least the value of the game (similarly playing σ_2 guarantees player two at least -$v(G)$).

An **ϵ-Nash equilibrium** is a pair of strategies (σ_1, σ_2) such that:

$$u(\sigma_1, \sigma_2) + \epsilon \geq \max_{\sigma'_1 \in \Delta(S)} u(\sigma'_1, \sigma_2) \tag{3}$$

$$u(\sigma_1, \sigma_2) - \epsilon \leq \min_{\sigma'_2 \in \Delta(S)} u(\sigma_1, \sigma'_2) \tag{4}$$

An ϵ-Nash equilibrium also has the property that playing σ_1 guarantees the first player a utility of $v(G) - \epsilon$.

We can now state the definition of our proposed measure.

Definition 1 *Given a zero-sum symmetric game G with utility u, define a list of strategies $\sigma_1, \ldots, \sigma_N$ to be an **ϵ-ranked list** if for all $i > j$, $u(\sigma_i, \sigma_j) > \epsilon$. The **range of skill** or $ROS_\epsilon(G)$ is the length of the longest ϵ-ranked list.*

The Algorithm

Range of skill gives a measure of game difficulty that depends more on the game's strategic choices than various descriptors of its size. In this section we turn the measure's intuition into an algorithm. The resulting algorithm's computational complexity also depends more on the game's strategic choices than more common bounds focused on its size.

[2] We will use s to denote pure strategies and σ to represent randomized strategies.

Let the **best-response oracle** for the first player to be an oracle that, given σ_2, computes a strategy $BR(\sigma_2) \in S_1$ such that:

$$u(BR(\sigma_2), \sigma_2) = \max_{s'_1 \in S}(s'_1, \sigma_2) \tag{5}$$

The best-response oracle for the second player is defined similarly. All of the algorithms in this section assume such an oracle exists and can be computed efficiently.

Our algorithm builds on the ideas of McMahan and colleagues (McMahan, Gordon, & Blum 2003) and their Double Oracle Algorithm. In particular, they maintain a set of strategies for each player which they extend using a best response oracle.

Algorithm 1: (Double Oracle Bundle Algorithm) (McMahan, Gordon, & Blum 2003)

1. Initialize $S'_1 \subseteq S_1$ and $S'_2 \subseteq S_2$. These could be arbitrary singletons.
2. Repeat until satisifed.
   (a) Find a Nash equilibrium (σ_1, σ_2) in the restricted game $G = (S'_1, S'_2, u)$.
   (b) Compute $BR(\sigma_1)$ and add it to S'_2.
   (c) Compute $BR(\sigma_2)$ and add it to S'_1.
3. Return (σ_1, σ_2).

The algorithm eventually converges, but may have to generate every strategy in the game. There is also a similarity to earlier work on the Lemke-Howson algorithm (Lemke & Howson 1964) and constraint generation (Gilmore & Gomory 1960; Dantzig & Wolfe 1960).

At the highest level, our goal is to be more particular while generating strategies. In particular we want to insure the sequence of strategies generated satisfy the conditions on the range of skill, hence limiting the total number of strategies required to be generated. The primary idea behind our research is the concept of a *generalized best response*.

Definition 2 *Given a game $G = (S_1, S_2, u)$, a **generalized best response** to a set of strategies $S'_1 \subseteq S_1$ is a strategy $\sigma'_2 \in \Delta(S_2)$ such that (σ'_1, σ'_2) is a Nash equilibrium in the restricted game $G = (S'_1, S_2, u)$. Define σ'_1 to be one of the **safest** S'_1 mixed strategies. If (σ'_1, σ'_2) is instead part of an ϵ Nash equilibrium, it is an **ϵ generalized best response**.*

Notice that a generalized best response is actually an equilibrium strategy of a restricted game. Therefore it can actually be a mixed strategy in more than just the trivial circumstances.

A generalized best response can be computed in two ways. The first is by using a linear program, similar to the techniques of Koller and Meggido (Koller & Megiddo 1992). However, a simpler technique is to use basically half of the Double Oracle Algorithm.

Algorithm 2: (Generalized Best Response I)

1. Given $S_1' \subseteq S_1$, initialize $S_2' \in S_2$ arbitrarily.
2. Repeat until satisfied.
   (a) Find a Nash equilibrium (σ_1, σ_2) to $G = (S_1', S_2', u)$.
   (b) Compute $BR(\sigma_1)$ and add it to S_2'.
3. Return σ_2.

In the limit, the above algorithm computes a Nash equilibrium of the game (S_1', S_2, u). It also can be used to return an ϵ-generalized best response of arbitrary precision ϵ.

We can now define our algorithm. For the sake of simplicity, and to make our theoretical results simpler, we will focus on defining it for symmetric games.

Algorithm 3: (Range-Of-Skill Algorithm)

1. Initialize $\Sigma' \subseteq \Delta(S)$ as a singleton.
2. Repeat until satisfied.
   (a) Find an ϵ'-generalized best response to Σ', σ.
   (b) Add σ to Σ'.
3. Return one of the safest mixed strategies Σ' as an ϵ-Nash equilibrium.

The range of skill algorithm builds a set of strategies based on generalized best response, rather than best-response. As such, the strategies themselves are equilibria for restricted games of increasing size. The algorithm can be extended to asymmetric games by maintaining different sets of strategies for each player. Moreover, this algorithm can be applied to an extensive-form game, or any game where a generalized best response can be computed efficiently.

The double oracle algorithm could possibly generate every strategy in the game, the range of skill algorithm is bounded by our proposed measure of game difficulty.

Theorem 1 *If $\epsilon' = 0$ (the generalized best responses are exact) then the Range-Of-Skill Algorithm provides an ϵ-Nash equilibrium in $ROS_\epsilon(G)$ iterations.*

Proof: First, consider the range-of-skill algorithm when $\epsilon = 0$. Define σ_1 to be the first strategy in Σ_1', and σ_t to be the strategy generated on the tth iteration of the range of skill algorithm. Observe that, if $T < T'$:

$$\min_{t<T} u(\sigma_T, \sigma_t) \geq \min_{t<T} u(\sigma_{T'}, \sigma_t) \quad (6)$$

$$\geq \min_{t<T'} u(\sigma_{T'}, \sigma_t) \quad (7)$$

If this were not the case, then σ_T would not be a generalized best response to $\{\sigma_1, \ldots, \sigma_{T-1}\}$. Define $\epsilon_T = \min_{t<T} u(\sigma_T, \sigma_t)$. Since $\epsilon_2 \geq \epsilon_3 \geq \ldots \epsilon_T$, then $\{\sigma_1, \ldots, \sigma_T\}$ is an ϵ_T-ranked list. Thus, $ROS_\epsilon(G)$ provides a bound on the number of iterations before the Range-Of-Skill Algorithm provides an ϵ-Nash equilibrium. ■

We now look at the application of this new algorithm to the game of poker.

Poker and Equilibria

Heads-Up Limit Texas Hold'em is a two-player variant of poker, and the one used in the 2006 AAAI Computer Poker Competition. Unlike in draw poker games, there is absolutely no skill involved in getting a good hand: instead, the player's betting strategy entirely determines the quality of their play, which is usually measured in small-bets per hand, where a small-bet is the smallest denomination that can be wagered (usually two chips).

A match of Limit Hold'em consists of a number of hands, in which private and public cards are dealt and the players bet, i.e., put chips in the pot. The player who doesn't fold or, if no one folds, has the best hand wins the chips in the pot. At the beginning of the hand, the players put blinds into the pot: one player (the small blind) puts one chip in the pot, and the other (the big blind) puts two chips into the pot, while the small blind alternates every hand. Then, they each receive two cards privately. A round of betting occurs, where a bet or raise is always the small bet, in which no more than three raises (a bet, a raise, and a re-raise) can occur. The next round (the flop), three public cards are put on the board. Then, another round of betting occurs, where a bet or raise remains the small bet and no more than four raises can occur, which remains the max for the remaining rounds. The next round (the turn), one public card is placed on the board. Then, another round of betting occurs, where a bet or raise is twice the small bet (a big bet). In the final round (the river), one last card is placed on the table, and the final round of betting occurs, where a bet or raise is still limited to the big bet.

If no one has folded, a showdown occurs. Each player makes the best five card hand amongst his or her private cards and the five public cards on the board. The player with the best five card poker hand wins the pot; if the hands are of equal value, a tie occurs, and the pot is split (each player gets back his or her chips).

The number of states in this game is huge: there are over 1.3×10^{15} card sequences alone, not considering the betting. Therefore, traditionally the game is abstracted. There are a number of common abstractions used to create a more tractable restricted game.

1. Symmetries in the cards. Since a card's exact suit holds no special status, permutations of the four suits are all equivalent games. Likewise, the order in which the private or flop cards are dealt is also insignificant. These symmetries can be abstracted away.
2. Eliminate dominated strategies. For example, remove fold strategies when no chips are required to call.
3. Force the players' strategies to behave the same on two different card sets. Due to the fact that there are seven cards visible to a player during a game, some assumption along this line must be done in order to fit the strategy into memory. The set of card sets that are indistinguishable are called **buckets**.
4. Treat situations with four bets in a round as though there were only three, as there are few situations where such betting is appropriate.

5. Fix the players' betting strategy for a single round. This causes a drastic reduction in the size of the game.

These restrictions must be selected with a great deal of care. The more benign abstractions often do not offer a large enough reduction in the size of the game. Many of the more dramatic reductions are used at the cost of introducing serious flaws in the resulting equilibrium programs.

For example, the work of Billings' and colleagues (Billings *et al.* 2003) forced the players' betting in the first round to follow a fixed strategy. This caused significant weaknesses in the bot that have been exploited significantly by both humans and other bots (Billings *et al.* 2004). Gilpin and Sandholm (Gilpin & Sandholm 2006) solved the first two rounds of the game assuming that the strategy would always call afterwards. Even though a second equilibrium solution replaced these later rounds, the disconnect in the equilibrium resulted in a significant weakness (Zinkevich & Littman 2006).

These examples suggests it is important to solve for all of the rounds simultaneously. However, even with other aggressive forms of abstraction it is intractable to use the techniques of Koller and Megiddo (Koller & Megiddo 1992) on a full four-round game without fixing at least one round. The reason is actually two-fold: the size of the four-round game is difficult to fit into memory and there is numeric instability when dealing with the products of four full rounds of probabilities as is required by sequence form.

The range of skill algorithm presented in the previous section, though, offers an alternative to the techniques of Koller and Megiddo. Using the range of skill algorithm we can solve a complete four-round abstraction of Limit Texas Hold'em. In particular, we make two major abstraction choices. First, we bucket all card sequences down to only 625 for each player. This bucketing is done by compressing the player's private cards and the board cards down to a single number called hand strength, which measures the probability of winning with the hand against a random hand. Buckets are formed as the quintiles of this hand strength on every round given the previous rounds' buckets so that the probability of transition to any bucket in the next round has a nearly equal probability. This results in $5^4 = 625$ buckets. Second, we assume that only three bets are allowed per round reducing the number of betting sequences by a factor of 5. Observe that storing a strategy for this restricted game, the primary requirement of the range of skill algorithm, requires 625 $\times$ 3425 floating point values or around 17MB. Contrast this with representing the whole game, the primary requirement for the linear program, which requires 625 times more memory or almost 11GB.

Empirical Results

We used the range of skill algorithm to find an approximate Nash equilibrium to the abstracted game described in the previous section. This bot was called SmallBot2298 (second generation bot after 298 iterations of the range of skill algorithm). SmallBot2298's abstraction assumed that there are two raises after the blinds in the first round (when there are normally three) and there are three raises in the last three rounds (when there are four). If, while playing the real game, one of these constraints was violated, it would be due to a raise by the opponent. We created BigBot2298 that could play the full game: it calls any such raise, and plays how SmallBot2298 would have if the raise had not happened. We investigate the effectiveness of BigBot2298 both in terms of an analysis of its exploitability and its performance against other near optimal strategies.

Due to the fact that a best response is much easier to compute than a Nash equilibrium, we can actually compute the suboptimality of SmallBot2298 in games with various levels of abstraction. To begin with, SmallBot2298 is within 0.01 small bets per hand of optimal in the abstracted game in which it plays[3]. Moreover, BigBot2298 is within 0.025 small bet per hand of optimal in the game where the betting is not abstracted but the cards are. This was determined by computing the best response to BigBot2298 in this game. The low additional exploitability justifies the choice of this abstraction. We also considered its exploitability versus a bot with a much finer level of hand strength abstraction.. In particular, we considered a bot that used the same card abstraction technique, but with ten buckets per round for a total of 10,000 sequences instead of 625. The best-response to SmallBot2298 in this game was 0.053 and 0.065 with and without the betting abstraction, respectively. These results, split by player, are summarized in Table 1.

Further, we ran BigBot2298 against the competitors from the bankroll competition of the 2006 AAAI Poker Competition. BigBot2298 and each competitor played 40,000 hands. In particular, we randomly generated the cards for 20,000 hands, and the hand was played twice with each bot in both seats. The results are displayed in Table 2 (combined with the results from the competition).

Observe that not only does BigBot2298 have a statistically significant positive expected winnings against all the other bots, but its average win rate of 0.34 small bets per hand is higher than Hyperborean's 0.28 small bets per hand win rate. Therefore, it would have won the competition.

Generalized Best Response: A Theoretical Improvement

If the Range-Of-Skill Algorithm is implemented with Generalized Best Response I as a method of computing the generalized best response, in theory one could generate every deterministic strategy in order to get this generalized best response. Therefore, we discuss here another variant which allows us to obtain runtime bounds based upon linear programming runtime bounds.

In particular, one can utilize the techniques of (Koller & Megiddo 1992). From the first player's perspective, if the second player is fixed, a game of poker is like a Markov decision process. This Markov decision process has a set of states S, some of which are terminal $T \subseteq S$ and some of which are non-terminal $N = S \backslash T$. Given a state $s \in N$, there is a set of actions $A(s)$ that can be used in that state. There is also a set of initial states $I \subseteq S$. Given a state

[3] The closest bots in the 2006 AAAI Computer Poker Competition differed by approximately 0.05 small bets per hand.

	5 Buckets, abstracted betting	5 Buckets, full betting	10 Buckets, abstracted betting	10 Buckets, full betting
Player 1	0.0936	0.1150	0.1377	0.1584
Player 2	-0.0725	-0.0620	-0.0322	-0.0204
Average	0.0106	0.0265	0.0528	0.0648

Table 1: The amount won (in small bets per hand) by a best response against the Player 1 and Player 2 strategies of SmallBot2298 in various abstractions of poker. These figures have been computed analytically.

	BigBot2298	Hyperborean	Bluffbot	Monash-BPP	Teddy	Average
BigBot2298		0.061 ± 0.015	0.113 ± 0.011	0.695 ± 0.015	0.474 ± 0.023	0.336
Hyperborean	-0.061 ± 0.015		0.051 ± 0.017	0.723 ± 0.016	0.407 ± 0.025	0.280
BluffBot	-0.113 ± 0.011	-0.051 ± 0.017		0.527 ± 0.020	-0.190 ± 0.043	0.043
Monash-BPP	-0.695 ± 0.015	-0.723 ± 0.016	-0.527 ± 0.020		1.168 ± 0.043	-0.015
Teddy	-0.474 ± 0.023	-0.407 ± 0.025	0.190 ± 0.043	-1.168 ± 0.043		-0.465

Table 2: A cross table indicating the number of small bets per hand won by the row player from the column player in an extended version of the Computer Poker Competition. The results with BigBot2298 were generated for this paper, with other results taken from (Zinkevich & Littman 2006).

$s \in S \backslash I$, there is a predecessor pair $P(s) = (s', a)$ such that playing a in s' can lead to s.

The states of this Markov decision process are betting sequences and card (or bucket) sequences. Observe that in this Markov decision process, there is exactly one path to each state.

For any state $s \in S$, define $p_s^{\sigma_1}$ to be the probability that the first player using strategy σ_1 will select all of the actions required to reach state s, given that all of the actions by the other player (and actions of nature) required also occur. Define $p_{s,a}^{\sigma_1}$ to be the probability of the first player selecting all of the actions required to reach state s and then action a, given that all of the actions by the other player (and actions of nature) required also occur. For all $s \in N$:

$$p_s^{\sigma_1} = \sum_{a \in A(s)} p_{s,a}^{\sigma_1}. \tag{8}$$

If playing action a in state s can immediately lead to s' (i.e., $P(s,a) = s'$), then:

$$p_{s,a}^{\sigma_1} = p_{s'}^{\sigma_1}. \tag{9}$$

For any fixed strategy σ_2 of the second player, each terminal state has a probability of occurring (given the actions of the first player would lead to it) and an expected utility for the first player (fixed if it is a fold; dependent upon the posterior distribution over the opponent's hand if it is a showdown). Therefore, we can define the **utility vector representation**, a function $u^{\sigma_2} : T \to \mathbf{R}$ such that:

$$u(\sigma_1, \sigma_2) = \sum_{s \in T} p_s^{\sigma_1} u_s^{\sigma_2}. \tag{10}$$

Thus, given a set of strategies $S' \subseteq \Delta(S_2)$ for the second player, one can define the worst case performance of the first player as:

$$\min_{\sigma_2 \in S'} u(\sigma_1, \sigma_2) = \min_{\sigma_2 \in S'} \sum_{s \in T} p_s^{\sigma_1} u_s^{\sigma_2}. \tag{11}$$

A generalized best response is:

$$\max_{\sigma_1} \min_{\sigma_2 \in S'} u(\sigma_1, \sigma_2) = \min_{\sigma_2 \in S'} \sum_{s \in T} p_s^{\sigma_1} u_s^{\sigma_2}. \tag{12}$$

We can discover p_s^* and $p_{s,a}^*$ conforming to a generalized best response to S' by maximizing v subject to:

$$p_s = \sum_{a \in A(s)} p_{s,a} \quad \forall s \in N, \tag{13}$$

$$p_{s,a} = p_{s'} \quad \forall s, s' \in S, a \in A(s) \text{ where } P(s') = (s,a), \tag{14}$$

$$p_s \geq 0 \quad \forall s \in T, \tag{15}$$

$$p_{s,a} \geq 0 \quad \forall s \in N, a \in A(s), \tag{16}$$

$$p_s = 1 \quad \forall s \in I, \tag{17}$$

$$v \leq \sum_{s \in T} p_s u_s^{\sigma_2} \quad \forall \sigma_2 \in S'. \tag{18}$$

We call this algorithm Generalized Best Response II.

The above can also be done from the second player's perspective.

Observe that the number of equations and variables is roughly proportional to the number of states $|S|$ plus $|S'|$. Moreover, for a particular $\sigma_2 \in \Delta(S_2)$, the function $u_s^{\sigma_2}$ can be computed from $p_s^{\sigma_2}$ with a single traversal of the states of the full game. This can be done as soon as it is generated (instead of for each generated best response).

If we define g to be the number of game states (nodes in the extensive form game), v to be the number of states in the MDP (the view), then:

Theorem 2 *The Range of Skill Algorithm which uses Generalized Best Response II, given a symmetric game G, can compute an ϵ-Nash equilibrium by solving $ROS_\epsilon(G)$ linear programs of size less than $ROS_\epsilon(G) + v$ each, and iterating over all g states $ROS_\epsilon(G)$ times.*

The following can be derived from (Grotschel, Lovasz, & Schrijver 1988, pages 69–70,75):

Theorem 3 *Given $C \in \{-N \dots N\}^{n \times n}$, $d \in \{-N \dots N\}^n$ with m nonzero entries, assuming $\{x : Cx \leq d\}$ is full-dimensional (i.e., has strictly* *positive volume), one can find $x \in \mathbf{R}^n$ such that $Cx \leq d$ using the ellipsoid method in $O(n^4 m \log N)$ time.*[4]

$O(m \log N)$ is the number of bits required to represent C and d. To grasp this result, note that the ellipsoid method repeatedly reduces the volume under consideration. Secondly, the bounds on the number of nonzero elements in C and the size of the integers in C affects the minimum nonzero volume that $\{x : Cx \leq d\}$ could have.

Determining a Nash equilibrium with one linear program would require a linear program with $\Omega(v)$ variables and $\Omega(v)$ constraints, and $\Omega(g)$ nonzero elements[5] in the coefficient matrix, resulting in a bound of $\Omega(v^4 g)$. On the other hand, computing one generalized best response requires no more than $O(v^4(vROS_\epsilon(G)))$ time, and since there are $ROS_\epsilon(G)$ iterations, this is $O(v^5(ROS_\epsilon(G))^2)$ time (the iteration over all the game states is dwarfed by this amount). Therefore, if $O(ROS_\epsilon(G)^2) < O(g/v)$, the range of skill algorithm could be faster than the traditional algorithm.

Conclusion

In this paper, we demonstrated how an iterated technique for computing a Nash equilibrium could be efficient with memory usage and solve abstractions of Texas Hold'em larger than those solved to date. In particular, the main benefit was that all of the betting rounds were solved together, make a strategy that was a cohesive hole. Strikingly, it was found that the strategy was not only an approximate Nash equilibrium in the game in which it was computed, but had very few weaknesses in a larger game which was much closer to Texas Hold'em. Moreover, it outshone any of the bots in the Bankroll Competition.

However, there are also some striking points about the empirical results which show where improvement could be made in the future. For instance, while the strategy which always raises is at the beginning of the search for a Nash equilibrium, even though Teddy actually does almost always raise, still SmallBot2298 did not do as well as Monash-BPP against Teddy. This is a reminder that, even though an equilibrium is a very safe strategy, an approximate Nash equilibrium is not always the most exploitive strategy of a weaker bot, and suggests that in the future development, exploitation and safety should be balanced.

References

Billings, D.; Burch, N.; Davidson, A.; Holte, R.; Schaeffer, J.; Schauenberg, T.; and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. In *International Joint Conference on Artificial Intelligence*, 661–668.

Billings, D.; Davidson, A.; Schauenberg, T.; Burch, N.; Bowling, M.; Holte, R.; and Schaeffer, J. 2004. Game tree search with adaptation in stochastic imperfect information games. In *Computer and Games*.

Dantzig, G., and Wolfe, P. 1960. Decomposition principle for linear programs. *Operations Research* 8:101–111.

Gilmore, P., and Gomory, R. 1960. A linear programming approach to the cutting stock problem. *Operations Research* 849–859.

Gilpin, A., and Sandholm, T. 2006. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *National Conference on Artificial Intelligence*.

Grotschel, M.; Lovasz, L.; and Schrijver, A. 1988. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag.

Hsu, F. H. 2002. *Behind Deep Blue: Building the Computer that Defeated the World Chess Champion*. Princeton University Press.

Koller, D., and Megiddo, N. 1992. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior* 528–552.

Koller, D.; Megiddo, N.; and von Stengel, B. 1994. Fast algorithms for finding randomized strategies in game trees. 750–759.

Lemke, C., and Howson, J. 1964. Equilibrium points of bimatrix games. *Journal of the Society of Industrial and Applied Mathematics* 413–423.

McMahan, H.; Gordon, G.; and Blum, A. 2003. Planning in the presence of cost functions controlled by an adversary. In *International Conference on Machine Learning*.

Mérő, L., and Mészáros, V. 1990. *Ways of Thinking: The Limits of Rational Thought and Artificial Intelligence*. World Scientific.

Romanovskii, I. V. 1962. Reduction of a game with complete memory to a matrix game. *Soviet Mathematics* 678–681. (English translation of Doklady Akademii Nauk SSSR 1444,62–64.).

Samuel, A. L. 1959. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development* 210–229.

Schaeffer, J.; Björnsson, Y.; Burch, N.; Lake, R.; Lu, P.; and Sutphen, S. 2003. Building the checkers 10-piece endgame databases. *Advances in Computer Games 10* 193–210.

Zinkevich, M., and Littman, M. 2006. The aaai computer poker competition. *Journal of the International Computer Games Association* 29. News item.

[4]There will be $O(n^2 m \log N)$ iterations (Grotschel, Lovasz, & Schrijver 1988, page 75) where each iteration in the ellipsoid method takes $O(n^2)$ time to compute a Löwner-John ellipsoid (Grotschel, Lovasz, & Schrijver 1988, pages 69–70).

[5]This assumes that a significant fraction of the states in the overall game tree are terminal, which is the case if every node has at least two children.