Using Sliding Windows to Generate Action Abstractions in Extensive-Form Games

John Hawkin and **Robert C. Holte** and **Duane Szafron**
{hawkin, holte}@cs.ualberta.ca, dszafron@ualberta.ca
Department of Computing Science
University of Alberta
Edmonton, AB, Canada T6G2E8

Abstract

In extensive-form games with a large number of actions, careful abstraction of the action space is critically important to performance. In this paper we extend previous work on action abstraction using no-limit poker games as our test domains. We show that in such games it is no longer necessary to choose, a priori, one specific range of possible bet sizes. We introduce an algorithm that adjusts the range of bet sizes considered for each bet individually in an iterative fashion. This flexibility results in a substantially improved game value in no-limit Leduc poker. When applied to no-limit Texas Hold'em our algorithm produces an action abstraction that is about one third the size of a state of the art hand-crafted action abstraction, yet has a better overall game value.

Introduction

Our objective is to develop techniques for creating agents that can make better decisions than expert humans in complex stochastic, imperfect information multi-agent decision domains. Multi-agent means that there at least two agents. Such decision problems can often be posed as extensive-form games. An extensive-form game is represented by a tree, where each terminal node represents the utility or payoff for each agent. Each interior node represents one of the agents with the edges leaving that node representing the potential actions of that agent. To play an extensive-form game each agent must have a strategy. A strategy is a probability distribution over all legal actions available to that agent for every possible history of game actions.

In this paper, we use poker as a testbed for developing and validating techniques that can be used to create better agents. One common approach to finding good agent strategies is to use an ϵ-Nash equilibrium solver. A strategy profile is a set of strategies for each player. An ϵ-Nash equilibrium solver generates strategy profiles where no single agent can increase utility by more than ϵ by unilaterally changing its strategy. Unfortunately, ϵ-Nash equilibrium solvers cannot generate solutions for very large games, so abstraction is often used to create smaller games. The solution to the smaller abstracted game is then used to play the original game. A good abstraction technique is one that maintains important aspects of the full game, while reducing the overall game tree size. State-space abstraction combines similar nodes by applying a metric to chance outcomes. For example, in Texas Hold'em poker, the 169 different (up to suit isomorphisms) two-card pre-flop hands may be combined into a number of buckets based on the hand strength metric (Waugh 2009), which combines hands such as Ace-Ace and King-King into the same bucket. State-space abstraction for Texas Hold'em poker has been studied in great detail over the last few years (Gilpin and Sandholm 2006; 2007; Gilpin, Sandholm, and Sorensen 2007; Waugh et al. 2009a). In the two player limit variation of Texas Hold'em, each agent selects from a fixed number of legal betting actions at each decision node with a maximum of three decisions at each node: fold, call or bet a fixed amount. Current state abstraction techniques are sufficient to reduce the 10^{18} game states to 10^{14} states so that state of the art ϵ-Nash equilibrium solvers can be applied (Johanson 2007). The resulting solution strategies are competitive with human experts when used in the unabstracted game (Johanson 2007).

In no-limit Texas Hold'em, however, there are many actions: fold, call or bet any number between a minimum and the remaining stack-size. Besides selecting the bet action, an agent must select a bet size from a large discrete value space. For two player no-limit Texas Hold'em with 500 big blind stacks, there are 10^{71} states (Gilpin, Sandholm, and Sorensen 2008). State-abstraction alone does not make this domain tractable - we must also abstract the action space by removing actions, ideally leaving only those actions that are the most essential for good strategies.

The major challenges of action abstraction generate two research problems. First, we must determine which actions to remove from the game. Second, we must determine how to act when the opponent makes an action that is not in our action abstraction. The latter problem, known as the translation problem, has been examined by Schnizlein (Schnizlein 2009; Schnizlein, Bowling, and Szafron 2009). The former problem was studied in our previous paper (Hawkin, Holte, and Szafron 2011). In this work, a transformation is introduced that can be applied to domains where agents make actions and choose parameter values associated with these actions. Examples of such domains are trading games and no-limit poker games. In the case of no-limit poker games, the action in question is a bet, and the associated parame-

ter is the size of the bet. In addition to this transformation, this paper introduces an algorithm to minimize regret in the transformed game. After finding a strategy that minimizes regret, the strategy can be mapped to an action abstraction of the no-limit game. This paper showed that such abstractions select the same bet sizes as Nash-equilibrium strategies in small poker games and that this technique produces better strategies than ϵ-Nash equilibrium strategies computed from manually selected action abstractions in no-limit Leduc poker.

The algorithm introduced in our previous work (Hawkin, Holte, and Szafron 2011) defines a bet size range for each bet action and selects a bet size from that range. The same range was picked for all bets, based on expert knowledge, and the problem of selecting good ranges was ignored. We extend this work here by considering the choice of ranges in detail. First, we show that range choice is essential in creating a good action abstraction. Second, we show that selecting appropriate ranges is a non-trivial problem: there are practical reasons why large ranges cannot be used, and in general no single small range is appropriate. Third, we demonstrate a method of automatically selecting ranges using multiple short runs of a regret-minimizing algorithm similar to the one used in our previous paper (Hawkin, Holte, and Szafron 2011). This range-selection technique generates better abstractions than our previous algorithm in no-limit Leduc poker, while using ranges that are 3.5 times smaller. Fourth, we show that our action abstraction performs better than state of the art hand-crafted action abstractions used in two player no-limit Texas Hold'em agents that are about three times larger than the abstractions generated by our technique.

Rules of heads up no-limit poker

We apply our techniques to two player no-limit poker games. Each player starts with a fixed number of chips known as a stack. There are two variations. In one variation each player puts the same number of chips in the pot, called an ante. In the other variation player one puts chips in the pot (the big blind) and player two puts half as many chips in the pot (the small blind). In the ante variation player one acts first in each round, while in the blinds variation player two acts first in the initial betting round and player one acts first in all subsequent rounds. At least one private card is dealt to each player, sometimes more depending on the poker variant being played.

Once the ante or blinds are posted, a betting round occurs. During this betting round a player may fold (surrender the pot), check/call (match any outstanding bet, referred to as check if there is no bet to match), or bet/raise (match any outstanding bet and increase it). If the blind variation is used, the small blind acts first and faces an outstanding bet, which is the size of the small blind. If the ante variation is used, the first player to act faces no outstanding bet. The betting round continues until: one player folds, both players check or one player bets and then the other player calls. The bet size is selected by the betting player, with minimum size equal to the maximum of the big blind and the last bet increase made during the current round. If a player bets all remaining chips, it is called an all-in bet. After the first betting round, a number (depending on the poker variant) of community cards are revealed and another betting round occurs. Further community cards and betting rounds can occur, depending on the poker variant. If no player folds before the end of the last betting round, each player makes a poker hand using their own cards and the community cards and the highest ranked poker hand wins the pot. If there is a tie, the pot is split.

Solving for bet sizes

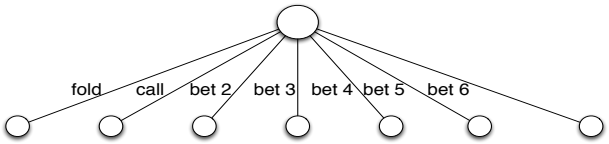
Our previous work (Hawkin, Holte, and Szafron 2011) introduces a bet-sizing algorithm for generating action abstractions in no-limit poker. In that work we outline a transformation that is applied to two player no-limit poker that creates a new game with extra agents, which we call the bet-sizing game throughout this paper. A regret minimization algorithm is then applied in order to compute strategy profiles for the bet-sizing game. The strategies of the extra agents can then be mapped to bet sizes for the no-limit game, creating an action abstraction.

The bet-sizing game retains useful properties of the bigger game while reducing memory requirements. The transformation is defined as follows. At every tree node where betting (or raising) is a legal action, all bet actions, except all-in, are replaced by a single bet action with no amount specified and all child nodes are coalesced into a single node. In addition, a new subtree with three nodes is inserted between the betting node and the coalesced node. The new subtree belongs to a new agent called a "bet-sizing" agent, who has two actions: low bet, denoted L, and high bet, denoted H. This transformation is illustrated in Figure 1, which is adapted from our previous work (Hawkin, Holte, and Szafron 2011). The "bet-sizing" agent privately chooses either the low or high bet action, both of which lead back to the coalesced node. No other agent knows which action was taken. A new bet-sizing agent i is introduced at each decision point in the game, so the values of L_i and H_i can be different for every bet-sizing agent i, with the constraint that $L_i < H_i$. Note that L_i and H_i are expressed as pot fractions: $L = 0.75$ means a bet size of three quarters of the current pot. We will refer to the two agents that have fold, call, bet and all-in actions as players one and two, or the "main players", and the extra agents as "bet-sizing" agents.

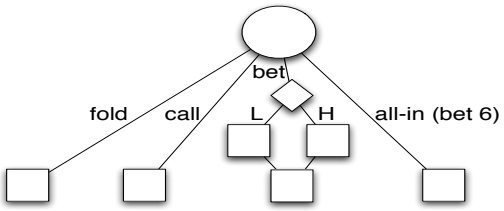
Although the bet-sizing transformation can be applied to bets for both main players, it could be applied to one player's bets, while the other player uses a fixed betting abstraction. Previously we picked a single L value and a single H value for the entire tree. This paper addresses the question of how to pick values of L and H that allow us to generate action abstractions for one player which maximize value against a particular, pre-determined action abstraction of the other player. The effective bet size is defined as (Hawkin, Holte, and Szafron 2011)

$$B(P(H)_i) = (1 - P(H)_i)L_i + P(H)_i H_i. \qquad (1)$$

This is the expected value of the bet size made by bet-sizing agent i using strategy $P(H)_i$.



(a) Regular no-limit game



(b) Bet sizing game

Figure 1: Decision point in a no-limit game, and the bet-sizing game version of the same decision point.

	$0.80 - 0.82$		$0.82 - 0.98$		$0.98 - 1.0$	
Length	All Bets	Top 10 Bets	All Bets	Top 10 Bets	All Bets	Top 10 Bets
1m	152	8	8	2	0	0
4m	151	8	9	2	0	0
6m	150	8	9	1	1	1
8m	145	8	13	0	2	2
10m	142	8	16	0	2	2
20m	138	8	20	0	2	2
100m	126	9	31	0	3	1
200m	126	9	28	0	6	1

Table 1: Bet sizes after different length runs.

The case for variable ranges

In this section we show that when the algorithm introduced in our previous paper (Hawkin, Holte, and Szafron 2011) is applied to large games such as no-limit Texas Hold'em, there is much value to be gained from using different L and H values at different points in the tree.

We applied the bet-sizing transformation, with $L = 0.8$ and $H = 1$ (suggested by experts) for all bets, to a small card abstraction of no-limit Texas Hold'em with 200 big blind stacks. We modified the algorithm from our previous work (Hawkin, Holte, and Szafron 2011) and applied it to the first player, obtaining a new agent whose game value exceeded the best game values of fold call pot all-in agents by $\approx 17\%$ percent after 200 million iterations. The second, fourth and sixth columns of Table 1 show the number of bet sizes in the given range after different length runs.

Table 1 shows that the vast majority of bet sizes are < 0.82, which is surprising, given that previous expert knowledge dictated that if only a single bet size is used everywhere, it should be pot sized. So many bet sizes being close to the low boundary after only 1 million iterations suggests we should move the range lower. It's possible, however, that the bets with these small values have little effect on the game value (for example those betting decisions could be reached with extremely small probability). To test if this was the case we developed a metric, $R^T_{i,|\cdot|}$, that ranks bet importance. The $R^T_{i,|\cdot|}$ value measures the utility that could be gained, after T iterations, by moving bet i.[1] The third, fifth and seventh columns of Table 1 show the bet sizes of the 10 bets with the highest $R^T_{i,|\cdot|}$ values. We can see that while 8 of these bets were very close to $L = 0.8$, two bet sizes moved towards $H = 1$ over the first 8 million iterations. These two bets had the highest and third highest $R^T_{i,|\cdot|}$ values.

The game value of the abstractions created changed significantly during the first 8 million iterations, while these important bets were moving. During the final 192 million iterations, however, the game value stayed relatively constant. This result, coupled with the fact that there are important bets at both ends of our range, suggests that allowing some of the bets to go lower and others to go higher may result in increased game value. The simplest way to achieve this goal is to continue using static ranges, but make them larger. Unfortunately, as we explain in the next subsection, there are significant disadvantages to using large ranges.

Tree creation - the small range constraint

When creating the game tree for the bet-sizing game, there are two issues:

- What ranges do we use?
- How many bets do we allow in each bet-sequence (what is the depth of the game tree)?

Consider an abstraction of a no-limit poker game with initial stacks of 15 big blinds, where only half-pot and pot bets are legal. Figure 2 shows the betting tree. Each node is labeled by the pot size in big blinds after the agent to act has put in chips to call the outstanding bet, but before adding the raise chips. For example, the root node (player two) is labeled 2 since when player two is about to raise, player one has already put in the big blind and player two has put in both the small blind (0.5 big blinds) and another 0.5 big blinds to call (as a prerequisite to making the raise action) for a total pot of 2 big blinds. The right child node contains 6 big blinds, since if player two raises by the pot (2 big blinds), the pot would then contain: the current pot (2), plus the raise (2), plus the amount that player one would need to add (2) before adding the next raise amount.

In this game the players can make three half-pot bets, two pot bets, or two half-pot bets and one pot bet, before an additional bet would require more than the 15 big blinds in the initial stack. For example, to follow the node labeled 16 by a half-pot bet would require player one to add 8 chips to the pot after having put $16/2 = 8$ chips into the pot, requiring an initial stack of 16 big blinds. If we transform this game to a bet-sizing game with $L = 0.5$ and $H = 1$ everywhere, do we construct a game tree with depth 2 bets or 3 bets?

Figure 3 shows the betting tree in the bet-sizing game, where the bold edges indicate the third bet in any betting sequence. The dotted ovals represent information sets, since after the bet-sizing agent makes their private action, none

[1] See the appendix for more details

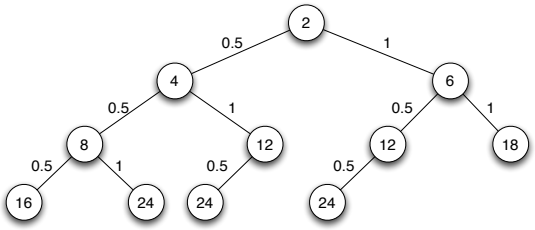


Figure 2: Betting tree for a 15 big blind stack game, with an abstraction that allows only half pot and pot bets.

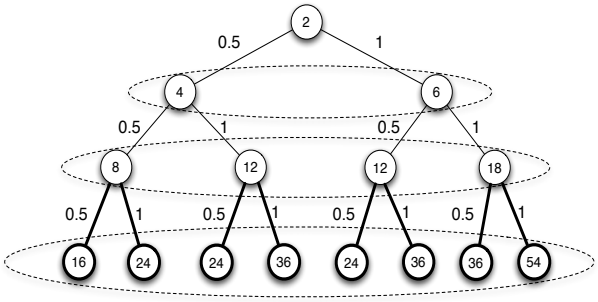


Figure 3: Betting tree for a bet-sizing game transformation of a 15 big blind stack game, allowing two bets not including the bold actions, or three bets if bold actions are included.

of the other players know the resulting pot size. Unfortunately, if we include three bet-sequences in the tree, some of the sequences are invalid. For example, three pot-size bets result in a pot size of 54 chips, where each player has contributed 27 chips, which is 12 more chips than the initial stack size. Alternatively, if we create a tree of depth 2, then some legal bets are missing from the tree. For example, it is legal to bet half-pot or pot after two half-pot bets. If these actions are missing from our tree, our strategy may not be able to take advantage of a bet action that results in a higher game utility. Unfortunately, the legality of the third bet is dependent on the size of the previous bets and this information is hidden by the information sets. The disparity between the shortest and longest legal bet-sequence size is dependent on the size of the betting range so it can be minimized by selecting small ranges, but cannot be eliminated in general. Therefore, to avoid making invalid bets we use the H value to select the tree-depth. If the algorithm selects many small bets, it may not be able to select as many small bets as are possible in the full game, unless we have some way of changing the tree size to allow more small bets. Recall that in the previous section we saw the importance of allowing bet sizes to increase or decrease beyond fixed range boundaries. Therefore, we need small ranges that can change dynamically while the strategy computation algorithm is running. In addition, if the algorithm favors a smaller bet size, we need a mechanism for reducing both L and H so that the tree size gets larger (due to a smaller H) and the preferred bet size can move lower towards the new L. If the algorithm favors a larger bet size, we need to increase both L and H, which allows the preferred bet size to get larger, while decreasing the tree size.

Variable range boundaries

Table 1 suggested that our expert-informed choice of fixed $[0.8, 1]$ ranges in our Texas Hold'em strategy computation was likely suboptimal, since the bet sizes for the most important bets hit the range boundaries. Some bets should be smaller than 0.8 and others should be larger than 1. However, we showed in the previous section that large ranges $[L, H]$ cause problems due to variable sized betting sequences. Our approach is to minimize the impact of tree size variation by fixing the size of the range, while moving the two range boundaries.

Instead of running the algorithm for many iterations (200 million) with a single fixed range, we do a series of short runs (a few million iterations). We start with a single default range $[L_d, H_d]$. We initialize all ranges to this default range on the first short run, using H_d to determine how many bets are allowed in the tree. At the end of the run, we change all ranges so that the new range for a particular bet is the same size as the old range, but is centered on the bet size computed by the algorithm during that short run. Therefore, if a bet size for any bet hits a range boundary or was near to a range boundary at the end of a run, the bet size will be able to move further in that direction on the next run. If for any bet H increases to the point that it is larger than an all-in bet size, we reduce H to the all-in bet size and set L to accommodate the fixed range size. If L decreases so that it is smaller than the minimum bet size for any bet in the tree, this bet is removed from the tree. If the H values along any bet-sequence decrease enough to add a bet of size H_d to that sequence, we add a bet using the default range $[L_d, H_d]$.

Each time a short run completes, we obtain a set of bet sizes, which we use as our new betting abstraction. We measure the quality of this new abstraction in the usual game-theoretic way - we apply an ϵ-Nash equilibrium solver, such as CFR (Zinkevich et al. 2007), and obtain an ϵ-Nash equilibrium. We then obtain best responses to this strategy profile for both players, and this gives us upper and lower bounds on the game value of our new betting abstraction. A best response is a strategy for one player that maximizes that player's expected utility when played against a specific fixed strategy of the other player. To answer the question of how well our variable-range bet-sizing algorithm performs, we go through this process after every short run and plot the resulting upper and lower bound best response curves.

Algorithmic changes

The algorithm introduced in our previous paper (Hawkin, Holte, and Szafron 2011) was based on CFR, a widely used algorithm for computing ϵ-Nash equilibria in poker games (Zinkevich et al. 2007). CFR is an iterative self-play algorithm that uses regret matching to adjust probabilities every iteration. The algorithm from our previous work uses CFR

for players one and two, but for the bet-sizing agents the strategy is updated according to[2]

$$s_i^{t+1} = \begin{cases} \frac{(t-1)s_i^t + sr_i^{t+1}}{t} & \text{if } t < 10000 \\ \frac{(9999)s_i^t + sr_i^{t+1}}{10000} & \text{otherwise.} \end{cases} \quad (2)$$

where s_i^t is the effective bet size made by bet-sizing agent i on iteration t defined as $B(P(H)_i)$ from Equation 1, and sr_i^{t+1} is the effective bet size on iteration $t+1$ as computed by an unmodified CFR algorithm. In CFR, the average (not the current) action probabilities converge to an ϵ-Nash equilibrium. Therefore, for each action sequence the algorithm maintains a sum of the probability this sequence was played each iteration. Any time a player has 0% probability of reaching an information set, the addition step can be skipped for all action sequences that reach the subtree beneath that information set. This cutoff can always be used, independent of how the bet-sizing agents are updated. Therefore, we now introduce an alternative equation for updating the bet-sizing agents:

$$s_i^{t+1} = \begin{cases} s_i^t + \frac{B_i^t(sr_i^{t+1} - s_i^t)}{t} & \text{if } t < 10000 \\ s_i^t + \frac{B_i^t(sr_i^{t+1} - s_i^t)}{10000} & \text{otherwise} \end{cases} \quad (3)$$

Here B_i^t is the probability all players play to reach this information set. Equation 2 updates s_i^t for each bet sizing player on every iteration. If $B_i^t = 0$ then Equation 3 implies that $s_i^{t+1} = s_i^t$. In this case, the algorithm can make a single strategy update pass for each team of bet-sizing agents at the same time that it updates the totals used to calculate the average strategy of the corresponding main player. In tests of the algorithm, where Equation 2 was replaced by Equation 3 and a few other optimizations were made at the coding level, the results were equivalent within a few percent. However, the changes resulted in a speedup of 3 times on smaller poker games and as high as 100 times on Texas Hold'em. We used Equation 3 to generate all of the results in this paper.

Empirical results

In our previous work (Hawkin, Holte, and Szafron 2011) we applied the bet-sizing game transformation to a restricted form of no-limit Leduc poker with a cap of two bets per round and starting stacks of 400 antes. Using $L = 0.8$ and $H = 1.5$ everywhere, a betting abstraction was generated whose game value was greater than the pot-size betting abstraction by 12% and 7.7% for players one and two respectively. With a two bet per round cap and large stacks, the tree-size problem described in this paper was avoided. The betting sequence length is at most 4 bets and the starting stack is large enough to make 4 bets of 1.5 pot each.

We applied our variable-range algorithm to this game, with default values of $L = 0.8$ and $H = 1$ for all bet sizes. Using short runs of length 2, 4, 6, 8 and 10 million, our abstractions all converged within 10 short runs, beating the value of the pot-size abstraction by 21% and 9% for players one and two respectively, as compared to the fixed-range algorithm with 12% and 7.7% gains. When we used more than 10 short runs, the game value deviated by at most 1%. Our variable-range algorithm can out perform a fixed-range algorithm, even when the width (0.7) of the fixed range is larger than the width (0.2) of our variable ranges. The generated betting abstractions contained many important bet sizes outside the range $[0.8, 1.5]$ - for the 10 million iteration run of player one, 3 of the top 4 bets, as ranked by $R^T_{i,|\cdot|}$, were greater than 1.5.

We also applied our technique to an unrestricted Leduc game, with 200 big blind stacks. Using the same ranges and short run lengths, the abstractions we created improved over a betting abstraction that makes only pot sized bets by $\approx 30\%$ and $\approx 43\%$ for player one and player two respectively.

Texas Hold'em

Finally we applied our methodology to 200 big blind no-limit Texas Hold'em poker. We abstracted the state space of the game using the hand strength metric (Waugh 2009) with 5 buckets each round and perfect recall of past actions. This led to an abstraction with 5 buckets on the pre-flop, 25 on the flop, 125 on the turn and 625 on the river. While imperfect recall is often used in Texas Hold'em state abstractions (Waugh et al. 2009b), we used perfect recall to facilitate analysis, as it is intractable to calculate best responses in imperfect recall games. Again $[0.8, 1]$ was used as the starting range for all bets.

In the Texas Hold'em experiment described by Table 1 most bets were close to an edge of their range after 1 million iterations, with the 10 most important bets being close to an edge within 8 million iterations. With this result in mind we tried short runs of various lengths, from 2 to 10 million iterations. We found that while 2 million iterations was not enough, 8 and 10 million iteration runs had very similar results, with 10 being marginally better. All of the results we discuss below use short runs of length 10 million iterations.

We found that for both players, the game value of the abstractions we obtained would initially increase after each short run, eventually levelling off. We used 8 runs for player one and 6 runs for player two. The game value of these abstractions improved over a betting abstraction that makes only pot sized bets by $\approx 48\%$ and $\approx 88\%$ for player one and player two respectively.

Good no-limit agents typically use at least two bet sizes in their abstractions - usually pot and half pot. Since half pot bets are small, they increase tree depth, so the number of half pot bets is usually restricted (Schnizlein 2009; Gilpin, Sandholm, and Sorensen 2008). We created a number of betting abstractions that use unrestricted pot bets, plus one half pot bet, once per round on specific rounds. These fixed bet agents can be compared to the agents generated by our bet-sizing algorithm. The abstractions were made asymmetrically - only one player was given extra half pot options. These abstractions and our generated abstractions are listed in Table 2, along with the number of (I, a) pairs (edges in the game tree) they contain. The number of extra (I, a) pairs due to added half pot bets does not depend on which player

[2]There was an indexing error in the original equation from our previous paper (Hawkin, Holte, and Szafron 2011), which has been corrected here.

Name	Pot fractions in abstraction	# (I, a) pairs
POT	1	1,781,890
H_{PF}	1, 0.5 once on pre-flop	3,867,090
H_F	1, 0.5 once on flop	3,820,140
H_T	1, 0.5 once on turn	3,646,890
H_R	1, 0.5 once on river	3,143,140
H_{FTR}	1, 0.5 once on flop, turn and river	9,115,390
$P1_8$	Our agent, player 1, 8 runs	2,930,050
$P2_6$	Our agent, player 2, 6 runs	3,121,775

Table 2: (I, a) pairs in 5 bucket Hold'em abstractions.

has the added bets. The amount of memory used by CFR is proportional to the number of (I, a) pairs.

We used CFR to compute upper and lower bounds on the game value between a player using each abstraction in Table 2 and a fold call pot all-in player. Figure 4 shows the results for some of the abstractions that were used for player two. The Y-axis represents big blinds won by player two. The bounds are very tight - for the basic POT abstraction a run of about 200 million iterations would in practice be considered long enough to produce an acceptable ϵ. To obtain these bounds we ran the CFR algorithm for between 2 and 4 billion iterations on each of the bet-sizing abstractions and 5 billion iterations on each of the other abstractions. Abstractions H_T and H_R are not included in the plot as they were only slightly better than POT.

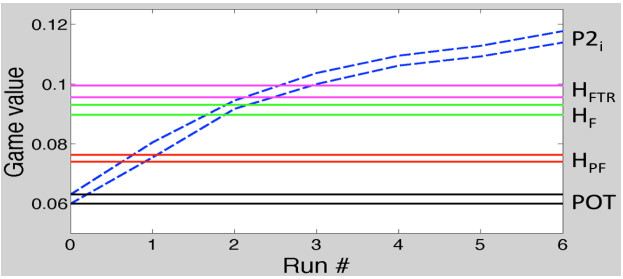


Figure 4: Bounds on game values of various betting abstractions for player two against a fold call pot all-in player one

As shown in Figure 4, abstraction $P2_6$ out performs all other abstractions, beating H_{FTR} by $\approx 19\%$ after 6 short runs. For player one, abstraction $P1_8$ had bounds on its game value very close to those of H_{FTR}, and easily beat the rest of the abstractions (not shown in Figure 4). Additionally, we can see from Table 2 that abstractions $P1_8$ and $P2_6$ are about one third the size of H_{FTR}. Smaller betting abstractions are always favoured over larger ones with similar value, as this allows for refinement of the state abstraction. H_{FTR} is a state of the art action abstraction - it was designed to have restrictions on half pot bet usage identical to those of the winning entries in the no-limit instant run-off division of the 2010 and 2011 computer poker competition.[3]

A histogram showing the distribution of the top 10% of bets for abstractions $P1_8$ and $P2_6$, ranked according to $R^T_{i,|\cdot|}$ and rounded to the nearest 0.1, is shown in Figure 5. We can see that no one bet size is preferred - a variety of bets from 0.2 to 1.5 are used. The important bet sizes are different for each player - $P1_8$'s range from 0.2 to 0.7 along with a couple of large bets > 1, while $P2_6$'s vary from 0.4 to 1.

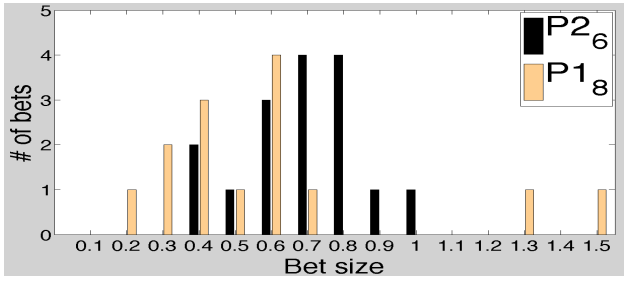


Figure 5: Distribution of important bet sizes for abstractions $P1_8$ and $P2_6$

Conclusion

We have shown that when the approach introduced in our previous paper (Hawkin, Holte, and Szafron 2011) is used to abstract the action space of extensive-form games, the choice of L and H values throughout the tree is of utmost importance. It is, in fact, more important to choose these ranges correctly than it is to find exact values within them. Our approach of using multiple short runs of a regret-minimizing algorithm, followed by adjustments of all L and H values, creates action abstractions in no-limit Texas Hold'em that are both smaller in size and better in game value than current state of the art action abstractions.

The generated abstractions use a wide variety of bet sizes, with the most important ones ranging from 0.2 to 1.5 pot. This result shows the complexity of the action space of no-limit Texas Hold'em, and the need for further work on action abstraction in similar domains. The tendency to use different bet sizes in different situations also has a practical advantage over action abstractions that use a small number of bet sizes. When creating a game-theoretic agent to operate in this domain, the designer must decide on a small set of bet sizes for opponent agents. The large variety of bet sizes used by the agents we created ensures that game-theoretic opponents will have to rely heavily on dynamic translation during matches. Creating such problems for opposing agents is an advantageous property of any action abstraction.

Acknowledgements

We would like to thank the members of the University of Alberta Computer Poker Research Group for their valuable insights and support, and Compute Canada for providing the computing resources used to run our experiments. This research was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), the Alberta Informatics Circle of Research Excellence (iCORE) and Alberta Ingenuity through the Alberta Ingenuity Centre for Machine Learning.

[3] http://www.computerpokercompetition.org/

Appendix - Importance metric

Each iteration, the bet-sizing agents minimize immediate counterfactual regret (Zinkevich et al. 2007). We compute two values:

$$R(L)_i^t = u_i^t(L) - u_i^t(s_i^t) \tag{4}$$

and

$$R(H)_i^t = u_i^t(H) - u_i^t(s_i^t) \tag{5}$$

These two equations are the utility differences for bet-sizing player i betting H or L on iteration t instead of the current effective bet size $s_i^t = B(P(H)_i^t)$. If increasing s_i^t gains value, then $R(H)_i^t > 0$ and $R(L)_i^t < 0$. The opposite is true if decreasing s_i^t gains value. Since all other probabilities are fixed during regret computation, $R(H)_i^t$ is linear in s_i^t. This is true because the amount of money that is won or lost, u_i^t, is linear in s_i^t. Therefore, the magnitudes of these values are proportional to the distance s_i^t is from the range boundaries. For example, as s_i^t gets closer to H, $|R(L)_i^t|$ increases and $|R(H)_i^t|$ decreases proportionally. Therefore, we define

$$R_{i,|\cdot|}^t = |R(L)_i^t| + |R(H)_i^t|. \tag{6}$$

The larger this value is, the more utility we stand to gain by moving this bet. Finally, we define

$$R_{i,|\cdot|}^T = \sum_{t=1}^{T} R_{i,|\cdot|}^t \tag{7}$$

References

Gilpin, A., and Sandholm, T. 2006. A competitive texas hold'em poker player via automated abstraction and real-time equilibrium computation. In *AAAI*, 1007–1013.

Gilpin, A., and Sandholm, T. 2007. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *AAMAS*, 1168–1175.

Gilpin, A.; Sandholm, T.; and Sorensen, T. B. 2007. Potential-aware automated abstraction of sequential games, and holistic equilibrium analysis of Texas Hold'em poker. In *AAAI*, 50–57.

Gilpin, A.; Sandholm, T.; and Sorensen, T. B. 2008. A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In *AAMAS*, 911–918.

Hawkin, J.; Holte, R.; and Szafron, D. 2011. Automated action abstraction of imperfect information extensive-form games. In *AAAI*, 681–687.

Johanson, M. 2007. Robust strategies and counter-strategies: Building a champion level computer poker player. Master's thesis, University of Alberta.

Schnizlein, D.; Bowling, M.; and Szafron, D. 2009. Probabilistic state translation in extensive games with large action sets. In *IJCAI*, 276–284.

Schnizlein, D. 2009. State translation in no-limit poker. Master's thesis, University of Alberta.

Waugh, K.; Schnizlein, D.; Bowling, M.; and Szafron, D. 2009a. Abstraction pathologies in extensive games. In *AAMAS*, 781–788.

Waugh, K.; Zinkevich, M.; Johanson, M.; Kan, M.; Schnizlein, D.; and Bowling, M. 2009b. A practical use of imperfect recall. In *Proceedings of the Eighth Symposium on Abstraction, Reformulation and Approximation (SARA)*, 175–182.

Waugh, K. 2009. Abstraction in large extensive games. Master's thesis, University of Alberta.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2007. Regret minimization in games with incomplete information. In *NIPS*, 1729–1736.