A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs

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

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

Troels Bjerre Sørensen
Dept. of Computer Science
University of Aarhus
Århus, Denmark
trold@daimi.au.dk

ABSTRACT

We present *Tartanian*, a game theory-based player for heads-up no-limit Texas Hold'em poker. *Tartanian* is built from three components. First, to deal with the virtually infinite strategy space of no-limit poker, we develop a discretized betting model designed to capture the most important strategic choices in the game. Second, we employ potential-aware automated abstraction algorithms for identifying strategically similar situations in order to decrease the size of the game tree. Third, we develop a new technique for automatically generating the source code of an equilibrium-finding algorithm from an XML-based description of a game. This automatically generated program is more efficient than what would be possible with a general-purpose equilibrium-finding program. Finally, we present results from the AAAI-07 Computer Poker Competition, in which *Tartanian* placed second out of ten entries.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Miscellaneous; J.4 [**Computer Applications**]: Social and Behavioral Sciences—*Economics*

General Terms

Algorithms, Economics

Keywords

Equilibrium finding, automated abstraction, Nash equilibrium, computational game theory, sequential games, imperfect information games, heads-up no-limit poker

1. INTRODUCTION

Poker is a complex game involving elements of uncertainty, randomness, strategic interaction, and game-theoretic reasoning. Playing poker well requires the use of complex, intricate strategies. Optimal play is far from straightforward, typically necessitating actions intended to misrepresent one's private information. For these reasons, and others, poker has been proposed as an AI challenge problem [4].

Cite as: A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs, Andrew Gilpin, Tuomas Sandholm, and Troels Bjerre Sørensen, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

There has been a recent flurry of research into developing strong programs for playing poker. Just as chess was once seen as an important challenge problem for AI, poker is now starting to be seen in the same way. At the recent Man Versus Machine Poker Competition, two professional poker players, Phil Laak and Ali Eslami, defeated the computer competitors, but by a small margin.

The bulk of the research into poker AI, including that demonstrated at the Man Versus Machine competition, has been on heads-up limit Texas Hold'em [16, 21, 3, 2, 7, 8, 10, 23, 24, 13]. In that game, the players only ever have at most three possible actions (fold, call, or raise). In *no-limit* Texas Hold'em, on the other hand, players may bet any amount up to the amount of chips remaining in their stack. This rule change significantly alters the optimal strategies, and also poses new research problems when developing a computer program for playing the game.

In this paper we present *Tartanian*, our game theory-based player for heads-up no-limit Texas Hold'em poker. After presenting related work (Section 1.1), we describe the rules of the game (Section 2). We present an overview of *Tartanain*, including the three main components, in Section 3. Sections 4–6 discuss each of the three components in more detail, respectively. In Section 7, we present the results of the 2007 AAAI Computer Poker Competition in which *Tartanian* placed second out of ten entries. Finally, in Section 8, we present conclusions and suggest directions for future research.

1.1 Related work on no-limit Texas Hold'em programs

As mentioned above, most AI work on Texas Hold'em poker has been for the limit variety. However, there are a few exceptions that have focused on no-limit.

The most notable contribution to no-limit has been the computation of near-optimal strategies for the *later stages* of a no-limit *tournament* [17, 5]. (In a tournament, the players start with the same number of chips, and play is repeated until only one player has chips left. Typically the minimum bets increase after a certain number of hands, so eventually the stacks are very low relative to the minimum bet.) That work focused on the computation of *jam/fold* strategies, that is, strategies in which the players either fold or bet all of their chips as their first action. In contrast, we study the unlimited space of strategies, which is drastically richer and contains better strategies than jam/fold.

Rickard Andersson's master's thesis [1] is more closely related to the work described in this paper since that work also develops strategies for a heads-up no-limit Texas Hold'em game. However, in that work round-based abstraction is used: the different betting rounds of the game are separated into phases and solved separately. That approach has been used before, but suffers from many known drawbacks [10]. In contrast, we solve the game model in one large optimization. Also, that work considered a game where each player only has 40 chips, whereas we consider a game where each has 1000 chips. Since the size of the betting space grows exponentially in the number of chips each player has, this is a significant difference. Also, the size of the card abstraction that we consider is drastically larger than what was considered in that earlier work.

2. RULES OF HEADS-UP NO-LIMIT TEXAS HOLD'EM POKER

There are many variants of poker. In this paper we focus on two-player (heads-up) no-limit Texas Hold'em poker. As in the 2007 Association for the Advancement of Artificial Intelligence (AAAI) Computer Poker Competition, we consider the variant known as *Doyle's game*, named for the accomplished professional poker player Doyle Brunson who publicized this game. The game rules are as follows.

Blinds Two players, the *small blind* and *big blind*, start every hand with 1000 chips. Before any cards are dealt, the small blind contributes one chip to the pot and the big blind contributes two chips.

Pre-flop Both players receive two *hole cards*, face down, from a standard deck of 52 playing cards. The small blind then has the options of *folding* (thus ending the game and yielding all of the chips in the pot to the other player), *calling* (contributing one more chip), or *raising* (calling one more chip and then adding two or more chips to the pot). In the event of a call or a raise, the big blind has the option to take an action. The players alternate playing in this manner until either one of the players folds or calls. Note that it is possible for a player to go *all-in* at any point by raising all of his remaining chips. Also, the size of the raise must always be at least as large as any raise already made within the current betting round.

Flop Three *community cards* are dealt face up. The players participate in a second betting round, with the big blind going first. The first bet must be at least two chips. If the players are already *all-in* then no betting actions take place.

Turn One community card is dealt face up. The players again participate in a betting round as on the flop.

River A final community card is dealt face up. The players again participate in a betting round as on the flop and turn.

Showdown Once the river betting round has concluded (and if neither player has folded), a *showdown* occurs. Both players form the best five-card poker hand using their two hole cards and the five community cards. The player with the best hand wins the chips in the pot. In the event of two equally ranked hands, the players split the pot.

The differentiating feature of *Doyle's game* compared to other variants of Texas Hold'em is that each player begins every hand with the same number of chips (1000 in our case). This is an important distinction since the quantity of a player's chips greatly influences his optimal strategy. Incorporating this rule makes for a more fair game since both players start every hand on equal footing.

3. OVERVIEW OF *TARTANIAN*

We constructed our poker-playing program, *Tartanian*, from three conceptually separate components. Here we provide an overview of each component.

1. **Discretized betting model.** In no-limit poker, a player may bet any quantity up to the amount of chips he has remaining. Therefore, in principle, the betting action space is infinite (since a player could bet a fractional amount of a chip). Even if players are restricted to betting integral amounts of chips (as is the case in most brick-and-mortar casinos), the number of actions available is huge. (The small blind has nearly 1000 actions available at the time of the first action.) This issue does not arise in limit poker and so has until now received very little attention. To deal with this huge strategy space, we use a *discretized betting model.* This also entails a reverse model for mapping the opponent's actions—which might not abide to the discretization—into the game model. We describe the design and operation of these models in Section 4.

2. **Automated card abstraction.** In addition to abstracting the players' betting actions, it is also necessary to abstract nature's moves of chance (i.e., the dealing of the cards). Recent research has introduced abstraction algorithms for automatically reducing the state-space of the game in such a way that strategically similar states are collapsed into a single state. This can result in a significant decrease in problem size with little loss in solution quality. We apply our *potential-aware* automated abstraction algorithm [10], though this is the first time that that algorithm has been applied in the no-limit setting. We describe this application of *automated card abstraction* to no-limit Texas Hold'em in Section 5.

3. **Equilibrium finding.** Two-person zero-sum games can be modeled and solved as linear programs using simplex or interior-point methods. However, those algorithms do not scale to games as large as the ones we are considering. Recently, we have developed gradient-based algorithms which scale to games many orders of magnitude larger than what was previously possible [12, 6]. We apply these new algorithms to our problem, and we also develop a system for automatically constructing the source code for computing the crucial part of the equilibrium computation directly from a description of the game. This is particularly useful given the wide variety of betting models in which we may ultimately be interested. We detail this *equilibrium-finding* process in Section 6.

The following three sections describe these three components in detail, respectively.

4. BETTING ABSTRACTION

The most immediate difficulty encountered when moving from limit to no-limit Texas Hold'em is in the development of a betting model. In limit Texas Hold'em, the players only ever have at most three possible actions available to them (fold, call, or raise). This small branching factor in the action sequences allows the model builder to include all possible actions in the model of the game used for the equilibrium analysis.[1]

In no-limit Texas Hold'em, on the other hand, the number of actions available to the players can be huge. For example, when the small blind makes his first action, he can fold, call, or raise to any (integral) amount between 4 and 1000, for a total of 999 possible actions. (If the bets were not limited to be integral amounts then the branching factor would actually be infinite.) Information sets (decision points) with high degree occur elsewhere in the game tree as well. Even if bets are limited to integers, the size of the unabstracted game tree of no-limit heads-up Texas Hold'em is approximately 10^{71} nodes, compared to "only" 10^{18} nodes in the limit variant.

In the remainder of this section, we discuss the design of our discretized betting model. This consists of two pieces: the choice of which bet amounts we will allow in our model (Section 4.1) and the mapping of actions in the real game back to actions in our abstracted game (Section 4.2).

4.1 Betting model

Although there are potentially a huge number of actions available to a player at most points of the game, in practice among human players, a few bets occur much more frequently than others. These include bets equal to half of the size of the current pot, bets equal to the size of the current pot, and *all-in* bets. We discuss each of these in turn.

- Bets equal to half of the size of the current pot are good *value bets*[2] as well as good *bluffs*. When a player has a strong hand, by placing a half-pot bet he is giving the opponent 3:1 *pot odds*.[3] For example, if a half-pot bet is placed on the river, then the opponent only needs to think that he has a 25% chance of winning in order for a call to be "correct". This makes it a good value bet for the opponent who has a good hand.

  Half-pot bets also make good bluffs: they only need to work one time in three in order for it to be a profitable play. This bet size is advocated by many poker experts as a good-size bet for bluffing [11].

- Bets equal to the size of the current pot are useful when a player believes that he is currently "in the lead", and does not wish to give the opponent a chance to draw out to a better hand (via the additional cards dealt later on in the hand). By placing a pot bet, the player is taking away the odds that the opponent would need to rationally call the bet—with almost any drawing hand, that is, a hand that is not good currently, but has the potential to improve with additional cards. (Half-pot bets are also good for this purpose in some situations.) It is usually not necessary to bet more than this amount.

  Pot bets are particularly useful pre-flop when the big blind, who will be *out of position* (i.e., acting first) in later betting rounds, wishes to make it more expensive for the small blind to play a particular hand.

- In most situations it is a bad idea to go all-in because if the opponent makes the call, he most likely has the better hand, and if he does not make the call, then nothing (or very little) is gained. However, this is a commonly used move (particularly by beginners). In some situations where the pot is large relative to the players' remaining chips, it makes more sense to employ the all-in move.

  Another good reason for including the all-in bet in the model is that it provides a level of robustness in the model. This aspect will be discussed further in Section 4.2.

There are also a few bets that are particularly poor or redundant actions, and therefore we do not include them in our betting model in order to keep it relatively small, thus gaining computational tractability.

- Making bets that are small relative to the pot are usually a bad idea. When facing such a bet, the opponent has terrific pot odds to make a call. Since the opponent can make the call with almost any hand, not much information about the opponent's hand is revealed. Also, since the bet is so small, it is not of much value to a player with a strong hand.

- Once a player's quantity of remaining chips is small relative to the pot, he is in a situation known as *pot-committed*. When facing a subsequent bet of any size, the player will be facing great pot odds and will almost surely be compelled to call (because he can call with whatever he has left, even if that amount is drastically smaller than the pot). In this sense, a rational player who is pot-committed is basically in the same situation as a player who went all-in already. Thus bets that lead to pot-committed situations are, in a sense, nearly redundant. Therefore, in order to reduce the action space for computational tractability, we advocate not allowing bets that put the player in a pot-committed situation. Similarly, we advocate not allowing bets that put the opponent in a pot-committed situation if he calls.

- In theory, the players could go back and forth several times within a betting round. However, such a sequence rarely occurs in practice. The most common sequences involve just one or two bets. In order to

[1]Of course an abstraction of the playing cards is still necessary in models of limit Texas Hold'em intended for equilibrium analysis.

[2]A bet is considered a *value bet* if the player placing the bet has a strong hand and aims to bet in a way that will entice the opponent into calling the bet. This increases the size of the pot, thus increasing the amount that the player placing the bet will likely win.

[3]*Pot odds* is the ratio of the current size of the pot to the current amount that a player needs to call. They are often used by human players as a guide for making decisions of whether to call or fold.

keep the betting model small, we advocate a cap of three bets within a betting round.[4]

Taking all of the above considerations into account, we designed our betting model to allow for the following actions:

1. The players always have the option of going *all-in.*
2. When no bets have been placed within a betting round, the actions available to the acting player are *check*, *bet* half the pot, *bet* the pot, or go *all-in.*[5]
3. After a bet has been placed within a betting round, the actions available to the acting player are *fold*, *call*, *bet* the pot, or go *all-in.*
4. If at any point a bet of a certain size would commit more than half of a player's stack, that particular bet is removed from the betting model.
5. At most three bets (of any size) are allowed within any betting round.

The above model could most likely be improved further, particularly with the incorporation of a much larger body of domain knowledge. However, since our research agenda is that of designing game-independent solving techniques, we avoid that approach where possible. We propose as future research a more systematic automated approach to designing betting abstractions—and more generally, for discretizing action spaces in games.

4.2 Reverse mapping

Once the betting model has been specified and an equilibrium analysis has been performed on the game model (as described in Section 6), there still remains the question of how actions in the real game are mapped into actions in the abstracted game. For example, if the betting model contains half-pot bets and pot bets, how do we handle the situation when the opponent makes a bet of three-fourths of the pot? In this section we discuss several issues that arise in developing this *reverse mapping*, and discuss the different design decisions we made for *Tartanian.*

One idea is to map actions to the nearest possible action in terms of amount contributed to the pot. For example, if the betting model contains half-pot bets and pot bets, and the opponent bets four-fifths of the pot, we can treat this (in our model) as a pot-size bet. (Ties could be broken arbitrarily.) However, this mapping can be subject to exploitation. For example, consider the actions available to the small blind player after the initial blinds have been posted. At this point, the small blind has contributed one chip to the pot and the big blind has contributed two chips. According to our betting model, the options available to the small blind are to fold (adding zero chips), call (one chip), half-pot bet (three chips), pot bet (five chips), or all-in (999 chips). Clearly, there is a huge gap between contributing five chips and 999 chips. Suppose that the opponent in this situation actually contributes 500 chips. In absolute distance, this is closer to the pot bet than it is to the all-in bet. However, the bet is so large relative to the pot that for all practical purposes it would be more suitably treated as an all-in bet. If the opponent knows that we treat it as a five-chip bet, he can exploit us by using the 500-chip bet because we would call that with hands that are too weak.[6]

Another possible way of addressing the interpolation problem would be to use randomization.[7] Suppose an action is played where a player contributes c chips to the pot. Suppose that the closest two actions in the betting model correspond to actions where the player contributes d_1 and d_2 chips, with $d_1 < c < d_2$. We could then randomly select the first action in the betting model with probability $p = 1 - \frac{c-d_1}{d_2-d_1}$ and select the second action with probability $1 - p$. This would help mitigate the above-mentioned example where a 500-chip bet is treated as a pot-size bet. However, this would still result in it being treated as a pot-size bet about half of the time.

Instead of using the absolute distances between bets for determining which actions are "closest", we instead advocate using a relative distance. Again considering the situation where the opponent contributes c chips and the two surrounding actions in the model contribute d_1 and d_2 chips, with $d_1 < c < d_2$, we would then compare the quantities $\frac{c}{d_1}$ and $\frac{d_2}{c}$ and choose the action corresponding to the smallest quantity. In the example where the small blind contributes 500 chips in his first action, the two quantities would be $\frac{500}{5} = 100$ versus $\frac{999}{500} = 1.998$. Hence, according to this metric, our reverse mapping would choose the all-in bet as desired.

5. AUTOMATED CARD ABSTRACTION

As discussed in the previous section, the size of the unabstracted game tree for no-limit heads-up Texas Hold'em is approximately 10^{71} nodes. In addition to abstracting the players' betting actions, it is also necessary to perform abstraction on the game's random actions, *i.e.,* the dealing of the cards. Fortunately, the topic of automated abstraction of these signal spaces has received significant attention in the recent literature. We leverage these existing techniques in our player.

We developed the *GameShrink* algorithm [9] for performing automated abstraction in imperfect information games. This algorithm was based on *ordered game isomorphisms*, a formalization capturing the intuitive notion of strategic symmetries between different nodes in the game tree. For example, in Texas Hold'em, being dealt the hole cards $A\spadesuit A\clubsuit$ ver-

[4]After we developed our betting model, we observed that allowing an unlimited number of bets (in conjunction with a minimum bet size of half the pot) only increases the size of the betting model by 15%. Therefore, in future versions of our player, we plan to relax this constraint.

[5]Due to a bug in the equilibrium-finding code that was discovered less than one week before the 2007 AAAI Computer Poker Competition, we were unable to incorporate the half-pot betting action in that model. Thus, the experimental results presented in Section 7 do not reflect the full capabilities of our player. Since it is reasonable to expect that the presence of an additional action could only improve the performance of an agent (the agent always has the option of not taking that action), we expect that the experimental results in this paper are a pessimistic representation of *Tartanian's* performance.

[6]The experimental results in Section 7 reflect the performance of a version of our player that used this simplistic mapping rule. In that section we discuss situations in which this mapping led to weak play.

[7]A similar randomization technique has been proposed previously for mitigating this problem [1].

sus $A\diamondsuit A\heartsuit$ results in a strategically identical situation. The *GameShrink* algorithm captures such strategic symmetries and leads to a smaller game on which the equilibrium analysis can be performed. The equilibrium in the abstracted game corresponds exactly to an equilibrium in the original game, so the abstraction is lossless. We used this technique to solve Rhode Island Hold'em [20], a simplified version of limit Texas Hold'em. A simple modification to the basic *GameShrink* algorithm yields a lossy version, which can be used on games where the losslessly abstracted game is still too large to solve. We used that lossy version to construct the limit Texas Hold'em player *GS1* [7].

Subsequently, we observed several drawbacks to that lossy version of *GameShrink*; this led to the development of an automated abstraction algorithm based on k-means clustering and integer programming [8]. The basic idea is to perform a top-down pass of the card tree (a tree data structure that contains a path for every possible deal of the cards). At each level of the card tree, hands are abstracted into *buckets* of similar hands, with the additional constraint that children of different parents cannot be in the same bucket. At each level, for the children of each parent in turn, k-means clustering (for various values of k) is used to cluster the children. Then, an integer program is solved to allocate how many (k) children each parent gets to have, under the constraint that the total number of children at the level does not exceed a threshold that is pre-specified based on how fine-grained an abstraction one wants. Then the process moves to the next deeper level in the tree. We used this technique to develop the limit Texas Hold'em player *GS2*.

The metric we initially proposed for use in the k-means clustering and integer programming approach was based simply on the winning probability of a hand (based on a uniform roll-out of the remaining cards). However, this does not take into account the (positive and negative) *potential* of hands. Furthermore, a hand's strength becomes apparent over time, and the strengths of different hands are revealed via different *paths*. We developed a potential-aware metric to take this into account. We further improved the basic top-down algorithm by making multiple passes over the card tree in order to refine the scope of analysis, and *GS3*, a limit Texas Hold'em player, was developed based on this abstraction algorithm [10].

In *Tartanian*, we use the same automated abstraction algorithm as we used for *GS3*. The number of buckets we allow for each level are the inputs to the algorithm. We used 10 buckets for the first round, 150 for the second round, 750 for the third round, and 3750 for the fourth round. These numbers were chosen based on estimates of the size of problem that our equilibrium-finding algorithm, described below, could solve to high accuracy in a reasonable amount of time.

Once the discretized betting model and reverse mapping have been designed, and the card abstraction has been computed, we are ready to perform the final step, equilibrium computation. We will describe that next.

6. EQUILIBRIUM COMPUTATION

The Nash equilibrium problem for two-player zero-sum sequential games of imperfect information with perfect recall can be formulated using the sequence form representation [19, 14, 22] as the following saddle-point problem:

$$\max_{\mathbf{x}\in Q_1} \min_{\mathbf{y}\in Q_2} \mathbf{x}^{\mathrm{T}} A\mathbf{y} = \min_{\mathbf{y}\in Q_2} \max_{\mathbf{x}\in Q_1} \mathbf{x}^{\mathrm{T}} A\mathbf{y}. \tag{1}$$

In this formulation, $\mathbf{x}$ is player 1's strategy and $\mathbf{y}$ is player 2's strategy. The bilinear term $\mathbf{x}^{\mathrm{T}}A\mathbf{y}$ is the payoff that player 1 receives (player 2 receives the negative of this amount) when the players play the strategies $\mathbf{x}$ and $\mathbf{y}$. The strategy spaces are represented by $Q_i \subseteq \mathbb{R}^{|S_i|}$, where S_i is the set of sequences of moves of player i, and Q_i is the *set of realization plans* of player i. Thus $\mathbf{x}$ ($\mathbf{y}$) encodes probability distributions over actions at each point in the game where player 1 (2) acts. The set Q_i has an explicit linear description of the form $\{z \geq 0 : Ez = \mathbf{e}\}$. Consequently, problem (1) can be modeled as a linear program (see [22] for details).

The linear programs that result from this formulation have size linear in the size of the game tree. Thus, in principle, these linear programs can be solved using any algorithm for linear programming such as the simplex or interior-point methods. For relatively small games, that suffices [15, 20, 3, 9]. However, for many games the size of the game tree and the corresponding linear program is enormous and thus intractable. Recently, there has been interest in finding ϵ-*equilibria* using alternative algorithms. Formally, we want to find strategies $\mathbf{x}^*$ and $\mathbf{y}^*$ such that

$$\max_{\mathbf{x}\in Q_1} \mathbf{x}^{\mathrm{T}} A\mathbf{y}^* - \min_{\mathbf{y}\in Q_2} (\mathbf{x}^*)^{\mathrm{T}} A\mathbf{y} \leq \epsilon. \tag{2}$$

Nesterov's *excessive gap technique* (EGT) [18], an algorithm for solving certain non-smooth convex optimization problems, has been specialized to finding ϵ-equilibria in two-person sequential games [12]. We further improved that basic algorithm via 1) the introduction of heuristics that speed up the algorithm by an order of magnitude while maintaining the theoretical convergence guarantees of the algorithm, and 2) incorporating a highly scalable, highly parallelizeable implementation of the matrix-vector product operation that consumes the bulk of the computation time [6].

Since the matrix-vector product operation is so performance-critical, having custom software developed specifically for this purpose is important for the overall performance of the algorithm. In Section 6.1 we discuss tools we have developed for automatically generating the C++ source code for computing the required matrix-vector product based on an XML description of the game.

6.1 Automatic C++ source code generation for the matrix-vector product

As mentioned above, the most intensive portion of the EGT algorithm is in computing matrix-vector products $\mathbf{x}^{\mathrm{T}}A$ and $A\mathbf{y}$. For small games, or games where the structure of the strategy space is quite simple, the source code for computing this product could be written by hand. For larger, more complicated games, the necessary algorithms for computing the matrix-vector product would in turn be more complicated. Developing this code by hand would be a tedious, difficult task—and it would have to be carried out anew for each game and for each betting discretization.

We can see two alternatives for handling this problem. The first, and most obvious, is to have a tree-like representation of the betting model built in memory. This tree could be built from a description of the game. Then, when the matrix-vector product operation is needed, a general algorithm could traverse this tree structure, performing the necessary computations. However, the performance of this algorithm would suffer some since there is the overhead of traversing the tree.

```
<bml name='CustomBetting'>
      <round number='1'>
            <decisions>
                  <decision player='2' sequence='' parent='-1'>
                        <action name='F'  number='0' />
                        <action name='C'  number='1' />
                        <action name='R1' number='2' />
                        <action name='A'  number='3' />
                  </decision>
                  <decision player='1' sequence='C' parent='-1'>
                        <action name='k'  number='0' />
                        <action name='r1' number='1' />
                        <action name='a'  number='2' />
                  </decision>
                  <decision player='2' sequence='Ca' parent='1'>
                        <action name='F'  number='19' />
                        <action name='C'  number='20' />
                  </decision>
                  <decision player='1' sequence='A' parent='-1'>
                        <action name='f'  number='32' />
                        <action name='c'  number='33' />
                  </decision>
                  <!-- other decisions omitted... -->
            </decisions>
            <leaves>
                  <leaf seq1='2'  seq2='19' type='fold'     sequence='CaF' payoff='2.0' />
                  <leaf seq1='2'  seq2='20' type='showdown' sequence='CaC' potshare='1000.0' />
                  <leaf seq1='32' seq2='3'  type='fold'     sequence='Af'  payoff='-2.0' />
                  <leaf seq1='33' seq2='3'  type='showdown' sequence='Ac'  potshare='1000.0' />
                  <!-- other leaves omitted... -->
            </leaves>
      </round>
      <!-- other rounds omitted... -->
</bml>
```

Listing 1: A snippet of the BML for our first-round betting model. The `r1` action indicates a pot-size bet.

A second approach, which offers better performance, is to generate the C++ source code automatically for the game at hand. This eliminates the need for a tree-like representation of the betting model. Instead, for each node of the tree we simply have one line of source code which performs the necessary operation.

For this approach to work, we need some way of specifying a betting model. We accomplish this with our *Betting Model Language (BML)*, an XML-based description of all possible betting models for no-limit Texas Hold'em. Listing 1 contains a snippet of the BML file used by our player.

The BML file consists of a `<round>` section for each betting round (only parts of the first betting round are shown in Listing 1). Within each `<round>`, there are `<decision>` entries and `<leaf>` entries. The `<decision>` entries specify the actions available to each player at any stage of the game, as well as specifying certain indices (given via the `number` key) which are used by the equilibrium-finding algorithm for accessing appropriate entries in the strategy vectors.

The `<leaf>` entries encode the payoffs that occur at terminal sequences of the game. When a `<leaf>` has `type` equal to `'fold'`, then it contains a `payoff` value which specifies the payoff to player 1 in that case. Similarly, when a `<leaf>` has `type` equal to `'showdown'`, then it contains a `potshare` value which specifies the amount of chips that each player has contributed to the pot so far. (Of course, the actual payoffs in showdown leaves also depend on the players' cards.)

Listing 2 contains a snippet of C++ code produced by our software for translating BML into C++. As can be seen, the code is very efficient as each leaf of the game tree is processed with only a few instructions in one line of code each.

7. EXPERIMENTAL RESULTS

Tartanian participated in the no-limit category of the 2007 AAAI Computer Poker Competition. Each of the 10 entries played head-to-head matches against the other 9 players in Doyle's no-limit Texas Hold'em poker. Each pair of competitors faced off in 20 *duplicate matches* of 1000 hands each. A duplicate match is one in which every hand is played twice with the same cards, but the players are switched. (Of course, the players' memories are reset so that they do not remember the hands the second time they are played.) This is to mitigate the element of luck inherent in poker since if one player gets a particularly lucky hand, then that will be offset by giving the other player that same good hand.

Table 1 summarizes the results.[8] *Tartanian* placed second out of the ten entries. The ranking system used in this competition was *instant runoff bankroll*. In that system, the total number of chips won or lost by each program is

[8]The full competition results are available on the web at `http://www.cs.ualberta.ca/~pokert/`.

```
void TexasMatrixNoLimit::multvec_helper_round1_fold
     (Vec& x, Vec& b, const unsigned int i, const unsigned int j, const double prob) {
        b[i +  2] += x[j + 19] * prob *  2.0; // CaF
        b[i + 32] += x[j +  3] * prob * -2.0; // Af
        /* other payoffs omitted... */
}

void TexasMatrixNoLimit::multvec_helper_round1_showdown
     (Vec& x, Vec& b, const unsigned int i, const unsigned int j, const double prob, const double win) {
       b[i +  2] += x[j + 20] * win * prob * 1000.0; // CaC
       b[i + 33] += x[j +  3] * win * prob * 1000.0; // Ac
       /* other payoffs omitted... */
}
```

Listing 2: A snippet of the automatically-generated C++ code for computing the matrix-vector product.

compared to all of the others. The entrant that loses the most is eliminated and finishes in last place; this ranking process iterates until there is a single winner.

Once the ranking process had only three remaining entries (*Tartanian*, *BluffBot*, and *Hyperborean*), 280 more duplicates matches were held in order to obtain statistical significance. Based on this total of 300 duplicate matches, *Tartanian* beat *Hyperborean* by 0.133 ± 0.039 small bets, but lost to *BluffBot* by 0.267 ± 0.032.

An interesting phenomenon was that *Tartanian's* performance against *PokeMinn* was significantly worse than against any other opponent—despite the fact that *PokeMinn* fared poorly in the competition overall. We manually investigated the hand histories of this match-up and observed that *PokeMinn* had a tendency to place bets that were particularly ill-suited to our discretized betting model. For example, a common bet made by *PokeMinn* was putting in 144 chips pre-flop. As mentioned in Footnote 6, the version of our player in the competition was using the simplistic absolute rounding mapping and so it would treat this as a pot-size bet. However, it actually makes much more sense to treat this as an all-in bet since it is so large relative to the size of the pot. We expect that our improved rounding method based on relative distances, described in Section 4.2, will appropriately handle this.

8. CONCLUSIONS AND FUTURE RESEARCH

We presented *Tartanian*, a game theory-based player for heads-up no-limit Texas Hold'em poker. To handle the huge strategy space of no-limit poker, we created a discretized betting model that attempts to retain the most important actions in the game. This also raised the need for a reverse model. Second, as in some prior approaches to game theory-based poker players, we employed automated abstraction for shrinking the size of the game tree based on identifying strategically similar card situations. Third, we presented a new technique for automatically generating the performance-critical portion of equilibrium-finding code based on data describing the abstracted game. The resulting player is competitive with the best existing computer opponents.

Throughout, we made many design decisions. In this research so far, we have made educated guesses about what good answers are to the many questions. In particular, the design of the discretized betting model (and reverse model) and the choice of the number of buckets for each level of the card abstraction were largely based on our own understanding of the problem. In the future, we would like to automate this decision-making process (and hopefully get better answers). Some concrete paths along these lines would be the development of an automated discretization algorithm for the betting model. This could attempt to incorporate a metric for the amount that is lost by eliminating certain strategies, and use this to guide its decisions as to what strategies are eliminated from the model. Another research direction involves developing a better understanding of the tradeoffs between abstraction size and solution quality. We would also like to understand in a more principled way how to set the number of buckets for the different levels of the abstracted card tree.

9. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under ITR grant IIS-0427858. We also acknowledge Intel Corporation and IBM for their gifts.

10. REFERENCES

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

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

[3] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 661–668, Acapulco, Mexico, 2003. Morgan Kaufmann.

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

[5] S. Ganzfried and T. Sandholm. Computing an approximate jam/fold equilibrium for 3-agent no-limit Texas hold'em tournaments. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Estoril, Portugal, 2008.

[6] A. Gilpin, S. Hoda, J. Peña, and T. Sandholm. Gradient-based algorithms for finding Nash equilibria

	BB	TART	HYP	SR	G1	G2	MIL	MB1	PM	MB2	TOTAL
Bluffbot	–	0.166	0.237	0.576	2.093	2.885	3.437	0.475	1.848	2.471	1.577
(BB)	–	±0.074	±0.080	±0.102	±0.346	±0.306	±0.243	±0.153	±0.252	±0.138	±0.101
Tartanian	−0.166	–	−0.079	0.503	3.161	0.124	1.875	4.204	−42.055	5.016	−3.406
(TART)	±0.074	–	±0.148	±0.148	±0.597	±0.467	±0.377	±0.323	±0.606	±0.192	±0.17
Hyperborean	−0.237	0.079	–	−0.048	6.657	5.455	6.795	8.697	14.051	22.116	7.063
(HYP)	±0.080	±0.148	–	±0.171	±0.493	±0.483	±0.551	±0.424	±0.723	±0.589	±0.181
SlideRule	−0.576	−0.503	0.048	–	11.596	9.73	10.337	10.387	15.637	10.791	7.494
(SR)	±0.102	±0.148	±0.171	–	±0.295	±0.359	±0.595	±0.523	±0.685	±0.405	±0.182
Gomel1	−2.093	−3.161	−6.657	−11.596	–	3.184	8.372	11.450	62.389	52.325	12.690
(G1)	±0.346	±0.597	±0.493	±0.295	–	±0.287	±0.705	±0.854	±1.264	±0.599	±0.218
Gomel2	−2.885	−0.124	−5.455	−9.730	−3.184	–	15.078	11.907	58.985	40.256	11.650
(G2)	±0.306	±0.467	±0.483	±0.359	±0.287	–	±0.830	±0.848	±0.892	±0.610	±0.211
Milano	−3.437	−1.875	−6.795	−10.337	−8.372	−15.078	–	5.741	12.719	27.040	−0.044
(MIL)	±0.243	±0.377	±0.551	±0.595	±0.705	±0.830	–	±0.675	±1.124	±0.736	±0.202
Manitoba1	−0.475	−4.204	−8.697	−10.387	−11.450	−11.907	−5.741	–	18.817	50.677	1.848
(MB1)	±0.153	±0.323	±0.424	±0.523	±0.854	±0.848	±0.675	–	±1.236	±0.910	±0.241
PokeMinn	−1.848	42.055	−14.051	−15.637	−62.389	−58.985	−12.719	−18.817	–	34.299	−12.01
(PM)	±0.252	±0.606	±0.723	±0.685	±1.264	±0.892	±1.124	±1.236	–	±1.370	±0.411
Manitoba2	−2.471	−5.016	−22.116	−10.791	−52.325	−40.256	−27.04	−50.677	−34.299	–	−27.221
(MB2)	±0.138	±0.192	±0.589	±0.405	±0.599	±0.610	±0.736	±0.910	±1.370	–	±0.358

Table 1: Results from the 2007 AAAI Computer Poker Competition. The players are listed in the order in which they placed in that competition. Each cell contains the average number of chips won by the player in the corresponding row against the player in the corresponding column, as well as the standard deviation. The numbers in the table reflect 20 pairwise matches each; in the AAAI competition a further 280 matches were conducted between each pair of the three top-ranked entries in order to get statistical significance, and *Tartanian* finished second.

in extensive form games. In *3rd International Workshop on Internet and Network Economics (WINE)*, San Diego, CA, 2007.

[7] A. Gilpin and T. Sandholm. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Boston, MA, 2006. AAAI Press.

[8] A. Gilpin and T. Sandholm. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, Honolulu, HI, 2007.

[9] A. Gilpin and T. Sandholm. Lossless abstraction of imperfect information games. *Journal of the ACM*, 54(5), 2007.

[10] A. Gilpin, T. Sandholm, and T. B. Sørensen. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 50–57, Vancouver, Canada, 2007. AAAI Press.

[11] D. Harrington and B. Robertie. *Harrington on Hold'em Expert Strategy for No-Limit Tournaments, Vol. 1: Strategic Play.* Two Plus Two, 2004.

[12] S. Hoda, A. Gilpin, and J. Peña. A gradient-based approach for computing Nash equilibria of large sequential games. Available at http://www.optimization-online.org/, 2007.

[13] M. Johanson, M. Zinkevich, and M. Bowling. Computing robust counter-strategies. In *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Vancouver, BC, 2007.

[14] D. Koller and N. Megiddo. The complexity of two-person zero-sum games in extensive form. *Games and Economic Behavior*, 4(4):528–552, Oct. 1992.

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

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

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

[18] Y. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM Journal of Optimization*, 16(1):235–249, 2005.

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

[20] J. Shi and M. Littman. Abstraction methods for game theoretic poker. In *CG '00: Revised Papers from the Second International Conference on Computers and Games*, pages 333–345, 2002. Springer-Verlag.

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

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

[23] M. Zinkevich, M. Bowling, and N. Burch. A new algorithm for generating equilibria in massive zero-sum games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Vancouver, Canada, 2007.

[24] M. Zinkevich, M. Bowling, M. Johanson, and C. Piccione. Regret minimization in games with incomplete information. In *Proceedings of the 23rd Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, Vancouver, BC, 2007.